Matrix Shuffler 2: Improving a Web App for Matrix Reordering

Lukas Auer, David Heidinger, Nina Tschikof, and Christina Vogel

Group 2
DAT.C406UF Information Visualisation 3VU SS 2026
Graz University of Technology

29 Jun 2026

Abstract

Matrix visualisations are an effective way to explore tabular data by revealing patterns through row and column reordering. Matrix Shuffler 1 provides an interactive web application for this purpose, but suffers from a tightly coupled architecture that limits maintainability and extensibility.

This project presents Matrix Shuffler 2, a redesigned version of the application with a modular architecture, which separates rendering from user interaction. The original PixiJS renderer was replaced by a Canvas2D implementation, creating a cleaner foundation for future rendering technologies. In addition, several usability improvements were introduced, including Bertin-inspired visual encodings, an interactive minimap, zooming and panning, an improved user interface, and simplified build and deployment options. Overall, Matrix Shuffler 2 preserves the functionality of the original application while providing a more maintainable architecture and an improved user experience.

Contents

Chapter 1

Introduction

Matrix visualizations have long been used to represent relationships between entities in a structured and compact way. While early matrix representations appeared in statistical and scientific applications during the late nineteenth century, their analytical potential was limited by the fixed ordering of rows and columns. During the 1950s, researchers started to recognise that reordering rows and columns could reveal patterns that remained hidden in the original arrangement [Perin et al. 2014, page 2083]. Since manually reordering is a difficult and time-consuming process, supportive technologies started to emerge.

In 1969, Jacques Bertin described his matrix reordering method [Bertin 1969], which was later expanded upon in a book [Bertin 1977], and then translated into English [Bertin 1981]. He built a series of physical reorderabe matrices, which he called Dominos [Dragicevic 2012]. Final arrangements of a matrix were photographed for archival.

However, rearranging large matrices quickly became impractical and fueled the development of algorithmic approaches and interactive visualization tools. Modern matrix visualisation systems therefore combine automatic ordering algorithms with user interaction to support exploratory data analysis across a wide range of domains [Perin et al. 2014].

Bertifier [Perin et al. 2025] is a web application inspired by Jacques Bertin's work. Bertifier provides an interactive environment for exploring tabular data through matrix reordering and visual encoding techniques. Users are able to import a dataset into a tabular visualisation. The tool represents values not only using colours, but also supports multiple graphical encodings like circles, bars, gradient, and Bertin encodings. These encodings allow users to emphasise different characteristics of the underlying data while preserving the structure of the matrix. The system further integrates automatic reordering algorithms with manual interaction, enabling users to iteratively refine the visualisation according to their analytical goals. Bertifier demonstrated that interactive matrix visualisations provide an effective alternative to traditional tabular representations particularly for discovering clusters, correlations, and structural patterns in multidimensional datasets.

Chapter 2

Matrix Shuffler 1

The work in this project builds upon Matrix Shuffler 1, an interactive web application for matrix visualisation and reordering created by Esma Karic, Andrej Knaus, and Laura Thaçi [Knaus et al. 2025a; Knaus et al. 2025b]. Matrix Shuffler 1 is built using the following web and desktop technologies:

- *Frontend*: The user interface is developed with Vue.js and TypeScript.
- *Styling*: The application uses plain CSS for styling.
- *Native Desktop*: By using Tauri 2, the web application can run as a standalone desktop application.
- *Build Tools and Package Manager*: Vite is used as the build tool, and PNPM is used for package management.
- *Libraries*: The main libraries used are PixiJS for rendering the matrix visualisation, Pinia for state management, and Handsontable for the table view.

Matrix Shuffler 1 provides tools for exploring, transforming, and analysing tabular data through visual matrix representations. The main features can be divided into data management, matrix manipulation, and exporting. Users can import custom datasets in CSV or TSV formats or load one of the provided example datasets. Once loaded, the data is accessible through a table view, allowing users to inspect the raw data values alongside the visualisation. The application includes data transformation features to support pattern discovery. Users can transpose the dataset to swap rows and columns, and apply different normalisation techniques. The supported normalisation methods are row-wise, column-wise, and global scaling using a Min-Max approach, which helps to show relative differences in the data.

Users can manually drag and drop rows and columns, or randomly shuffle them. For automated reordering, the system implements several sorting methods:

- *Statistical and Similarity Sorting*: Rows or columns can be sorted based on statistical metrics such as sum, mean, median, maximum, minimum, and variance, or alphabetically. Furthermore, similarity sorting orders the matrix based on the correlation coefficient relative to a selected target row or column.
- *Algorithmic Reordering*: The system also provides algorithmic methods for reordering. The 2D Sort method iteratively sorts rows and columns based on their calculated average or sum weights until the order converges. The Greedy Seriation algorithm iteratively reorders rows and columns based on distance metrics to group similar patterns together.

