Pixel Art Editor in C

GROUP - 7

Nujhat Neha
ID: 2421313642

Sadia Afrin Sharmi
ID: 2423019642

Arifa Akter Mim
ID: 2422852642

Md Mahabubul Alam
ID: 2422388042

Abstract - The pixel art editor simulates pixel manipulation on a fixed grid using basic console I/O, demonstrating core digital image processing principles. This report evaluates the program's algorithmic structure, computational efficiency, scalability potential, and the limitations of using recursion within constrained memory. It highlights optimization opportunities and underscores the border relevance of grid-based methodologies in computational design and graphics.

1. METHODOLOGY

The program was developed using C programming language, making use of standard input/output libraries and basic control structures. We have used GCC (GNU Compiler Collection) compiler for compiling the program. It is widely used and provides robust error handling and debugging tools. IDE: Code::Blocks was used for writing, testing and debugging the code. Operating system: Development and testing were conducted on Windows 11. The program follows a modular structure, with each function serving a specific purpose. Key each functions include initializeGrid():for setting up an empty grid, displayGrid(): for displaying the grid state, and drawPixel():for modifying a pixel, fillColor():for filling colors, erasePixel(): for erasing pixels.

The program follows a modular structure, with each function serving a specific purpose. Key functions include initializeGrid() for setting up an empty grid, displayGrid() for displaying the grid state, and drawPixel()/erasePixel() for modifying specific pixels. The fillColor() function implements a recursive flood-fill algorithm to change connected cells of a target color, while handleUserInput() manages user interactions and function calls based on user input. The program is contained in a single source file but can be modularized for complex scenarios. The core data structure is a 2D char grid[ROWS][COLS] array, representing a fixed-size grid for pixel storage and updates, ensuring efficient memory usage and fast access. User interaction is handled through a menu-driven system, enabling operations such as drawing, erasing, and filling.

2. IMPLEMENTATION

The program is centered around a grid that serves as a canvas, with several meticulously designed functions enabling its efficient manipulation, providing users a straightforward yet versatile tool for pixel art editing. The initializeGrid() function is fundamental to the program's operation, initializing all elements of the grid to blank spaces and ensuring a clean, uniform starting point. This function ensures that every cell in the grid is uniformly prepared, eliminating any potential inconsistencies in the initial state. The displayGrid() function plays a critical role in providing visual feedback to the user, allowing the grid to be output to the console with clearly defined borders, making the current state of the grid easily interpretable. This visual representation not only helps users track their changes but also ensures that the interface remains intuitive and user-friendly. Pixel modifications are managed through the draxPixel() function, which allows users to alter specific pixels by providing valid coordinates, ensuring precise control over the grid. This validation step is essential as it prevents unintended changes, maintaining the integrity of the design. The ability to modify individual pixels with accuracy is a cornerstone of pixel art creation, as it allows for intricates detailing and design flexibility. Conversely, the erasePixel() function resets a cell to its blank state, effectively "erasing" it and giving users the ability to undo specific actions or clear sections of the grid without impacting other parts or their artwork. This function is particularly useful for iterative design process, where adjustments and corrections are frequently needed. For large scale changes, the program, incorporates the fillColor() function, which leverages a recursive flood-fill algorithm to update connected cells of a specific color with a new one. The algorithm ensures that only connected cells are affected, preserving the boundaries of distinct areas and maintaining the artistic integrity of the design. To streamline user interaction, the handleUserInput() function provides and interactive menu that facilitates command execution, allowing users to navigate and manipulate the grid with ease. This menu-driven system enhances user experience by organizing operations logically and reducing the likelihood of errors. Collectively, these features exemplify a cohesive and well-designed pixel art editor, balancing precision, usability, and functionality, highlighting effective algorithmic design and modular programming.

3. BUGS & ISSUES

This program has several issues and potential bugs that could cause unintended behavior or make the user

experience less smooth. Here's a descriptive explanation:

First, there are input validation problems. For example, when users enter invalid data (like letters instead of numbers for menu choices or grid coordinates), the program might behave incorrectly or enter an infinite loop because the input buffer isn't always cleared. To fix this, we should validate all inputs and ensure the input buffer is flushed after failed input attempts.

