2048 Terminal Game

Product Requirements Document

by Samvel Aghakhanyan & Lusine Sahakyan

May 20th, 2025

This work was done for AUA BSES ENGS110: Introduction to Programming course

Product Information

The 2048 terminal game is a terminal-based version of the famous 2048 puzzle. 2048 is a single-player sliding block puzzle game where the objective is to combine numbered tiles on a 4x4 grid to create a tile with the value 2048. The game starts with two tiles, each having a value of 2. The player uses arrow keys to slide all tiles in one of four directions (up, down, left, or right). When two tiles with the same number collide during a move, they merge into one tile with their sum as the new value. After each move, a new tile (2) appears at a random empty spot. The game continues until the player forms a 2048 tile or no moves are left (game over). The same rules apply in the terminal version we built.

We chose to create this as a final game because it covers the very essential topics of C. The product is for anyone who enjoys playing simple, classic games, especially using terminal.

The key features of the product are the 4x4 grid, keyboard inputs as playing buttons, game-over detection, error-handling for invalid inputs and option to quit the game anytime.

There are limitations such as single player only or no graphical interface.

Objectives/Goals

The primary objective of this project is to implement a playable version of the 2048 puzzle in C, applying fundamental programming concepts such as arrays, conditionals, loops, functions, and user input handling. This task helps deepen our understanding of procedural programming and problem-solving through algorithms and data manipulation.

Another major goal was to provide a smooth and responsive terminal experience, with features such as random tile spawning, win/loss detection, and replayability.

Assumptions & Constraints

This project is developed under several assumptions. We assume that the user will be running the program in a Linux terminal with a standard keyboard layout. We also assume the user is familiar with basic terminal operation, such as compiling and running C programs using gcc.

There are a few constraints that shaped the design and implementation. Firstly, the interface is entirely text-based, meaning no graphics, animations, or color are used. All interactions happen through keyboard input, restricted to the characters W, A, S, D (or lowercase), and Q to quit. The grid size is fixed at 4x4, and only tiles with the value of 2 are randomly generated. Additionally, the game logic had to be implemented without external libraries or frameworks, using only standard C functionality.

Background & Strategic Fit

2048 was originally developed as a simple browser game that became widely popular due to its intuitive mechanics and addictive gameplay. Its logical structure and deterministic rules make it an excellent candidate for implementation in a programming education context.

This game fits well with the strategic goals of our programming course, as it reinforces core topics such as arrays (2D grids), loops, conditionals, input handling, and modular program design. We selected 2048 specifically because it offered the right balance of algorithmic complexity and achievable scope for a beginner project, with clear feedback on success through observable output.

Scope: User Stories & Requirements

The program is designed to fulfill the following user stories:

- As a player, I want to move tiles using the keyboard so I can control the game.
- As a player, I want the grid to update visibly after each move so I can track my progress.
- As a player, I want to win when I reach the 2048 tile.
- As a player, I want to be told when I lose, so I know when the game is over.
- As a player, I want the option to play again after finishing.

To support these stories, we implemented a grid that initializes with two random tiles at the start of each game. The user can input W, A, S, or D to move the tiles in the corresponding direction, and the grid updates according to the rules of 2048. A new tile (value 2) is spawned after every successful move. The game checks for a win condition (tile 2048) and a loss condition (no possible moves). After either outcome, the user can choose to play again or quit.

The program is modular, with functions responsible for grid initialization, movement, merging, user input, and display. Input is validated and handled case-insensitively. There is no support for saving game state, undoing moves, or resizing the grid.

Product Features

The 2048 terminal game includes the following features:

- A 4x4 grid printed using ASCII characters and updated after every move.
- Movement in four directions (up, down, left, right) through the keys W, A, S, and D.
- Merging of adjacent tiles with the same number.
- Automatic generation of a new tile (value 2) after each valid move.
- Win detection when a tile reaches 2048.
- Loss detection when no moves are possible.
- Ability to quit the game at any time by pressing Q.
- Option to restart the game after winning or losing.

These features provide a playable and faithful adaptation of the original game within the limitations of a text-only environment.

Release Criteria

The program is considered ready for release when the following conditions are met:

- The game compiles without warnings or errors using gcc.
- The grid prints clearly and updates after every valid move.
- The user can move tiles with WASD or wasd input.

- A new tile appears on the grid after every successful move.
- The win and loss conditions are correctly detected and communicated to the player.
- The game handles invalid input gracefully and prompts the user again.
- The player is given the option to play again after the game ends.
- Pressing Q exits the game immediately and cleanly.

Success Metrics

We consider the project successful if:

- The game runs reliably without crashes or undefined behavior.
- All game mechanics (merging, movement, win/loss detection) behave as expected.
- The code is readable, modular, and follows good programming practices.
- User input is handled smoothly, including both upper and lower case.
- Players are able to reach 2048 with skillful play, confirming correctness of logic.
- The replay and quit features function correctly at the end of the game loop.

Additionally, success can be measured by how engaging and playable the experience is within the limitations of the terminal.

Exclusions

The scope of this project intentionally excludes several advanced features in order to keep the implementation focused and aligned with course goals. These exclusions include:

- No graphical user interface or use of color in the terminal.
- No mouse input or touch-based controls.
- No tile values of 4; only 2s are generated for simplicity.
- No undo or backtracking functionality.
- No saving or loading game progress.
- No customization of grid size or difficulty level.
- No animations or sound.

These features could be considered in an extended version of the game, but were not included in this implementation due to time and complexity constraints.