Finally, Matrix Shuffler 1 allows users to export the modified dataset as a new CSV file. The matrix visualisation can also be exported as a PNG image or an SVG graphic, which preserves the current visual encodings and ordering.

Chapter 3

Matrix Shuffler 2

Matrix Shuffler 2 was developed as a redesign of Matrix Shuffler 1, with the goal of creating a more maintainable system. While the first version already provided interactive matrix visualisation and encodings, the implementation tightly coupled rendering, interaction, and application logic. This made it difficult to extend the application with additional rendering technologies or reuse existing functionality. The main objective of Matrix Shuffler 2 was therefore to separate rendering from interaction and establish a more renderer-independent architecture. Instead of embedding mouse interaction directly inside the renderer, interaction should be handled independently and shared between different renderers. This allows future implementations based on SVG (D3) or WebGPU to reuse the same interaction logic, while only replacing the rendering component. Another goal was to replace the original PixiJS renderer with a Canvas2D implementation, providing a simpler rendering pipeline while preserving the interactive functionality of Matrix Shuffler 1. Besides the architectural improvements, the application was extended with additional interaction techniques, improved visual feedback, and support for Bertin-style encodings. The software is open source and is available on GitHub [Auer et al. 2026b]. A live demo is deployed to GitHub Pages [Auer et al. 2026a].

3.1 Revised Software Architecture

The redesigned architecture follows the principle of separating application state, rendering, and user interaction into independent components. Instead of relying on a single component that performs all tasks, Matrix Shuffler 2 distributes responsibilities across dedicated stores, renderer components, and interaction utilities. The application state is managed using Pinia stores. The dataset store contains the current matrix together with the row and column ordering. The visualisation store manages rendering-related settings such as colour schemes, cell size, label rotation, and normalisation. User interaction is handled separately by the interaction store which maintains hover states, drag-and-drop information, and the current mouse position. A major change compared to Matrix Shuffler 1 is the introduction of a dedicated interaction layer. The `MatrixInteractionOverlay` component captures mouse events independently of the renderer. Hover detection, row and column selection, drag-and-drop operations, and tooltip positioning are therefore implemented only once and can be reused by different rendering technologies. The renderer itself is only responsible for drawing the current state of the matrix. The resulting architecture is easier to maintain and extend, and provides a better foundation for future rendering technologies.

3.2 Canvas2D Renderer

One of the major implementation tasks was replacing the original PixiJS renderer with a Canvas2D renderer. The renderer dynamically redraws the complete matrix whenever the dataset or visualisation settings change. Matrix dimensions are automatically adjusted according to the current cell size, label rotation and matrix layout. This ensures that different datasets remain readable without requiring manual

adjustments. Besides drawing the matrix itself, the renderer implements hover highlighting, drag-and-drop indicators, configurable colour schemes and different normalisation strategies. The current visualisation can be exported as CSV, PNG or SVG using a shared export interface. Although the current implementation focuses on Canvas2D, the renderer was developed in a way that allows future rendering technologies to implement the same functionality while sharing the surrounding application logic.

3.3 Improved Interface Layout

To refine the user experience, several adjustments were made to the layout and visual presentation of the application. A key change involved reorganising the top panel, which now houses the controls for visual encodings alongside the colour schemes. By moving the colour scheme selection out of the side settings and into the top bar, users can access and change the primary visual aesthetics more directly. To provide better feedback within these menus, both the colour scheme and encoding dropdowns now display a checkmark next to the currently selected option. Additionally, the options in the encoding dropdown feature descriptive tooltips to help users understand what each visualisation method does.

The collapsible side panels were also updated. While the left side panel remains unchanged in its functionality, the right-hand settings panel and its opening flap received visual modifications to improve the overall look and feel. Furthermore, the contents of the Settings Panel were streamlined: some options were removed or relocated to reduce clutter. To ensure consistency in language throughout the interface, the terminology inside the Settings Panel was updated to use British English spelling.

3.4 Bertin Encodings

In addition to the standard visual representations, Matrix Shuffler 2 introduces two new cell encodings directly inspired by Jacques Bertin's visual variables and the Bertifier web application:

- Bar Chart: This encoding represents the numerical value of a cell through the length of a small horizontal bar contained within the cell boundaries. This mapping to size allows users to quickly compare the magnitudes of different cells at a glance.

- Hatched Bar Chart: This encoding functions similarly to the Bar Chart encoding, but introduces a hatched texture to the bars. It is referred to as the Dual Bar Chart encoding in the original Bertifier web app, since it leverages Bertin's texture variable. This secondary visual encoding makes it highly effective for distinguishing data points in scenarios with limited colour palettes, such as black-and-white printing, or for users with colour vision deficiencies. An example of this encoding can be seen in Figure 3.1.