The recursive implementation of [fillColor] presents a major risk of stack overflow if the area to be filled is large or complex. This happens because each recursive call adds a layer to the stack, and for a 10x10 grid, this could grow too large. To resolve this, an iterative flood-fill algorithm using a stack or queue is a safer and more efficient alternative. Infinite loops in the main loop or input handling are also a concern. If invalid input is repeatedly provided for menu choices, the program might continue asking for input indefinitely without an option to gracefully exit or recover. To address this, we could limit the number of retries or add a fallback option to reset to the main menu after failed input attempts. The [fillColor] function has another problem in its behavior: it stops prematurely if the starting cell already matches the new color, which might not align with user expectations. Providing feedback in such cases or modifying the function logic to handle this scenario will improve usability. There's also no way to undo an action, such as mistakenly filling or drawing in the wrong area. Implementing an undo stack to store previous grid states would be a helpful feature for users.

The program's [while (1)] loop in the main function only exits when the user selects option 4, but unexpected input or program bugs might cause it to loop indefinitely. Introducing a more robust exit condition or providing alternative ways to terminate the program can prevent this. Another missing feature is the ability to save and reload the grid. Currently, if users close the program, they lose all their work. Adding functionality to save the grid to a file and load it later would make the program much more useful. Finally, the fillColor function might unintentionally modify large areas of the grid when used on an empty cell, as it propagates through all connected empty cells. Adding checks to prevent this or notifying the user of the potential outcome can help avoid unexpected results.

4. RESULT & TESTING

When the program runs, the user is presented with a 10x10 grid initialized with empty spaces and a menu to choose actions: draw a pixel, erase a pixel, fill color, or exit. In the draw pixel option, the user specifies a row, column, and a character to place on the grid, and the selected position is updated with the character. Invalid inputs, such as out-of-bound coordinates, are handled with an error message. The erase pixel option allows the user to clear a specific cell by entering its row and column, with invalid positions also displaying an error. The fill color feature uses a flood-fill algorithm, enabling the user to replace a connected region of adjacent cells of the same color with a new color. This works efficiently, even for large regions. The exit option gracefully terminates the program. The program was tested with multiple scenarios, including valid and invalid inputs for drawing, erasing, and filling. The grid updates correctly after each operation, and the interface remains user-friendly, displaying changes in real time. The recursive flood-fill function performed as expected, successfully filling connected regions without errors or infinite recursion.

5. ANALYSIS

This program is a simple 10x10 pixel art editor written in C. It creates a grid where users can draw, erase, and fill colors interactively. The grid is initialized with blank spaces (' ') and displayed with separators for visual clarity. Users interact with the program through a menu system that offers options to draw pixels, erase them, fill connected regions with a new color, or exit the program. The program validates menu choices and ensures users only select valid options. However, it has some limitations, such as missing row/column labels for easier navigation, limited error handling for invalid input formats, and potential inefficiencies in the recursive fill function for larger or complex grids. While functional, it could be improved by adding features like saving/loading grids, undo/redo functionality, better input handling, and a more efficient filling algorithm. The program is a great foundation for understanding grid manipulation and user-driven input handling.

6. CONCLUSION

The project demonstrates the successful implementation of core features such as drawing, erasing, and color filling within a grid, showcasing fundamental programming skills like recursion, input validation, and modular design. Despite certain bugs and limitations, it achieves its goal of simulating a pixel art editor.

7. REFERENCES

1. J. Hanly and E. Koffman (2012), Problem Solving and Program Design in C, 7th Ed., Pearson.
2. H. Schildt (2000), C: The Complete Reference, 4th Ed., McGraw-Hill.
3. Y. P. Kanetkar (2008), Let Us C, 8th Ed., Jones & Bartlett.
4. Deitel & Deitel (2012), C: How to Program, 7th Ed., Prentice Hall.
5. ChatGPT (2024), OpenAI – for technical assistance and concise documentation.