Universiteit Leiden
The Netherlands

Opleiding Informatica

Achieving the
maximal score in Azul

Sara Kooistra

Supervisor:
Dr. Rudy van Vliet

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 08/07/2023

Abstract

Azul is a strategic board game where players tile a mosaic wall, with various game elements affecting their scores. This study aims to determine the theoretically optimal score achievable in the game and explore how this score can be achieved. We conduct an analysis of mosaic wall tiling and its symmetries, followed by the practical implementation of these findings in a game simulation. The research establishes that the maximum attainable score is 240 points, including 95 bonus points. To reach this score, two key considerations must be taken into account: 1) Placing five tiles in separate rows and columns without touching other tiles, and 2) Ensuring all remaining tiles are placed horizontally and vertically adjacent to already placed tiles. Furthermore, we investigate when incomplete permutations of an $n \times n$ board can still lead to an optimal score. The implementation of our findings into a simulation and its results suggest that these tiling considerations may not have significant practical value in game play, as other factors strongly influence both score and win rate.

Contents

1 Introduction

Azul is a strategic board game, released in 2017, as the first of a series of five [Gee]. In this game players tile a mosaic wall of a Portuguese palace. They do so by taking turns picking tiles from the table and later placing these tiles onto their wall. Tiling the mosaic wall gives the players points, and the player with the highest score wins the game. Figure 1 shows a (two-player) game of Azul in progress.

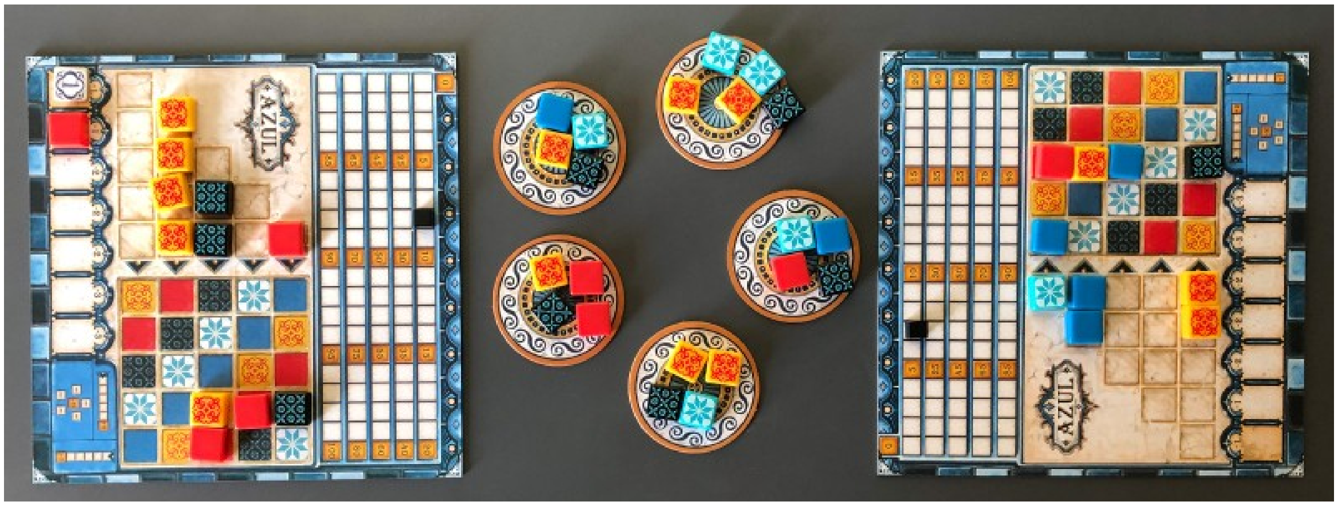


Figure 1: Azul game overview

Research has been done on many popular board games already, to find complexities, algorithms or most-profitable strategies. Examples are Scrabble [LMS15], Chess [FL81] and Rummikub [vRTV16]. Although Azul is an award winning board game [Sch18], and its luck factor is relatively low, no research has yet been done on its most profitable strategy.
Scoring points is a relatively simple aspect of the Azul game. It is, however, not obvious what the highest possible score is or how to get to it. In this thesis we therefore research what the highest possible score is, how this score can be achieved and what we can deduce from this knowledge considering a best playing strategy. To do so, we start with an overview of the game and its rules in section 2. Next, in section 3, the game and in particular the scoring system is thoroughly analysed by determining the highest possible score, lowest possible score, symmetrical patterns and possible tiling orders. The obtained theoretical knowledge will then be used in practice in a simulation of the game, where several algorithms play against each other. The algorithms and the results of the simulations are described in section 4. In section 5 of this thesis, we give our conclusion and describe potential further work.

2 Azul Game Rules

Azul is a game that can be played with a maximum of four players. In this thesis we will focus on the 2-player game, of which the set-up is as follows: There is a bag on the table with 100 tiles in it, 20 tiles each of five different colours. On the table between the two players there are 5 tile repositories called factory displays. Additionally, there is a single white tile with the number one on it. Every player has its own player board with a score line, pattern lines, mosaic wall and floor line. The player board together with the factory displays can be seen in Figure 2. As can be seen in this figure, the mosaic wall is a 5×5 grid where every row and column contains all five tile colours.
The game is played over multiple rounds, and ends when a player has covered a horizontal line of his mosaic wall with tiles. At the start of every round, the five factory displays will each be covered with four random tiles from the bag. Every round consists of two phases:
1. Getting tiles from the factory
2. Tiling the mosaic wall

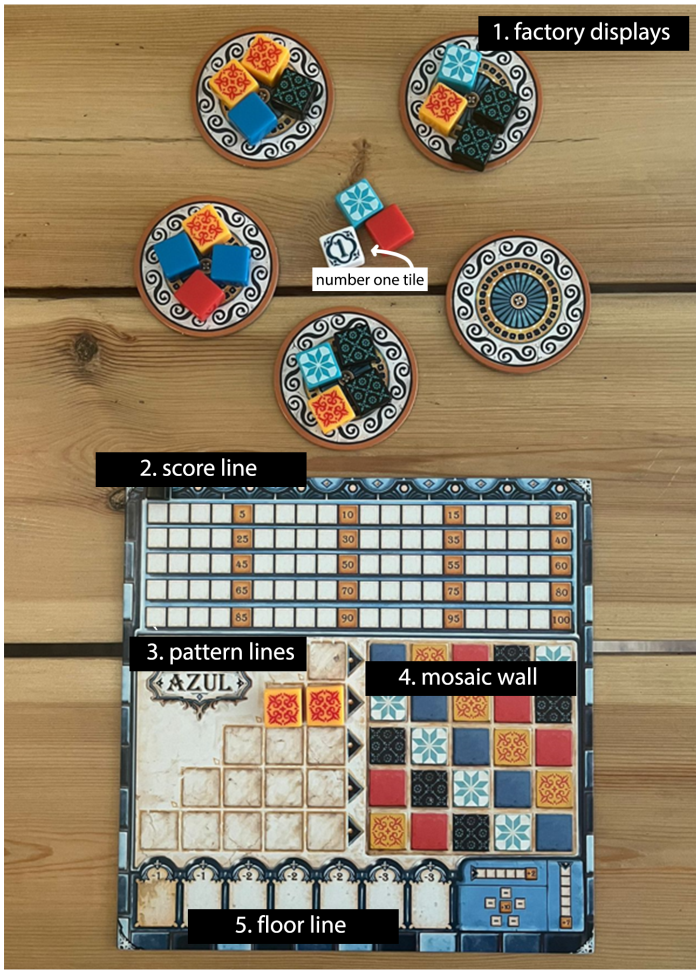


Figure 2: Azul playing field

Phase 1: Getting tiles from the factory

In phase 1, players take turns picking tiles from the factory displays/table. They can choose to:

1. Pick all tiles of one colour from a factory display, and move all the remaining tiles from that factory display to the table.

2. Pick all tiles of one colour from the table. If the white tile with a 1 on it is still in the field, take it and put it on your floor line.

