**Pixel Art Editor**

Course Code & Section: CSE 115.2

Project Group 6

Author Information

Nafis Ul Islam Nafis, Jamil Kabir Hridoy, Salman Farsy, Md. Shaiful Islam

2513053642, 1931032642, 2513446642, 2511965642

***Abstract*** - This project report presents the design and development of an advanced pixel art editor implemented in C using the SDL2 and SDL2\_ttf libraries. Aimed at both amateur and professional digital artists, the editor offers an intuitive interface alongside essential creative features such as multi-layer editing, customizable brush tools, undo-redo functionality and efficient file operations including save, load and BMP export. By leveraging the performance benefits of C and SDL2, the application ensures high responsiveness and cross-platform compatibility while maintaining a minimal resource footprint. The modular architecture promotes code clarity, scalability and ease of integration into larger graphical systems. This project demonstrates how lightweight, high-performance graphical software can be built from the ground up using low-level programming techniques.

***Keywords—SDL2, Pixel Art, Image Editor, Undo Redo, C Programming, Graphics Tool.***

**I.Introduction**

Pixel art remains a significant art style in digital media, especially in retro-style video games and avatars. The ability to control each pixel with precision is both an artistic and technical requirement. This project aims to design and implement a powerful yet lightweight pixel art editor that caters to the core needs of digital artists. The editor has been built using SDL2 for rendering and SDL2\_ttf for text display, all developed in the C programming language.

**II. Related Work**

Numerous pixel editors exist, such as Aseprite , Piskel and Pixie, which offer extensive features. However, they often rely on heavy frameworks or are commercial. This project aims to provide essential tools with minimal dependencies using C and SDL2, making it lightweight and portable for educational and hobbyist purposes. Open-source projects such as Piskel, though flexible are developed in high-level languages like JavaScript and often require a browser or runtime environment. Our editor, being developed in C, is much closer to system resources and offers better performance on limited hardware. It is also easier to integrate into other low-level applications such as game engines or hardware-focused projects.

**III. System Requirements**

**A. Software Requirements**

* SDL2 Library
* SDL2\_ttf for font rendering
* C Compiler (e.g., GCC)
* Operating System: Windows/Linux

**B. Hardware Requirements**

* Minimum 2 GB RAM
* 1 GHz processor
* Basic GPU for rendering support

**IV. System Design and Implementation**

The pixel editor is built around a modular and layered software architecture to ensure code clarity, ease of expansion and high performance.

**A.System Architecture**

* **Rendering Module**: Handles all drawing operations on the canvas and UI.
* **Event Handling Module**: Processes input from mouse and keyboard.
* **State Management**: Maintains the editor state including current tool, color, layers and undo-redo stacks.
* **UI Logic**: Implements buttons, toolbars and indicators.

**B. Data Flow**

1. User input is captured and translated to actions (e.g., draw, switch layer).
2. Editor state is updated based on the input.
3. Render system reflects changes on the display.

**C. File Structure**

* main.c: Entry point, handles initialization and the main loop.
* editor.h/.c: Editor state and functions.
* ui.c: Button and toolbar rendering.
* fileio.c: Save/load/export functionalities.

**D. Implementation Details**

* Modular coding style for each functionality.
* Header files for clean interface separation.
* All memory allocations are carefully managed to avoid leaks.

**E. Design Choices**

* Event-driven loop over polling for better input responsiveness.
* Fixed-size canvas to simplify rendering logic.
* Minimalist UI to maintain focus on pixel art creation.

**V. Functional Modules**

**A. Canvas and Pixel Grid**

* 32x32 grid with zoomed 20x20 pixel cells.
* Drawing, erasing, and grid toggling supported.

**B. Layer Management**

* Up to 3 independently editable layers.
* Seamless switching without cross-layer interference.

**C. Color Palette and Tools**

* 16 predefined colors via clickable buttons.
* Toggle for eraser and grid visibility.

**D. Undo/Redo Mechanism**

* Stack-based implementation allowing 10 levels of undo/redo.
* Efficient use of static arrays with circular stack logic.

**E. File Operations**

* Save/load using binary file serialization.
* Export to BMP using SDL\_Surface.

**F. User Interface**

* Button-driven toolbars for all tools.
* Real-time feedback through status display and visual toggles.

**VI. Code Architecture**

**A. Core Data Structures**

typedef struct {

Uint8 r, g, b;

} Color;

typedef struct {

Color pixels[HEIGHT][WIDTH];

} Layer;

typedef struct {

Layer layers[MAX\_LAYERS];

int currentLayer;

int currentBrush;

Color currentColor;

Layer undoStack[MAX\_UNDO];

Layer redoStack[MAX\_UNDO];

int undoTop;

int redoTop;

} EditorState;

**B. Event Flow**

The main event loop captures SDL events and updates editor state accordingly. Mouse events are mapped to the canvas grid and interpreted for drawing or button clicks.

**C. Rendering Logic**

Utilizes SDL2 functions like SDL\_RenderDrawRect, SDL\_RenderFillRect, and SDL\_RenderCopy to construct UI elements and the pixel grid dynamically.

**VII. Advanced Implementation Details**

**A. Brush Logic**

Brush size affects how many adjacent pixels are painted simultaneously. Uses simple loops to fill circular areas around cursor.

**B. Export to BMP**

Creates an SDL\_Surface, populates it from the current canvas and uses SDL\_SaveBMP() for export.

