Universiteit Leiden
The Netherlands

Opleiding Informatica

Analysing the Maximum and Minimum Scores in Azul and the Orders in Which They can be Achieved

Thomas Verwaal

Supervisors:
Rudy van Vliet & Luc Edixhoven

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

05/08/2024

Abstract

In Azul players earn points by placing tiles on their 5×5 mosaic wall. The order in which tiles are placed has a significant impact on the score. In this thesis we prove a conjecture from [Koo23] and analyse the effects two constraints have on the maximum and minimum score and the orders in which these can be achieved. These constraints are based on the game rules, they both enforce that tiles are placed from top to bottom. The applicability of the theoretical key factors for achieving the maximum score is then tested in experiments in which different algorithms play Azul.

We find that the maximum score (145) can be achieved in only two orders if tiles have to be placed from top to bottom in five rounds of five tiles. We have proven that this generalizes to $n \times n$ boards for $n \geq 2$. In these orders the first n tiles are placed on the diagonal and all other tiles are placed with horizontally and with vertically adjacent tiles. We also find that if tiles have to be placed from top to bottom, the minimum score increases from 89 to 92. From the experiments we conclude that the two main factors in achieving the maximum score are sometimes hard to apply in practice and that they are less important than filling pattern lines as much as possible. Taking the opponent into account at the correct moment also proves to be a good strategy.

Contents

1 Introduction

In this thesis we analyse the board game Azul, building on the thesis of Sara Kooistra [Koo23]. In Azul players collect tiles to place on their 5×5 mosaic wall. The way in which tiles are collected and in what order they are placed on the mosaic wall can have a significant impact on the resulting score. Azul thus requires a good strategy to score as many points as possible. In Figure 1 a game of Azul with two players is depicted.

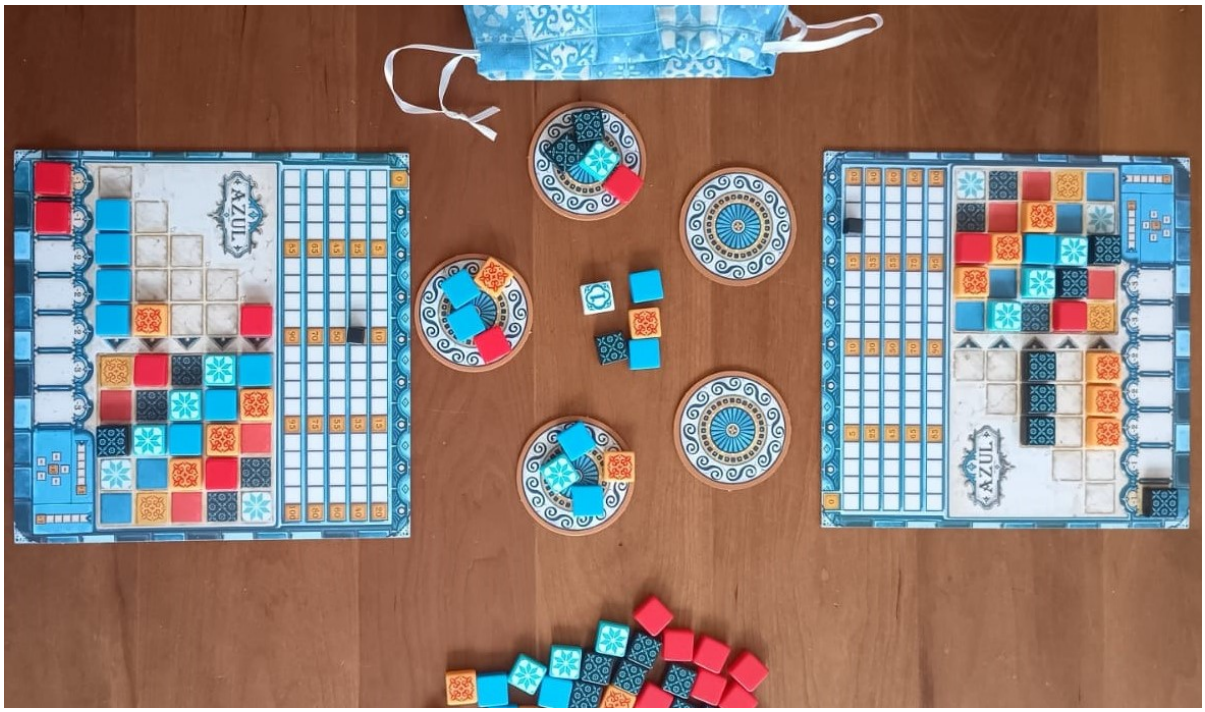


Figure 1: A game of Azul with 2 players

In computer science research on board games is a common theme. This research mostly focuses on strategies/algorithms, complexity and AI players. For example [VG23] looks at strategies to play "Don't Get Angry" and [BCG82] looks at the winning ways for a large number of games. Apart from [Koo23] no research regarding Azul has been conducted. [Koo23] analysed the tiling of the mosaic wall and created different algorithms to play Azul. In this thesis we extend the analysis and implement two additional algorithms. We will first explain the game rules in Section 2. We then prove a conjecture from [Koo23] and analyse the effects of two constraints on the maximum and minimum scores and the orders in which they can be achieved in Section 3. In Section 4 we discuss new algorithms that we have created and the results of several experiments with these algorithms. In section 5, we summarize the conclusions from our study and discuss potential future research.

2 Azul Game Rules

Azul is a game in which players collect tiles to place on their boards and earn points. The player that has the highest score at the end of a game wins. It can be played with two to four players. In this thesis we focus on the two player game.
The game has multiple components: tiles, a bag, factories and player boards. There are five colours of tiles and a total of 100 tiles (20 per colour). The tiles are stored in the bag. There is also one special tile, the negative one tile. This tile is not stored in the bag. Each player has their own board that consists of four parts, a score line, pattern lines, a mosaic wall and a floor line. The factories and a player board are depicted in Figure 2. The game is played in multiple rounds and each round has two phases. At the start of each round, 20 tiles from the bag are placed on the factories, four tiles per factory.

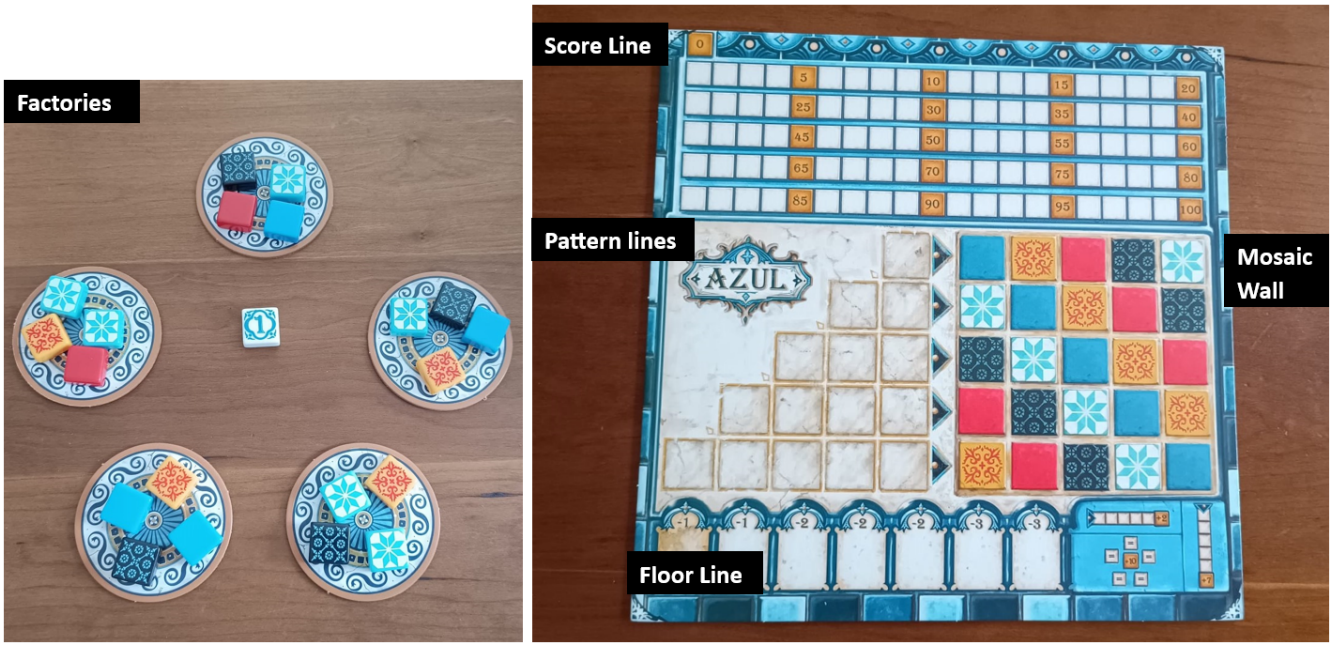


Figure 2: factories (left) and an empty player board (right).

In the first phase the players take turns collecting tiles from the factories. Players can take all tiles of the same colour from a single factory or from the middle of the table. If a player grabs tiles from a factory, all other tiles from this factory are moved to the middle of the table. At the start of each round, the negative one tile is placed in the middle of the table. The first player to grab from the middle of the table has to place this tile on its floor line. This player is the starting player for the next round. The players place the tiles they take on their pattern lines, or on their floor line if there is no place on their pattern lines. If the floor line is full, the tiles are placed in the box. Tiles can only be placed on a pattern line if they have the same colour as tiles present on the pattern line and if that colour is not present on that row of the mosaic wall. This phase ends when there are no more tiles on the factories and in the middle of the table.

In the second phase players earn points. For every completely filled pattern line, one tile is placed on the matching colour in the corresponding row of the mosaic wall. This is done from top to bottom. The remaining tiles of these pattern lines are moved to the box. Tiles are moved from the box to the bag at the start of a round if there are no more tiles in the bag and not all factories are filled.