The tile(s) that the player picks must be put on one of their pattern lines. Tiles can only be put on a pattern line when there is not a different colour on that pattern line and the same colour is not already put on that line of the mosaic wall. Tiles that do not fit in the chosen pattern line must be put on the floor line, filling it up from left to right. In the sporadic case that the floor line is already completely covered, the tile can be put aside to be put back in the bag later.
This phase ends when all tiles are picked from the factory displays as well as from the table.

Phase 2: Tiling the mosaic wall

For all pattern lines that are completely filled, players now shift one tile to the corresponding colour on the row of the mosaic wall. This must be done from top to bottom. The other tiles on the pattern line are put aside and will be put back into the bag when the bag is empty.

Scoring system

Every new tile on the mosaic wall gives a score directly when it is shifted on there. The scoring of a new tile is as follows:

- +1 point for the new tile
- +1 point for every tile that is in the same row or column as the new tile and connected to it directly or through other tiles in that row or column.
- +1 point if there are tiles both horizontally and vertically connected to the new tile.

An example of partially tiling the wall with the corresponding scores can be seen in Figure 3.

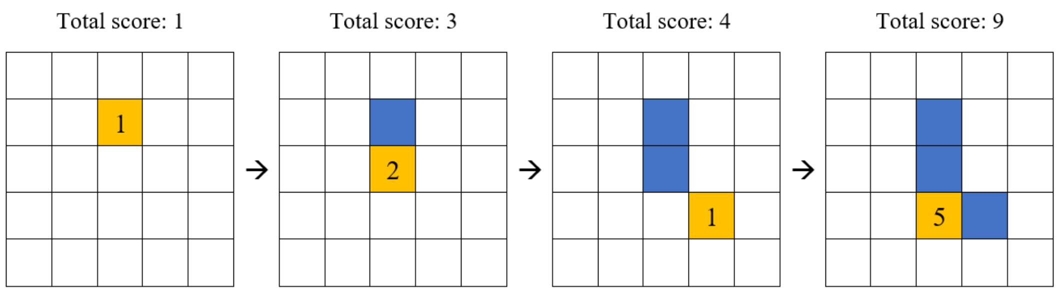


Figure 3: Example of tile scoring. Blue tiles are on the wall already, yellow tiles are newly added tiles with the number in it being the score they yield.

The scored points are added to the players score line. Tiles on the floor line count as negative points, where the first two are −1 point each, the next three are 2 points and the last two are 3 points (as marked above the cells of the floor line). The points on the floor line are added up and then subtracted from the score on the score line. A player's score can, however, never become negative. The game ends at the end of the round in which at least one player has a horizontal line filled on their mosaic wall. Players then get bonuspoints:

- per completed horizontal line, +2
- per completed vertical line, +7
- per completed colour, +10

3 Analysis

3.1 Highest possible score

To answer the research question, we will determine the theoretical maximum score attainable in Azul. Additionally, we will determine how this score can be achieved and what possible tile orders can be used for that purpose. In our analysis we focus on completely tiling the mosaic wall. We mostly ignore the fact that the game ends after the round where the player has filled a first horizontal line, which occur before a complete tiling of the wall.

3.1.1 Determining highest score

To determine the highest possible score, we focus on the second phase of the game, tiling the mosaic wall. Since points are scored solely in this phase, the highest possible score depends only indirectly on the first phase. The order in which the wall is tiled determines the score the tiled wall yields. Tiling a cell that leads to the highest score for that tile, does not always lead to the highest total end score. An example of two identical coverings, a covered 2x2 square, can be seen in Figure 4. Although in order A each individual tile scores the most points possible for that tile, order B leads to a higher total score.

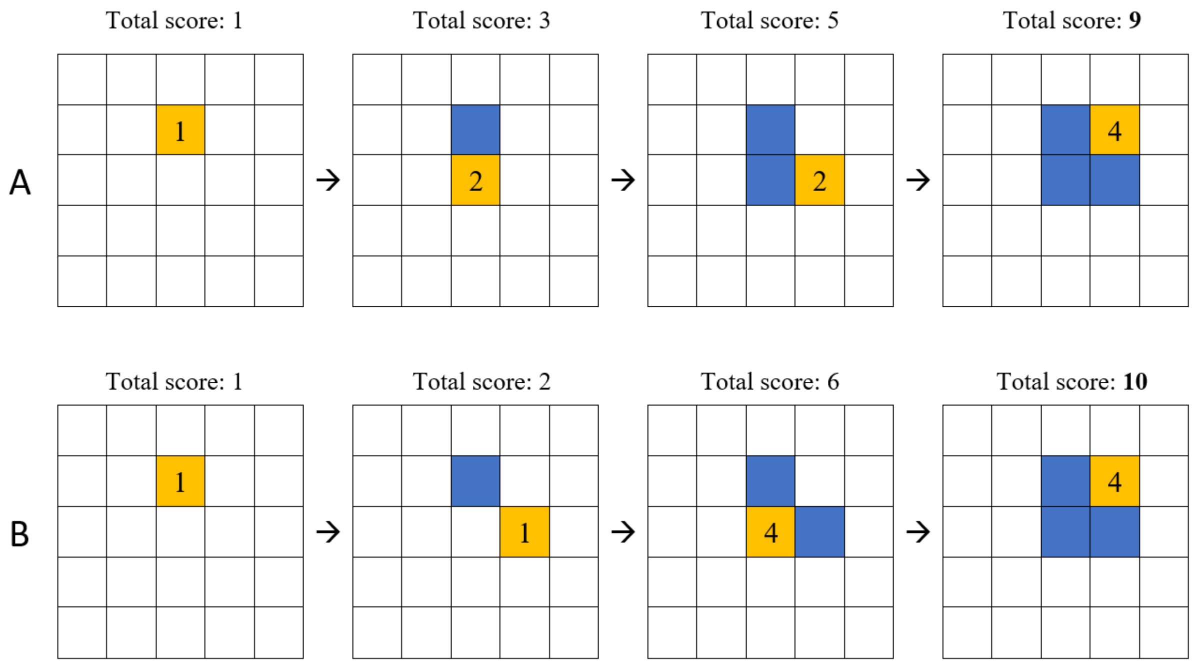


Figure 4: Different order leads to different score

Since each new tile on the wall provides extra points, the highest possible score is logically achieved with a fully tiled wall. As has been explained in section 2, points are achieved for a new tile on the board, all horizontally connected tiles, all vertically connected tiles, and an additional point if there are both horizontally and vertically connected tiles. We can break the score of a new tile into a horizontal and a vertical component. A formula for the tile score could then be:

$$score_x = 1 + \text{number of horizontal adjacents}$$
$$score_y = 1 + \text{number of vertical adjacents}$$

$$score = \begin{cases} score_x & \text{if score}_y = 1 \\ score_y & \text{if score}_x = 1 \\ score_x + score_y & \text{otherwise} \end{cases} \tag{1}$$

If we take a look at completely filling one horizontal line on an empty wall, we achieve the maximum number of points if all new tiles are horizontally adjacent to all present. The score achieved would then be $1 + 2 + 3 + 4 + 5 = 15$. The same holds for a vertical line on the wall. Figure 5 shows how these points add up. There are, of course, other possible orders to fill up a line that will lead to the same maximum score.

