MQP-CS-37986

Zero-th Annual Martin Gardner Exploration Project: 2048

A Major Qualifying Project Report Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

In

Computer Science

Submitted By:

Ethan Catania, CS
Tucker Raymond, CS

Date:
April 23rd, 2024

Advisor:
Michael Engling

Acknowledgements

We would like to thank WPI for giving us the opportunity to complete this project as well as the platform and resources to be able to do so. We would also like to thank our project advisor, Professor Michael Engling, who was a constant source of ideas and inspiration for our project. Professor Engling allowed our team generous flexibility in regard to project direction, which enabled us to create a project tailored specifically to the interests and strengths of our team. We would also like to thank Michael Voorhis, the Systems Administrator for the WPI Computer Science department, for his help in setting up a Microsoft SharePoint site containing information and extra resources regarding our project specifically, as well as general information about the MQP itself.

Abstract

The purpose of this project was to explore the popular game 2048 in terms of expanding the functionality of the existing application, creating solutions for optimal play, discovering useful heuristics to help players perform better in the game, and creating images using the sliding-tile nature of the game. Our expansions to the application's functionality include the ability to dynamically modulate many aspects of the game, such as the placement and value of newly inserted tiles, as well as the size of the board. Additional functionality was implemented to automate gameplay, such that our program makes moves according to the current board state of the game.

Table of Contents

Acknowledgements 2
Abstract 3
Table of Contents 4
Introduction 6
- **Game Choice Investigation** 6

Background 8
- **Similar Games** 8
- **Previous 2048 Solvers** 8
- **Snake Chain** 9
- **Perimeter Defense Formation** 10

2048 Game Expansion 11
- **Game Alterations** 11
- **Gameplay Solver** 13
  - **Chain** 13
  - **Desirable Characteristics of Chaining** 16
  - **Spawning of Tiles** 18
  - **Largest Tile** 20
  - **Empty Tiles** 21
  - **Monotonicity** 21
  - **Adjacent** 22
  - **Shift Load** 23
  - **Equation Formulation** 24
  - **Future Observation** 24
  - **Formulation of Heuristics** 27
- **Image Creation** 30
  - **Systematic Creation of 2048 Boards** 30
    - **Composite Rows/Columns** 32
  - **Color Map Creation** 33
  - **Game of Life** 36
  - **Stereograms** 39
    - **Polyomino Stereograms** 41

Incidental Applications 47
- **2048 Visual** 47
- **Chain Paths** 47
- **SharePoint Website** 47

Results/Conclusion **48**
Optimal Play Algorithm Performance **48**
Heuristics for Human Players **50**
Image Creation Results **51**
Future Work **52**
References **54**
Appendix A: Stereogram Viewing Keys **56**
Polynomial Tile Set Key **56**
Polynomial Tile Board Key **57**
Viewing Key for Stereographic Texture Board with Predefined Tile Positions **58**
Opposite Designation Polyomino Tiles in Two Distinct Orientations **59**
Appendix B: Links to External Applications **60**
Potential Project Investigations **60**
GitHub Repository for Project Application **60**
GitHub Repository for Supplemental 2048 Visual Web Page **60**
All Chain Paths **60**

Introduction

As the first iteration of this MQP, the overall goal was to investigate, analyze, and expand upon some type of puzzle or game. This project stemmed from the love and admiration which the advisor for this project, Professor Michael Engling, has for mathematical puzzles and games, combined with new game ideas and perspectives from the student members of the team. The main inspiration for this project belongs to Martin Gardner, who was responsible for much of the popularization of logical and mathematical puzzles through the "Mathematical Games" columns he created for the *Scientific American* magazine for nearly 25 years.

Martin Gardner's influence on mathematics and puzzles is profound, despite the fact that he had no formal mathematics education past high school. His interest in hexaflexagons, shapes with interesting topological properties able to be made by folding and gluing together strips of paper, led him to make them the focus of his first column in *Scientific American*. The interest garnered by Gardner's first article was so great that he was invited by the editor of the paper to write a monthly column for them (Mulcahy, 2014). One of the keys to Gardner's success was his ability to learn and relay complex ideas and thought-provoking puzzles in a witty and understandable fashion such that his readers could understand the content of his writing without extensive prior understanding of mathematical notation or theory (Martin Gardner, 2021). One of the goals of this project was to continue in this spirit and have the outcomes of this project be in-depth yet understandable as it relates to our analysis and exploration of 2048.

Game Choice Investigation

To decide which game or puzzle to use as the focus for our project, the team started with an initial list of eleven games chosen from personal experience, research, and recommendations from our project advisor. A link to more information regarding each game investigated can be found in Appendix B. From this list of games, we evaluated each one on the grounds of its avenues for expansion, complexity, prior research, level of personal interest within members of the team, and whether potential project ideas for each game or puzzle could feasibly be created by a two-person team in the allotted time. The team narrowed the possibilities from this evaluation to two games - 2048 and Quadrillion, and ultimately chose 2048 for a multitude of reasons. Having prior personal experience with 2048, both of the team members possessed a strong grasp and level of familiarity with the game. The team also believed that 2048 would be a more familiar game to onlookers with more interesting and actionable concrete project ideas and possible game expansions than if Quadrillion had been used as the focus of the project. Finally, the official source code for 2048 created by Gabriele Cirulli is open-source and freely available for use under the MIT license. This allowed the team to directly begin creating an application using the code from the original project, rather than having to recreate the original functionality of the game.

Our initial ideas for game expansion within 2048 revolved around using tiles to create a visually appealing animation or image using the natural spawning and movement of the tiles in the game, altering the game structure and allowing the user more dynamic control over gameplay, changing the physical structure of the board, as well as finding optimal strategies and useful heuristics to be used for improved play.

Background

2048 is a sliding tile game which was released on March 9, 2014 under the MIT License as free, open-source software. The game consists of a 4 x 4 grid where each grid cell can be occupied by a single tile containing a value of the form 2^n where n >= 1. The user makes moves in this game by choosing a cardinal direction, and having all current tiles on the board shift in that direction. After a move is made, a new tile is created on the grid in a random unoccupied cell with a value of 2 90% of the time and a value of 4 10% of the time. Adjacent tiles of the same value can be merged to create a tile with the combined value of both of the merged tiles. The goal of the game is to make a series of moves such that the highest tile on the board has a value of 2048, however the user can elect to continue playing past this point to reach the highest tile value possible until the game inevitably ends.

Similar Games

2048 and other similar games took inspiration from the game Threes, which was released a month prior to 2048 on February 6, 2014 (Conditt, 2015). Threes introduced the concept of a sliding-tile game in which adjacent tiles can be merged to create tiles with a larger value. The creation of Threes was followed by another game, 1024, which the creator of 2048, Gabriele Cirulli, cites as the inspiration for his version of the game. Both Threes and 2048 were met with huge success, quickly reaching hundreds of thousands of plays; however, the fact that the original 2048 game was released for free whereas Threes was purchasable for two dollars made Cirulli's game the more popular and culturally relevant of the two (Schreier, 2014). Numerous clones of 2048 have emerged as a result of its popularity and countless variations of the game now exist with unique changes to the original structure, gameplay, and appearance of 2048 (Hoskins, 2022; "2048: Fibonacci", 2014).

Previous 2048 Solvers

There are many existing algorithms and programs that have been created to solve and optimize 2048, with many well-performing applications utilizing artificial intelligence to determine moves for optimal play. Existing AI programs have used a multitude of board state properties and heuristics in conjunction with learning models such as n-tuple neural networks and convolutional neural networks to be able to evaluate a given 2048 board, with one of the best programs utilizing n-tuple neural networks and temporal difference learning achieving an average score of 600,000 points and reaching the 32,768 tile in 70% of games (Naoki Kondo, Kiminori Matsuzaki, 2019; Jaśkowski, Wojciech, 2016). Different AI players have implemented algorithms such as minimax, Monte-Carlo tree search, and average depth-limited search to iterate through possible future game states within 2048, with one of the most effective algorithms being expectimax (P. Rodgers and J. Levine, 2014; Ovolve, 2020; Naoki Kondo, Kiminori Matsuzaki, 2019). Using an expectimax as opposed to a minimax algorithm performs better for actual gameplay in 2048 because the tile placements which change the value of the current game state

are determined randomly, so always assuming that the “opponent” will choose the option which minimizes the value of the next state will not net the most desirable possible outcome. There are numerous versions of 2048 that have been created, for which an expectimax algorithm would likely perform worse than another algorithm. Evil 2048 is an example of such a derivative, in which the tile is placed on the board in the worst possible position for the player, and is one in which a minimax algorithm would likely perform better than an expectimax algorithm (Richardson, 2015; Hoskins, 2022).

We were able to use a sequence of papers from John Lees-Miller about 2048 to gain a better general understanding of the mathematical complexity of 2048. From the enumeration of all possible 2048 states on a 4 x 4 board, it was observed that the number of possible 2048 boards is well over 1.3 trillion (Lees-Miller, 2017). This early insight made brute force approaches unfeasible and prompted our team to investigate approaches that dynamically applied scores to board states based on an evaluation function rather than depending on the enumeration of every possible board state.

Snake Chain

When 2048 is being played by a human, one of the most understandable and effective heuristics that is used to solve the game is creating what will be referred to as a snake chain. This strategy involves keeping the tile with the highest value on the board in one of the corners of the board, and merging to create a sequence of tiles against the edge of the board in increasing value leading up to the highest tile in the corner. This structure makes it such that when the “chain” of tiles reaches a state in which each next tile in the chain starting from the root tile has a value equal to 2^{n-1} of the previous tile (2^n), all of the tiles in the chain merge into one another and the highest tile will remain at the root. This can be achieved most easily by choosing a corner to keep the highest tile on the board, and a row or column on which to place tiles in the chain, and always preferring moves that increase the value of the tiles in the chain until they are large enough to merge into the root tile, namely the two directions corresponding to the two moves that leave the corner tile immutable.

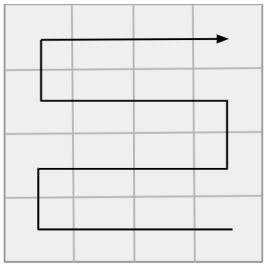


Figure 1: An example of the snake chain with its root in the bottom right corner of the board

The snake chain is the best chain of all possible chains to use due to its stability and the advantageous positioning of new tile placements. Maintaining all of the highest tiles on one side

of the board makes it so that new tiles which spawn in are closer in proximity to the tiles of lower value already on the board, and more likely to be able to merge into other tiles. Following the snake chain also allows the player at least two directions of movement by which their chain will not move. This makes it easier for players to visually maintain, and not erroneously make moves which disrupt the chain.

Perimeter Defense Formation

Many programs built to excel at playing 2048 do not use the snake chain strategy, but rather a strategy known as Perimeter Defense Formation (PDF). The structure of a game board utilizing PDF contains a 3 x 3 monotonic square of tiles against one of the edges of the board consisting of the largest tiles on the board, and manipulating the remaining tiles on the board to eventually increment one of the tiles in the chain and compact it to reach a larger-valued root tile. This method is often employed when attempting to reach the upper limit of the possible tile values on a 2048 board, as if performed correctly, the player has a greater chance to achieve a new highest tile than if other methods such as the snake chain were used (*Perimeter Defense Formation,* 2014). This strategy was not employed by the team, as PDF finds its main utility in achieving extremely high-valued tiles such as the 32,768 tile. Through the course of this project, our solving program never reached high enough tiles to warrant the use of PDF over a snake chain, and thus it was never implemented.

2048 Game Expansion

Our two main goals for this project revolve around creating a program to play the game optimally, or well enough to win the game, as well as using the style and structure of the sliding tiles to create cohesive images. For the basis of our project, we used the official GitHub repository made public by the creator of the game, Gabriele Cirulli, under the MIT License.

Game Alterations

In the process of using Gabriele Cirulli's 2048 GitHub repository as a starting point for our project application, our team made some integral functional and visual changes to the source code for it to be able to accommodate the new functionality which the team integrated into it.

In the original 2048 game, the CSS styles for each type of tile were hardcoded into the game based on their position within the board. Due to the fact that our application provides the functionality to alter the size of the game board, the CSS styles for tile positions within the game were changed to be created dynamically. These styles are created based on the current board size of the game, and updated when the user changes the size of the board. We had originally maintained their existing styles and only added in additional styles as more classes were needed, but found it more concise and practical to simply create all of the necessary classes dynamically on actualization of GameManager.

The most common platform ratio is 1920 x 1080 with a 0.64 : 0.36 ratio (*Desktop Screen Resolution Stats Worldwide, n.d.*). To use the middle 5/6 to present the title, buttons, score, tiles, etc., we are restricted to using 1600 pixels for the height and 900 pixels for the width. With a board consisting of n tiles, and padding for the border, the space our board occupies can be represented as:

(padding * 2) + (tile width * n) + (tile margin * n - 1) = total width (or height)

Our limiting dimension is height with a max height of 900. Using our formulated equation from above to create an inequality with the original tile width, tile padding and margins:

$900 < (15 * 2) + (121 * n) + (15 * n - 1)$
$870 < 121n + 15(n - 1)$
$870 < 136n - 15$
$885 < 136n$ when $n > 6$

This means to accommodate the height of our application to comfortably fit on the viewport we want to dynamically change the size of the board once the board size is greater than

6 and adjust the dimensions of the tiles on the board such that the width and height for each individual tile is equal, and the square shape of the board is maintained.

We also added functionality to the program to allow for the deliberate changing of tile spawning positions and values. We have added HTML functionality in the forms of <input> elements to be able to change where tiles are placed on the board, and the values that those tiles have as the game is being played through. Currently, there are nine different tile spawning styles - two for each corner of the board, placing the new tile on the closest unoccupied cell based on either rows or columns if the desired corner position is occupied, and one spawning a tile in a random position. The two tile-spawning styles for each corner operate identically given the desired corner cell is not occupied. Keeping in style with the game, we constructed the application such that only tiles of value two and four can be inserted into the board, so any boardstate one makes with the application is a valid boardstate that could have come about from a game naturally. The user can choose for each tile input, to make the inserted value two, four, or random. To implement this, our program creates and manipulates global variables within the Game Manager class to denote the tile insert style and tile insert value when adding new tiles to the board.

Another functionality added to our application, as was necessitated by the goals of this project, was the ability to automate the process of playing a game, such that the application was able to successfully and properly make consecutive moves on a 2048 board until the game ends. This feature was utilized in both the optimal gameplay and image creation facets of this project, but in different capacities, as simulating real gameplay required the position and value of spawned tiles on a board to be random, and the implemented method of image creation required that inserted tile positions and values be deterministic. The speed with which the program makes moves automatically can also be adjusted, allowing for the ability to simulate large numbers of 2048 games in a short amount of time.

Gameplay Solver

The evaluation of a 2048 board stems from the pursuit of the ideal 2048 board state. An ideal board consists of a filled board of tiles, having a tile with value 2^{17} in any corner, with each consecutive descending power of 2 adjacent to the next highest value present on the board.

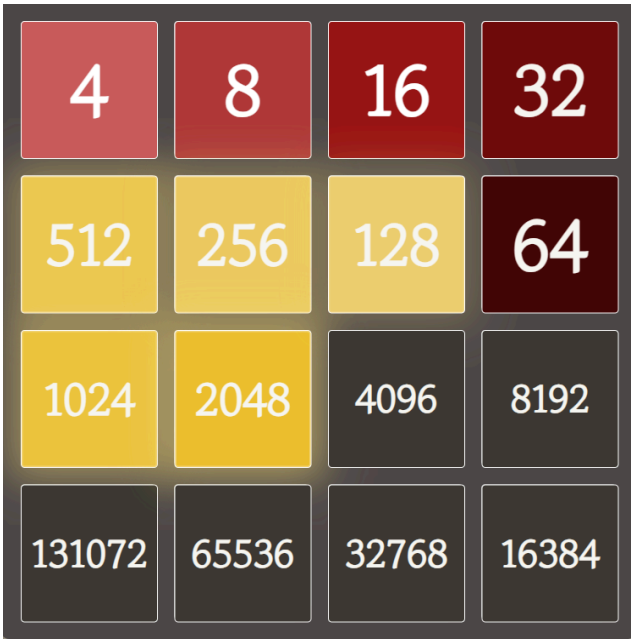


Figure 2: One possibility of the ideal 2048 board state

Chain

A chain is a cardinally adjacent group of tiles starting at the largest tile on the board (the root tile) and spanning to each sequential tile along the chain that is of value $2^{n\text{-}DFR}$, where n is an integer representing the exponent by which 2 would need to be raised by to equal the value of the root tile, and DFR = the integer distance from root tile to the given tile. The ideal chain is one that starts in one of the corners and extends from that corner to another corner whilst following an S shaped pattern and occupying all tile positions on the board. This chain structure is the most optimal use of space on the board that requires the least amount of merges to reach a larger tile state. When any tile within the ideal chain is increased in value, the entire chain is guaranteed to merge in such a way to produce a tile of value 2^{n+1} in the position of the previous root tile.