Scoring

Players earn points for every tile placed on their mosaic wall and gain negative points for every tile on the floor line. A player's score is kept on their score line. The scoring is as follows:

- If the tile is placed without any adjacent tiles, 1 point.
- If the tile has horizontally adjacent tiles, count all the tiles horizontally linked to the newly placed tile (including the placed tile). Gain that many points.
- If the tile has vertically adjacent tiles, count all the tiles vertically linked to the newly placed tile (including the placed tile). Gain that many points.

Note that a newly placed tile counts twice if it has adjacent tiles both in the horizontal and in the vertical direction In Figure 3 an example of the number of points gained by three new tiles is depicted.

For every tile on the floor line the player gets negative points. The number for each tile is indicated above the floor line. The first two tiles give 1 negative point, the next three give 2 negative points and the last two give 3 negative points. After subtracting the negative points from the score line, these tiles are moved to the box. The score of a player can never become negative. After the scoring a new round starts. If at this point a player has filled a horizontal row the game ends. At the end of the game players also earn bonus points. A player gets 2 points for every filled row, 7 points for a filled column and 10 points for a completed colour. The player with the most points wins.

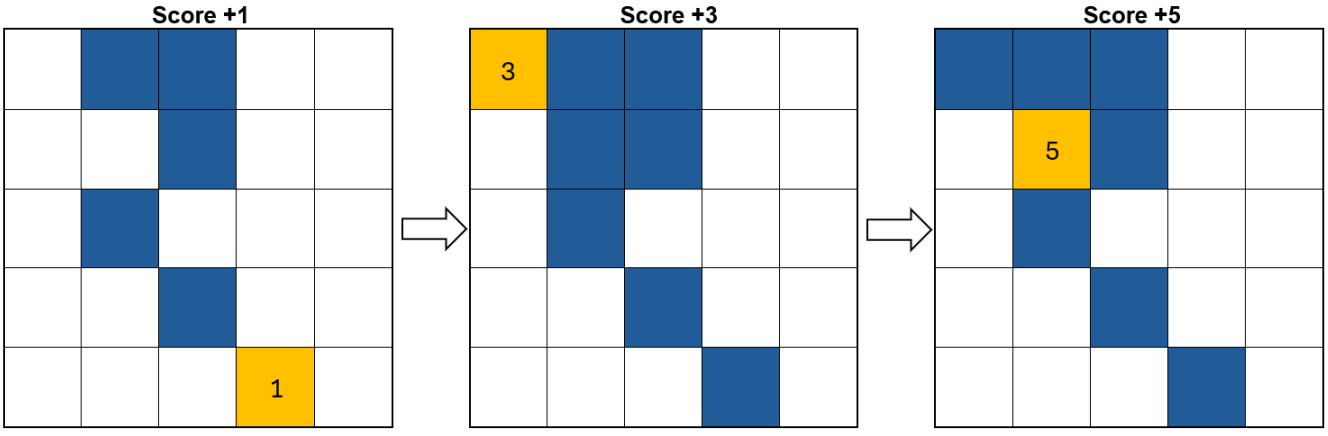


Figure 3: An example of how points are earned.

3 Analysis

In this section we analyse the maximum and minimum scores and the orders in which these can be achieved in Azul. We first discuss earlier research and then prove a conjecture formulated in that research. We then separately analyse the effect two new constraints have on reaching the maximum and minimum scores and the orders in which these can be achieved. The first constraint is: to fill the board, the last tile in every row needs to be placed in the last round, these tiles need to be placed from top to bottom. The second constraint is: all tiles have to be placed from top to bottom in five rounds of five tiles. These constraints will appear to have some impact on the minimum score and a significant impact on the number of orders in which the maximum and minimum score can be achieved.

3.1 Background and Previous Work

In [Koo23] the maximum and minimum achievable scores in Azul were determined to be 145 and 89 respectively (without bonus points). The maximum score was generalized for $n \times n$ boards and is equal to $n^3 + n^2 - n$. Two rules were derived to which a tiling order has to satisfy to achieve the maximum score on $n \times n$ boards:

- n tiles need to be placed in both a new row and a new column.
- All other tiles need to be placed with at least one horizontally and one vertically adjacent tile.

The following problem regarding achieving the maximum score on $n \times n$ boards was defined: **AzulOptimalTiling**: "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 $n^3 + n^2 - n$ points?"
These k tiles can be represented as a permutation. A permutation is a list of column numbers, whose indices match the row in which the tile is placed. A permutation is complete if $k = n$, it is incomplete if $k < n$. For example, the complete permutation $(2, 3, 5, 1, 4)$ represents the configuration in Figure 4.

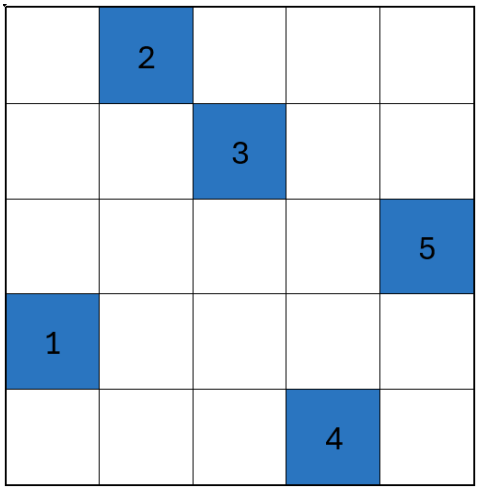


Figure 4: Configuration of 5 tiles of the permutation $(2, 3, 5, 1, 4)$.

In [Koo23] it was shown that if a permutation contains the subpermutation/pattern (2, 4, 1, 3) or (3, 1, 4, 2), it can never lead to the maximum score. Note that (3, 1, 4, 2) is the mirrored image of (2, 4, 1, 3). In this different numbers w, x, y, z with $w < x < y < z$ may take the roles of $1, 2, 3, 4$ and w, x, y, z have to be in that order. For example the permutation $(2, 3, 5, 1, 4)$ contains the subpermutation $(2, 5, 1, 4)$. A permutation without these subpermutation is called a separable permutation. This led to the following conjecture: "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". In Section 3.2.2 we prove that an instance of AzulOptimalTiling is a yes-instance, if it is a complete separable permutation or if it is a incomplete permutation that can be completed without creating the (2, 4, 1, 3)- or (3, 1, 4, 2)-pattern. As the conjecture had already been proven in the other direction, this completes its proof.

In [Koo23] an algorithm was constructed to determine the number of orders in which the maximum score on a 5×5 board can be achieved. This algorithm uses bottom up dynamic programming. We have reconstructed this algorithm to be able to determine the number of orders in which the maximum and minimum score can be achieved when we add the two constraints. The pseudocode for determining the number of orders in which the maximum score can be achieved is as follows:

```
for every possible covering do
    if not valid_covering() then
        continue
    covering.max_score = 0
    covering.routes = 1
    for every square on the covering do
        if square is filled then
            predecessor = covering without square filled
            score = predecessor.max_score + points earned by filling square on predecessor
            if score > covering.score then
                covering.score = score
                covering.routes = predecessor.routes
            else if score == covering.score then
                covering.routes += predecessor.routes
return full_covering.score and full_covering.routes
```

This code can be easily adapted to calculate the number of orders in which the minimum score can be achieved. This was illustrated in [Koo23]. The number of orders in which the maximum and minimum score can be achieved without any constraints are 3.15×10^{14} and 1.56×10^{18} respectively. We have added an IF-statement such that our constraints can be applied later. This IF-statement is marked red and was not used in determining the numbers mentioned above. It can be used to call a function that checks if a covering is valid when a constraint is applied. This is described in Sections 3.3 and 3.4.

We use the state space in Figure 5, which was also drawn in [Koo23], to visualize the effects of our constraints. In this figure all possible orders for tiling a 2×2 board are depicted when no constraint is applied. The orange arrows show the orders in which the maximum score can be achieved. In Sections 3.3 and 3.4 we show the resulting state spaces after removing orders and configurations that do not comply to the constraints.

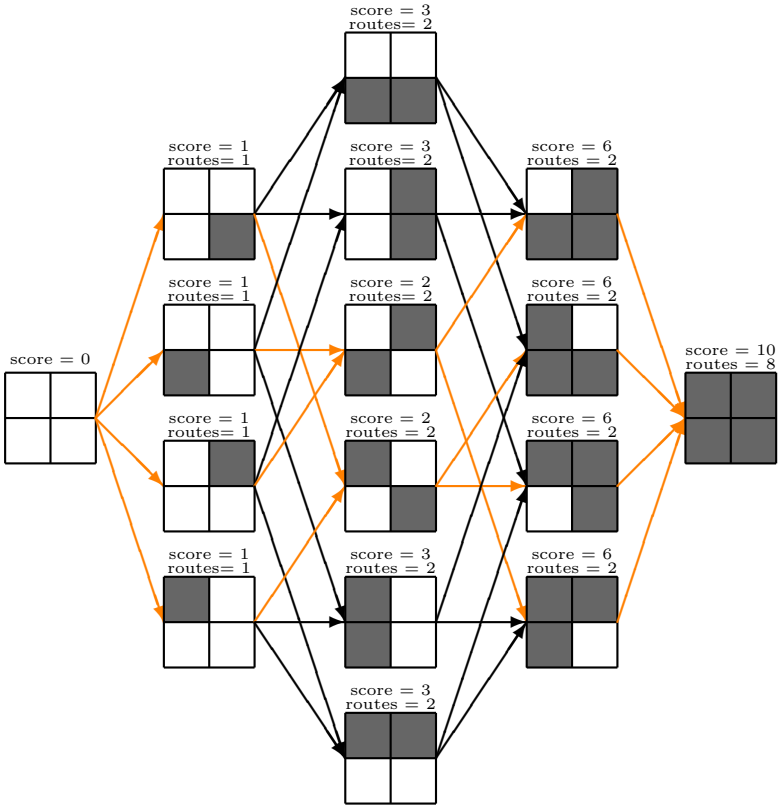


Figure 5: The state space of a 2×2 board.