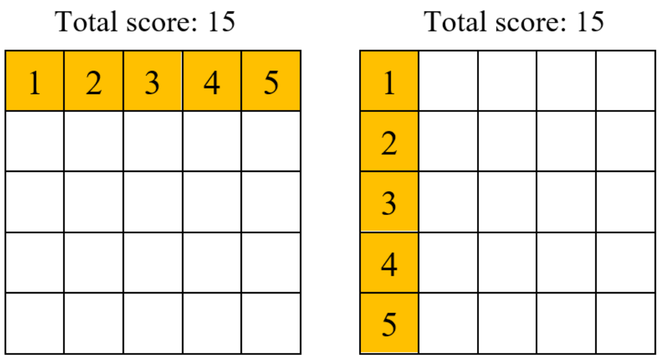
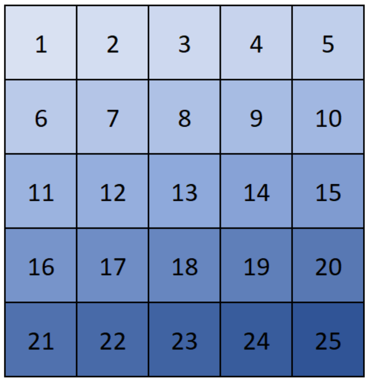


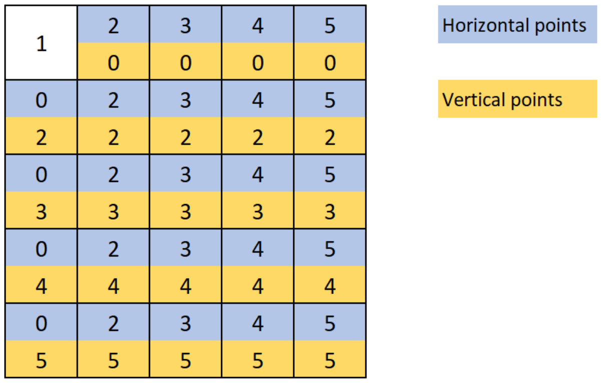
Figure 5: Maximal scores for parts of mosaic wall

If we completely fill up the board, we have five rows and five columns, which would mean a maximum of $(5 + 5) * 15 = 150$ points. This would be the case if all tiles placed on the field are counted in two directions. However, not all tiles can be placed adjacent to tiles in both directions. Thus not all tiles can gain points in two directions and thus the score of 150 points is not achievable. This is because a tile that is placed in an empty row or column, can never be adjacent to both a tile in the same row and a tile in the same column. In other words: such a tile only yields $score_x$ or $score_y$, not $score_x + score_y$ in equation 1. For example, if we tile the 5x5 grid from left to right and from top to bottom, we gain points as illustrated in Figure 6, which leads to 141 points.



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

(a) Simple tiling order

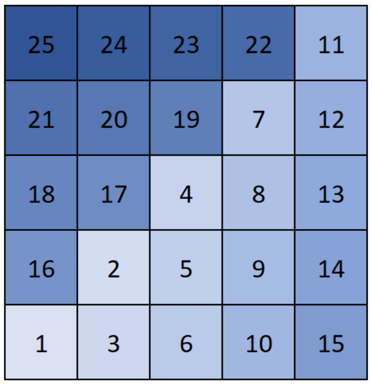

Horizontal points
Vertical points

1	2	3	4	5
	0	0	0	0
0	2	3	4	5
2	2	2	2	2
0	2	3	4	5
3	3	3	3	3
0	2	3	4	5
4	4	4	4	4
0	2	3	4	5
5	5	5	5	5

(b) Horizontal and vertical score

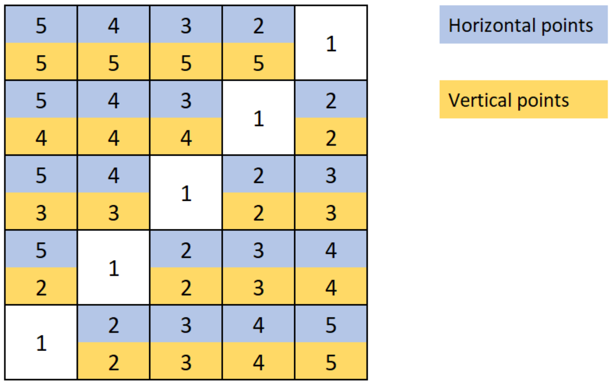
Figure 6: Score components for simple tiling order, yielding a score of 141 points

To lose as few points as possible we want to minimize the number of ”new tiles” on the board, i.e. tiles that do not yield points in two directions. At least every row and every column will need one ”first” tile, which is always a tile that does not yield points in two directions. Therefore we will place all new tiles in both a new row *and* a new column, for example by creating a diagonal line of new tiles. This leads to five tiles *not* yielding points in two directions, and thus to a maximum score of 145 points. We must make sure that all tiles apart from these five are tiled adjacent to a tile both in horizontal and in vertical direction. This will automatically lead to all rows and columns being filled up from out one of these five existing tiles, since every new tile must be adjacent to two existing tiles, and every row and column starts with only one tile in it. An example of a tiling pattern that leads to the highest score with its vertical and horizontal components split up can be seen in Figure 7.



25	24	23	22	11
21	20	19	7	12
18	17	4	8	13
16	2	5	9	14
1	3	6	10	15

(a) Tiling order for highest score


Horizontal points
Vertical points

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

(b) Horizontal and vertical score

Figure 7: Score components for optimal tiling order

The bonus points still need to be added to this score to get to the highest possible total score. We add 2 points for every completed horizontal line, 7 points for every completed vertical line, and 10 for every completed colour. Since the board is totally covered we get $145 + 2 \times 5 + 7 \times 5 + 10 \times 5 = 240$ points.

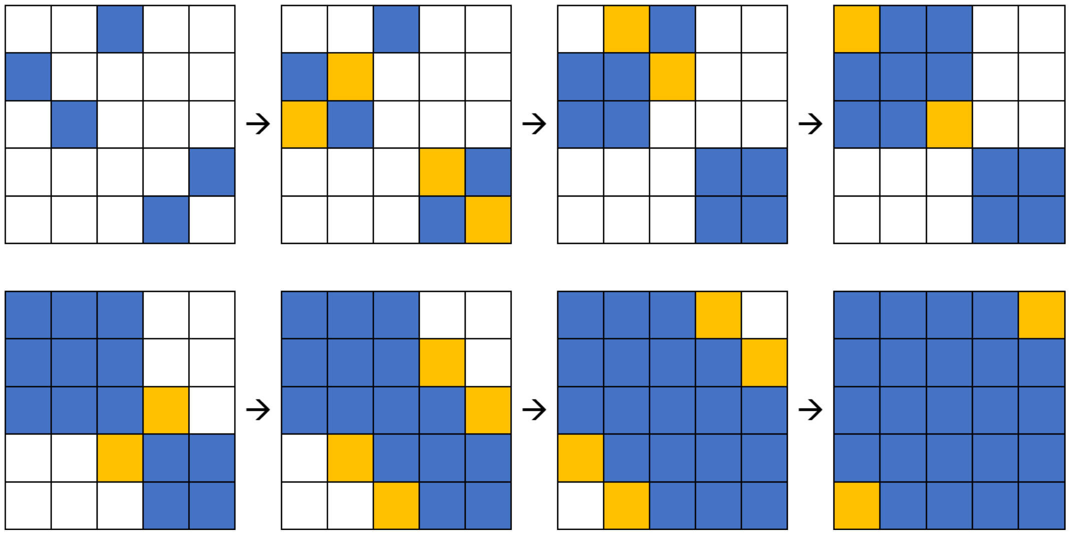
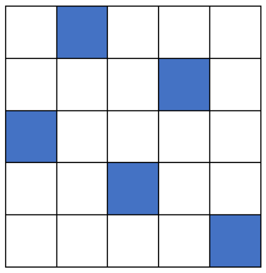


Figure 8: Example of how we can fill up a 5-tile wall configuration to achieve the maximum score

3.1.2 Configuration of five loose tiles

The five tiles that are put on the wall in a new row and column, can be tiled in many different orders and configurations. The exact order in which these five tiles are placed has no influence on gaining or not gaining the highest possible score. The positions of these tiles, however, do matter, since not every configuration with every tile in a separate row and column can eventually lead to the highest possible score. In this section, we investigate which configurations can lead to the highest score, and which cannot. Figure 8 shows an example of a five tile configuration that can lead to the maximal score. It also shows how we can fill up the wall by only putting tiles on cells that are both horizontally and vertically adjacent to a covered cell.
Figure 9a shows a configuration of the board with five tiles that cannot lead to the maximal score, since no empty cell is in both directions adjacent to a covered cell.