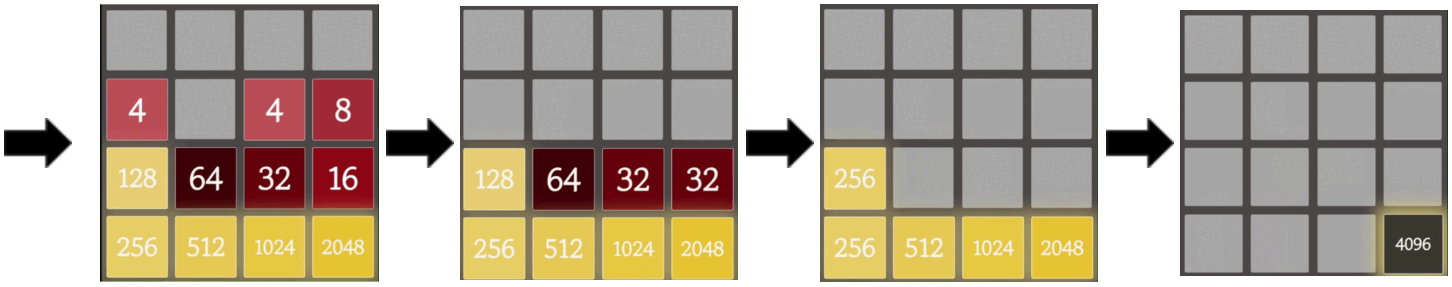


Figure 3: Compaction of a chain structure

Because of this compacting nature, the longer a given chain is, the easier it is to compact, as the necessary value to compact the entire chain decreases as a function of the value of the root tile of the chain and its length. As the chain grows, there is less space on the board with which to combine tiles, but if a complete chain exists (starting at root 2^n to 2^1), then the necessary value to compact the entire chain into one tile of value 2^{n+1} is only two. Once we obtain a tile that can merge into our chain, we make moves to merge those lesser tiles together and subsequently guarantee the maintenance of the structure we had previously defined. With optimal play, a cyclical pattern can be observed that repeatedly builds structures that provide a conducive topological environment for creating large tiles and embedding those large tiles into the structure for iterative large tile creation.

It is also possible with our implementation to shift the orientation of the chain. The position of the largest tiles on the board determine where we will start chaining from. Having one of the largest tiles in the corner ensures we will start at that position and chain in the direction that results in the largest chain. If no largest tile is in the corner, we will then prefer largest tiles that are present along the edge of the board. Otherwise we will choose the first largest tile that was encountered as our starting point for the chain. Our implementation will always attempt to make a snake chain from any starting position. Chains will always search in the direction of the most space on the board with respect to the row and column of the selected largest tile. For example if a tile is in a center tile position, we will attempt to chain in the row towards the direction with 2 spaces until the edge of the board, rather than the direction with just one tile until the edge of the board. The same follows for columns and we will only try to chain in the column towards the direction that has more space until the edge of the board. We will select the longer chain among the row and column directions as our chain for evaluation.

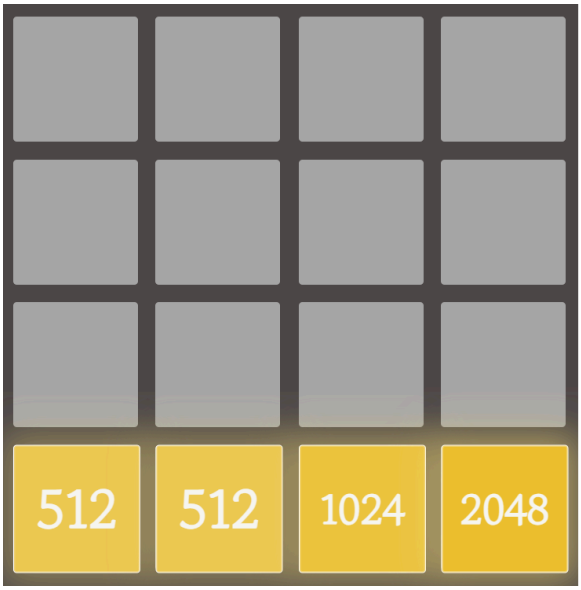


Figure 4: An example of a chain

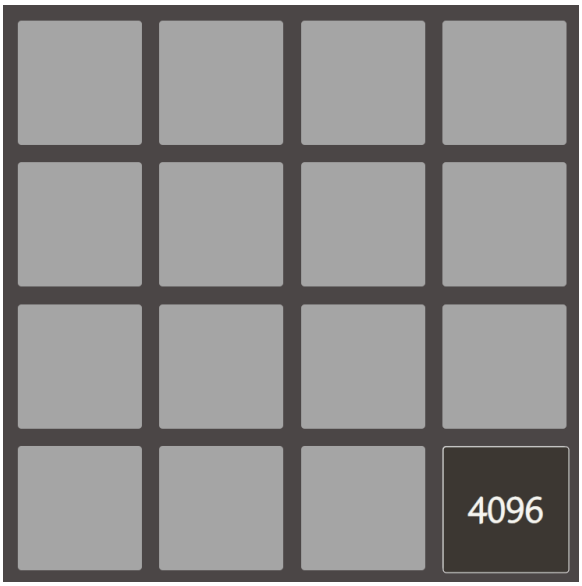


Figure 5: Chain from Figure 4 after 3 moves to the left disregarding tile spawns

Desirable Characteristics of Chaining

A chain may start at any position on the board and extend to any length between 1 tile and 16 tiles. We will refer to the desired sequence of board positions for a chain to occupy as a *chain path*. The length of a chain is based on the presence of recursively cardinally adjacent tile values along a chain path. Chain paths that span the entirety of the board (length 16) will be referred to as *complete chain paths*.

As previously defined, any point in a chain may be a catalyst for compaction, meaning chains that contain smaller values are easier to compact. As the root tile increases in value and as our chain grows in length, the compaction of a chain is more likely to occur as increasingly smaller tiles are present near the end of the chain. To reach increasingly higher tile values, having a *complete* chain path allows for higher potential chain lengths. It should be noted that as the chain itself increases in length, the number of empty tiles with which a tile can spawn decreases. This means the formation of a chain is flexible with regards to its location and ease of compaction, but the player must still be able to merge into that chain in order to compact it and continue the game. When referencing flexibility among chains, we are referring to the lack of stipulations around the positioning of the root tile and the fact that any tile in a chain can be incremented to compact the entire chain from that tile to the root tile. On a 4 x 4 board, there are 69 unique complete chain paths, including reflections and rotations.

Upon making moves to compact the chain, it is possible that the chain path does not inherently guarantee a maintainable, compactible chain structure. A chain that is guaranteed to compact, is more useful than a chain that can only compact conditionally. We must make a move to compact mergeable tiles within a chain, and therefore must subject all tiles on the board to that same direction of movement. It is possible for a tile on the board to be immutable to a specific move when the row or column with which that tile is contained is completely filled with unique valued tiles, or that tile is touching the edge of the board. Rows that are filled are immutable to horizontal moves and columns that are filled are immutable to vertical moves. Tiles adjacent to edges of the board, are immutable to moves made towards those edges. We will refer to this property as *locking,* and the same principle may follow with regards to a row or column (i.e. a locked row contains 4 locked tiles). It should be noted that a tile or row/column can only be locked with respect to a direction of movement and can be locked in multiple directions. Locking is ideal for chain compaction as if the tiles composing the chain are locked, that guarantees the structure of the predefined chain after moves are made.

Likewise, and more often the case, tiles may be able to move freely with respect to a given direction. When attempting to merge part or the entirety of a chain, having more locked tiles that are elements of that chain, increases the likelihood of its compaction. The most conducive structure for locking tiles is the ideal snake-shaped chain. Complete chain paths are flexible and easy to compact, but have a lack of ensured compaction. The snake shape path is a complete path that also provides the stability of ensured compaction within the chain. The snake shaped path is the most suitable complete path with which to combine tiles systematically into larger tiles. The structure ensures that when there are chains that are still yet to be compacted, filling a row/column entirely will lock that row/column and ensure that when compaction is occurring, no tiles that are still to be compacted are shifting in location. This cannot be said for every complete chain path on a 4 x 4 2048 board. As a chain path introduces more turns and

becomes less similar to the snake shaped path, there are more tiles within the chain that must move during the process of compaction, subsequently destroying the defined structure.

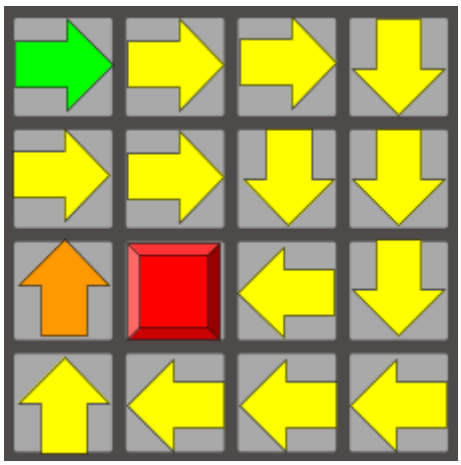


Figure 6: Complete chain lacking guaranteed compaction

From Figure 6, we have one example of a complete chain path. If this path was completely filled with a chain from the root (denoted as a green arrow), to each intermediate chain tile (denoted as yellow arrows), to the end of the chain (denoted as a red box), upon compaction our path would be disrupted.

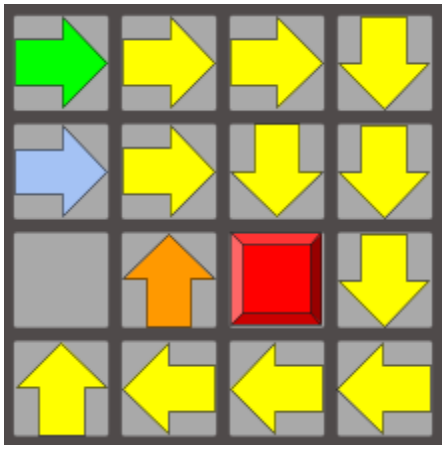


Figure 7: Disrupted chain

To compact Figure 6 we must move in the “right” direction, yielding the board state shown in Figure 7. Upon this move, the disrupted tile (denoted as an orange arrow) must also shift, as merging the end tile of the chain would make that disrupted tile, which was previously part of the chain path, no longer locked with respect to left and right moves. By necessity, in order to compact the chain present, we must disrupt the nature of the chain. If we also account for the new tile insert which must spawn in the position the orange arrow was previously

occupying, if we continue to compact our chain to the position of the blue arrow, we will not be able to merge into the rest of our chain.

Spawning of Tiles

After each move a tile will spawn in a random unoccupied position with a 90% chance of having value 2 and a 10% chance of having value 4. The probability of a tile spawning in any position depends on the number of empty positions and the number of merges that occur as a result of moving from that board state.

Although no tile spawn is inherently bad, when tiles spawn in certain positions, they may leave the board in a less desirable state than before the inserted tile. In the context of creating a snake chain, spawns that force moves and spawns that prevent tile merges are among the least favorable. The most likely occurrence of spawns that force moves will be referred to as rectangle states, as they resemble a rectangle and only allow one direction of movement. Spawns that prevent merges are much more fluid as they can occur between any two occupied tiles that have an empty tile between them. Both of these cases make the continuation of chain building within 2048 much more difficult.

A rectangle state is a board state in 2048 in which only one direction of movement is possible. This occurs when the user makes a move that leaves a filled collection or rows/columns that cannot be merged with each other, with one of those rows having an unoccupied cell. If a tile spawns in that unoccupied cell and cannot be merged, the user is forced to move towards the rows/columns that are completely empty, effectively shifting every tile in the cluster of occupied tiles to a new position on the board. A rectangle state is extremely unfavorable for ideal(snake) chain maintenance as after making the forced move, the next tile to spawn is likely to spawn in a position previously occupied by a tile in the chain. Upon trying to restore the previous chain structure by moving in the opposite direction, the newly spawned tile will be surrounded by the largest tiles on the board, making it incredibly difficult to increase the value of that tile whilst maintaining the same predefined chain structure. When rectangle states occur, they tend to destroy our chain structure and make the continuation of the game exceedingly difficult.

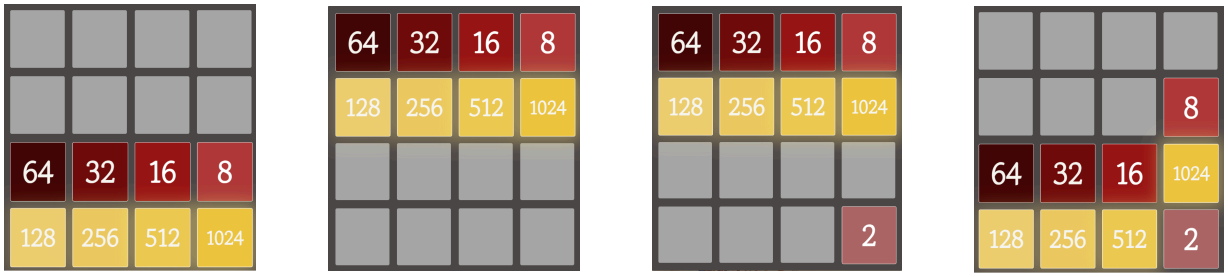


Figure 8: Rectangle state progression

Using a 4 x 4 board, there are only two possible rectangle states that can occur from each of the four edges of the board. For each edge, there exists a rectangle state in the form of 2 filled rows/columns, and 3 filled rows/columns. Reaching either of these states from any edge will result in the behavior demonstrated in Figure 8. Having a single filled row/column from any edge is the least possible detrimental rectangle state on a board as it forces a direction of movement, but because our chain structure occupies at most one row/column, we can simply maintain our chain structure on the other edge of the board from which that chain was previously located. Conversely, having four filled rows/columns is the most detrimental board state as a user cannot make any move and will result in the game ending.

A tile can also spawn in a position that makes a merge that may have been possible before the spawn impossible on the next move. Given there exists two mergeable tiles that are separated by at least one empty tile, if a tile spawns in an intermediate position between those two tiles, the merging of those tiles is now only possible if the value of the spawned tile is able to be increased to be equal to that of the other tiles, or if we are able to make a sequence of moves to stop that tile impeding merge progress. That sequence of moves increases the number of subsequent tile spawns, increasing the number of moves we have to make to reach higher valued tiles while reducing the space with which we have to create larger tiles.