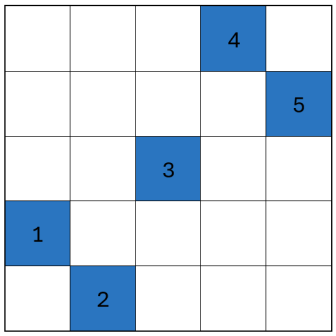
3.2 Separating Trees

In this section we prove that an instance of AzulOptimalTiling is a yes-instance, if it is a complete separable permutation. We first explain what complete separable permutations are for Azul and how separating trees can be used to represent them. We then construct a proof using induction on the structure of a separating tree that represents a separable permutation.

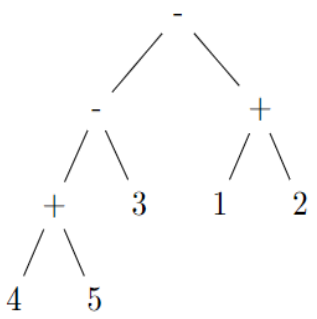
3.2.1 Separable permutations and separating trees

A separable permutation is a permutation without the (2, 4, 1, 3)- and (3, 1, 4, 2)-patterns [BEN16]. The configuration for $n = 5$ in Figure 6a corresponds to the permutation $(4, 5, 3, 1, 2)$, which is separable. A permutation is separable if and only if it has a so-called separating tree [BBL98]. A separating tree is a special type of binary tree for a permutation $(p_1, ..., p_n)$ of $(1, ..., n)$ where the leaves are in the order $(p_1, ..., p_n)$ from left to right in the tree [BBL98]. In Azul n corresponds to the first n tiles in AzulOptimalTiling and $(p_1, ..., p_n)$ with the configuration of these tiles. Internal nodes in a separating tree are either positive or negative. If a node is positive then the highest leaf in its left subtree is exactly one lower than the lowest leaf in its right subtree. If a node is negative then the lowest leaf in its left subtree is exactly one higher than the highest leaf in its right subtree. It is not hard to prove that a node in a separating tree always has a contiguous series of numbers in its leaves. As a result, each subtree containing m leaves corresponds to a box of m contiguous rows

and m contiguous columns on the Azul board. The rows are determined by the position in the tree. The columns are determined by the numbers in the leaves. The separating tree corresponding to the separable permutation $(4, 5, 3, 1, 2)$ can be seen in Figure 6b.



(a) Configuration of 5 tiles.



(b) Separating tree

Figure 6: Configuration of 5 tiles and separating tree of the separable permutation $(4, 5, 3, 1, 2)$.

3.2.2 Separable permutations are yes-instances of AzulOptimalTiling

We are going to prove that every complete separable permutation represents a configuration in Azul from which the maximum score can be achieved. For this we use induction on the structure of the separating tree representing the permutation. In particular, we prove for every subtree containing m leaves, that the corresponding $m \times m$ box can be filled with first m free tiles that correspond with the leaves of the subtree and all the other tiles with at least one horizontally and one vertically adjacent tile.

Base case: A subtree with one leaf represents one square of the Azul board, i.e. a 1×1 box, which can indeed be filled with first one free tile on that square and then zero other tiles.

Induction hypothesis: Suppose our claim holds for two subtrees A and B. We now consider a subtree C consisting of a root with A and B as its children.

Assume that tree A is the left subtree of C and has m_1 leaves and tree B is the right subtree of C and has m_2 leaves. Then tree C has $m_1 + m_2 = p$ leaves. The separating tree C thus represents a $p \times p$ box that needs to be filled according to our claim. By the induction hypothesis for both trees A and B, the first p (free) tiles in the box of tree C can be placed in the same way as the first m_1 and m_2 tiles are placed in the boxes of trees A and B.

After placing the first p free tiles, the remaining $p^2 - p$ tiles need to be placed adjacent to one horizontally and one vertically adjacent tile. The box corresponding with tree C contains two smaller boxes, which correspond with the $m_1 \times m_1$ and $m_2 \times m_2$ box of the trees A and B respectively. These two boxes will be called box A and B, as is illustrated in Figure 7.

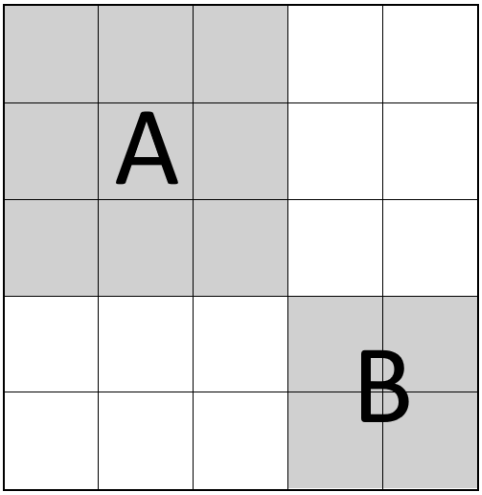


Figure 7: Boxes of trees A and B in the box of tree C.

By the induction hypothesis for subtrees A and B, these two boxes can be filled by placing tiles adjacent to at least one horizontally and one vertically adjacent tile. There are now $p^2 - m_1^2 - m_2^2$ unfilled boxes remaining. To prove that these can be filled with at least one horizontally and one vertically adjacent tile, we have to look at the definition of the separating tree.

The root of tree C is either positive or negative. If it is positive, then the highest column number in box A (corresponding to the left subtree of the root) is exactly one lower than the lowest column number in box B (corresponding to the right subtree of the root). If it is negative, then the lowest column number in box A is exactly one higher than the highest column number in box B. We assume that the box corresponding to tree C starts at column 1 and ends at column p. If tree C has a positive root, then the highest column number of box A is m_1 and the lowest column number of box B is $p - m_2 + 1 = m_1 + 1$. Since boxes A and B have been completely filled, in particular the bottom right of box A and the top left of box B have been filled. The bottom left square of box A is at (m_1, m_1) and the top right square of box B is at $(m_1 + 1, m_1 + 1)$. These two squares can be used to fill the remaining $p^2 - m_1^2 - m_2^2$ squares. First filling row $m_1 + 1$ from right to left, then row $m_1 + 2$ from right to left, and so on till row p. Then filling column $m_1 + 1$ from bottom to top, then column $m_1 + 2$ from bottom to top, and so on till column p. This is illustrated in Figure 8. The above logic can also be applied if tree C has a negative root. This completes the induction step

When we apply our claim to the entire separating tree, corresponding to the entire $n \times n$ Azul board we find that an instance of AzulOptimalTiling is a yes-instance, if it is a complete separable permutation. Of course, it is also a yes-instance, if it is an incomplete permutation that can be completed without creating the (2, 4, 1, 3)- or (3, 1, 4, 2)-pattern.

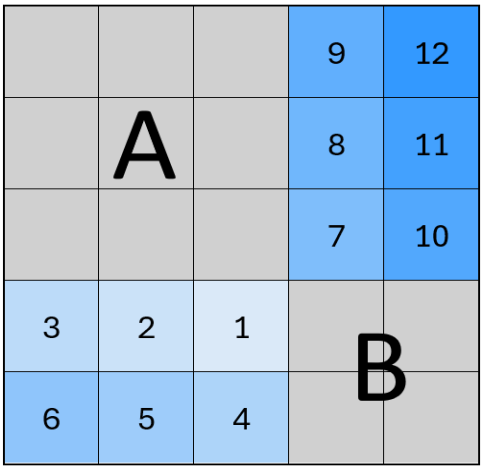


Figure 8: Example order to fill the box corresponding with tree C.

3.3 Effects of the First Constraint

The first constraint we add to our analysis of the maximum and the minimum score follows directly from the rule about the end of the game: to fill the board completely, the last tile in every row needs to be placed in the same round, these tiles need to be placed from top to bottom.
The effect of this constraint on the possible orders on a 2×2 board is illustrated in Figure 9. The orange lines represent the orders in which the maximum score can be achieved. From this figure it becomes clear that there are fewer accepted coverings and therefore fewer orders that result in the maximum score.
To determine the effect of the constraint on the maximum and minimum score and the number of orders in which they can be achieved, we use the code described in Section 3.1. In the IF statement marked red in that code we call the following function: valid_covering():

```
for every row do
    tiles[row] = number of tiles in row
    if tiles[row] == 5 then
        rowFilled = True
for row = 2 to row = 5 do
    if tiles[row] == 5 and tiles[row-1] < 5 then
        return False
if rowFilled then
    for every row do
        if tiles[row] < 4 then
            return False
return True
```

With the addition of this constraint the maximum score stays the same but the number of orders in which it can be achieved reduces from 3.15×10^{14} to 1.33×10^{11}. The minimum score increases from 89 to 92 and the number of orders in which it can be achieved reduces from 1.56×10^{18} to 2.00×10^{15}. This constraint significantly reduce the number of possible orders in which the game can be played and the result shows that both the maximum and minimum score can be achieved in fewer orders. In Section 3.6 the cause of the increase in the minimum score will be discussed.

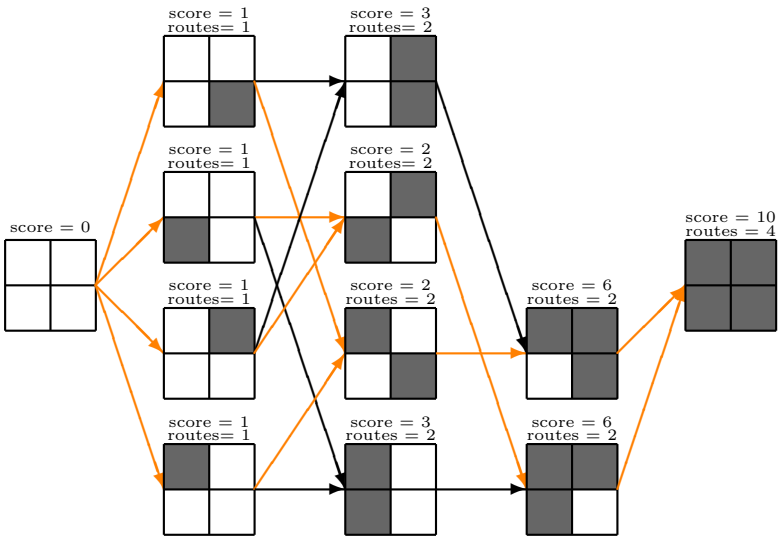


Figure 9: The effect of the first constraint on the state space of a 2×2 board.