(a)

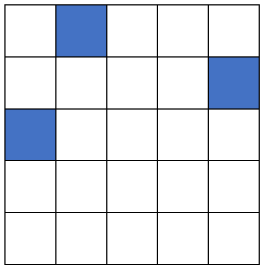

(b)

Figure 9: Two wall configurations that cannot lead to the maximum score

Not only with five tiles on the board, but also with fewer tiles placed, it is possible that an optimal score has already become unattainable. An example is the configuration in Figure 9b.
We now want to find out when a configuration can still lead to the earlier determined highest score, and when it cannot. To provide a more comprehensive understanding of the game dynamics, and to increase the likelihood of discovering patterns within the configurations, we will generalize this

to an $n \times n$ board. The maximum score can be defined by $n * (n + 1)$ times n columns minus n points, being $n^3 + n^2 - n$ points.

We define the following problem:
OptimalAzulTiling: Given an $n \times n$ board containing $k \leq n$ tiles, each having both a unique row and a unique column. Is it possible to tile the complete board in an order that yields an Azulscore of $n^3 + n^2 - n$ points excluding bonus points?
Figure 8 shows a yes-instance to this problem, Figure 9a and 9b show no-instances to this problem. We recall that, to achieve the highest score on an $n \times n$ board, our tiling pattern should meet the following conditions:

- n tiles are placed in a new (meaning currently empty) row and a new column.
- At the time of tiling, the other $n(n-1)$ tiles must be adjacent both horizontally and vertically to other tiles present on the board.

To check if an instance is a yes-instance to the problem, we can test all possible ways of expanding the current configuration to n tiles (all in unique rows and columns) and after that cover all cells with both horizontal and vertical adjacents until there are no empty cells left with both horizontal and vertical adjacents. The second part can be done by repeatedly iterating over all cells of the board and adding tiles to empty cells that are both vertically and horizontally adjacent to a tiled cell. We will stop when all cells are tiled or when no tiles are added to the wall after iterating over all cells. In the worst case we add one tile every time we have checked all n^2 cells of the board, which means we need to check all n^2 cells n^2 times. This can thus be done in $\in O(n^4)$ with n being the board size. The first part, however, trying all possible configurations of one tile per row and per column, has a higher time complexity. With k tiles already placed on the board, the number of possible configurations is $(n - k)!$. The worst case complexity of this algorithm is thus in $O(n!)$. Given a configuration with $k \leq n$ tiles, a solution to the problem can be guessed and verified in polynomial time. This proves that the decision problem is in NP.
Can we generalise the yes- and no-instance to a form so that we can find a more efficient algorithm? Or could this decision problem be NP-complete? We can denote a wall configuration as an array of numbers, where an element's index is the row number and its value the column number. The permutation from Figure 8 would then be written as $(3, 1, 2, 5, 4)$, Figure 9a would be $(2, 4, 1, 3, 5)$ and Figure 9b would be $(2, 5, 1, \sqcup, \sqcup)$. The permutation (2,4,1,3,5) contains the subpermutation $(2, 4, 1, 3)$. This subpermutation represents four tiles (w, x, y, z) that vertically appear in exactly that order, but whose horizontal positions are ordered as y < w < z < x. A subpermutation with this property is called a permutation pattern $(2, 4, 1, 3)$. The other permutation that is a no-instance to our decision problem, $(2, 5, 1, \sqcup, \sqcup)$ can only be completed in two ways, which are $(2, 5, 1, 3, 4)$ and $(2, 5, 1, 4, 3)$. Both of these contain the permutation pattern $(2, 4, 1, 3)$ as described.

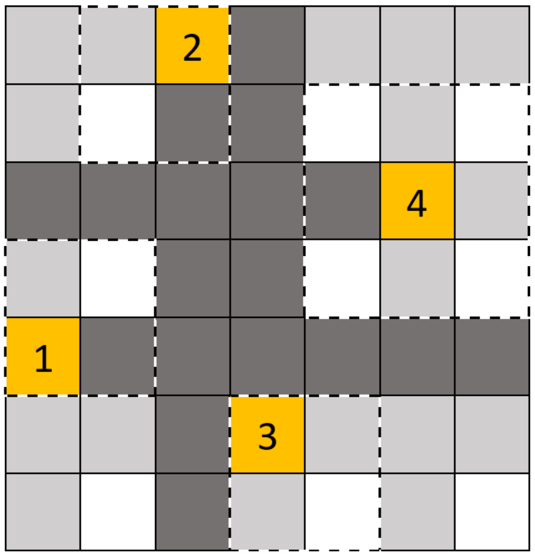


Figure 10: Interlocked boxes for a (2,4,1,3) permutation

Permutations with the pattern (2,4,1,3) lead to a situation where the four tiles are interlocked by each other and can in no way be connected anymore. The same holds for the mirrored image of this pattern, (3,1,4,2). Figure 10 shows how a (2,4,1,3)-pattern creates interlocked boxes. The light and dark grey squares on the board represent the rows and columns where a tile has already been placed, and therefore no unconnected tile can currently be placed to achieve an optimal score. The white cells are cells where the $(n - k)$ "first" tiles can still be placed. The boxes surrounded by dotted lines may still be filled up following the two conditions stated above. The four tiles that were already placed, however, create borders between the boxes, which makes it impossible to connect them following the conditions for an optimal score. For example, tile 2 and its surrounded dotted area is locked away from tile 1 by tile 4, from tile 4 by tile 3, and from tile 3 by tiles 1 and 4.

We will now have a closer look at complete permutations where $k = n$. Permutations containing neither the pattern (3,1,4,2) nor the pattern (2,4,1,3) are called separable permutations [BBL98]. Separable permutations are permutations that can be represented by a separating tree. This is a tree where the elements of the permutation are represented by the leaves of the tree, and all nodes are either positive nodes or negative nodes. In positive nodes, all descendants on the left are smaller than the ones on the right and in negative nodes it is the other way round. An example of a separable permutation with its separating tree can be seen in Figure 11.

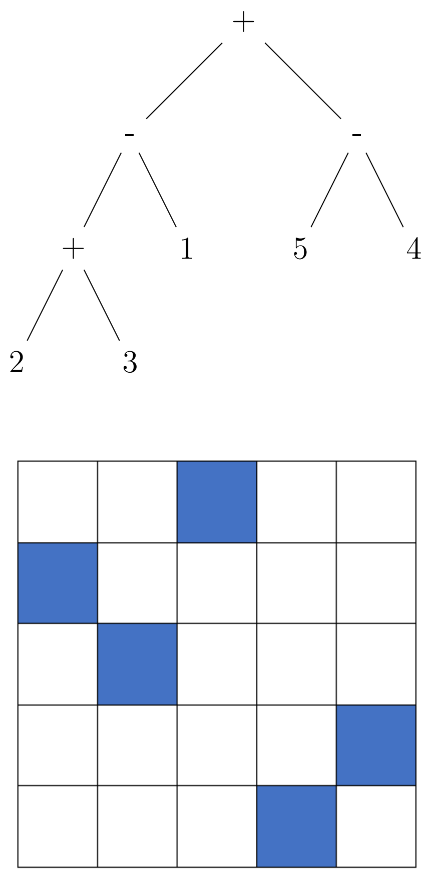


Figure 11: Separable permutation (3,1,2,5,4) with its separating tree

Julian West showed that the number of separable permutations of length n is equal to the $(n-1)$th Schröder number [Wes95]. We have already seen that the non-separable permutations are no-instances to our decision problem. This means that if the number of yes-instances of all configurations with n tiles on an $n \times n$ board is equal to the $(n-1)$th Schröder number, then all separable permutations should be yes-instances to our decision problem.
We have written a simple program in C++ where all permutations of n tiles are checked. The program calculates how many of these permutations are yes-instances to our decision problem. The results for $n = 1$ to $n = 11$ can be found in Table 1.
The number of yes-instances indeed match the $(n-1)$th Schröder number. This proves that for $n = 1$ to $n = 11$ the only n-tile configurations that can not lead to the highest score are configurations with the pattern $(2, 4, 1, 3)$ or $(3, 1, 4, 2)$. This leads to the following conjecture for general $n \geq 1$: "An instance of AzulOptimalTiling for $n \geq 1$ is a yes-instance, if and only if it is a complete permutation without $(2, 4, 1, 3)$- and $(3, 1, 4, 2)$-patterns, or it is an incomplete permutation that can be completed without creating these patterns" .
Assuming that our conjecture is correct, the only thing we need to do to decide if a(n incomplete) permutation is a yes-instance to our problem, is check if it does not contain the above patterns and can be completed without creating the patterns. It has been proven already that when only looking at separable permutations, the problem to decide if a general pattern ϕ (not necessarily of length 4) can be found in a permutation ρ is NP-complete [NRV16]. On the other hand, we have found that

n	number of yes-instances	n	number of yes-instances
1	1	7	1,806
2	2	8	8,558
3	6	9	41,586
4	22	10	206,098
5	90	11	1,037,718
6	394		

Table 1: Number of yes-instances for $n = 1$ to $n = 11$

deciding if the permutation pattern $(3, 1, 4, 2)$ or $(2, 4, 1, 3)$ occurs in a complete permutation is a polynomial problem. It is still open whether or not it is possible to decide in polynomial time if an incomplete permutation can be completed without creating the patterns $(3, 1, 4, 2)$ and $(2, 4, 1, 3)$. Perhaps, this problem is NP-complete. This thesis does not delve further into this question. It would be interesting to pursue further research on this topic.