Reaching the ideal board given random tile spawns is nearly impossible because the player encounters states in which there is only one tile spawn position and value that can allow the player to continue the game. With a nearly filled ideal board, the game reaches a state where the final row of the board is in the form [x , x , 8, 16] with x's representing empty tiles. Because we have two empty spaces and need to make a tile of value 8, we need two consecutive 4-valued tiles to spawn or we need two consecutive tiles of value 2 to spawn followed by a tile of value 4. If the first tile spawned is of value 4, it must be in the leftmost position so the user can move to leave the empty space next to the previously spawned tile (i.e. [4, x, 8, 16] → [4,4,8,16]). Spawning only 2s in the final row will leave us with :

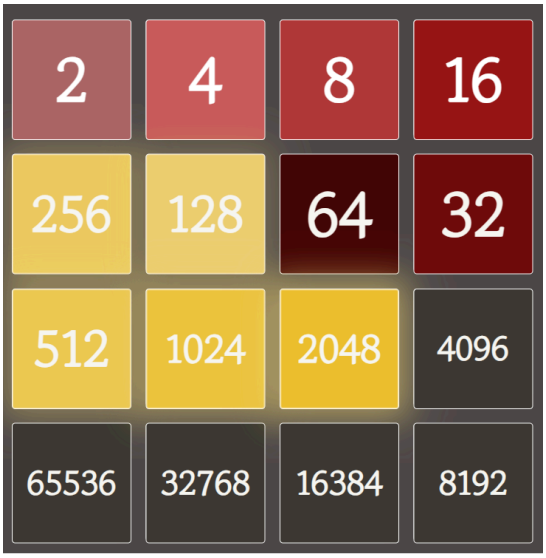


Figure 9: Full snake chain from value 2 to 65,536

As the space remaining on the board decreases, we are forced to rely on the random nature of tiles spawns to be able to create a tile of value 32 in Figure 9. This requires a tile of value 16 to be made having only four unoccupied positions, and another tile of value 16 only having three unoccupied positions on the board. Due to the non-deterministic nature of the game, reaching and even approaching the maximum board state is incredibly infeasible even with perfect moves. It is likely that even the best solvers will be unable to reach the max board state.

Largest Tile

There always exists at least one largest tile present on the board at any time. This value ranges from 2 to 131,072 on a 4 x 4 board. In order to progress in the game, this tile (or set of tiles) and the location of such is the single largest determining factor of the flexibility of chain formation. The largest tile (root) can be located in one of the *corners*, along the *border*, or *neither* the border nor the corner. There exists 552 complete chain paths within a 4 x 4 2048 board which can be observed in Appendix B4. It should be noted that when referring to unique paths we are assuming paths that can be obtained by reflections or rotations of other paths are considered to be the same unique state.

From the enumeration of all complete chains, the corner has the most with 26 unique complete paths. This position is the most stable as it provides two guaranteed locked directions with respect to the root of a chain, and up to four directions if the row and column of the root is also locked. It should be noted that the moves themselves are not guaranteed, but rather if the move is able to be made, it ensures the position of the root tile remains the same after the move is made. The snake chain provides the best balance between the flexibility of chains and the stability of compaction, and can only occur given the largest tile value (or one of them) is located in the corner.

In contrast, the border only allows for one guaranteed locked direction of movement with respect to the root tile, and up to four locked directions if the row and column of the largest tile is also locked. Each border tile has 25 unique complete chain paths associated with it but the ideal chain cannot be made from a border tile. These complete paths are far less stable than the ones associated with the corner due the the likelihood that a complete path with its root tile starting from the border will have tiles within its path that do not remain locked during the process of chain compaction, and the chain structure may be broken.

Having neither the corner nor the border as the largest tile position is the least stable and flexible type of chain, as there are up to four directions of movement to leave the tile in the same position if the tile is locked in all directions and there are no guaranteed locked directions. Although the center has the largest set of complete chain paths, none of those paths are stable enough to be guaranteed to compact and cannot achieve the ideal board state.

Empty Tiles

Another metric of evaluation of a board state is the number of empty tiles present on the board. There are always between 14 and 0 empty tiles on a 4 x 4 board due to the game beginning with two tiles on the board, and having one tile spawn after each move made by the player. The number of empty tiles determines the difficulty and room for error with which a player has to create larger tiles and provides a relative metric for how close the game is to being over. Due to the fact that the game terminates if the number of empty tiles reaches zero and there are no more moves to be made, we can attempt to stay away from this end state by having as many unoccupied tiles present on the board as possible. By maximizing the number of empty tiles on the board, we allow more space for larger tiles to be created, while also reducing the likelihood that an unfavorable tile spawn will result in the end of the game.

Monotonicity

Monotonicity, as it relates to 2048, means having values that are decreasing or increasing in the same direction for every row and column on the board (Kohler et al., 2019). It is advantageous for rows and columns to be structured this way as it allows for consecutive sequences of tile merges. Merging within a row/column given a monotonic structure will maintain the monotonicity of that row or column.

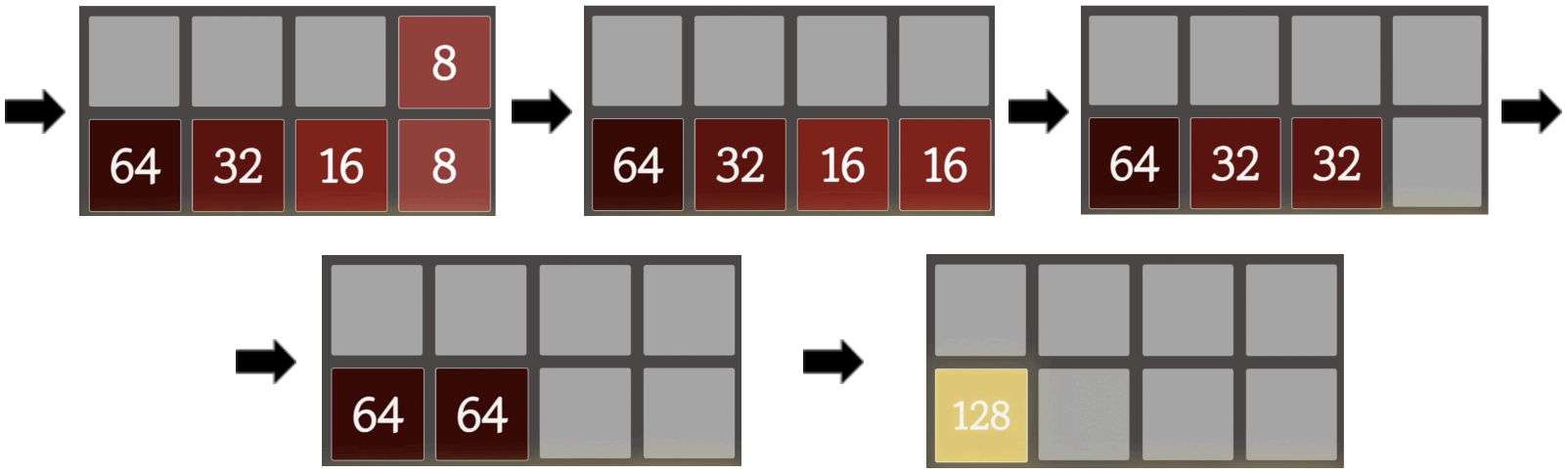


Figure 10: Merging into a monotonic row

Monotonic rows and columns ensure that smaller valued tiles will not become trapped by larger tiles on the board, barring an unfavorable tile spawn. Monotonicity is not necessarily a structure of a board but rather a property that rows and columns can have. Thus the monotonicity of a board refers to the number of monotonic rows/columns of tiles which that board contains.

The ideal snake chain is the most monotonic complete chain path possible as it is monotonic for every row and column, barring two rows or columns that are decreasing in the direction opposite to that of the row/column that contains the root tile. All other complete chain paths may contain some monotonic rows/columns, but the ideal snake chain is among complete chain paths that have the lowest amount of turns, providing the most conducive environment for monotonic rows and columns. By the definition of a chain, every tile in the chain is less than or equal to the precursor, and occupying entire rows/columns sequentially across the board ensures that all values in that row/column are decreasing in the same direction as the chain path. Although this board is not entirely monotonic, 6 of the 8 rows and columns are decreasing in the same direction. There is no possible complete chain path in which every row and column is monotonic, as any chain that moves in opposite directions among the same row or column is inherently less monotonic than the snake chain which never does.

Adjacent

In order to maintain a favorable board state, we want to avoid moving into board positions containing tiles that prevent other tiles from merging by physically blocking them. Whilst using a chain structure, we want to merge tiles that will subsequently compact to create a new root tile. Throughout the course of the game utilizing a chain structure, a smaller tile might become trapped and disrupt the structure of the chain. We want to ensure that all tiles near our chain are also being merged whenever possible to avoid blocking better moves that would have

been possible later if not for that blocking tile. By having the adjacent tiles to our chain be of greater value, we are clustering our largest tiles together while also clustering our smaller and newly spawned tiles together. As our chain grows larger, increasing the value of adjacent tiles allows us to compact chains that may not be complete rather than having to create an entire chain to be able to compact our structure.

Our implementation of the adjacent parameter was to consider all tiles on the board that were not part of the chain structure and sum a proportion of the tile value in relevance to the distance away from the root tile. Each tile value was multiplied by 1/n where n is the euclidean distance of a given tile to the root tile. All weighted tile values were then summed and averaged to generate the adjacent score.

Shift Load

As our board becomes more occupied, it is likely that there are directions in which moves in those directions result in locked rows and columns. Given that a row cannot move, it is possible that we can shift the above/below rows left or right, such that they might merge into the row on the subsequent move. Columns can also have the same properties, but can only be shifted on vertical moves.

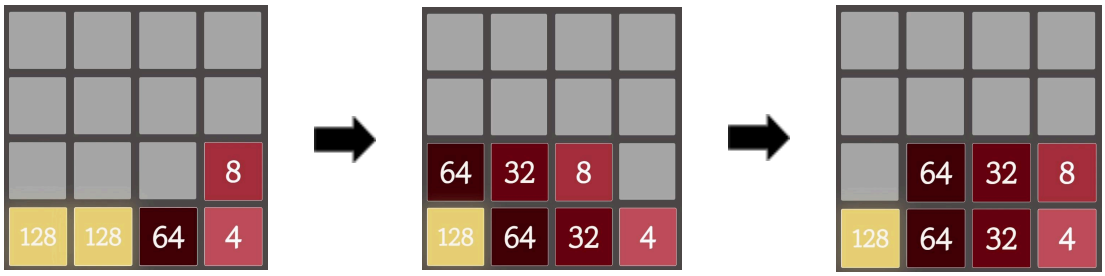


Figure 11: Demonstration of shifting and loading tiles

By aligning tiles of identical value within columns for horizontal moves and within rows for vertical moves, we can guarantee merges on the next move, given that after the move to shift some rows or columns, there is no tile in between the newly aligned tiles. This can serve to cluster our larger tiles together, ensuring that newly spawned tiles are further away from our larger tiles and are less likely to occupy spaces that would prevent other tiles from merging. Shifting allows for more potential tile merges and leads to more empty space on the board whilst maintaining the chain structure.

Equation Formulation

The evaluation function of our program is responsible for the production of a score for a board state. Using the aforementioned heuristics, any board configuration can be evaluated and given a score. The equation used for calculation is a linear equation that is the summation of all previously mentioned characteristics, each multiplied by an independent coefficient value.

```
let eq =  (lg * val1) + (monoScr*val2 ) + (merges*val3) + (chainScr *val4 ) + (shiftLoad*val5 ) + (adjLoad *val6)
return {lg: lg , multip: chainScr , mono: monoScr, adjLoad: adjLoad  , shiftLoad: shiftLoad ,numEmpty: merges, eq: eq};
```

Figure 12: Application calculation of evaluation function

The goal of this equation was to have the values represented by the chosen heuristics be relative to the likelihood that those heuristics present on the given board would result in board states that are conducive to maintaining and compacting a chain more easily. Therefore, the goal of board evaluation was to be representative of the desirability of that state, with each heuristic contributing appropriately to the overall evaluation score for that state.

Future Observation

In order to determine what move to make, we evaluate potential future board states of the current board state. Our parameters at first consisted of comparative statistics, which would evaluate the point score for a board state based on the point differential between the score of the given state, and that of the game state after a move, as opposed to using an objective evaluation function. Our evaluation would use the comparison of the largest tile position, the difference in chain score, the difference in the length of the chain, the difference of the values in a chain, and the presence of a locked row in the row/column which contained our largest tile. One of the reasons we did not use these parameters in our final evaluation was that the evaluated score of a board was dependent on the board state prior to it and thus individual board states were not able to be judged objectively. Comparative statistical analysis was repetitive and made errors in the calculation harder to identify and thus instead of looking at comparing states and making the best move comparatively, we decided to evaluate individual future board states independently. This allowed us to observe the positive correlation amongst the chain variable (denoted as “MTP”), the positioning of the largest tile (denoted as “LG”), the presence of larger valued adjacent tiles (denoted as “ADJ”), and the minimal correlation amongst the remaining variables.

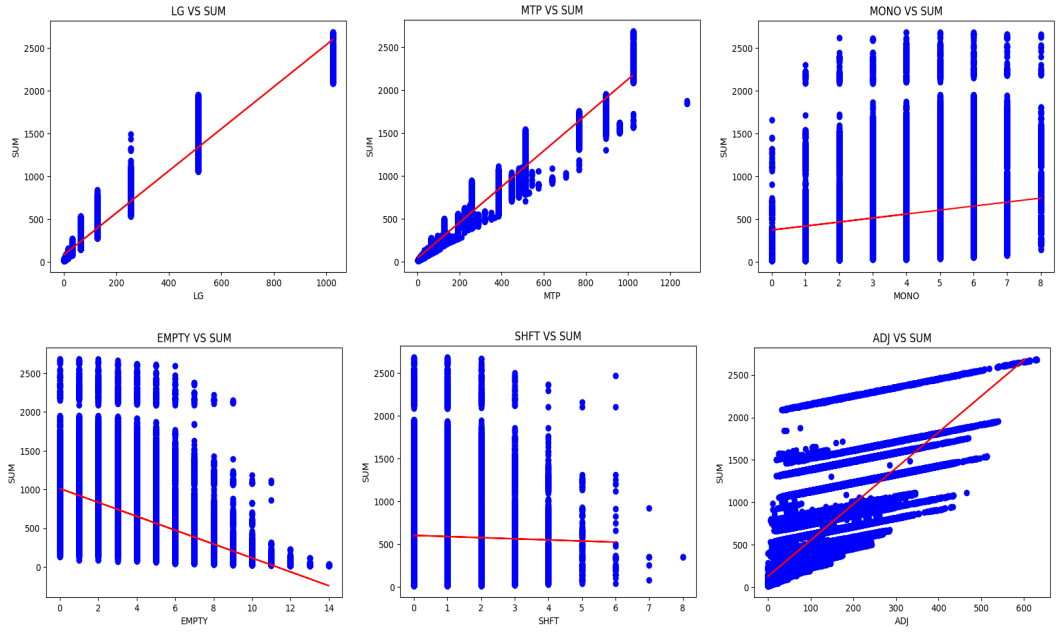


Figure 13: Correlation among initial evaluation parameters

In addition, we were looking at simply the result of a grid after making a move and before a tile would spawn on the resulting board. This was informative but it was not complete. Neglecting to account for the impact that spawning tiles has on a given board was ill-informed, as individual tile placements can have large effects on the score of a given board and the scores of boards resulting from it. From the original game, inserted tiles have a 90% chance of having a value of 2 and 10% chance of having a value of 4. Only one tile spawns after each move and tiles can only spawn in positions that are unoccupied.

Evaluating boards with the inclusion of tile spawns is much better than not considering them, but we were only evaluating our search space with depth 1, and not evaluating many continuations of the current board state. This was detrimental to the performance of our program, as it may be the case that a state is less advantageous than the current state one move into the future, but leads to a much higher-valued state after a series of moves. The performance is improved by looking into more game state trees resulting from moves and tile spawns. This allows us a much more complete view of the possible future game states that might occur as a result of a certain move. Ideally we would look into all possible continuations of each move until the game ends, but this is extremely computationally expensive as each observed game state resulting from a move and a tile spawning has its own set of moves and tile spawns that could occur from that state. This results in an exponential increase in computation as the program continues to evaluate an increasing number of game states resulting from searching through a greater number of possible moves in the future. We want the most informative and least computationally expensive means of determining more favorable states, and thus are looking 2

moves into the future and selecting the move that was the precursor to the board that had the highest score as a result of moves and tile spawns.

As the number of iterations of future board states increases, the program will be able to access and evaluate a greater number of possible game states but at the cost of computation time. To avoid an exceedingly large number of computations, we only want to explore into more future branches of a given board state that are within some range of our best seen score. For example, much of the basis for our ideal play centers around the placement of the largest tile on the board because of the flexibility and stability a user gains by having it in the corner. The largest tile score tends to contribute largely to the evaluation of a board state and therefore, it is likely that any board state in which the largest tile deviates from that position will have a much lower score than boards where the largest tile remains in the corner. However, it may be true also that making a move that shifts the largest tile away from the corner leads to a game state with greater value. By eliminating branches of our search when the evaluation score of that board state is exceedingly low, we can eliminate a substantial number of moves that we would likely never make, but maintain exploration of potentially significant game states. By having fewer states to look through, we can cut off the number of computations we have to make per iteration, and subsequently look at more moves and tile spawns into the future.