3.4 Effects of the Second Constraint

The second constraint we add is: all tiles have to be placed from top to bottom in five rounds of five tiles. This rule does not directly follow from the game rules, because it is possible (in fact, very likely) that in a round of the game a player moves less than five tiles from the pattern lines to the mosaic wall. However the constraint reflects the situation that a player wants to finish the game as quickly as possible, i.e, in five rounds.

This constraint significantly reduces the number of possible orders in which the game can be played even further. In particular, the effect of this constraint on the possible orders on a 2×2 board is illustrated in Figure 10. The orange lines represent the orders in which the maximum score can be achieved. From this figure it becomes clear that there are even fewer accepted coverings than with the first constraint and therefore also fewer orders that lead to the maximum score.

To determine the effect of the constraint on the maximum and minimum score and the number of orders in which they can be achieved, we again use the code described in Section 3.1. In the IF statement marked red in that code we call the following function: valid_covering():

```
for every row do
    tiles[row] = number of tiles in row
if tiles[5] + 1 < tiles[1] then
    return False
for row = 2 to row = 5 do
    if tiles[row] > tiles[row-1] then
        return False
return True
```

The coverings that comply with this constraint are a subset of those that comply with the first constraint. Therefore we compare the number of orders with the number of orders when the first constraint is applied. With the addition of this constraint the maximum score stays the same, but

the number of orders in which it can be achieved reduces from 1.33×10^{11} to only 2. The minimum score for this constraint stays 92 but the number of orders in which it can be achieved reduces from 2.00×10^{15} to only 1096. These results shows that both the maximum and minimum score can be achieved in less orders. In the next section we will prove that there are indeed only two possible orders in which the maximum score can be achieved. This will be done for $n \times n$ boards. In Section 3.7 we discuss the origin of the 1096 orders that yield the minimum score.

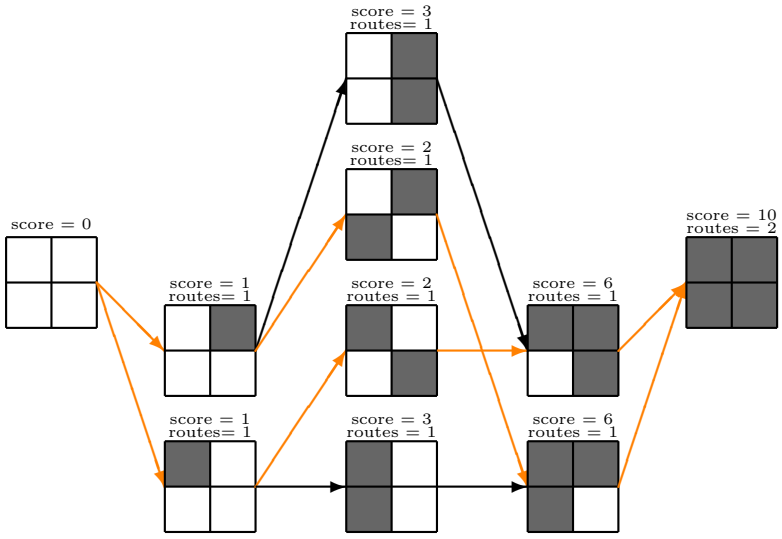


Figure 10: The effect of the second constraint on the state space of a 2×2 board.

3.5 Proof of Two Orders

In the previous section we saw that when tiles need to be placed from top to bottom in five rounds of five tiles (second constraint), there are only two possible orders to achieve the maximum score. In these orders, the first five tiles appear to be placed on the diagonal, either from top left to bottom right or from top right to bottom left. The complete tiling order for the former case can be found in Figure 11. We will now prove that these are indeed the only two possible orders for general n, illustrating the proof with 5×5 boards. In this proof, square coordinates are represented as (row, column), the square in the top left is at (1,1) and we start at round 1 (not 0).

We know from [Koo23] that to achieve the maximum score, the first tile in every row needs to be placed in a different column and that all other tiles need to be placed adjacent to a tile both in the horizontal and the vertical direction (rule 1).

We now add the second constraint to the game such that tiles have to be placed from top to bottom in n rounds of n tiles. Because of this the first n tiles need to be placed in different rows and columns. With this extra constraint and rule 1, we can define another rule for achieving the maximum score: If we want to place a tile in (i, j) next to a tile at $(i, j-1/j+1)$ in round $k \geq 2$ with the tile at $(i+1, j)$ as vertically adjacent, the tile at $(i+1, j)$ needs to be present at the end

of round $k-1$ (rule 2). This rule follows directly from the placement of tiles from top to bottom every round.

The first step in our proof considers the placement of the first two tiles. Assume we place the first tile in $(1, L)$. Because of rule 1 we know that the next tile in row 1 needs to be placed in column $L+1$ or $L-1$. From rule 2 it follows that the first tile in row 2 needs to be placed in column $L+1$ or $L-1$, because otherwise the second tile in row 1 does not have a vertically adjacent tile in the second round.

From the first step, we can conclude that we need to start with an $m \times m$ box in the first rows of the board (with $m \geq 2$), in this box the first m tiles are placed on the diagonal of this box. Since the diagonal from top left to bottom right and the diagonal from top right to bottom left are symmetric, we continue the proof with the diagonal from top left to bottom right.

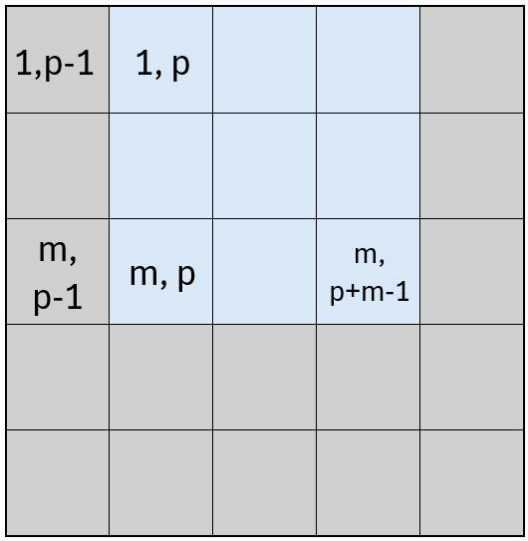
1	6	11	16	21
22	2	7	12	17
23	18	3	8	13
24	19	14	4	9
25	20	15	10	5

Figure 11: Order to achieve the maximum score when tiles need to be placed from top to bottom in five rounds of five tiles.

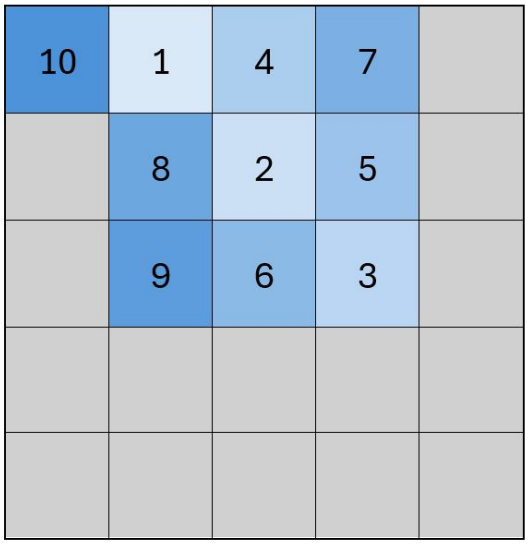
Steps 2 and 3 will together prove that we have to start with the first n tiles on the diagonal. This proof is based on the result from step 1, we have to start with an $m \times m$ box with $m \geq 2$. If we do not start with the first n tiles on the diagonal of the entire board, we have to place a tile in the first row, outside of the box in round $m+1$ (or before). This tile can be placed either to the left or to the right of the box. In step 2 we will prove, that we can never leave the box on the left side while maintaining maximum points. In step 3 we will prove that we can never leave the box on the right side while maintaining maximum points.

Step 2: Assume that the first tile that is placed in row 1 outside our $m \times m$ box, is placed to the left of this box. We define three spots in our box: the top left as $(1, p)$, the bottom right as $(m, p+m-1)$ and the bottom left as (m, p). This is illustrated by Figure 12a, for a 3×3 box, where the squares used in this proof have been marked. In Figure 12b the attempted tiling order used in the proof can be seen, where tile 10 does not comply with rule 1 and squares underneath the box are not considered. The first tile that is placed in row 1 outside our box is at $(1, p-1)$ and it needs to be placed in round $\leq m+1$. From rule 2 it then follows that square $(2, p-1)$ needs to be filled in round $\leq m$ which cascades down to square $(m, p-1)$ needing to be filled in round ≤ 2.

However, since all tiles need to be placed with a horizontal adjacent tile, square $(m, p-1)$ can at earliest be filled in round $m+1$, which is a contradiction since $m+1 > 2$.



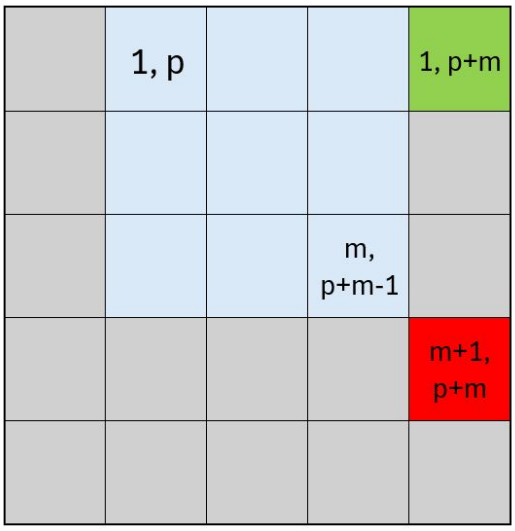
(a) Used tiles in step 2.