3.1.3 Total number of possible orders

We have determined that the highest achievable score for the original game (with $n = 5$) is 240 points, and we have also found a way to get to this highest possible score. However, if we mirror this tiling pattern over a vertical or horizontal axis, we also get a tiling pattern with a score of 240 points. How many sequences are there that lead to this optimal score? To determine this, a program was written in C++. The program was also used to determine the lowest possible score with 25 tiles on the board, which we will discuss in Section 3.2.1.
We now explain how our program computes the number of sequences leading to the maximal score. In the explanation, we use the term "picture" for (partial) tilings of the board. We have implemented bottom up dynamic programming, based on the different possible pictures. This way, we do not need to examine all 25! sequences. In our program, all $2^{25} = 33.55$ million pictures are represented by binary numbers. Every digit here stands for a cell of the 5×5 grid. The cells are numbered from left to right and from top to bottom. Every covered cell is represented by a 1, every empty cell is represented by a 0. As an example, in Figure 12 the picture corresponding to [0010011000010000000101010] is shown.

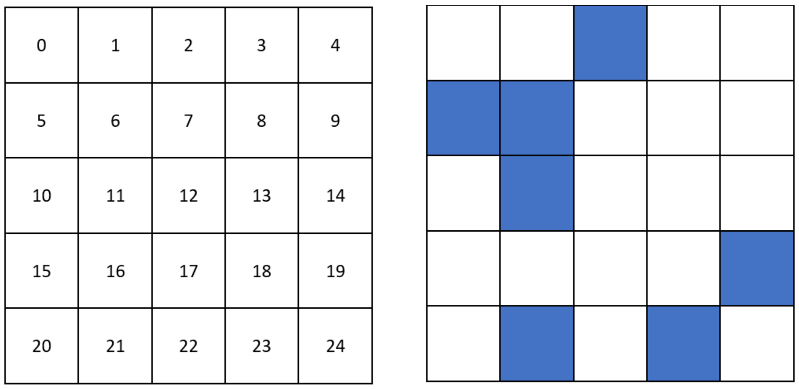


Figure 12: Covering corresponding to [0010011000010000000101010]

We define objects for all possible pictures, containing i.a. their corresponding bitstring and an integer variable *score*, which is set to 0 at the start of the program. Now, from $n = 1$ to $n = 25$, the program iterates over all pictures with n tiles and determines the maximal score that it can be tiled for. To do so, it walks over all pictures with $n - 1$ tiles that can precede the current picture, adds up their score and the score of the new tile, and checks if it is larger than the score of the current picture. If so, the score is updated to this new highest score.
To find the number of possible patterns the wall can be tiled in to get to the highest score, every object has a counter variable. When a score is found that is larger than the current score, the counter is set equal to the counter of the preceding picture. If a score is found that is equal to the current score, the counter is incremented with the counter of the preceding picture.
An example of this process with a 2×2 mosaic wall can be seen in Figure 13. The orange lines represent routes that lead to an optimal score.

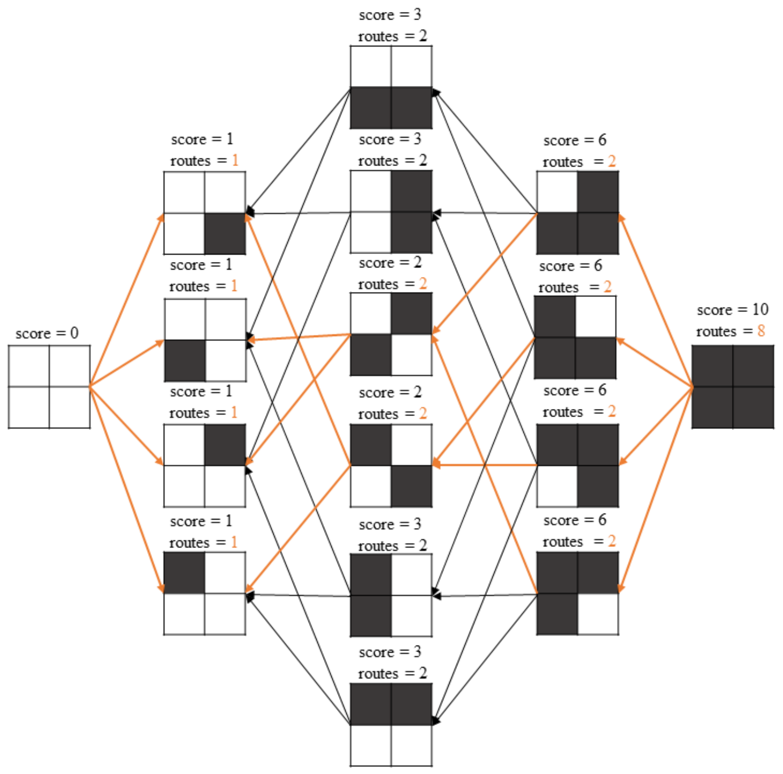


Figure 13: Illustration of determining number of most-profitable tiling patterns

Pseudocode
The pseudocode of the program to calculate the number of routes to the maximum score is as follows:

```
1:  for i=0; i≤ $n^2$; i++ do
2:      for j=0; j<number of possible pictures with i tiles; j++ do
3:          currentCovering = find covering[i][j] in allCoverings;
4:          currentCovering.score = 0;
5:          currentCovering.numberOfRoutes = 1;
6:          for k=0; k< $n^2$; k++ do
7:              if bit k of currentCovering is set then
8:                  // define lastCovering as currentCovering with bit k unset
9:                  lastCovering = index of currentCovering $-2^{n^2-1-k}$;
10:                 score = lastCovering.score + tilescore(lastCovering);
11:                 if score ≥ currentCovering.score then
12:                     if score > currentCovering.score then
13:                         currentCovering.score = score;
14:                         currentCovering.bestPrevious = lastCovering;
15:                         currentCovering.numberOfRoutes = 0;
16:                     currentCovering.numberOfRoutes += lastCovering.numberOfRoutes;
17: return currentCovering.numberOfRoutes;
```

Results
To find the number of possible sequences in which the wall can be tiled to get to 240 points, we look at the picture with 25 tiles. The score at this picture is 145 points as expected, and the number of routes is 1.57×10^{14}. The total number of sequences of tiling the complete board is $25! = 1.55 * 10^{25}$. This means that around $1 * 10^{-9}\%$ of all tiling patterns leads to the highest possible score.

3.2 Lowest possible score for tiled wall

3.2.1 Expected lowest possible score

To get a better image of the range of scores we will also determine the lowest possible score that can be achieved with a fully covered mosaic wall.
For the lowest possible score, we again look at the possible score for a single row on the wall. The tree in Figure 14 shows all possible sequences of scores when we put five tiles in a row or column. The first tile will always add one point to the score, the last tile five. The smallest number of points that can be achieved is 11.