When evaluating potential future board states, one method our team implemented to attempt to increase the performance of the algorithm was to make the evaluation of each state representative of the probability in which that state would occur. This was employed for the purpose of making the point values calculated by the evaluation function for a given board state more comparable to one another due to the fact that the certainty of a future board state was factored into the calculation of its score. This functionality would allow the program to choose a move based on the evaluation score of its board as well as its likelihood, preventing it from skewing in favor of valuable states which may be unlikely to occur. This was done by multiplying the calculated value of a state by the evaluation function by the probability that the tile which spawned on that board would be the tile to spawn: $\frac{p}{u}$ where p is the probability based on the value of the tile and u is the number of unoccupied tiles which the tile could have spawned in on the given board. Starting from a given board, each future state for up to two moves was analyzed using this percentage calculation, and collected to compute a final score for each of the four potential moves from the given board to determine which move to take.

When deciding which move to take while taking into account the probability of a state occurring, the results of our algorithm were substantially worse and gave significantly different parameter weights than that observed by the model choosing the next move to make based on the highest-valued future state which is encountered in its search. In attempting to use both the highest sum of scores, and the highest average score to determine which move to take, the model never reached the 2048 tile, and was thus scrapped.

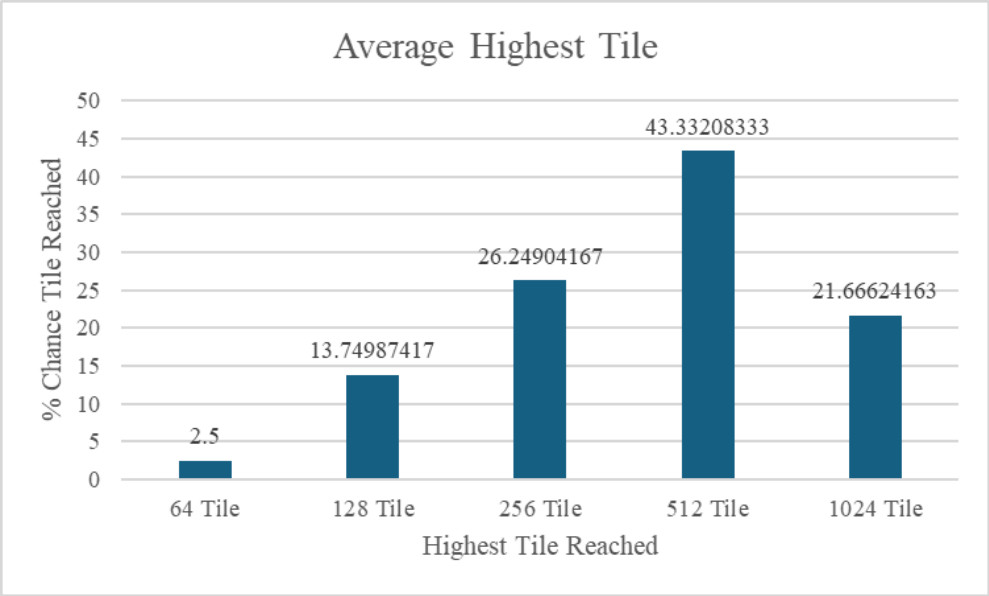


Figure 14: Average highest tile reached for model accounting for probability of given states occurring

In addition, the most optimal weights calculated for the “lg” parameter, presumably one of the most impactful features when evaluating a board based on the chain structure, were often negative. This result was counterintuitive as it suggests that keeping the largest value away from the corner of the board led to improved board states. This may have been a result of complications related to propagating and accounting for game state probabilities when evaluating subsequent game states, as the relative contribution of any individual game state was significantly diminished if a large number of other possible states also existed.

Formulation of Heuristics

Initially, we observed games of 2048 and existing research to determine properties of a game that are conducive to reaching higher tiles. We initially formulated a list of some such areas of interest such as having the largest tile in the corner, locking an important row, using a chain structure, and increasing tile values within that chain. From these observations, we were making moves exclusively upon conditional checks to determine the best move to make. For example, if the largest tile wasn't in a corner, but it could be moved there, we would be guaranteed to make that move. These conditional checks worked fine for simple observations, but as the heuristics we wanted to take advantage of became less clear and harder to quantify, determining moves purely based on the existence or lack thereof of certain properties of a given board state became increasingly difficult. There was also the problem of determining the importance of each relevant condition when multiple conditions were true simultaneously. We determined that our best course of action was to give each of the four available moves from a given state a score from the

evaluation function and make the move with the best score. This proved much more efficient to make better moves but also suffered from the same issue of making the best move when more than one condition might be optimal. The problem seemed to be the weights associated with the aforementioned heuristic. The value produced from each heuristic is directly associated with the values on the board which apply to the given heuristic. For example, if a tile having value 1024 was the largest tile on the board and in the corner, it would receive a score of 1024 for the largest heuristic. Likewise if that same valued tile was the largest and it was in the center of the board, it would receive a score of 1, heavily favoring boards in which the largest tile was located in the corner, and assigning the value of the heuristic to be representative of the board state. It became clear that changing the weights and identifying changes through move observation was not a good metric of determining optimal weights.

To solve this problem, we generated all permutations of a given set of values to determine if there was a simple change such as negating a value or doubling a value that would make the results better. Although this did allow for some minor improvements in the weights being used for scaling the heuristic values, it seemed that we were never able to consistently get to tiles with values larger than 1024.

Consequently, we created a genetic algorithm to play games of 2048 and recursively improve the results that were generated. By starting with either random sequences of integer values of length six or with previous weights that had performed best, and running hundreds of games with those weights, our algorithm ranks those weights based on how likely that set of weights reached higher-valued tiles when evaluating those hundreds of games. The score provided to weights is equal to the expected value of highest tile reached in a game times the probability of that tile being reached. Our algorithm is more likely to select weights that have a higher probability of reaching larger-valued tiles and passing their input parameters into further iterations of the program. That is, parents were selected upon the relative probability of that parent performing well amongst the population, and having a parent that scored twice as well as another parent would mean that it would be twice as likely to be selected for future generations. These selected parents would then be subjected to crossover between pairs of parents. Every two parents have a variable probability that one of the indexes in both gene strings will swap values. This is to ensure that parents that perform well carry on their genes and have the chance to gain other desirable properties amongst other well performing parents. These crossed-over parents would then have a variable chance to mutate a random weight and insert a new random value in that for that weight. This makes new gene strings more likely to explore gene possibilities that are less similar to the previous parents. This new set of parents would then be used to run hundreds of games again and the process would be repeated. Having a higher mutation rate allows a more diverse set of branches to be explored by allowing newer parents to differ more from previous parents. Having a higher crossover rate allows more branches to be explored that are similar to its parents. (GfG, 2024)

After this process was run a few times, we would take the best weights and use them as the parents for future generations. Initially, we had a higher mutation rate to explore more possibilities amongst the gene strings. After running multiple iterations of the genetic algorithm using randomly generated weights, we were able to identify ranges of weights that tended to perform better. We then increased the crossover rate and reduced the mutation rate to allow for future parents to be more similar to those that were identified as performing better. Being able to algorithmically vary the weights used allowed our formation of weights to represent dependence between variables and balance the scores associated with each heuristic such that each individual score contributed appropriately to the overall evaluation score of a board state.

Image Creation

One of the express purposes of this project was to use the sliding-tile nature of the game to create images using the tiles on the board. The two ideas that the team formulated regarding image creation on a 2048 game board were to apply color textures to the tiles to create images or patterns and use stereographic tile textures to be able to view hidden images within the board. In pursuing these ideas, the team implemented a method of systematic 2048 board creation and tile texture application. The difficulty of creating particular 2048 boards given programmatic control of the entire game environment is trivial by way of generating the necessary tile values in the desired positions. Therefore, the team chose to construct boards in a manner more native to the original game. The team chose to only be able to insert tiles of value two and four, which impacts the ability of the program to create images. As a result, it will be constrained by the standard game rules concerning the permissible values for tile spawns, and results in each set of moves and tile spawns taken to get to the final board state being one that could theoretically occur in a normal game of 2048. In addition, we have created multiple sets of stereographic tile textures to be used on 2048 board tiles, which reveal hidden images when viewed in a certain way.

Systematic Creation of 2048 Boards

To create an image with 2048 tiles, an n x n board of colors is created by the user to represent the board which the user intends to visualize. Users of the program specify their intended image using a simple HTML and CSS UI by selecting the desired color from those provided, and clicking on the tiles in the desired positions on the interactive color board. Our program takes in a representation of the different colors on such a board, and returns a set of moves for the program to take and a set of color-tile mappings to apply to each tile. Using those moves and mappings of tile values to colors, the program will create the desired image if it is able to do so.

The method used to create 2048 boards is through the construction of rows or columns of tiles by which to compose a final image. To make an image, our program exclusively constructs a row/column against one of the edges of the board. Once there are n tiles in that row/column and each tile has no adjacent tiles that share the same value, the program makes a move to slide that row/column to the opposite side of the board. This process is repeated until the board has been completely filled with tiles and no further valid moves are able to be made. The row/column on the board used for constructing rows/columns of tiles to be used in the final image remains the same for every row/column created for the image. Individual rows/columns of tiles are created by making a move within the game and placing a tile with either value two or four at the nearest unoccupied position closest to either end of the row/column on the board designated for creation of rows/columns of tiles. Rows/columns used for an image made must be completely filled and immutable to the moves perpendicular to that of the direction they are being slid. If this is not true, and a move is made to slide the row/column of tiles, the moves

made while creating the next row/column will either shift the position of the tiles within the previously created row or column on the board or unintentionally merge tiles within it. Adjacent rows/columns on a final board must also not have any of the same values at the same index, to prevent unwanted tile merges and image structure breakdown when creating consecutive rows or columns.

Figure 15: an example of a filled row of tiles on a 4 x 4 board

To be able to systematically create specific rows or columns, the application creates a map containing all possible rows or columns of tiles containing no adjacent tiles with the same value, along with the moves and tile placements that are necessary to create that row or column. This map is updated whenever the size of the board changes within the application.

With this method of row or column creation, the size of the board determines the maximum tile that can be created. The maximum value of a tile that can be created in a row or column of length n is 2^{n+1}, as always spawning in tiles with a value of four and making moves in the same direction will result in a row or column of the form: [4,8,16, …2^n,2^{n+1}]. By creating 2^n - 1 of these rows or columns and sliding them to the other side of the board, one ends up with a rotation of a board in? this form:

$$
\begin{bmatrix} 4,8, \dots & 2^{n+1} \end{bmatrix} \\
\begin{bmatrix} 8,16, \dots & 2^{n+2} \end{bmatrix} \\
\begin{bmatrix} \quad \dots \quad \end{bmatrix} \\
\begin{bmatrix} 2^{n+1}, \dots & 2^{2n} \end{bmatrix}
$$

Boards of this form contain the largest tile value possible with this method as they are the result of merging the largest tile able to be made in a single row or column the maximum number of times allowed by the size of the board. The largest tile that can be created on such a board has a value of 2^{2n}, and therefore an n x n board must have at most 2n uniquely-valued tiles to use to create a given color board with this method.

Composite Rows/Columns

Our program also is able to create a row/column of a desired image by creating the same (composite) row/column and sliding it to the other side of the board multiple times, such that each of the tile values in the final resultant row/column after all moves is a multiple of each tile value in its composite row/column equal to the number of rows/columns used for its creation. Introducing the ability to have rows or columns on the board composed of multiple lesser-valued rows/columns made it necessary to track the number of times a given row/column would need to be constructed and slid to the other side of the board in order for its tiles to be of the desired value.

The number of composite rows/columns of which a row/column on a given board can be comprised of depends on the positioning of that row/column within the image, and from which edge of the board the program is creating them from. Any given row/column must not be made from merging more than $2^n - 1$ composite rows or columns, as any such row/column would be unmakeable due to size constraints on a board of size n x n. One of the rows/columns, depending on which the program is using to build images, at either end of the board, must be constructed with no composite rows or columns, as once the program creates all necessary rows/columns except for the last, the row or column on the edge of the board used for creating the rows/tiles of tiles will be the only row/column left with unoccupied positions on the board, such that when the final row/column is created, it is not able to be slid to the other side of the board, and the game is left in a final immutable state. Thus, the selection of the board edge for generating and sliding rows/columns of tiles depends on whether one of the rows/columns against the edge of the desired board can be created without any composite rows/columns, and on whether sufficient space exists to create each other row/column on the board when inserting tiles from either edge.

The first row/column created on the board is the furthest row/column from the board edge from which tiles are being inserted, and can be constructed with a greater number of composite rows/columns than any other row/column on the desired board. This is due to the fact that there are no preexisting rows/columns on the board, and the program can use the entirety of the board to create rows/columns with the necessary values to construct that row/column. As rows/columns of a final board are created, less of the boardspace is available to create composite rows/columns, decreasing the number of possible rows/columns which might be used to create a given row/column. The number of composite rows/columns which a final row/column can be composed of based on its distance to the row or column against the edge of the board from which rows/columns are being created is $2^{d+1} - 1$ where d is the distance from the current row being created to the edge row creating tiles.

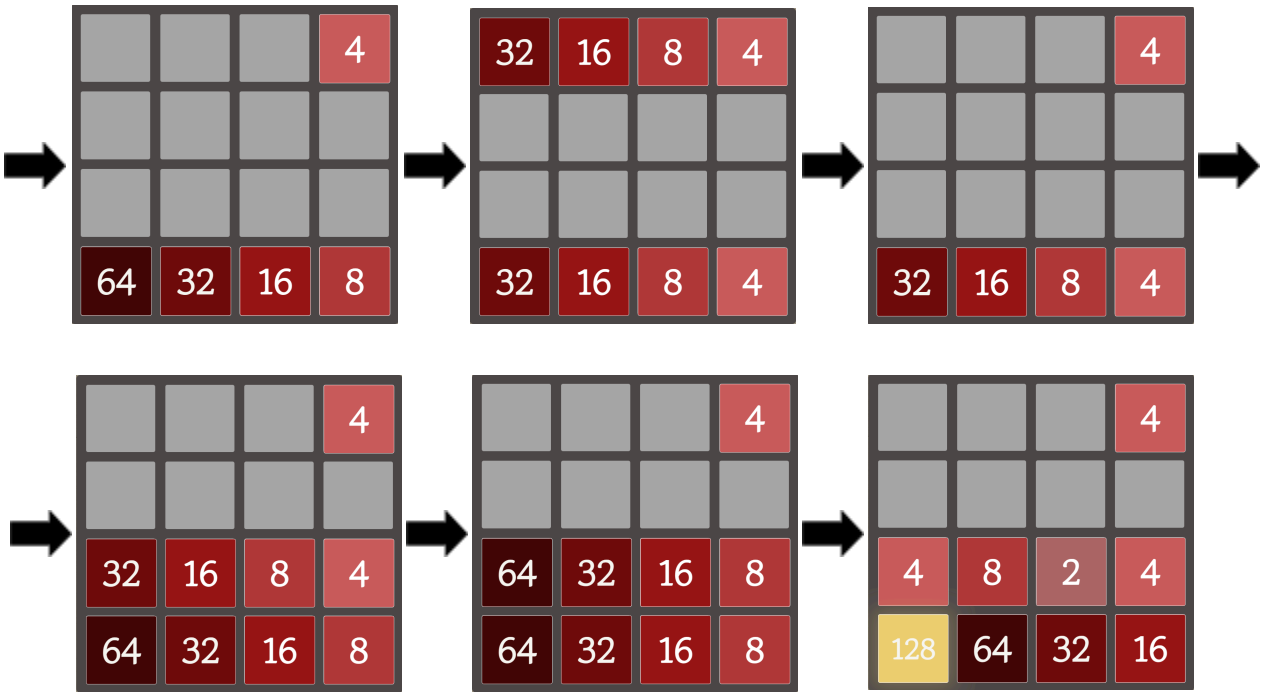


Figure 16: Final row created by sliding four identical composite rows to the bottom of the board

Color Map Creation

To change a 2048 board into a board of colored tiles, 2048 tile values are mapped to the colors that appear within the final desired image, such that each tile value is associated with a single color, but a color can have multiple tile values associated with it. The positioning of colors in the final color board will determine the number of tile values necessary to delegate to each color. On a final color board, if a tile has no tiles adjacent to it that are the same color as it, it can be represented with a single tile value, as in the process of image creation, that tile will never be adjacent to any tiles it can merge with. If a final color board contains a tile which has an adjacent tile with the same color, then tiles of that color will need to be represented by at least two tile values, such that they do not merge into one another when they are in their desired positions on the final board, as further moves may cause those adjacent tiles to merge together if they have the same value. These restraints bound the number of colors that can be used on a given board based on its size and position of each tile. The user is only able to create color board, taking n as the size of the board (n x n) in which:

$$ColorsNoSameAdjacent + ColorsSameAdjacent * 2 <= 2n$$

This inequality only marks the physical limitation of the number of colors that can be used as a function of the size of the board, and the adherence of a board to this color limitation does not guarantee that the image creation program will find a tile-mapping to satisfy the desired color image.