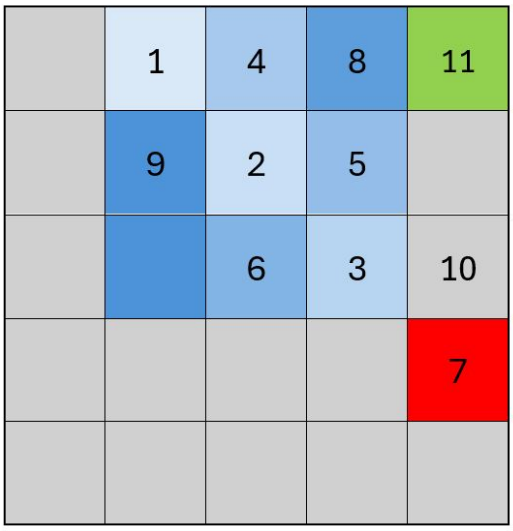
(b) Attempted tiling order in step 2.

Figure 12: Supporting figures for step 2.

Step 3: Assume that the first tile that is placed in row 1 outside our $m \times m$ box, is placed to the right of this box. Let us define two spots in our $m \times m$ box: the top left as $(1, p)$, and the bottom right as $(m, p+m-1)$. Since the diagonal ends at $(m, p+m-1)$ we know that after the first round, square $(m+1, p+m)$ is not filled. This square is coloured red in Figure 13a. In Figure 13b the attempted tiling order used in the proof can be seen, where tile 11 does not comply with rule 1 and most squares underneath the box are not considered. The first square that is placed in row 1 outside our box is at $(1, p+m)$ (green) and it needs to be filled in round $\leq m+1$. For this tile to achieve maximum points it follows from rule 2 that in round $\leq m$, the square in $(2, p+m)$ needs to be filled. This cascades down to the condition that in round 1 $(= m+1-m)$ square $(m+1, p+m)$ needs to be filled. We observed before that after the first round, square $(m+1, p+m)$ is not filled. We thus have a contradiction and therefore can never leave the box on the right side. From steps 2 and 3 it thus follows that we have to start with a diagonal from top left to bottom right because we can otherwise never place a tile in row 1 outside of our box on either the left or right side without losing points.



(a) Used tiles in step 3.



(b) Attempted tiling order in step 3.

Figure 13: Supporting figures for step 3.

As fourth step we have to prove that after placing the first n tiles on the diagonal, there is only 1 order in which the maximum number of points can be achieved. Rule 1 implies that in round $k \geq 2$ each tile must be placed next to a tile in the same row. This holds in particular for the tiles in row 1. Because we start with square $(1, 1)$ in round 1, square $(1, k)$ must be filled in round k. To fill the square at $(1, k)$, the square at $(2, k)$ needs to be filled in round $\leq k - 1$, which follows from rule 2. Consequently squares $(3, k), (4, k), ..., (k, k)$ need to be filled in round $\leq k + 1-$row number. This leads to the partial order that can be seen in Figure 14.

After filling a row like this, the squares in the left lower triangle of the board are not yet filled. These have only one remaining option every round that complies with rule 1, which leads to an order of filling these squares from right to left, as we already depicted in Figure 11.

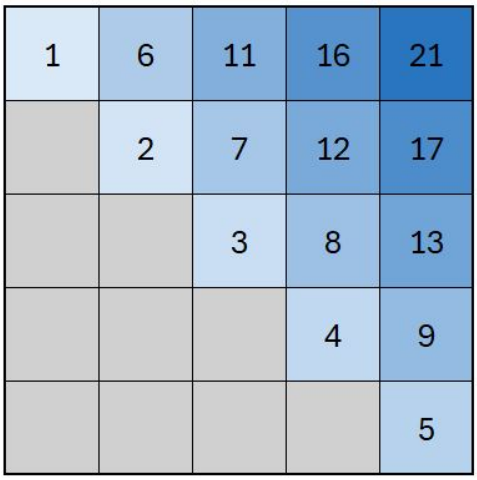


Figure 14: Partial filling order after the diagonal.

3.6 Minimum Score Increase

In Sections 3.3 and 3.4 we found that if we add the first or the second constraint on the tiling orders the minimum score increases from 89 to 92. In this section we discuss why the minimum score increases. The coverings that comply with the second constraint are a subset of those that comply with the first constraint. Therefore we only look at orders that also comply with the second constraint. There are multiple orders in which a row or column can be filled such that it yields the minimum number of points (11). The five tiles in these orders either yield $(1+1+1+3+5)$ or $(1+2+1+2+5)$ points, not necessarily in that order. Two orders and the points each tile yield in those orders are depicted in Figure 15.

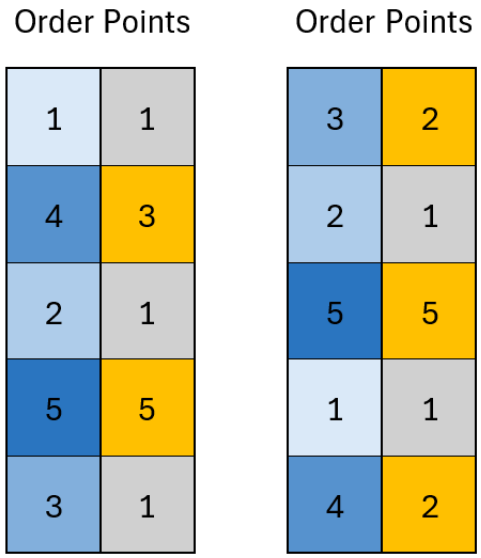
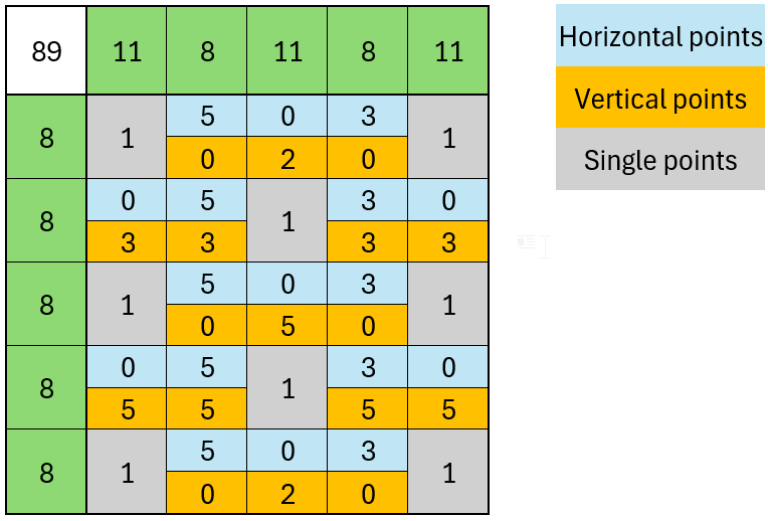


Figure 15: Two orders in which a column can be filled and the points each tile yields.

To understand how a minimum score for the entire board can be achieved we divide the points earned into horizontal, vertical and single points. Horizontal points are the points that are earned because a tile is placed next to tiles in the same row. Vertical points are the points that are earned because a tile is placed next to tiles in the same column. Single points are the points that are earned because a tile is placed without any adjacent tile, neither in the horizontal, nor in the vertical direction. When we count points as described above, the minimum number of points for a row/column can consist of either 8 horizontal/vertical points and 3 single points or 9 horizontal/vertical points and 2 single points. The minimum total score of 89 without constraints can for example be achieved with 5×8 horizontal points, $4 \times 8 + 9$ vertical points and 8 single points. An order with this score distribution is depicted in Figure 16. In this and later figures the single points are counted for the column in which they are earned.

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

(a) Order that yields 89 points.



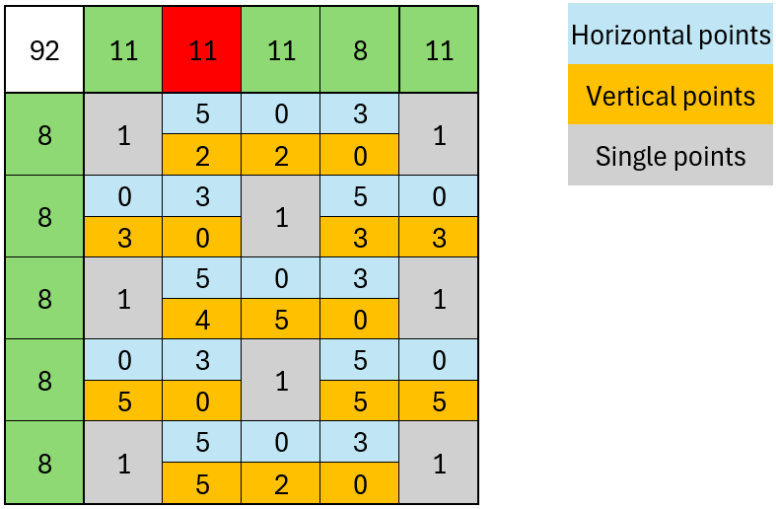
(b) Score distribution and points earned by this order.

Figure 16: Tiling order that yields the minimum score (89) and the score distribution of this order.

To explain why adding the second constraint increases the minimum score to 92, we compare the order in Figure 16a with a similar order that yields a score of 92 when the constraint is applied. This order consists of 5×8 horizontal points, $3 \times 8 + 9 + 11$ vertical points and 8 single points. This order with this score distribution is depicted in Figure 17.

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

(a) Order that yields 92 points.

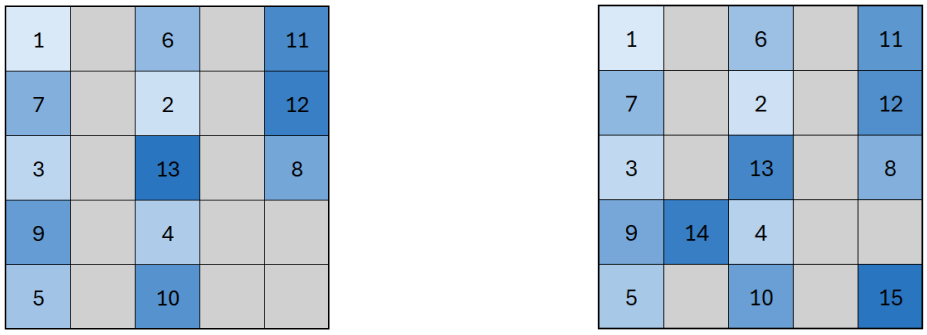


(b) Score distribution and points earned by this order.