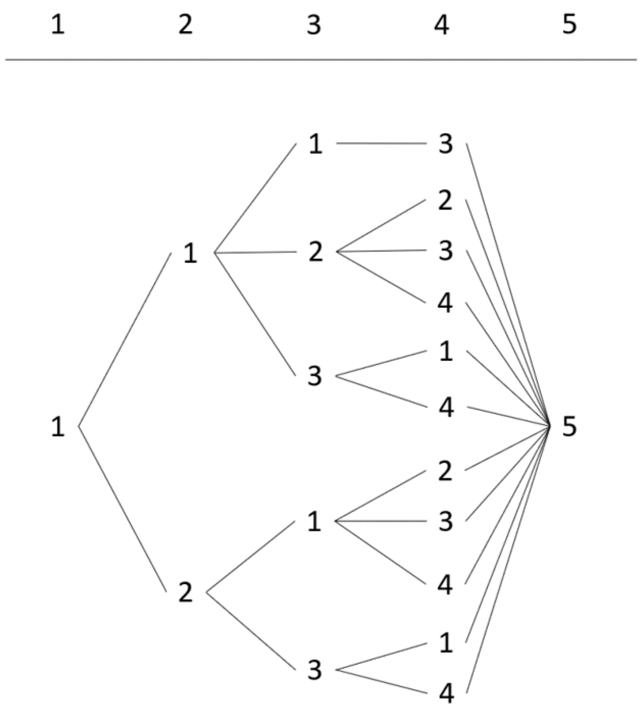


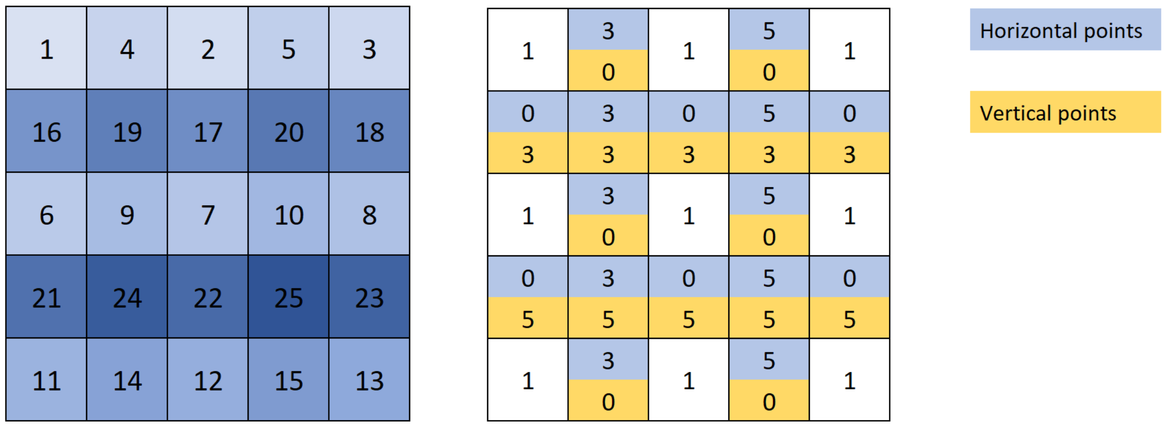
Figure 14: Possible points per tile in single row or column

One possible order to get to eleven points can be seen in Figure 15. In this order, tiles are placed in detached cells as long as possible, before choosing cells that are adjacent to a covered cell.

1	4	2	5	3

Figure 15: Possible order for least points in row: $1 + 1 + 1 + 3 + 5 = 11$

When we use this pattern to fill up the whole wall in both dimensions, we get the tiling order shown in Figure 16. This order leads to a total score of 89 without bonus points. To check if this is actually the lowest score we can get to with 25 tiles on the wall, we will alter and use the program that was described in section 3.1.3.



(a) Tiling order for highest score (b) Horizontal and vertical score

Figure 16: Score components for lowest possible score

3.2.2 Checking lowest possible score

To find the minimum score with 25 tiles we initialise the score of covering 0 (the empty grid) to 0 and alter the algorithm from Section 3.1.3 as follows, with modifications in red:

```
for i=1; i≤ boardSize^2; i++ do
    for j=0; j<number of possible pictures with i tiles; j++ do
        currentCovering = find covering[i][j] in allCoverings;
        currentCovering.score = 1000;            ▷ Initialise higher than maximum possible score
        for k=0; k< boardSize^2; k++ do
            if bit k of currentCovering is set then
                \\ define lastCovering as currentCovering with bit k unset
                lastCovering = index of currentCovering - $2^{boardSize^2-1-k}$;
                score = lastCovering.score + tilescore(lastCovering);
                if score ≤ currentCovering.score then
                    if score < currentCovering.score then
                        currentCovering.score = score;
                        currentCovering.bestPrevious = lastCovering;
                        currentCovering.numberOfRoutes = 0;
                    currentCovering.numberOfRoutes += lastCovering.numberOfRoutes;
return currentCovering.score;
```

The program for the minimum possible score with 25 tiles returns 89. This result confirms the expectation that was described in Section 3.2.1. The tiling pattern in Figure 16 is thus indeed a pattern to achieve the lowest possible score with 25 tiles.

3.3 Symmetries

The Azul mosaic wall is a 5×5 grid. There are in total 25 cells which may be tiled or untiled, and thus $2^{25} = 33,554,432$ possible board coverings, i.e. pictures.
We can divide all pictures into groups defined by the number of tiles needed for the picture, like we did in our algorithms. With 0 tiles there is only one possible picture, which is the empty board. Likewise, there is one possible picture with 25 tiles, which is the full board. In general, the number of pictures with k tiles and n cells is $C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$. The distribution of the number of pictures per number of tiles for a 5×5 board can be seen in Figure 17.

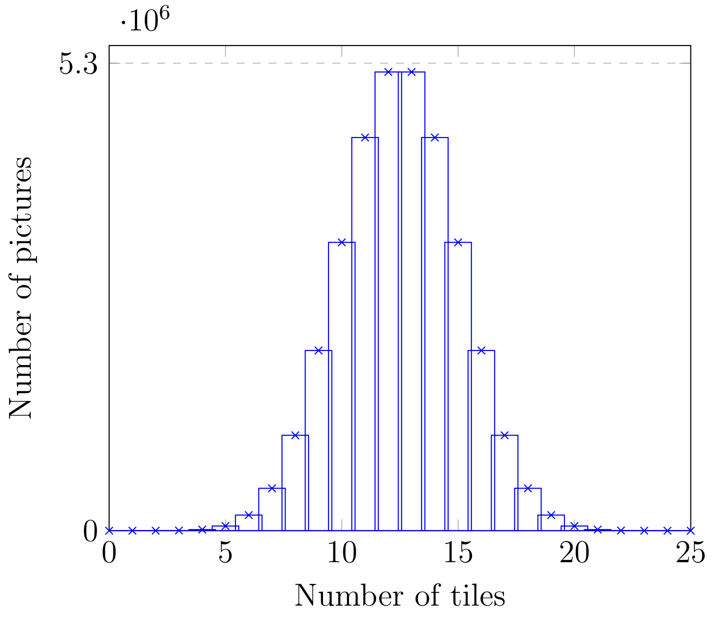


Figure 17: Distribution of pictures over number of tiles

When we put a tile the top left cell of an empty 5×5 wall, we have a similar situation as when we put a tile on the bottom left cell of the empty wall. You only need to rotate the second board with 90 degrees or flip it over horizontally to find the first one. Rotating or mirroring a board has no influence on the score that can be achieved (disregarding the bonus points). The resulting boards are called symmetric. An example of a picture and its vertically mirrored picture can be seen in Figure 18.
The set of symmetries that are applicable is the following: reflection symmetry - horizontal, vertical, diagonal left, diagonal right; rotational symmetry - 0 °, 90 °, 180 °, 270 °. This set of symmetries forms a group, meaning that applying multiple symmetries will always lead to one of these eight original symmetries. We define an object as a picture and all pictures we can find by using the group of symmetries described above. If all pictures would have seven symmetrical pictures, that would mean that there are only $\frac{2^{25}}{8} = 4,194,304$ unique objects, meaning symmetrically distinct coverings. However, there are also pictures that do not change when they are for example flipped horizontally. There are even pictures that are symmetric both rotational and reflective, meaning they do not have any symmetrical picture, e.g. the empty grid.