3.5 Minimap

To improve navigation in large matrices, a Minimap was added to our implementation. The Minimap is displayed as a fixed-size canvas that can be toggled on and off using a dedicated button in the toolbar. It provides a miniature overview of the entire matrix, allowing users to quickly understand which part of the dataset is currently visible.

The current viewport is represented by a red rectangle. By clicking and dragging this rectangle, users can directly pan the main visualisation, without using the middle mouse button. This enables faster navigation, particularly for large matrices.

To avoid covering important parts of the visualisation, the minimap itself can be repositioned by dragging its header to any location on the screen. In addition, separate zoom controls allow users to adjust the magnification of the minimap. Zooming in provides finer control when dragging the viewport, while zooming out offers a better overview of the relationship between the visible area and the complete matrix. The minimap can be seen in Figure 3.1.

Figure 3.1: Matrix Shuffler 2: The reorderable matrix showing the classic Cerials dataset with Hatched Bar Chart encoding. The Minimap can be seen to the far right.

3.6 Zooming and Panning

As datasets grow in size, rendering the entire matrix within a single view without losing legibility becomes impossible. To address this, Matrix Shuffler 2 implements zooming and panning mechanics, allowing users to explore large datasets seamlessly. Users can intuitively zoom in and out of the matrix visualisation using the mouse scroll wheel. This adjusts the magnification level in real time, ensuring that individual cells and their encoded data remain legible even when dealing with highly dense matrices. For navigation across the dataset, the application provides a straightforward panning functionality. Users can freely pan the viewable area by clicking and dragging across the matrix with the middle mouse button. These mechanics, working in tandem with the newly introduced Minimap, provide a highly responsive and flexible approach to data exploration, ensuring that users can easily navigate between high-level overviews and detailed cell-level inspections.

3.7 Build Options

While the fundamental technologies for building Matrix Shuffler 2 remain similar to the first version, the deployment and distribution process has been significantly improved. The most notable change to the build process is the introduction of Gulp as a task runner. Instead of relying solely on raw Tauri CLI commands, developers can now use the `build:desktop` command (via Gulp) to streamline the generation of the desktop application.

Furthermore, the distribution strategy for end-users has been expanded. A live demo is also deployed using GitHub Pages [Auer et al. 2026a], where users can try all the functionality without having to download or install anything.

In Matrix Shuffler 1, the build process by default skipped the creation of platform-specific installers (such as MSI, DMG, or AppImage) with Tauri for faster build times. In Matrix Shuffler 2, fully packaged setup files for all major operating systems (Windows, macOS, and Linux) are now generated and provided directly as downloads on the GitHub repository [Auer et al. 2026b]. Additionally, a standalone portable `.exe` file is offered for Windows, allowing users to run the application immediately without requiring any installation or setup process.

Chapter 4

Discussion

When comparing Matrix Shuffler 2 to the original version, the biggest difference is how the code is structured underneath. In Matrix Shuffler 1, the rendering (using PixiJS) and the mouse interactions were heavily mixed together. This made the application work well for its time, but it was very difficult to change or update. Matrix Shuffler 2 fixes this by completely separating state, interaction, and rendering. It replaces the complex PixiJS setup with a simpler Canvas2D renderer and moves all the drag-and-drop logic into a separate, reusable overlay.

From a user's perspective, Matrix Shuffler 2 also feels much more complete. The first version had some good basic features, but navigating large datasets was difficult. Matrix Shuffler 2 solves this by adding a Minimap and much smoother zooming and panning controls with the mouse wheel and middle mouse button. The interface has also been cleaned up, moving important controls like colour schemes to the top bar and adding helpful tooltips. Furthermore, while Matrix Shuffler 1 mainly relied on colours and circles to show data, the new version introduces Bertin-inspired Bar Chart and Hatched Bar Chart encodings. Finally, Matrix Shuffler 1 required users to build the app themselves to create platform-specific installers, whereas Matrix Shuffler 2 provides ready-to-download setup files and a portable executable.

The changes made in Matrix Shuffler 2 bring several major advantages to both developers and end-users:

- *Better Maintainability*: Since the interaction logic is now completely separated from the drawing logic, the code is much easier to read, debug, and expand. Future developers can easily swap out the Canvas2D renderer for something else (like SVG or WebGPU), without having to rewrite any of the mouse interaction code.
- *Improved Usability*: The cleaner UI, combined with tooltips and checkmarks, makes the application less confusing for new users. The addition of the Minimap and better panning controls also mean that working with large matrices is no longer a frustrating experience.
- *More Accessible Data*: By adding the Bar Chart and Hatched Bar Chart encodings, the application no longer relies on colour alone to show data differences. This improves accessibility for users with colour vision deficiencies and for black-and-white printing.
- *Easier Distribution*: Offering pre-built installers and a portable `.exe` file directly on GitHub is a massive time-saver. Users who just want to analyse their data can now download and run the tool immediately, without needing to understand Node.js, npm, or how to set up a development environment.