Figure 17: Tiling order that yields the minimum score (92) when the second constraint is applied and the score distribution of this order.

In Figure 16a we see that the order in which the tiles are placed for every row and column, is an order that yields the minimum number of points for a row/column, cf. Figure 15. In Figure 17a every row and column also has such an order except for the second column.
The order in Figure 17a is equal to the order in Figure 16a up to the placement of tile 14. In the

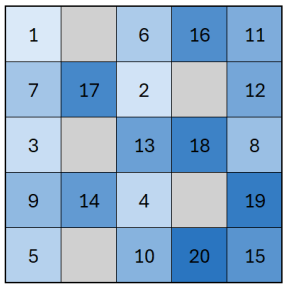
first three rounds we try to place tiles in columns $1, 3$ and 5, to make sure that no horizontal points are scored yet, and that in rounds 4 and 5, we score the minimum of $3+5$ horizontal points per row with tiles in columns 2 and 4. We try to place the tiles such that the minimum number of vertical points remains possible for every column. We also try to comply with the second constraint. This works well up to the fourth tile in the third round (tile 14), as depicted in Figure 18a. If we would place the fourth and fifth tile (15) in column 5 we would gain too many points for this column, $1+1+3+4+5$ instead of $1+1+3+1+5$ which is still possible.
Instead of placing tile 14 in the third round in column 5 we can place it in column 2. This already gives 1 point extra, as compared to the minimum of 89 without constraints. The reason for this is, that if the neighbouring columns are filled, there are no more single points to be gained, the horizontal points are fixed ($3+5$ per row), and the minimum number of vertical points is $0+0+0+3+5=8$. However, the minimum number of vertical points for a column in which the fourth square is filled first is $0+0+2+2+5=9$ points, see Figure 15. The resulting configuration is depicted in Figure 18b. Placing tile 14 in the second column thus leads to an increase of 1 point. Tile 14 could also be placed in the fourth column but this would eventually lead to even more than 92 points.



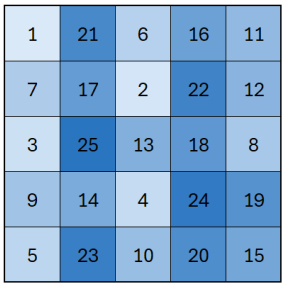
(a) (b)

Figure 18: Placement of tiles 14 and 15

In the fourth round we can start filling columns 2 and 4 and fill the last tile in column 5, as depicted in Figure 19a. If we would ignore the constraint in the fifth round, then the second column could indeed yield $(0+0+2+2+5=9)$ vertical points, with the order depicted in 19b. However, due to the second constraint this order is not possible and the second column yields 2 points more $(0+0+2+4+5=11)$, as depicted in Figure 17.



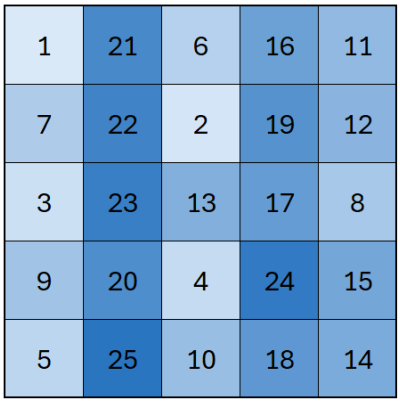
(a) Board after placing the tiles of the fourth round



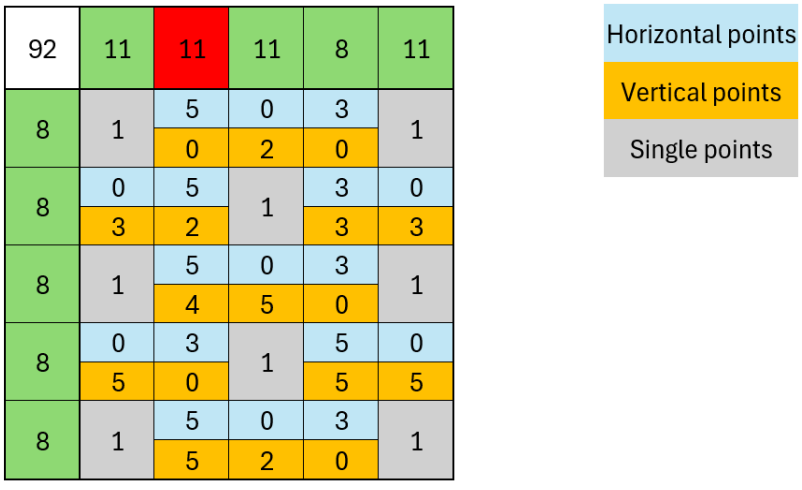
(b) Placement of tiles in the fifth round if no constraint is applied

Figure 19: Board after placing the tiles of the fourth round and round five without any constraints

From this reasoning it might seem like tile 14 is partly responsible for the cause of the increase of the minimum score from 89 to 92. This is, however not the case, the placement of tile 14 is only an extra problem that comes with the second constraint. This is one of the reasons why there are fewer possible orders that lead to the minimum score for the second constraint compared to the first constraint.
If only the first constraint is applied, the order in which the second and fourth column are filled must also differ from the order in Figure 16a. If we would keep the order in which the fourth column is filled the same as in Figure 16a, then the second column would have to be filled from top to bottom, yielding $0+2+3+4+5=14$ vertical points, leading to a total of 95 points. Hence, to prepare the second column best for the fifth round, we should place a tile on the fourth square of this column before that round. This leads to the order depicted in Figure 20. The effect on the score is the same as with tile 14 under the second constraint: the tiles in the second column yield $0+0+2+4+5=11$ points leading to a total of 92.



(a) Order that yields 92 points.



(b) Score distribution and points earned by this order.

Figure 20: Tiling order that yields the minimum score (92) when the first constraint is applied and the score distribution of this order.

An order that achieves 89 points thus needs every row and column to have an order that yields the minimum number of points for a row/column. Since for both constraints tiles need to be placed from top to bottom in the last round, the order in which columns 2 and 4 are filled needs to change. Therefore column 2 no longer has an order that yields the minimum number of points and thus the minimum score increases to 92.

3.7 1096 Orders for the Minimum Score

In Section 3.4 we found that if we add the second constraint on the tiling orders the number of orders in which the minimum score can be achieved is only 1096. It was beyond the scope of this thesis to investigate what exactly all of these orders are. We did, however, analyse the impact of the first three tiles on the possibility of achieving the minimal score. There are only 18 ($3 \times 3 \times 2$) configurations of these three tiles that can still yield the minimum score. These and the number of orders in which they can yield the minimum score are depicted in Figure 21. The configurations that have the same number of orders are symmetric to each other. In all of these configuration the first 3 tiles are placed in either column 1, 3 or 5. If no constraint was applied, the minimum score could also be achieved with the first tile in either column 2 or 4.
This observation supports the idea that, under the second constraint, in the first three rounds we want to place tiles in columns 1, 3 and 5 as much as possible, as discussed in Section 3.6. To further investigate this idea we wanted to determine how many tiles need to be placed in columns 1, 3 or 5 before a tile can be placed in either column 2 or 4. We used the code in Sections 3.1 and 3.4 to determine that, to achieve the minimum score, the first 10 tiles have to be placed in columns 1, 3 or 5 and that thus tile 11 can sometimes be placed in either column 2 or 4. The 1096 orders in which the minimum score can be achieved thus all start with the first 10 tiles in columns $1, 3$ and 5.