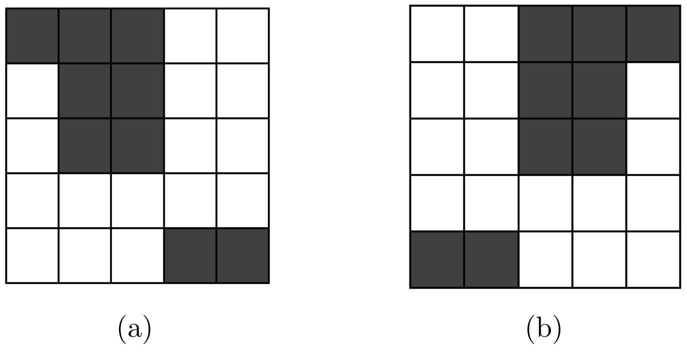


Figure 18: Vertically mirrored pictures

Burnside's lemma

Burnside's lemma is a theorem in group theory that can be used to count distinct objects of a set taking symmetry into account [Jin18]. These distinct objects are called orbits. We can use Burnside's lemma to calculate how many orbits there are in our game. To put it differently, we can calculate the number of unique configurations of the field, being in no way symmetrical to each other.
Burnside's lemma states [Ol20] that the number of objects $= \frac{\text{sum of symmetrical pictures}}{\text{number of symmetries}}$. To find the number of objects we should thus first find the sum of symmetrical pictures. To do so, for every symmetry we need to determine the number of pictures being symmetrical over that symmetry line or angle. We determine the number of fixed elements/cells for all eight symmetries, i.e. the cells of which the state is fixed by other cells. This can be seen in Figure 19.

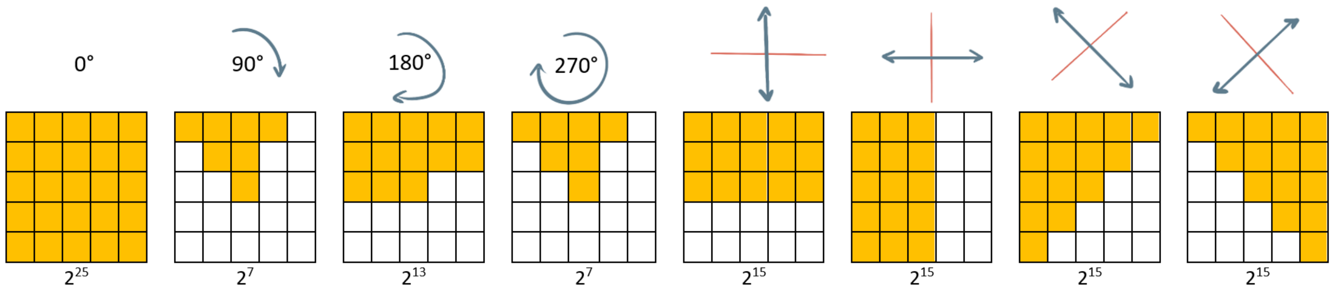


Figure 19: Fixed elements per symmetry (white cells) with the corresponding number of invariant pictures

The number of objects (so the number of unique pictures) is then given by the following formula:

$$\begin{aligned}
\text{objects} &= \frac{\text{sum of symmetrical pictures}}{\text{number of symmetries}} \\
&= \frac{2^{25} + 2^{7} + 2^{13} + 2^{7} + 2^{15} + 2^{15} + 2^{15} + 2^{15}}{8} \\
&= 4,211,744 \text{ objects}
\end{aligned}$$

So there are $4.2 * 10^6$ objects, which is a little more than $\frac{1}{8}$ of the total number of pictures.
Since symmetric pictures can lead to equal scores, we could alter our algorithms in Section 3.1 and 3.2. We could restrict our previously described algorithms to one picture per orbit. However, this would only speed up our algorithm by less than a factor 8, and combined with the time required to determine the orbits, it would not make a significant difference. Therefore, we decided not to pursue this approach.

4 Algorithms

In the theoretical analysis described in Section 3, we mainly determined how to achieve the highest score in a perfect game. However, Azul contains many game elements that affect the ability to play the “perfect” game. For example, available tile colours are randomized, there is an opponent that will make moves taking away tiles and there is a tile in the middle which is a negative point and should always be picked by one of the two players. The main conclusions from our theoretical analysis are that it is beneficial to have five “loose tiles” in separate rows and columns, and to place all other tiles on a cell that is adjacent both a horizontal and a vertical cell that is already tiled. To research if, despite the fact that there are many game components, our findings have a positive effect on a player’s win percentage, a simulator of the game was written in C++. The player can select its move by three different algorithms, a Greedy algorithm, a “smarter” algorithm and a strategic algorithm. Players with different algorithms will play games against each other. The different algorithms are described below.

4.1 Greedy algorithm

The greedy player is a player that is focused on picking the largest number of tiles it can fully put onto a pattern line. If this is not possible, the algorithm chooses a move that leads to the fewest tiles on the players floorline (negative points). The pseudocode for the greedy algorithm is as follows.

```
for int i=0 to number of possible moves do
    for int j=0 to number of pattern lines do
        if tiles of move can fully be put in patternline then
            if numberOfTiles > largest then
                largest = numberOfTiles;
                bestMove = move i in j;
                least = 0;
            else
                if least != 0 then
                    tilesTooMany = abs(patternLine_space - numberOfTiles);
                    if tilesTooMany < least then
                        least = tilesTooMany;
                        bestMove = i in j;
return bestMove;
```

4.2 Smart algorithm

The smart player is a player that is focussed on making a move which will fill up pattern lines as much as possible, i.e. leaves the least untiled cells on its patternline. In case of a tie, it prefers a move where the tile on the mosaic wall will be put adjacent to another tile. The pseudocode for the smart algorithm is as follows:

```
for i=0 to number of possible moves do
    for j=0 to number of pattern lines do
        if tiles of move can fully be put in patternline then
            movefound = true;
            whitespace = patternLine_space - numberOfTiles;
            if whitespace <= least_whitespace then
                if whitespace < least_whitespace then
                    least_whitespace = whitespace;
                    one_adjacent_move = false;
                    most_tiles = 0;
                if !one_adjacent_move then
                    checkAdjacents();
                    if horizontal_adjacent || vertical_adjacent then
                        one_adjacent_move = true;
                        bestMove = currentMove;
                    else
                        if currentMove.size > most_tiles then
                            bestMove = currentMove;
                            most_tiles = currentMove.size;
        else
            if !movefound then
                tilesTooMany = abs(patternLine_space - numberOfTiles);
                if tilesTooMany < least_remaining then
                    bestMove = currentMove;
                    least_remaining = tilesTooMany;
return bestMove;
```

4.3 Strategic algorithm

The strategic algorithm works similar to the smart algorithm, apart from two things. The algorithm prefers tiles in the diagonal line of the mosaic board in the first round. Also, it prefers moves where the tile will end up with two adjacent tiles above one adjacent tile.

```
for i=0 to number of possible moves do
    for j=0 to boardSize do
        if tiles of move can fully be put in patternline then
            movefound = true;
            whitespace = patternLine_space - numberOfTiles;
            if whitespace <= least_whitespace then
                if whitespace < least_whitespace then
                    least_whitespace = whitespace;
                    diagonal_move = false;
                    one_adjacent_move = false;
                    two_adjacent_move = false;
                    most_tiles = 0;
                if !diagonal_move && firstround then
                    if moveInDiagonal() then
                        bestMove = currentMove;
                        diagonal_move = true;
                if !diagonal_move && !two_adjacent_move then
                    checkAdjacents();
                    if horizontal_adjacent && vertical_adjacent then
                        bestMove = currentMove;
                        two_adjacent_move = true;
                if !diagonal_move && !two_adjacent_move && !one_adjacent_move then
                    checkAdjacents();
                    if horizontal_adjacent || vertical_adjacent then
                        one_adjacent_move = true;
                        bestMove = currentMove;
                    else
                        if currentMove.size > most_tiles then
                            bestMove = currentMove;
                            most_tiles = currentMove.size;
        else
            if !movefound then
                tilesTooMany = abs(patternLine_space - numberOfTiles);
                if tilesTooMany < least_remaining then
                    bestMove = currentMove;
                    least_remaining = tilesTooMany;
return bestMove;
```