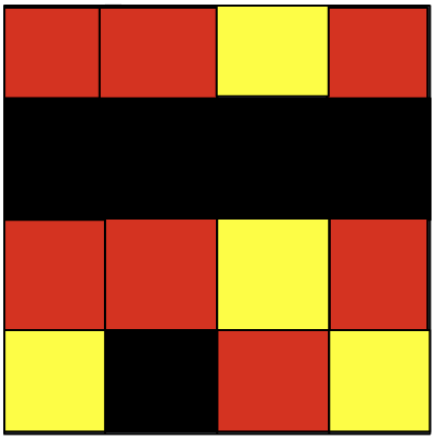
To find rows of tiles and color-value mappings that are able to create the desired image, the program finds the number of tile values necessary (t) to represent each color (1 or 2), and loops through all combinations of choosing t values from 2*n (the total number of tiles available). For each distinct combination of t tile values, the program enumerates all permutations of that combination, such that all possible combinations of tile values, and all permutations of each tile value representing each color are exhaustively searched. For each permutation of color values, the program then creates all row possibilities using the color mapping provided by the permutation, as the same combination of tile values and mapping permutation can yield multiple tangible rows due to the fact that multiple tiles can represent the same color. For each tangible row, the program checks to see if that row is contained within the computed list of filled rows without adjacent tiles of the same value, and can thus be created.

If a permutation of tile values according to the colors of the desired board allows for the creation of all necessary rows to create the image, the program makes further checks to ensure that no adjacent rows have tiles of the same value at the same index, as when those consecutive rows would be slid adjacent to one another on a 2048 board, the same-valued tiles would merge together rather than remain independent, breaking the structure for image creation. The program must also check to ensure that each row can be created based on its position on the board. Due to the fact that rows can be created using multiple composite rows, the program must also ensure that in the process of creating a given row, any of the values in any created intermediate rows do not merge with the values already present on the board.

In the process of execution, our program first iterates through the rows of the desired image for each combination of distinct tile values, and each permutation of tile values into color assignments. If the program generates a set of rows that can compose the desired image, it returns the associated set of moves, tile placements, and values necessary to create the image with rows. If the program fails to find a viable color mapping with rows, the program iterates through this process again using the columns of the image as its inputs instead of rows. If a viable mapping is found, the associated moves and tile placements are translated to apply to columns and returned, otherwise the function returns an empty set of moves, displaying an error message to the user.

4x4 Board: 2*4 total tiles available = [2, 4, 8, 16, 32, 64, 128, 256]

Needed Colors: 3 {Red, Black, Yellow}
Total Needed Tiles: 5



Number of Necessary Tiles per Color:
- Red, Black: 2 (touching adjacent tiles of same color)
- Yellow: 1 (no adjacent touching tiles of same color)

Combinations:[[2, 4, 8, 16, 32], [2, 4, 8, 16, 64], …]

Permutations: [[[2,4], [8,16], 32], [[2,4], [8,32], 16], …]

(using [[2,4], [8,16], 32])

Row 1: [[2,4,32,2], [2,4,32,4], [4,2,32,2], [4,2,32,4]]

Row 2: [[8,16,8,16], [16,8,16,8]]

Row 3: [[2,4,32,2], [2,4,32,4], [4,2,32,2], [4,2,32,4]]

Row 4: [[32,8,2,32], [32,8,4,32], [32,16,2,32], [32,16,4,32]]

If all rows can be made and have no vertically adjacent tiles of the same value, return associated set of moves of each row

---

Figure 17: Color map creation method description and visualization

Issues arise in our program as the board size increases, as the search space for filled rows/columns of the given board becomes exponentially large. As a result, the size of the board has been limited to eight when the ‘designer’ tab is open. Below is a table containing the number of stable, filled rows/columns of n tiles as the size of the board, n, increases.

Board Size	Number of Filled Rows/Columns	Total Distinct Encountered Rows/Columns
1	2	2
2	6	12
3	26	67
4	142	431
5	850	2,923
6	5,626	20,738
7	40,882	156,031
8	320,902	1,245,210

Figure 18: Table containing the number of filled rows that can be made in a single row, and the total distinct rows seen for board sizes 1 to 8

These numbers were calculated using the program itself, by creating a board of a given size and enumerating all of the possible rows of tiles that can be created with the given method of creating rows/columns. Boards larger than size 9 x 9 were unable to be calculated, as the machines running this program ran memory before all enumerations could be calculated. Thus, if images are to be created on larger boards, a different method aside from row/column enumeration must be used due to memory constraints.

Game of Life

Martin Gardner's October 1970 edition of his column for *Scientific American* would rival the success of his first column regarding hexaflexagons, when he wrote about John Conway's cellular automata dubbed the "Game of Life". Conway's Game of Life is played on a two-dimensional grid consisting of square tiles, where at each step of the game, a cell can be in one of two states - alive or dead. The game is played by constructing an initial configuration of alive and dead cells on the board, and taking discrete steps in which the state of each cell is altered based on a simple set of rules regarding the states of the eight tiles surrounding it:

- Any cell in the alive state with two or three alive neighbors remains alive in the next step, otherwise its state changes to dead in the next step.
- Any cell in the dead state with exactly three living neighbors changes to the alive state in the next step, otherwise its state remains the same.

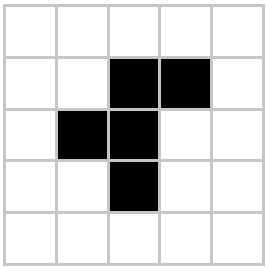


Figure 19: “R” pentomino in Conway’s Game of Life

These seemingly simple, but carefully constructed rules admit huge complexity and unpredictability as the iterations of an initial configuration play out. This result was one of Conway’s goals when designing the game, as he stated in an interview, “I just sort of thought if you couldn’t predict what [an initial configuration] did, then probably that’s because it was capable of doing anything” (Numberphile, 2014). This notion of unpredictability would be proven true through Conway’s proof of a universal computing machine within Life constructed using structures of live cells interacting with one another in complex ways to represent and store data, as well as perform logical operations on it (Berlenkamp et al., 1982). This proof has also been realized through multiple programmatic implementations of such a computer in the Game of Life (Rendell, 2011; Loizeau, 2018).

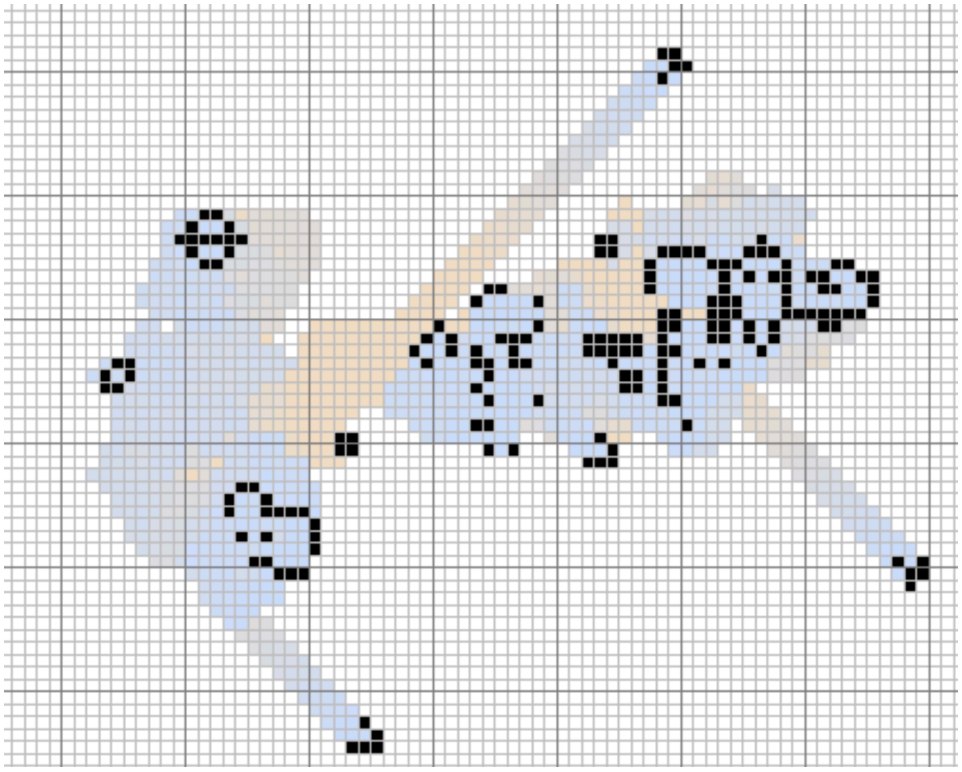


Figure 20: “R” pentomino in Conway’s Game of Life after 168 generations

Martin Gardner's multiple columns and chapters exploring Conway's Game of Life set out to explore and outline many of the findings that have been made regarding intriguing patterns and properties that can emerge given the unique rules of Life. Groups of live cells can create recognizable formations such as still-lifes, oscillators, gliders, glider guns, garden of eden patterns, self-replicating patterns, and much more (Gardner, 1983). Gardner was able to share the exciting complexity of Life and its intricacies to people around the world, and our project aims to commend Gardner and his accomplishments by simulating the game he popularized inside of 2048.

Using the aforementioned image creation method, our team was able to implement the creation of sequential 2048 boards representing iterations of Conway's Game of Life without excessive difficulty. The primitive rules of the game allow for easy computation of consecutive boards, and searching for rows or columns of tiles to compose the next board from. Additionally, the use of only two colors to create any given board in the Game of Life, representing the "alive" and "dead" states, limits the maximum number of tiles needed to represent a board to four, with at most two distinct tile values for each color. This limits the number of color tile assignments calculated for each permutation of distinct color values, allowing boards simulating Life to be created quickly and more efficiently than boards which would require a greater number of distinct tile values to represent them.

In our implementation of Life in 2048, the two different ways the game can be played is with or without tile wrapping, which can be altered by a user of our application. With the wrapping feature off, the board space is represented by a bounded square, having the neighbors of each tile be exactly as they appear, with cells on the edge of the board only having five neighboring cells and cells in any of the corners of the board only having three neighbors. When the wrapping feature is enabled, each next iteration of the game is calculated as if the game space was toroidal, in which the cells at the bottom of the board are neighbors of the cells at the top of the board, and the cells at the left side of the board are neighbors of the cells on the right side of the board.

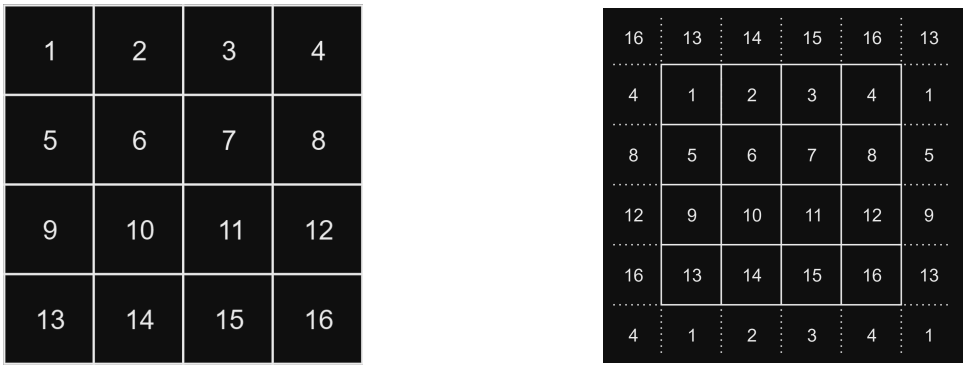


Figure 21: Visual representations of computed adjacent cells with and without wrapping

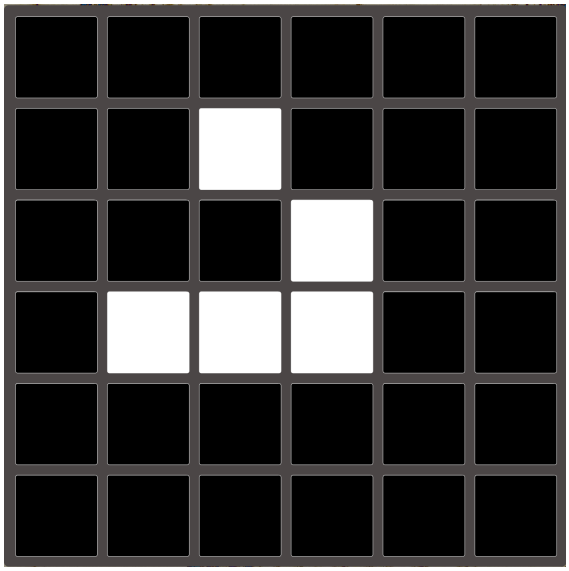


Figure 22: A “glider” from the Game of Life simulated in 2048 using image creation

Stereograms

A stereogram is a pair of similar images, which when viewed simultaneously either by the diverging or converging of the eyes, produces a three-dimensional image. It is generally the case that having two eyes (binocular vision) is necessary for accurate depth perception when looking at and focusing on things normally (Boyd, 2018). The spacing of the eyes, referred to as binocular disparity, is one of the main reasons for this, as it allows for both of the eyes to see the same images from two different perspectives, which the brain combines into a single view with a sense of depth using the differences in each image (Kelley, 2021). Stereograms take advantage of our binocular vision and our brain’s ability to converge two images into one and allow us to perceive portions of a two-dimensional image as closer to us, or further away. When stereograms are seen by diverging or converging the eyes, the line of sight for both eyes intersects at a point either in front of or behind the image. Doing this causes each eye to look at two different parts of the same image, and the viewer is able to discern a “hidden” image with perceived depth disparity resulting from precisely-spaced horizontally-repeating patterns within the image. This effect is accomplished by having two nearly identical images, with different portions of either the left or right image shifted horizontally a certain distance. If this is done correctly, the brain will interpret the two distinct images as one, and the slightly shifted portions of the image will appear as either closer or further away from the viewer depending on how they are viewing the image, and the distance and direction in which those portions were shifted.

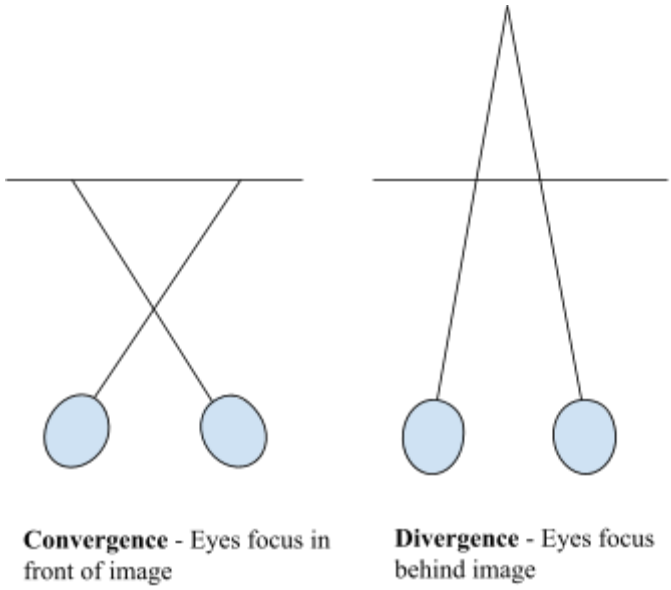


Figure 23: Stereogram viewing method visualization

Stereograms were originally created to be viewed with an instrument called the stereoscope, invented by Sir Charles Wheatstone in 1832, which did the work of manipulating the line of sight for each eye to be able to see a given image with depth without having to converge or diverge one's eyes (Britannica, 2024). The first random dot stereogram was created by Dr. Bela Julesz in 1959 (Tyler, 2014). The images created by Julesz were images of random dots which could be horizontally manipulated to create the illusion of depth rather than using regular images. The relative ease of creating a stereogram using random dots led our group to choose to create stereograms in this manner for this project.