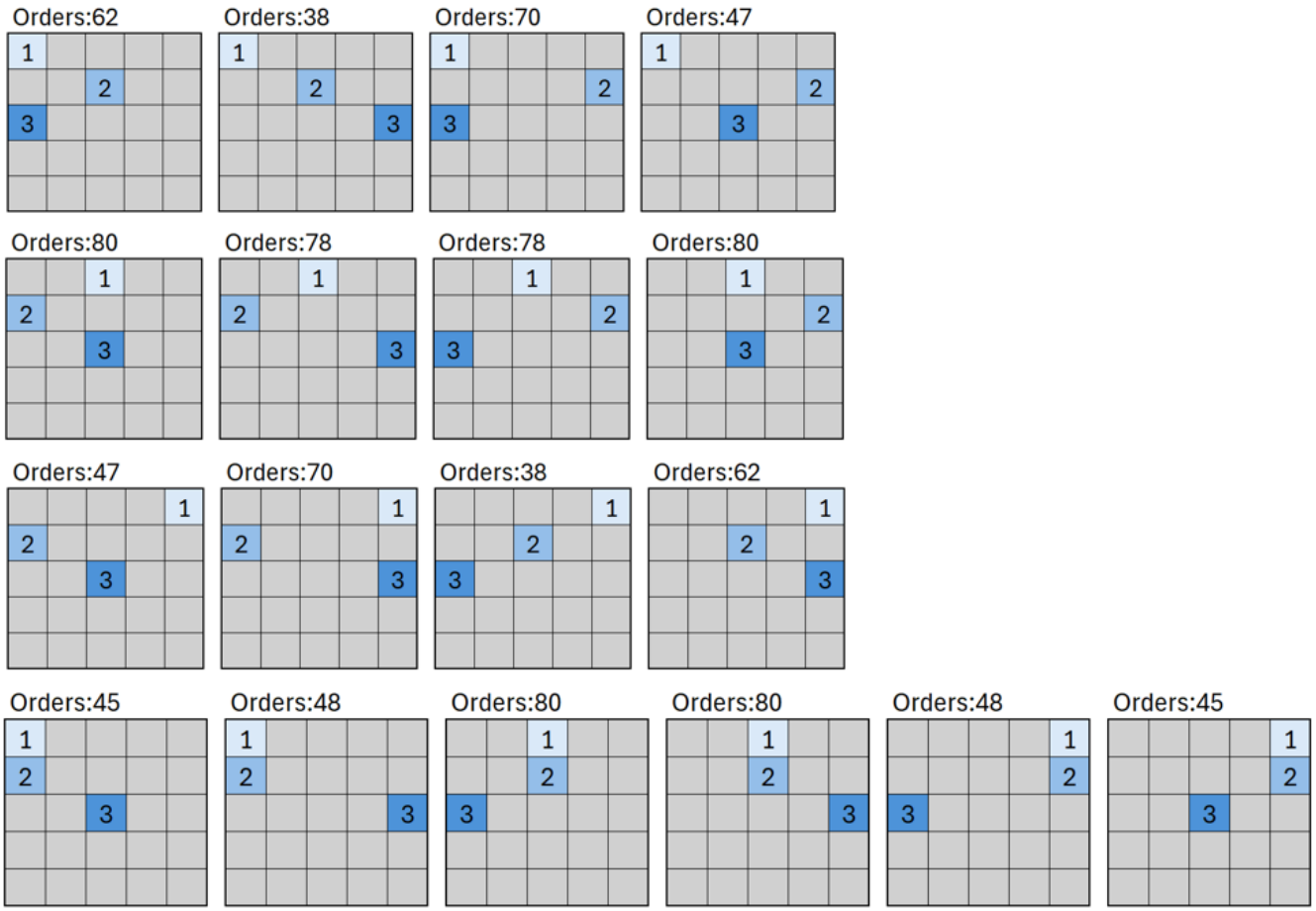


Figure 21: The 18 configurations of the first three tiles that can still yield the minimum score when the second constraint is applied.

4 Experiments and Results

In [Koo23] multiple algorithms were created to play Azul, a greedy algorithm, a smart algorithm and a strategic algorithm. The strategic algorithm was the best performing algorithm, it focuses on placing tiles in the diagonal line of the mosaic board in the first round, and on placing tiles on squares with, if possible, two adjacent tiles in later rounds. From our theoretical section, it follows that with the addition of two rules the maximum score can only be achieved in two orders. These two orders and the strategic algorithm are both centered around first placing tiles on the diagonal and then placing tiles adjacent to other tiles. This suggests that the theory behind the strategic algorithm is hard to apply in practice. We therefore wanted to create an algorithm that can beat the strategic algorithm. In this section, we discuss three algorithms that we have created. Since only pseudocode of the algorithms from [Koo23] was available, our interpretation of this pseudocode was used to recreate the algorithms. As will be seen later, we did not obtain the same results. Therefore, we will also discuss our implementation of these algorithms.

We created three new algorithms: a fast algorithm, an "opponent" algorithm and an improved strategic algorithm. The original idea to beat the strategic algorithm was the fast algorithm. This algorithm wants to end the game as fast as possible. Since the game ends when one player has filled one row, at least five rounds would be necessary. The reasoning behind this algorithm was that the strategic algorithm requires time to build up its board in order to achieve a high score. However after creating the first version of the fast algorithm we observed that when playing the strategic algorithm against itself the game already ends in five rounds in 90% of games. Hence, focusing on this particular aspect would not change a lot. The fast algorithm was therefore not developed further and we will not discuss it further.
The opponent algorithm does the move its opponent would do. The improved strategic algorithm follows the rules of the 'normal' strategic algorithm but does the move its opponent would do if it has to choose between two equally good moves.
To determine the effectiveness of the algorithms, a program that can simulate Azul was written in C++. The program follows the rules that were explained in Section 2, so the -1 tiles and bonus points are included. We first describe the different algorithms and then discuss the results of multiple experiments. In these experiments we look at the performance of the different algorithms and the effect of certain parts of the algorithms on the win percentage. We have also conducted one experiment to determine the impact of being the starting player.

4.1 Greedy Algorithm

The greedy algorithm picks the largest move that fits on a pattern line. If there is no move that fits on a pattern line it chooses a move that partially fits on a pattern line and has the smallest number of remaining tiles. If there is no move that can be placed on a pattern line it picks the smallest move in order to minimize the number of negative points. The pseudocode can be found below:

```
for every move in possibles moves do
    for every non full patternline in patternlines do
        if colour of move cannot be placed on the patternline then
            continue
        else if number of tiles of move fit in the patternline then
            if not goodMoveFound or number of tiles in move > bestMove.count then
                bestMove = move
                goodMoveFound = True
        else if not goodMoveFound and
                    number of tiles in move − patternlineSpace < bestMove.negativeCount then
            bestMove = move
            moveFound = True
            bestMove.negativeCount = number of tiles in move − patternlineSpace
    if not moveFound and number of tiles in move < bestMove.negativeCount then
        bestMove = move
        bestMove.negativeCount = number of tiles in move
return bestMove
```

4.2 Smart Algorithm

The smart algorithm has the same basic structure as the greedy algorithm, however when a move fits on a pattern line it does not focus on picking the largest move. It instead looks at the number of whitespace left on that pattern line. It picks the move that leaves the least amount of space on its pattern line. When two moves leave the same number of space on the pattern line it picks a move that leads to placing a tile adjacent to another tile on the mosaic wall. If there are two moves that both have an adjacent tile it picks the largest move. The pseudocode can be found below:

```
for every move in possibles moves do
    for every non full patternline in patternlines do
        if colour of move cannot be placed on the patternline then
            continue
        else if number of tiles of move fit in the patternline then
            whiteSpace = patterlineSpace − number of tiles of move
            if whiteSpace < bestMove.whiteSpace then
                bestMove = move
                goodMoveFound = True
            else if whiteSpace == bestMove.whiteSpace and not bestMove.adjacent then
                if move is adjacent then
                    bestMove = move
                    goodMoveFound = True
                else if bestMove.count < number of tiles in move then
                    bestMove = move
                    goodMoveFound = True
        else if not goodMoveFound and number of tiles in move − patternlineSpace <
                    bestMove.negativeCount then
            bestMove = move
            moveFound = True
            bestMove.negativeCount = number of tiles in move − patternlineSpace
    if not goodMoveFound and not moveFound and
            number of tiles in move < bestMove.negativeCount then
        bestMove = move
        bestMove.negativeCount = number of tiles in move
return bestMove
```

4.3 Strategic Algorithm

The strategic algorithm has the same basic structure as the smart algorithm, it also focuses on picking the move that leaves the least amount of whitespace on a pattern line. However, if two moves leave the same amount of space and it is the first round, it picks the move that would end up on the diagonal. If it is not the first round or if both moves would not end up on the diagonal, it picks the move that leads to placing a tile next to the most adjacent tiles on the mosaic wall. The pseudocode can be found below:

```
for every move in possibles moves do
    for every non full patternline in patternlines do
        if colour of move cannot be placed on the patternline then
            continue
        else if number of tiles of move fit in the patternline then
            whiteSpace = patterlineSpace − number of tiles of move
            if whiteSpace ≤ bestMove.whiteSpace then
                if whiteSpace < bestMove.whiteSpace then
                    bestMove = move
                    goodMoveFound = True
                else if move ends in diagonal and not bestMove.diagonal and firstRound then
                    bestMove = move
                    goodMoveFound = True
                else if number of adjacent for current move > bestMove.adjacent then
                    bestMove = move
                    goodMoveFound = True
                else if number of adjacent for current move == bestMove.adjacent and
                        bestMove.count < number of tiles in move then
                    bestMove = move
                    goodMoveFound = True
        else if not goodMoveFound and number of tiles in move − patternlineSpace <
                  bestMove.negativeCount then
            bestMove = move
            moveFound = True
            bestMove.negativeCount = number of tiles in move − patternlineSpace
    if not goodMoveFound and not moveFound and
            number of tiles in move < bestMove.negativeCount then
        bestMove = move
        bestMove.negativeCount = number of tiles in move
return bestMove
```

4.4 Opponent Algorithm

The opponent algorithm does the move the opponent would pick if it was their turn and if they would play strategic. It picks the tiles from the factory that the opponent would pick and then places these tiles on its own pattern lines using the strategic algorithm. The pseudocode can be found below:

```
move = opponent → strategic(possibleMoves)
possibleMoves = move
move = strategic(possibleMoves)
```

4.5 Improved Strategic Algorithm

The improved strategic algorithm plays the game strategically and if it encounters two equally good moves it picks the move that its opponent would choose. This algorithm does not prefer tiles on the diagonal in the first round whereas the 'normal' strategic algorithm does. Two moves are considered equally good when they leave the same number of whitespace on a pattern line and if they lead to the same number of adjacent tiles on the mosaic wall. The pseudocode can be found below:

```
for every move in possibles moves do
    for every non full patternline in patternlines do
        if colour of move cannot be placed on the patternline then
            continue
        else if number of tiles of move fit in the patternline then
            whiteSpace = patterlineSpace − number of tiles of move
            if whiteSpace ≤ bestMove.whiteSpace then
                if whiteSpace < bestMove.whiteSpace then
                    bestMove = move
                    goodMoveFound = True
                else if number of adjacent for current move > bestMove.adjacent then
                    bestMove = move
                    goodMoveFound = True
                else if number of adjacent for current move == bestMove.adjacent then
                    goodMoves = {bestMove, move}
                    bestMove = opponent → strategic(goodMoves)
        else if not goodMoveFound and number of tiles in move − patternlineSpace <
                    bestMove.negativeCount then
            bestMove = move
            moveFound = True
            bestMove.negativeCount = number of tiles in move − patternlineSpace
    if not goodMoveFound and not moveFound and
            number of tiles in move < bestMove.negativeCount then
        bestMove = move
        bestMove.negativeCount = number of tiles in move
return bestMove
```

4.6 Experiments and Results

We have conducted multiple experiments in which two algorithms played 100,000 games against each other. We chose to play 100,000 games because then the observed difference between runs was only 0.01% when an algorithm plays against itself. This difference was around 1.0% if two different algorithms play against each other. This number varies depending on which algorithms played against each other. For the experiments the number of draws was around 1%, so if one player won 50% of the games the other won around 49% of the games.