Even though Matrix Shuffler 2 successfully achieved its main goals, there are still several features that could be added in the future:

- *Alternative Renderers (SVG and WebGPU)*: Since user interaction is now separated from rendering, it is now much easier to plug in new rendering methods. For example, an SVG renderer (using D3.js) could be added to generate scalable vector graphics natively. A WebGPU renderer could also

be explored to improve performance when working with extremely large matrices.

- *More Algorithmic Solutions*: The application currently supports Greedy Seriation and 2D Sorting, but the new code structure allows for more algorithms to be added easily. Future updates could include techniques like Spectral Clustering (which groups data using a similarity matrix) or BiMax (a biclustering method that finds highly correlated sub-blocks in the data) to help users find hidden patterns automatically.
- *Chain Function*: A helpful addition for manual matrix reordering would be a “chain” or “glue” function. This would allow users to chain specific rows or columns together, so they always move as a single block when dragging them or running sorting algorithms.

Chapter 5

Concluding Remarks

The primary objective of this project was to resolve the structural limitations of Matrix Shuffler 1 by designing a more maintainable and modular system. With the development of Matrix Shuffler 2, this goal was successfully achieved. By decoupling the user interactions from the rendering pipeline, a flexible architecture was established, that not only supports the current Canvas2D implementation, but also lays a strong foundation for future rendering technologies. Furthermore, the redesign provided an opportunity to significantly improve the user experience through enhanced navigation tools, a cleaner interface, and more accessible visual encodings.

Ultimately, Matrix Shuffler 2 is a far more robust and user-friendly application than its predecessor. For anyone needing an interactive, scalable tool to visualise and explore complex matrices, Matrix Shuffler 2 provides a more maintainable architecture and an improved user experience compared to its predecessor. The software is open source and is available on GitHub [Auer et al. 2026b]. A live demo is deployed to GitHub Pages [Auer et al. 2026a].

Bibliography

Auer, Lukas, David Heidinger, Nina Tschikof and Christina Vogel [2026a]. *Matrix Shuffler 2 Demo*. 28 Jun 2026. https://davidheidinger0-hue.github.io/matrix-shuffler-2 (cited on pages 5, 7, 11).

Auer, Lukas, David Heidinger, Nina Tschikof and Christina Vogel [2026b]. *Matrix Shuffler 2 Repository*. 28 Jun 2026. https://github.com/davidheidinger0-hue/matrix-shuffler-2 (cited on pages 5, 7, 11).

Bertin, Jacques [1969]. *Graphique et mathématique: généralisation du traitement graphique de l'information*. French. Annales. Economies, sociétés, civilisations. 24.1 (1969), pages 70–101. doi:10.3406/ahess.1969.422034. https://www.persee.fr/doc/ahess_0395-2649_1969_num_24_1_422034 (cited on page 1).

Bertin, Jacques [1977]. *La graphique et le traitement graphique de l'information*. French. Flammarion, 01 Sep 1977. 288 pages. ISBN 2082111121 (cited on page 1).

Bertin, Jacques [1981]. *Graphics and Graphic Information-Processing*. De Gruyter, 01 Dec 1981. 280 pages. ISBN 3110088681 (cited on page 1).

Dragicevic, Pierre [2012]. *1968 – Jacques Bertin's Reorderable Matrices*. 2012. https://dataphys.org/list/bertins-reorderable-matrices/ (cited on page 1).

Knaus, Andrej, Laura Thaci and Esma Karic [2025a]. *Matrix Shuffler*. 30 Jun 2025. https://courses.isds.tugraz.at/ivis/projects/ss2025/ivis-ss2025-g2-project-matrix-shuffler.pdf (cited on page 3).

Knaus, Andrej, Laura Thaci and Esma Karic [2025b]. *Matrix Shuffler Repository*. 28 Jul 2025. https://github.com/AndrejKnaus/matrix-shuffler (cited on page 3).

Perin, Charles, Pierre Dragicevic and Jean-Daniel Fekete [2014]. *Revisiting Bertin Matrices: New Interactions for Crafting Tabular Visualizations*. IEEE Transactions on Visualization and Computer Graphics 20.12 (31 Dec 2014), pages 2082–2091. doi:10.1109/TVCG.2014.2346279. https://aviz.fr/wiki/uploads/Bertifier/bertifier-authorversion.pdf (cited on page 1).

Perin, Charles, Pierre Dragicevic and Jean-Daniel Fekete [2025]. *Revisiting Bertin's Matrices*. 28 Sep 2025. https://aviz.fr/bertifier (cited on page 1).