Our team was inspired to attempt to incorporate stereograms into our project, as the horizontally-repeating board structure of 2048 makes it so that viewing the tiles of a 2048 board with stereoscopic vision is relatively simple if one is familiar with the concept. The team created two types of stereographic tile textures - textures made in a set to be viewed with a variety of other textures from the set, and textures meant to be viewed at certain positions on a board in conjunction with other specific textures. The sets of textures created by the team are designed such that a unique image emerges when two tiles from a set are viewed together using stereoscopic vision. While stereogram creation specifically was not one of the main focal points of this project, and thus not included in our project application, the intriguing and almost magical nature of these images is something the team believes that Gardner would have enjoyed, and have included a few designs for tile textures on a 4 x 4 grid, as well as a set of tiles which create individual polyominoes when two tiles are viewed side-by-side, as is depicted in Appendix A.

Creating a set of tile textures in which each texture is meant to be seen in a particular position on a 2048 board was much less modular than creating a set in which its tiles might be viewed interchangeably, as textures of these kind can only be viewed as intended if the game board is completely filled with tiles, and each texture is in its correct location on the board. If applied to tiles in a 2048 game, the stereographic view of most given game states during play would likely be distorted and indiscernible. Thus, the team did not choose to examine textures of this kind with much rigor. Figure 24 is an example of such a board of tiles in their desired positions, for which the viewing key can be found in Appendix A.

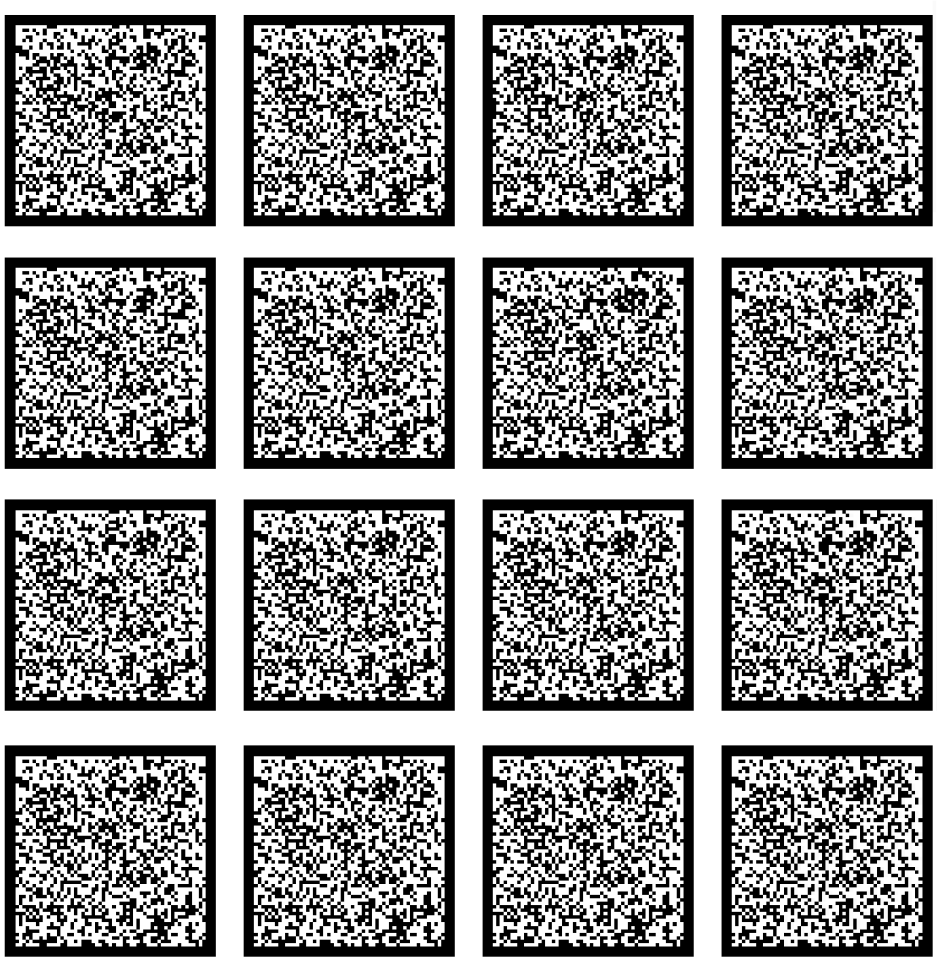


Figure 24: 4 x 4 grid of stereographic textures to be viewed in a specific orientation

Polyomino Stereograms

A polyomino is a geometric shape created by joining the edges of multiple squares, with the number of concurrent joined squares corresponding to the degree of the polyomino. The

polyominoes able to be viewed using the tile texture set provided in this report are polyominoes composed of either five or six squares, referred to as pentominoes and hexominoes respectively. The use of polyominoes as hidden images within the tile texture set created by the team was inspired by Martin Gardner. Gardner made reference to and used polyominoes as the subject of multiple columns that he wrote for Scientific American including “More about the shapes that can be made with complex dominoes (poly and pentominoes” and “Polyominoes, polyiamonds and polyhexes more about tiling the plane”, and as a result helped cultivate greater popularization and understanding of the topic (Miller, 1999; Toth et al., 2017).

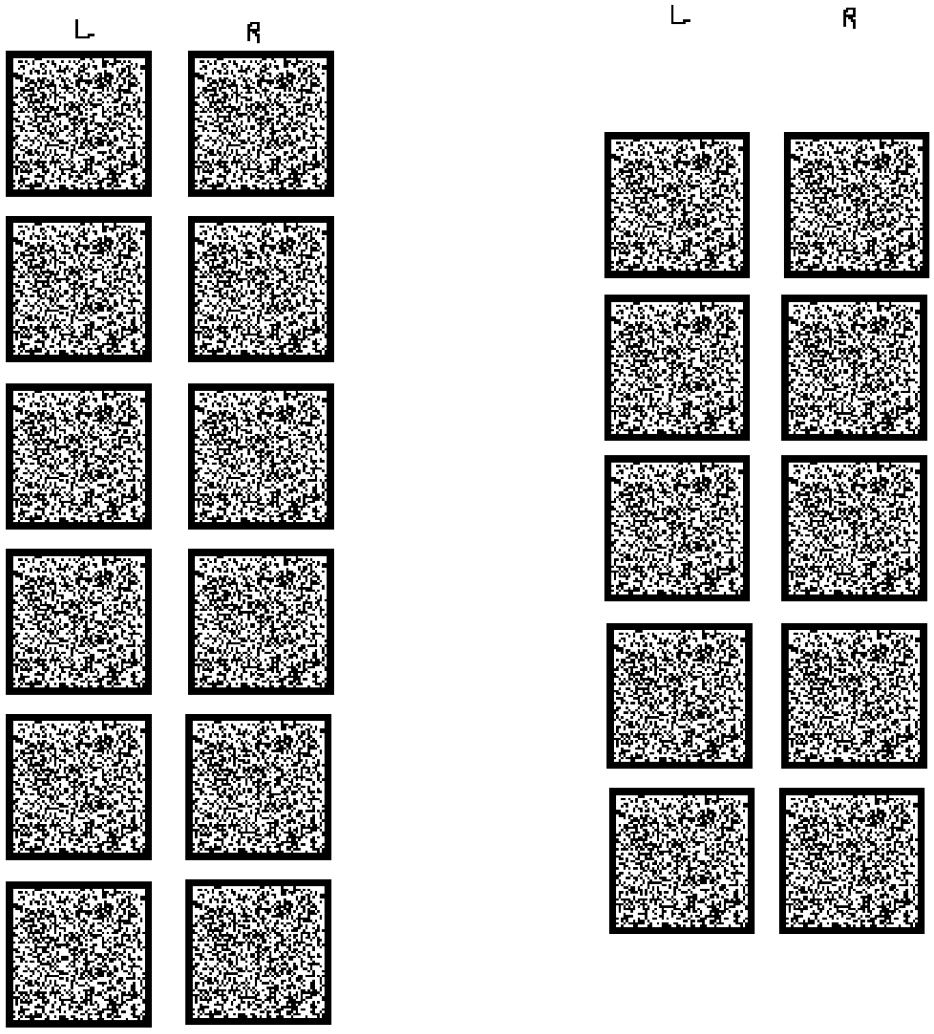


Figure 25: Set of left and right designations for each tile in polyomino texture set

The above figure depicts a set of stereogram tile textures which reveal polyominoes when viewed with stereoscopic vision. To make a set of tiles which could be viewed cohesively using stereoscopic vision alongside any other tile from the set, our group thought to make some part of

the final image identical for every texture in the set, and have each tile-pair contain one additional unique difference, such that the differences that exist on each tile do not interfere with one another when viewed together. To implement this, the set of textures we have created reveals certain pentominoes when a texture is viewed with its corresponding left or right counterpart, and hexominoes when a texture is viewed with a texture other than itself or its left/right counterpart. Every texture contains a central column, with each texture containing an additional section which protrudes from the middle column in a position unique from all other tiles in the set. A viewing key for this tile set can be found in Appendix A.

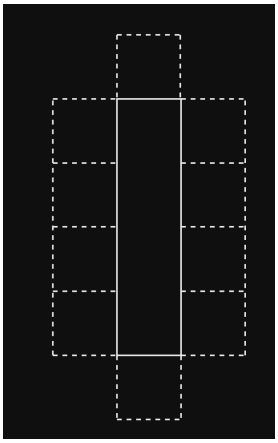


Figure 26: Representation of the central column and each possible protrusion visible on stereographic textures representing polyominoes

Pentominoes are seen when a texture is viewed with its counterpart due to both images containing an identical protrusion from the central column. Therefore when viewed with stereoscopic vision, only one protrusion is visible, making the resulting shape consist of five squares. Hexominoes are seen between two textures that are not identical or their counterparts due to the different protrusions that each tile contributes to the viewed image. Figure 26 contains a depiction of this central column along with the possible placements of the protrusions on each column.

Whether the stereographic polyominoes appear raised in front of the image, or sunken into the image depends on the positioning of the left and right tiles, and the form of stereographic vision being used to view it. In Figure 28, the hidden polyominoes within the tile textures appear raised off the rest of the image when viewed using convergence of the eyes, and sunken into the image when viewed with divergence. However, if the viewing order of the tiles is switched, with the tiles designated as left and right swapping places, then the hidden polyominoes within them would appear raised when viewed using diverging stereographic vision, and sunken into the image with converging vision. This effect occurs due to the way each eye perceives each distinct image differently based on the viewing method, as well as being a result of the careful construction of each tile texture.

The method used to create these images was to first create a black and white random-dot pixel image, duplicate it, and place the copy horizontally adjacent to the original image such that a stereographic image can be seen using these two images by converging or diverging the eyes. To give depth to a given section of the random-dot stereogram, certain sections of one of the images were copied, specifically those outlined in Figure 26, and pasted onto its adjacent duplicate, translated a small horizontal distance. This results in the brain being able to discern the exact same pattern of pixels from both eyes even though the physical entities each eye is viewing are distinct. All tile textures in the created set have designations as either right or left. Images within tiles designated as right were translated to the left to appear raised from the rest of the image, and tiles designated as left have sections translated to the right to appear raised, when seen using converging vision. An image depicting the same two tiles of opposite designation adjacent to one another in positions according to, and against their designations can be found in Appendix A. With this image, it can be seen that both the horizontal positioning and method of stereoscopic vision influence the final perceived image when this set of tiles is used.

Figure 27 shows two textures from this set, with the portions of the image able to be seen with depth depicted below them to further elucidate the manner in which the textures were constructed.

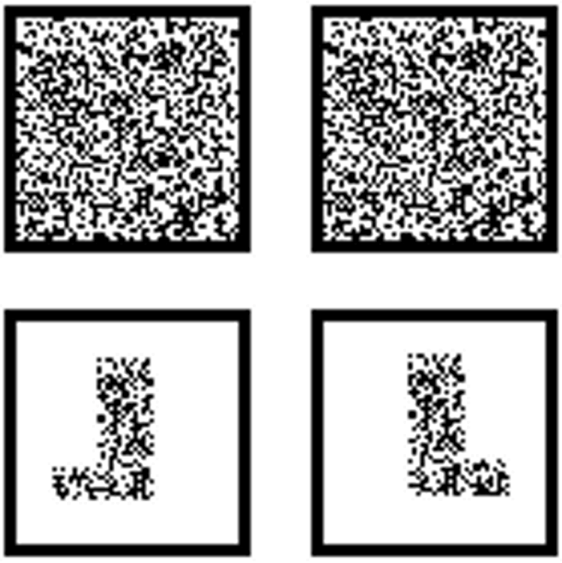


Figure 27: Two tiles from the created set, with representations of the column and uniquely-positioned protrusion for each texture

This way of using stereograms is not completely ideal for use within 2048, as in order to properly view the image created by two horizontally adjacent tiles, the two tiles must have different right/left designations. Textures from the same designation have the central column translated the exact same distance, leaving that part of the image identical in both tiles. Thus, if viewed with stereographic vision, the middle column will fail to appear with any depth, and only

the individual protrusions from each tile will be visible. For this effect to work properly on a 2048 board, the textures of tiles on the board must be decided based on the position of each tile, and its horizontally adjacent tiles, such that each row of tiles on a given board has textures of alternating right/left designation.

This alternating designation, however, results in the images contained within successive columns of tiles seen using stereographic vision alternating between appearing lifted up off the image, and sunken into the image regardless of whether the viewer converges or diverges their eyes. This is due to consecutive pairs of horizontally adjacent tiles being merged using stereographic vision alternating between designations, {R,L}, {L,R},{R,L}. To tile a 2048 board with these textures and have them be viewed as intended, the textures themselves could not be mapped directly to tile values, as that would allow for tiles with textures of the same right/left designation to be a horizontal neighbor of one another. These textures would instead have to be mapped to tiles based on their horizontally-neighboring tiles so as to avoid this from occurring.

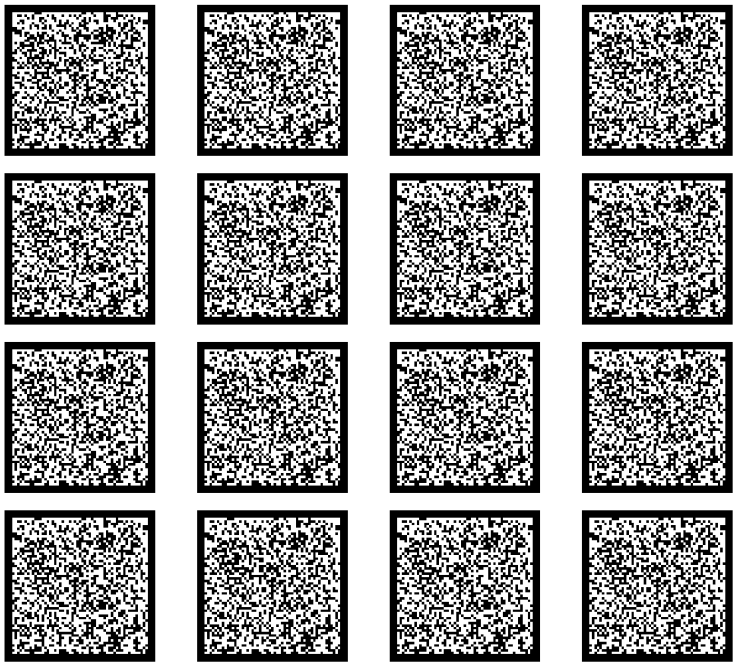


Figure 28: 4 x 4 set of tiles from the polyomino tile set containing alternating left and right designations for horizontally adjacent tiles

Figure 28 depicts a 4x4 board of tiles constructed from the tiles in Figure 25, with each column of tiles alternating between left and right designations. If the above image is viewed while diverging the eyes, the middle column from the three visibly merged columns should contain raised images while the two outside columns' images are sunk into the page, with these effects being reversed if the viewer converges their eyes instead. A viewing key for this image can be found in Appendix A.