4.6.1 Performance of algorithms

In the first experiment we played all algorithms against each other. The results of this experiment can be found in Table 1. The values in the table are the win percentages of the player in that row. In this experiment both players started 50,000 times. The percentages on the diagonal only show that the number of draws in these runs was 1.0%.
We first compare these results to the results from [Koo23]. Our interpretation of the greedy algorithm performs significantly better against the smart and strategic algorithms than reported in [Koo23]. In our experiments the win percentages of smart and strategic vs greedy were, 79.3% and 82.3%, whereas they were reported as 95.3% and 95.7%. The most likely reason for this difference is the way a move is picked when it does not fit on the pattern line, since this is not explained in [Koo23]. The results of the strategic vs the smart algorithm are more similar, 54.4% versus 52.6% in [Koo23]. This small deviation could be due to the difference between simulations. It is also possible that the reconstructed algorithms do not match exactly with the original algorithms.

	Greedy	Smart	Strategic	Opponent	Improved Strategic
Greedy	49.5%	20.1%	17.1%	57.3%	19.5%
Smart	79.3%	49.5%	44.7%	71.8%	40.0%
Strategic	82.3%	54.4%	49.5%	77.8%	44.3%
Opponent	41.5%	27.5%	21.6%	49.5%	16.1%
Improved Strategic	79.3%	59.0%	54.8%	83.2%	49.5%

Table 1: Win percentages of every algorithm against all other algorithms.
The win percentages are of the player in that row.

In Table 1 we see that the greedy algorithm and the opponent algorithm are by far the worst performing algorithms. Since the greedy algorithm beats the opponent algorithm when they play against each other, we may conclude that the opponent algorithm is a bit worse than the greedy algorithm.
The opponent algorithm picks the move its opponent would pick. After analysing this algorithm we see that it has two problems. The first problem is that it chooses a move that it cannot always place ideally on its own board. In some cases it picks a move it cannot even place partially on a pattern line and therefore has to place all the tiles on the floor line, which leads to negative points. The second problem is that the other player is not very much affected by losing its best move. In most cases the opponent has multiple options that leave the same number of whitespace on a

pattern line and therefore can simply pick another good move. This algorithm thus picks a move that is not ideal for itself and does not really affect the opponent with it. The opponent algorithm does itself have room for improvement. One option would be to create a sorted array of moves that its opponent would pick and then pick the best move from this array that the algorithm benefits from itself.
In the improved strategic algorithm the idea of playing against your opponent was integrated into the strategic algorithm. This is the best performing algorithm. It has an 5.4% higher win rate against the smart algorithm than the strategic algorithm and wins from the strategic algorithm in 54.8% of the games. An interesting observation is that the improved strategic algorithm has a lower win percentage against the greedy algorithm than the strategic algorithm. The improved strategic algorithm only uses the part of the theory about placing tiles adjacent to other tiles. It does not have a preference for the diagonal. This supports the conclusion from [Koo23] that the results from the theoretical section are sometimes hard to apply in practice.

4.6.2 Altered algorithms

In this section we want to investigate the impact of the preference for placing tiles adjacent to other tiles. To investigate this we have altered the improved strategic algorithm and the smart algorithm. We removed from both algorithms the preference for adjacent tiles. The improved strategic algorithm now picks the move its opponent would pick if two moves leave the same number of whitespace on a pattern line. The smart algorithm pick the largest move if two moves leave the same number of whitespace on a pattern line. We performed another experiment with these two altered algorithms. The results from this experiment can be found in Table 2. The values in the table are the win percentages of the player in that row. In this table we see that after altering the algorithms, their win percentages decrease slightly. The smart algorithm wins 0.4% less games against the greedy algorithm and 1.4% less against the strategic algorithm. The improved strategic algorithm wins the same number of games against the greedy algorithm and 0.9% less against the strategic algorithm.

	Greedy	Strategic
Smart	79.3%	44.7%
Smart Altered	78.9%	43.3%
Improved Strategic	79.3%	54.8%
Improved Strategic Altered	79.3%	53.9%

Table 2: Win percentages of altered smart and improved strategic algorithm against the greedy and strategic algorithm. The win percentages are of the player in that row.

The two main factors in reaching the maximum score, placing the first five tiles on the diagonal and placing all other tiles adjacent to at least one horizontal and one vertical tile, thus do not have a large impact on winning the game in practice. The altered smart algorithm is more similar to the greedy algorithm. The only difference is that instead of focusing on picking the largest move it first focuses on picking the move that leaves the least amount of whitespace on the pattern line and if

the whitespace is equal it picks the largest move. The main component of all algorithms is thus leaving the least amount of whitespace on its pattern lines.

4.6.3 Starting player

In the third experiment we played all algorithms against each other and kept the starting player for every game the same. The results of this experiment can be found in Table 3. The values in Tables 3 are the win percentages of the player in that row and the player in the row is also the starting player. In Table 4 the difference between Table 3 and 1 has been computed. The last column shows the average increase in win percentage for each algorithm.

	Greedy	Smart	Strategic	Opponent	Improved Strategic
Greedy	53.8%	21.5%	18.4%	59.4%	20.9%
Smart	79.8%	54.7%	45.4%	71.5%	40.9%
Strategic	83.2%	57.6%	50.8%	79.6%	46.5%
Opponent	43.4%	31.2%	23.5%	54.0%	18.8%
Improved Strategic	81.0%	61.5%	56.5%	85.7%	50.9%

Table 3: Win percentages of every algorithm against all algorithms. The win percentages are of the player in that row and the player in the row is also the starting player.

If we look at the diagonal in Table 4 it is clear that the starting player has a significant advantage. The size of this advantage in general is dependent on which algorithms play against each other. However if we look at the average values we see that in general the starting player wins 1.8% more. An interesting observation is that the smart algorithm wins 5.2% more games against itself if it is always the starting player while against other algorithms the increase in win percentage is below 1%.

	Greedy	Smart	Strategic	Opponent	Improved Strategic	Average Increase
Greedy	+4.3%	+1.4%	+1.3%	+2.1%	+1.4%	+2.1%
Smart	+0.5%	+5.2%	+0.7%	+0.3%	+0.9%	+1.6%
Strategic	+0.9%	+3.2%	+1.3%	+1.8%	+2.2%	+1.9%
Opponent	+1.9%	+3.7%	+1.9%	+4.5%	+2.7%	+1.6%
Improved Strategic	+1.7%	+2.5%	+1.7%	+2.5%	+1.4%	+2.0%

Table 4: Difference in win percentage between Table 3 and Table 1. The win percentages are of the player in that row and the player in the row is also the starting player.

5 Conclusion and Further Research

Azul is a board game in which the order in which tiles are placed significantly impacts the number of points earned. In particular, the placement of the first five tiles has a significant impact on achieving the maximum score. These first five tiles can be represented as a complete permutation. In [Koo23] it was suspected that an instance of AzulOptimalTiling is a yes-instance, if it is a complete separable permutation or if it is a incomplete permutation that can be completed without creating the (2, 4, 1, 3)- or (3, 1, 4, 2)-pattern. In this thesis we constructed a proof for this conjecture using induction on the structure of a separating tree that represents a separable permutation.
We then analysed the effects two constraints have on the maximum and minimum scores and the orders in which they can be achieved using bottom up dynamic programming in C++. When either constraint is applied the minimum score increases from 89 to 92. We explained the origin of this increase. The number of orders in which the maximum and minimum scores can be achieved also significantly decreases. If tiles have to be placed from top to bottom in five rounds of five, the number of orders in which the maximum and minimum score can be achieved are only 2 and 1096 respectively. We proved that when this constraint is applied on an $n \times n$ board that there are indeed only two orders in which the maximum score can be achieved. The two main factors in these two orders are first placing tiles on the diagonal and then placing tiles adjacent to other tiles. The 1096 orders in which the minimum score can be achieved all start with the first 10 tiles in columns $1, 3$ and 5.

To investigate the importance of the two main factors in achieving the maximum score we created two new algorithms to play against the strategic algorithm from [Koo23]. The opponent algorithm does the move its opponent would pick. This algorithm was the worst performing algorithm. The idea to do the move your opponent would do was integrated into the strategic algorithm This improved strategic algorithm picks the move its opponent would pick if two moves are similar. This algorithm does not have a preference for placing tiles in the diagonal. It wins in 54.8% of the games against the strategic algorithm. To investigate the impact of the preference for placing tiles adjacent to other tiles we removed this preference from both the smart and the improved strategic algorithm. These altered algorithms had a decrease in win percentage of only 1.4%. Since these algorithms did not have a preference for tiles on the diagonal and adjacent to other tiles we concluded that the theory behind the maximum score does not always have a large impact on winning the game in practice. The main component of the algorithms turned out to be leaving the smallest amount of whitespace on its pattern lines. We also conducted an experiment to determine the impact of being the starting player. The starting player wins 1.8% more on average.

In future research the optimal way of collecting tiles from the factories in regards to the opponent and future moves could be analysed. This could then be incorporated into algorithms that focus more on tile collection than on tile placement. The opponent algorithm could also be improved as described in Section 4.6.1. It would also be interesting to create an AI player that plays Azul and then analyse the moves it chooses.

References

[BBL98] Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283, 1998.

[BCG82] Elwyn R. Berlekamp, John Horton Conway, and Richard K. Guy. *Winning ways for your mathematical plays Elwyn R. Berlekamp, John H. Conway, Richard K. Guy.* Academic Pr, 1982.

[BEN16] S. Vialette B. Emerite Neou, R. Rizzi. Pattern matching for separable permutations. *SPIRE*, Oct 2016:260–272, 2016.

[Koo23] S. Kooistra. Achieving the maximal score in azul, 2023.

[VG23] Ladislav Végh and Stefan Gubo. Analyzing game strategies of the don't get angry board game using computer simulations. *International Journal of Advanced Natural Sciences and Engineering Researches*, 7:184–191, 11 2023.