4.4 Experiments and Motivation

Several experiments have been conducted to test the algorithms and thus the previously established principles in practice. On every run, two algorithms were chosen to play against each other. To ensure the validity of the results, and minimise the potential for bias, every test was done by playing ten series of 1000 games. Also, both the starting player and the order of tiles in the bag/on the factory tiles were randomly selected every single game.
Since there is a factor of "luck" in the game, we must perform a sufficient number of simulations to draw reliable conclusions about the influence of a particular algorithm on the win percentage and scores. We start with doing 1000 simulations where the two players are playing with the same algorithm. This way we can check if 1000 simulations is enough to eliminate the influence of luck from our results.
To check the two main points we took from our theoretical analysis, i.e. focus on the diagonal line and on adjacent tiles, we simulate the game with a strategic algorithm versus the smart algorithm. We run three tests with the following preferences for the strategic algorithm:

1. Both diagonal line and two adjacents
2. Only the diagonal line
3. Only the two adjacents

This way, we can check both the impact of the two focal points together and their individual effects. After this, we will also test the win percentage of the smart algorithm against the greedy algorithm, and the strategic algorithm against the greedy algorithm.

4.4.1 Results

We start with the greedy algorithm playing against itself, to check how many runs are needed to draw reliable conclusions. Figure 20 shows the average deviation from 50% for 10 to 10,000 runs. At 1000 runs, the deviation is around 1%. This means that one player won around 51% of the games. This deviation can be caused by the luck factor present in the game. The influence of this luck factor is largely, but not completely, averaged out by doing 1000 runs. To avoid making the number of runs too large while still obtaining reliable results, we will therefore work in batches of 1000 runs.

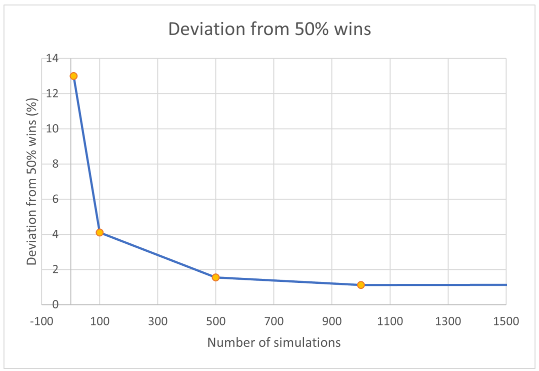


Figure 20: Influence number of runs on average win percentage

Series	Diagonal and two adjacents	Diagonal	Two adjacents
1	52%	50%	55%
2	52.6%	47.7%	54.8%
3	53.4%	50.2%	52%
4	51.4%	51.3%	51.4%
5	53.4%	49.4%	52.7%
Average	**52.56%**	**49.72%**	**53.18%**

Table 2: Percentage of wins for strategic algorithm against smart

Now we let the smart algorithm play against the three variants of the strategic algorithm. We perform five series of 1000 runs for each experiment. We do multiple series of 1000 runs for error reduction and to increase the reliability of the results. The results of 5 $\times$ 1000 runs can be found in Table 2

We can see that, on average, the strategic algorithm wins more often over the smart algorithm than the other way around. However, when we remove the preference for two adjacent tiles and use only the diagonal as a preference, the smart algorithm wins 0.56% more often than the strategic. This is a negligible difference. Conversely, when we forget about the diagonal and give preference only to the two adjacents, the strategic algorithm wins 53.18% of the time. From this, we can conclude that preferring two adjacents above one adjacents leads to a higher percentage of wins. Preferring the diagonal however, does not have the positive effect we had expected it to have.

Finally, we let both the strategic (diagonal and two adjacents variant) algorithm and the smart algorithm play against the greedy algorithm. The results can be found in Table 3.

Series	Smart	Strategic
1	95.4%	96%
2	95.3%	96.3%
3	95.1%	95.2%
4	94.9%	94.9%
5	95.6%	96.1%
Average	**95.26%**	**95.7%**

Table 3: Percentage of wins for smart and strategic algorithm against greedy

Both algorithms have a remarkably high win rate compared to the greedy algorithm. There is however no significant difference between the win percentage of the smart and the strategic algorithm.

5 Conclusion and Further Research

Azul is a board game with a variety of game elements. In this research, we mainly focused on the mosaic wall and implemented our most important findings into a simulation of the game. The highest score that can be obtained by tiling the mosaic wall turns out to be 145 points. By adding the bonus points that a player would get for a fully covered wall, a player can reach a maximum of 240 points. To achieve this score, the wall should be tiled in a specific way, where five tiles that are placed in an at that moment untiled row and column. It is also important that there is no (2,4,1,3)- or (3,1,4,2) pattern in the permutation of these five tiles. It is suspected that the problem to determine if this pattern can still be prevented in incomplete permutations for general board sizes n is NP-complete. Future research could be done to prove (or refute) this hypothesis.

Apart from the five tiles described above, other tiles can be put in different orders. Important is that all these tiles are placed both horizontally and vertically adjacent to an already covered cell. In total, there are $1.57 * 10^{14}$ possible routes the 5×5 wall can be covered in to achieve an optimal score.
We examined if we could minimise the number of instances of the board by only looking at one picture of every orbit. Burnside's lemma was used to determine the number of distinct configurations of the board, keeping symmetry into account. It was found that this number was a little more than 1/8 of the total number of pictures. We could speed up our algorithms by less than a factor 8 by only looking at the distinct objects. This would require extra time since we would have to determine the different orbits. It was therefore not found beneficial enough to have our program only look at one picture per orbit.

After the analysis, a simulator of the game was written in C++ to test our findings by having different algorithms play against each other. Several experiments have been done with the simulator to check the influence of the wall tiling order. It was found that focusing on the diagonal line and adjacent tiles did not have a significant positive impact on the win-rate of the player. Although this strategy is needed to theoretically be able to achieve the optimal score, it turns out to not provide a significant advantage when actually playing the game against an opponent.

In conclusion, our research primarily addressed the theoretically highest possible score in Azul. We acknowledge that the findings may not directly translate into practical value for playing the actual game. Further research could explore more profitable game strategies, such as analyzing tile selection from the factory and optimizing pattern line filling. Also, the impact of the -1 tile could be examined and determining the optimal moment to prioritise completing a horizontal line on the mosaic wall could provide valuable insights for enhancing gameplay strategies.

References

[BBL98] P. Bose, J.F. Buss, and A. Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65:277–283, 1998.

[FL81] A.S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for nxn chess requires time exponential in n. *Journal of Combinatorial Theory, Series A*, 31(2):199–214, 1981.

[Gee] Board Game Geek. Azul. https://boardgamegeek.com/boardgame/230802/azul. Accessed: 2023-08-13.

[Jin18] Jenny Jin. Analysis and applications of Burnside's Lemma, 2018.

[LMS15] M. Lampis, V. Mitsou, and K. Sołtys. Scrabble is PSPACE-Complete. *Journal of Information Processing*, 23(3):284 – 292, 2015.

[NRV16] B.E. Neou, R. Rizzi, and S. Vialette. Pattern matching for separable permutations. *SPIRE*, pages 260–272, 2016.

[Ol20] Miroslav Olšák. Burnside's lemma: counting up to symmetries. https://www.youtube.com/watch?v=D0d9bYZ_qDY&ab_channel=MiroslavOl%C5%A1%C3%A1k, 2020.

[Sch18] H. Schrapers. Award winning games 2018. *Spiel des Jahres*, pages 4–5, 2018.

[vRTV16] J.N. van Rijn, F.W. Takes, and J.K. Vis. The Complexity of Rummikub Problems. *CoRR*, abs/1604.07553, 2016.

[Wes95] J. West. Generating trees and the Catalan and Schröder numbers. *Discrete Mathematics*, 146(1):247–262, 1995.