Incidental Applications

In the process of completing this project, the team created a few incidental applications and sites which were relevant to it, and aided in its completion. The descriptions and purposes of these applications are detailed below.

2048 Visual

We created an additional program to help with quick visualization and state retrieval of given game states. In the process of discovering algorithms to result in more desirable game states, and finding ways to structure tiles in a way that could create images, we constantly found ourselves drawing and redrawing square boards to represent boardstates, and this application allows this to be done easily. It also has functionality such that it can print out the current board state as an array of integers for quick comparison and evaluation. Users can also input a board state in the form of a comma-separated array into the given <input> element to have the corresponding state be displayed on the board. A link to the GitHub repository containing the code for this program can be found in Appendix B.

Chain Paths

Another supplemental program that our team created was to enumerate the number of complete chain paths—paths of tiles that span every cell on a given board—to verify that the snake chain was the most optimal possible chain of tiles on the board. This program yielded the number of complete paths from every position on the board, and provided our team insights into the number of topologically unique paths from the different classes of starting tiles on the board (corner, edge and center) to determine the most optimal tile from which to begin a chain. This program also verified that the snake chain is the most monotonic complete chain on a square board, and is among complete chain paths with the lowest possible number of turns.

SharePoint Website

The SharePoint website set up for this project is meant to serve as an interactive platform to learn more about our project and the inspiration behind it, a place where people can find related external resources to other games, puzzles, and work done by Martin Gardner, as well as a starting point for the Martin Gardner Tribute Project, to give potential future students partaking in this MQP a sense of what the project is about as well as inspiration for their own project. This SharePoint site will hopefully become a platform used to organize the great many puzzles, games, videos, and oddities which the advisor of this project has encountered, and be able to easily share and have access to all of them from a central location.

Results/Conclusion

The purpose of this project was to appreciate the work which the late Martin Gardner has contributed to recreational mathematics through the expansion and modulation of a puzzle or game. Our team investigated and expanded upon the sliding-tile game 2048 by implementing a program to play the game optimally, and using the native structure of the game to create images within it.

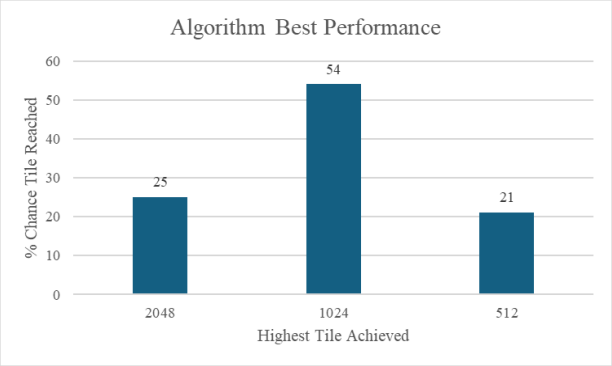


Figure 29: Input feature weights of best-performing algorithm

Optimal Play Algorithm Performance

The best performance achieved by the created solving agent can be seen in Figure 29, reaching the 2048 tile in 25% of games, reaching the 1024 tile in 79% of games, and having the highest tile be 512 in 21% of games. The weight coefficients applied to each board feature to accomplish this result can be seen in Figure 30, with the “Largest”, “Monotonicity”, and “Empty” parameters having the highest correlation with reaching higher tile values and the “Chain” variable having nearly no correlation.

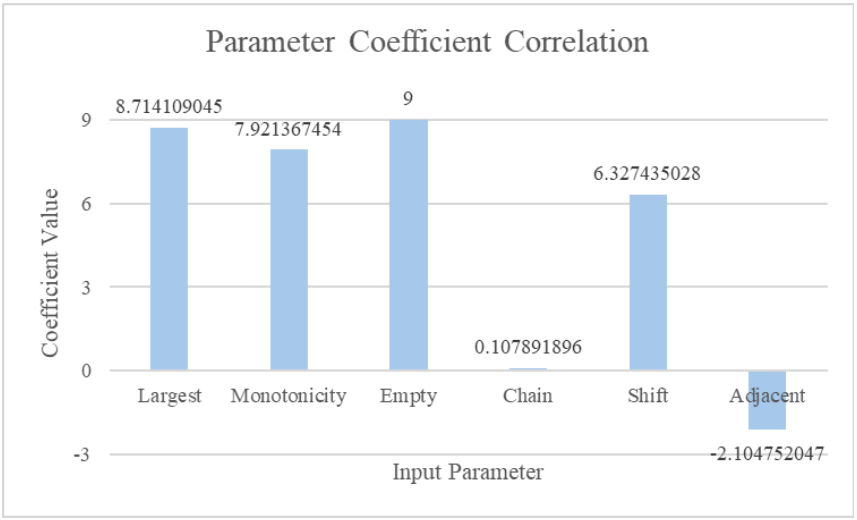


Figure 30: Average highest tile reached with best-performing algorithm

The "Largest” parameter has the most variability among genetic runs and it contributes the most to the overall evaluation score. As previously mentioned, the position of the largest tile on the board determines the number of complete chain paths that can occur from that position.

Positions that contain more favorable complete chain paths (i.e. corner), produce a score considerably higher than other positions (i.e. border or neither). The maximum value the "Largest" parameter can take is the largest value any parameter can have. This score tends to determine how important other parameters are for the evaluation of a board, especially since this parameter has a strong positive correlation with reaching larger tiles in comparison to other parameters. Having a higher weight for the largest point value means that other parameters will need to have larger weights in order for their values to be factored into the score of a board. The large point discrepancy present between board states that have the largest value in the corner and board states that do not will tend to maintain the largest tile value position in a corner once obtained.

The "Monotonic" score of a board has shown to be a useful metric when evaluating 2048 boards, as can be seen in its high correlation with achieving larger-valued tiles within the game. Monotonically-tiled boards provide a structure conducive for merging and maintaining the order of large tiles, due to the resemblance of a monotonic board to the ideal snake chain. The "Monotonic" score of a board is not dependent upon the values of the tiles on the board, making its individual contribution to the evaluation of a board consistent regardless of the magnitude of the tile values contained within it. The high correlation monotonicity had with reaching higher-valued tiles, in comparison to the neutral correlation which the "Chain" variable possessed, could mean that a monotonic board structure is more conducive to reaching higher tiles than a chain structure.

The best runs for our algorithm had a high, positive correlation of the "Empty" variable with reaching higher-valued states. This makes intuitive sense, as when the number of empty tiles reaches zero and there are no adjacent tiles of the same value anywhere on the board, the game ends. In addition, previous solvers of the game utilized the number of empty cells within a given board state to improve the performance of their programs (Xiao, 2021; Kohler et al., 2019). Thus, the number of empty tiles on a given board state plays an important role in determining how valuable that state is.

The optimal values for the "Chain" metric were the most surprising to the team, as the created program was predicated upon the idea of creating and maintaining the most optimal chain throughout the game. The best coefficients for the "Chain" score relayed very little correlation to the increased value of a given state, despite the gameplay of the algorithm resembling the construction and compaction of a chain. This is most likely due to the fact that a monotonic board containing the largest tile in the corner closely resembles a chain, giving the playing program the appearance of deliberately creating a chain, when many of the characteristics which make a snake chain desirable are accounted for within other parameters. From this, it may be true that the construction of a chain, while helpful for human players, is not itself a relevant property of game states which reach higher tile values, but is a recognizable

structure that contains properties which are conducive to reaching board states with higher valued-tiles.

The “Shift” parameter represents the number of merges present on a given board state and tends to increase as our board becomes filled. The “Shift” parameter is not as positively correlated as other more strongly correlated parameters but preferring moves that have more merges is conducive to reaching higher-valued tiles. Having more potential merges is the result of having less empty tiles on the board. This means that preferring move with more merges will get us closer to a losing state and may explain the lower correlation of shiftload when compared to other parameters. It may be the case that our parameter would have had a higher correlation if shiftload had considered merges that were possible in a direction parallel to the direction of a move as well.

The “Adjacent” parameter provides a metric of the remaining tiles on the board that are not part of the chain. We want higher tiles near our chain to improve the likelihood that we can compact our chain before it reaches its complete length (root $\rightarrow 2^1$). The goal of “Adjacent” was to always have larger tiles near our chain, such that smaller tiles would not get trapped by other tiles, preventing merges near our chain. By taking into account all tiles that were not part of the chain, the score for adjacent was significantly larger than the possible values other parameters could take. The negative correlation may have been explained by the large point discrepancy between parameters or a miscalculation in our method of computing the adjacent scores for board states. This was evident as increasing the weight associated with the adjacent score tended to populate larger tiles along the diagonals of the board, which was not conducive to prevent the blocking of tiles around a chain.

Heuristics for Human Players

Our experimentation of optimizing these parameters, and their previous iterations, has allowed our team to translate some of the desirable characteristics of boards that often reach higher tiles, and relay usable heuristics which human players will be able to use when playing the game. Attempting to create the ideal chain structure with the largest-valued tile in the corner of the board, making moves that move one’s chain away from the edge of the board only when necessary, is the easiest and most maintainable way of being able to consistently reach the 2048 tile as it ensures preservation of the chain structure. One method of mitigating the risk of a forced move disrupting the chain structure is to keep the rows/columns of the board which a given chain occupies, filled with tiles. This will prevent the chain from being shifted away from the corner of the board, and allow for easier maintenance of one’s chain.

In the process of creating a snake chain, there may be times when the player is forced to make a move which shifts the root of their chain away from the corner and the newly-inserted tile occupies the position previously held by the root. If this occurs, the player can only attain the

previous chain path if they are able to merge tiles into the inserted tile currently in the corner of the board until it is large enough to merge with the root tile of the current chain, or by manipulating the board through a sequence of moves which restores the chain structure. Given the player's chain is dislocated by an inserted tile, the player should disregard that inserted tile and attempt to create a compactible chain to the largest tile. This is due to the fact that if the player is able to compact their chain to a single row, the compaction of that chain can switch direction and subsequently compact into a new larger tile in the corner on the opposite side of that row.

Image Creation Results

One of the main outcomes of this project was the ability to programmatically create desired images within a 2048 board through the application of colored tiles to associated values on a board and the creation of that board. This encompasses the enumeration and construction of rows and columns of tiles and the application of color textures to tile values to attempt the recreation of a desired square board of colors while only being able to insert tiles of value 2 and 4. This was accomplished successfully and integrated into our application, and is relatively functional on boards with a size less than nine. This image creation method was used to be able to simulate John Conway's Game of Life within 2048 by creating sequential boards of tiles representing alive and dead cells within Conway's Life, with the option of choosing a bounded or toroidal gamespace.

Future Work

One of the areas which could be expanded upon in regards to this project is the scalability of the image creation method. The approach taken by the team creates a map of all possible rows/columns of tiles makeable on the given board if only one row/column was available. As the size of the board increases, the number of row/column mappings for that board increases rapidly. A more adaptable method of image creation not dependent upon the enumeration of large numbers of rows/columns may serve to improve the running time of the algorithm, and be able to generate a greater variety of boards.

Future implementations of stereographic tiles within 2048 may be improved through the creation of a more robust set of stereogram textures to apply to tiles. One way this could be done is to create a larger set of stereographic tile textures, and create a mapping of tile values to textures such that each tile has a unique set of stereographic textures associated with it, and have the texture of a given tile displayed to the user be decided based on the value of its horizontally-surrounding tiles, such that the designations of all tiles along each row always alternate between right and left to create the intended images. In this way, a single tile value will have multiple stereographic textures associated with them, but each texture will only be associated with a single tile value.

Another aspect of this project that could be expanded upon is the continuation of improving the capability of automove to reach higher tiles. Our algorithm was only able to get to 2048 around 30% of the time. Many other solvers are capable of reaching tile values of up to 32,768 (Naoki Kondo, Kiminori Matsuzaki, 2019; Jaśkowski, Wojciech, 2016). A large portion of this discrepancy is likely to be the use of our selected heuristics and our focus on the ideal snake chain structure. The premise of our algorithm was to reach the highest possible tile values, and thus we selected the board structure that, in theory, could reach the highest possible tiles on a 4 x 4 2048 board. Although other solvers mention the use of a chain structure, the solvers that tend to perform better tend to have less emphasis on chaining and more emphasis on the monotonicity of the board, the number of empty tiles, and the number of potential merges (Cox, 2024). It is likely our initial assumption about reaching higher tiles was formulated from a human players perspective rather than from an algorithmic one.

Using a chain structure can be very powerful and lead to high tiles being reached, but the plethora of ways a chain can be disrupted or otherwise lack assurance may detriment the progress of our solver more than it improves it. Human players will likely use methods such as chaining to determine which moves to make, but reaching tiles higher than 2048 becomes increasingly more difficult and it may be such that using chain structures is not conducive to reaching tiles higher than 2048 consistently. Those looking to pursue algorithmic solvers should compare the properties of chaining and monotonicity more stringently.

It would also benefit our implementation to look at more board states in the future that arise as a result of moves from a given board state. It is likely that any solver will benefit from more search space to evaluate, but ours is limited to just 2 moves into the future creating the possibility that we will not observe the best actions to take at any given state. Other individuals looking to improve the performance of this solving algorithm will likely need to formulate a better metric to prune branches of the search space.

Finally, those looking to benefit from our initial research will likely endeavor to use a form of neural network to replace the process of our genetic algorithm. Most well-performing solvers use neural networks to find the best weights to use for the parameters chosen. Using a neural network would likely determine better weights that are more suitable to reaching higher tiles.

References

Berlekamp E.R., Conway J., & Guy R. (1982). *Winning Ways for Your Mathematical Plays, vol 2.* Academic Press.

Boyd, K. (2018, March 23). *Depth Perception*. American Academy of Ophthalmology. https://www.aao.org/eye-health/anatomy/depth-perception

Britannica, T. Editors of Encyclopaedia (2024, February 2). *Sir Charles Wheatstone*. Encyclopedia Britannica. https://www.britannica.com/biography/Charles-Wheatstone

Cirulli, G. (2014). 2048. GitHub. https://github.com/gabrielecirulli/2048

Conditt, J. (2015, June 13). The Original Number-Pushing Puzzle Game, 'Threes,' Goes Free. Engadget, https://www.engadget.com/2015-06-12-threes-free.html

Cox, G. (2024, March 18). *What is the optimal algorithm for the game 2048?*. Baeldung on Computer Science. https://www.baeldung.com/cs/2048-algorithm

Desktop Screen Resolution State Worldwide. StatCounter Global Stats. (n.d.). https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide

Gardner, M. (1959). *Mathematical Puzzles and Diversions* (2nd ed.). New York, NY: Simon and Schuster.

Gardner, M. (1983). *Wheels, life, and other mathematical amusements*. W.H. Freeman.

Gardner, M. (1986). *Knotted doughnuts and other mathematical entertainments*. W.H. Freeman.

GfG. (2014, March 8). *Genetic algorithms.* GeeksforGeeks. https://www.geeksforgeeks.org/genetic-algorithms/

Hoskins, J. (2022). 2048.directory. GitHub. https://github.com/jimrhoskins/2048.directory

Jaśkowski, W. (2016). Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping. https://doi.org/10.48550/ARXIV.1604.05085

Kelley, S. (2021, August 11). *Depth Perception: How Are We Able To See In 3D?* All About Vision. https://www.allaboutvision.com/eye-care/eye-anatomy/depth-perception/

Kohler, I., Migler, T., & Khosmood, F. (2019). Composition of basic heuristics for the game 2048. *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 1–5. https://doi.org/10.1145/3337722.3341838

Kondo, N., & Matsuzaki, K. (2019). Playing Game 2048 with Deep Convolutional Neural Networks Trained by Supervised Learning, Journal of Information Processing. Volume 27, Pages 340-347, Released on J-STAGE April 15, 2019, Online ISSN 1882-6652, https://doi.org/10.2197/ipsjjip.27.340, https://www.jstage.jst.go.jp/article/ipsjjip/27/0/27_340/_article/-char/en

Lees-Miller, J. (2017, December 10). *The mathematics of 2048: Counting states by exhaustive enumeration*. jdlm.info. https://jdlm.info/articles/2017/12/10/counting-states-enumeration-2048.html

Loizeau, N. (2018). gol-computer. GitHub. https://github.com/nicolasloizeau/gol-computer

"Martin Gardner's Mathematical Legacy." *Martin Gardner*, 7 Mar. 2021,

https://martin-gardner.org/MathLegacy.html

Miller, J. (1999). Index to Mathematical Games. https://martin-gardner.org/MGSAindex.html

Mulcahy, C. (2014, October 21). *The Top 10 Martin Gardner Scientific American Articles*. Scientific American Blog Network. https://blogs.scientificamerican.com/guest-blog/the-top-10-martin-gardner-scientific-american-articles/

Numberphile. (2014, March 6). *Inventing Game of Life (John Conway) - Numberphile* [Video]. YouTube. https://www.youtube.com/watch?v=R9Plq-D1gEk

Ovolve. (2020). 2048-AI. GitHub. https://github.com/ovolve/2048-AI

Perimeter Defense Formation. (2014). 2048 Masters. https://2048masters.com/lessons/pdf/training-1/index.html

Richardson, AJ. (2015). Evil-2048. GitHub. https://aj-r.github.io/Evil-2048/

Rendell, P. (2002). *Turing Universality of the Game of Life. In: Adamatzky, A. (eds) Collision-Based Computing*. Springer, London. https://doi.org/10.1007/978-1-4471-0129-1_18

Rendell, P. (2011, December 7). *A Turing Machine in Conway's Game of Life, Extendable to a Universal Turing Machine*. http://www.rendell-attic.org/gol/tm.htm

Rodgers, P., & Levine, J. (2014) An investigation into 2048 AI strategies. *2014 IEEE Conference on Computational Intelligence and Games*, 1-2. https://doi.org/10.1109/CIG.2014.6932920

Schreier, J. (2014, March 31). 2048's Massive Popularity Triggers Cloning Controversy. Kotaku. https://kotaku.com/2048s-massive-popularity-triggers-cloning-controversy-1555599216.

Tóth, C., O'Rourke, J., & Goodman, J. E. (Eds.). (2017). *Handbook of discrete and computational geometry* (Third edition). CRC Press. https://www.csun.edu/~ctoth/Handbook/HDCG3.html

Tyler, C. (2014). *Bela Julesz Biographical Memoir*. National Academy of Sciences. https://www.nasonline.org/publications/biographical-memoirs/search-results.html?last_name_for_filter=julesz&include=deceased

Xiao, R. (2021, August 15). "What is the optimal algorithm for the game 2048?". Stack Overflow. https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048/22498940#22498940

Xiao, R. (2021, July 2). 2048-ai. GitHub. https://github.com/nneonneo/2048-ai

2048: Fibonacci. (2014, August 15). CoolMath Games https://www.coolmathgames.com/0-2048-fibonacci

Appendix A

Stereogram Viewing Keys

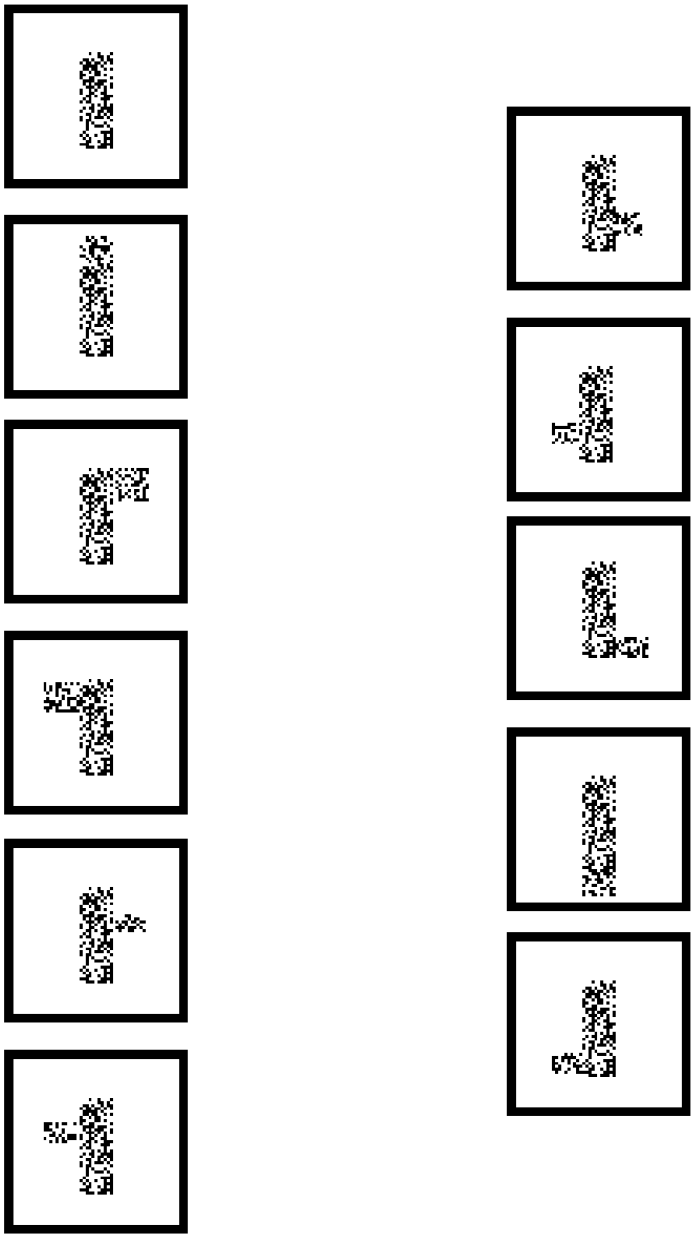


Figure A1 : Polyomino Tile Set Key

This image represents the different sections that can be perceived with depth for each pair of images in the original tile set when viewed with stereoscopic vision.

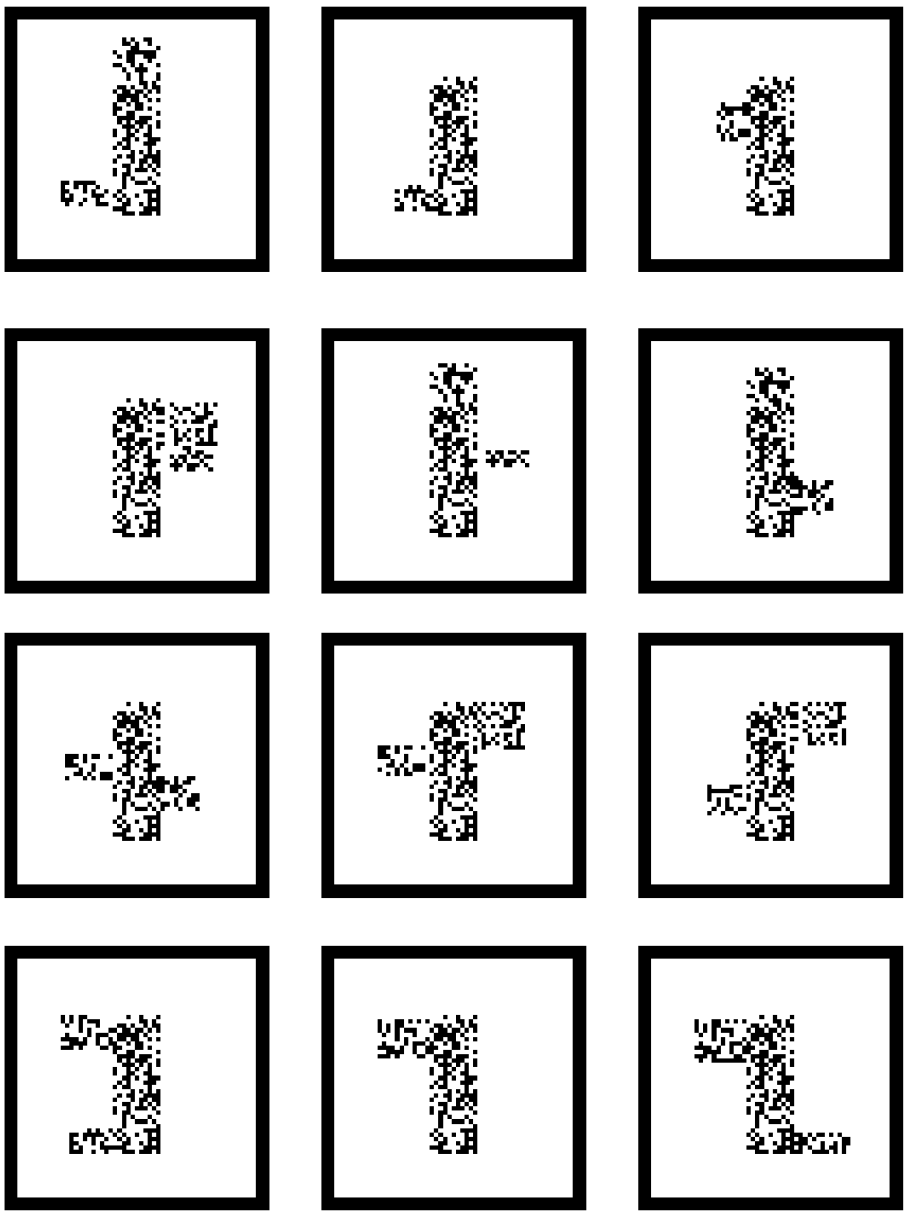


Figure A2: Polyomino Tile Board Key

This image represents the different sections of Figure 28 that can be perceived with depth when viewed with stereoscopic vision.

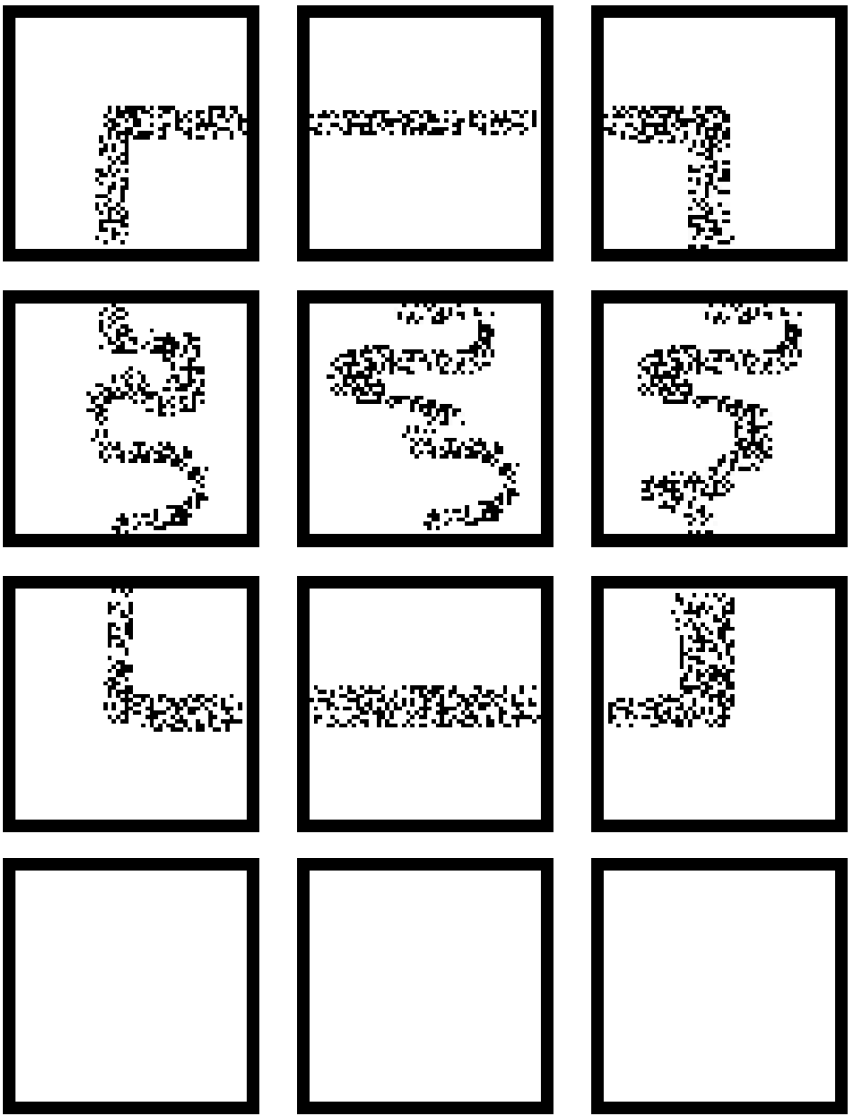


Figure A3: Viewing Key for Stereographic Texture Board with Predefined Tile Positions

This key represents the stereographic view of Figure 24. In this image, depending on whether the viewer converges or diverges their eyes, only one half of the stereographically visible segments will be perceived in front of the image, while the other half will be seen sunken behind it.

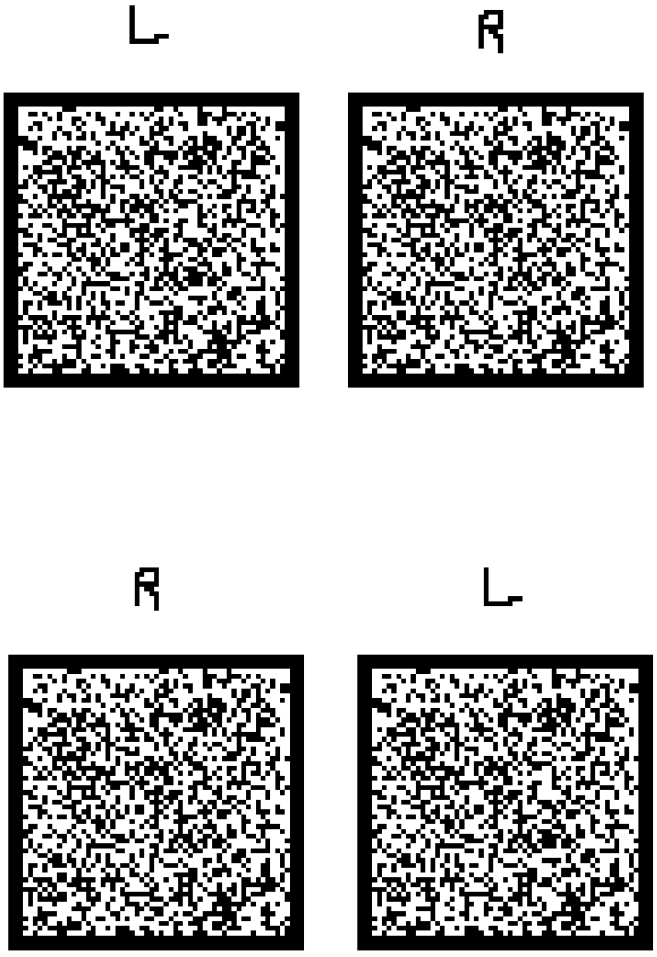


Figure A4: Opposite Designation Polyomino Tiles in Two Distinct Orientations

This image depicts the same two tile textures with different horizontal orientations, such that when viewed using stereographic vision, one set of tiles always appears raised from the image, and one set appears sunken into the image.

Appendix B

Links to External Resources

https://wpi0.sharepoint.com/:b:/s/gr-martingardnermqp/Eco3UzqCW4ZNnAI5YUs_IYoBU8P3Ml7MfI14126KiPtUXQ?e=bsAndN

Figure B1: Potential Project Investigations

This link directs to a document containing information regarding the eleven different puzzles and games investigated by the team in pursuit of a focus for this project. Hosted on SharePoint.

https://github.com/tRaymodn/2048

Figure B2: Github Repository for Project Application

https://github.com/tRaymodn/2048_Visual

Figure B3: Link to Code Repository for Supplemental 2048 Visual Web Page

All Chain Paths

Figure B4: All Chain Paths

Link to Google Sheets document containing all possible chain paths of a 4 x 4 board.