**C. Layer Blending**

Layer compositing is done using simple overwrite (non-alpha blended), with plans for alpha support.

**D. Optimization Techniques**

* Render batching to reduce draw calls.
* Memory pooling for managing history stack.

**E. Extensibility Features**

* Easily add tools by registering them to a toolbar array.
* Event dispatcher separates UI and tool logic for clean code maintenance.

**VIII. Challenges and Testing**

**A. Performance Bottlenecks**

Initial rendering caused lags. Resolved using hardware acceleration and render batching.

**B. Undo Stack Overflow**

Implemented static stack array and shift mechanism to drop oldest state.

**C. UI Precision**

Pixel rounding errors caused inaccurate drawing. Fixed via bounding boxes and snapped grid coordinates.

**D. Testing**

* Unit tested file I/O and state functions.
* Manual testing for interaction and canvas behavior.
* Visual testing for UI layout on different resolutions.
* Used static analysis tools to verify memory safety.

**IX. Conclusion**

The developed pixel art editor serves as a functional and educational tool for aspiring digital artists and developers. Its lightweight design, built entirely in C with SDL2, showcases how powerful graphical applications can be developed with minimal dependencies. The editor successfully integrates core features like layer manipulation, undo/redo stacks and image export capabilities.

It provides a strong baseline for academic projects, low-resource computing devices and integration into larger software systems such as game engines. The editor also exemplifies best practices in modular C development, memory management and event-driven programming.

**X. Future Scope**

To enhance the current capabilities, several advanced features can be introduced:

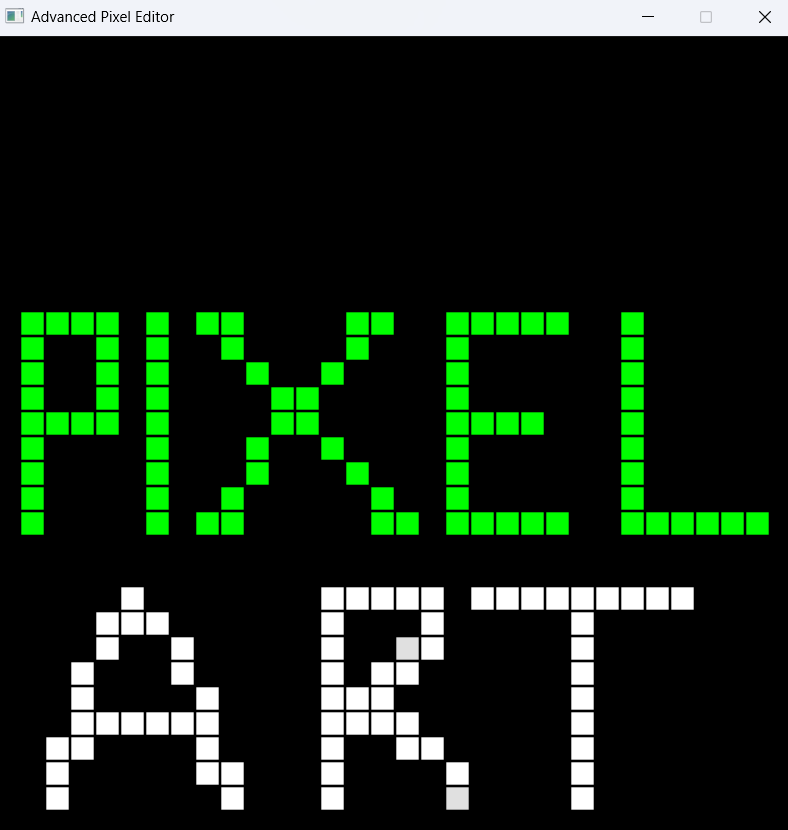
* **Custom Palette Management**: Allow users to define, import and export their own color palettes.
* **Animation Support**: Introduce onion skinning and frame-based animation timeline to support sprite creation.
* **Dynamic Canvas Resizing**: Enable variable canvas dimensions and zoom levels for flexible editing.
* **Alpha Blending and Transparency**: Implement alpha channel support for more nuanced artwork.
* **Brush Customization**: Provide users with a set of brush shapes and allow creation of custom brushes.
* **Tool Plugins**: Develop a plugin system so third-party developers can add tools.
* **Cross-Platform Builds**: Package the application for macOS and Android, increasing accessibility.
* **Collaborative Editing**: Allow multiple users to work on the same canvas using network synchronization.
* **File Format Support**: Add support for PNG and GIF with transparency and animation.
* **Keyboard Shortcuts**: Implement keybindings for faster tool access.

This project lays the foundation for a scalable and extensible graphical application. It can evolve into a fully-featured editor or be embedded as a component within larger game development toolchains.

**XI.** **Screenshots and Sample Output**

The following images illustrate the application’s user interface and sample outputs.

**A.Interface View**

****

* Displays the 32x32 canvas with 3 active layers stacked.
* Toggle buttons highlight active tool selection.

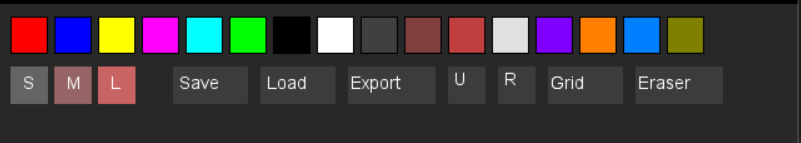
**B. Output Export**

****

**For output export we have three options.**

* Save
* Load
* Export

**C. Drawing Toolbar Interface**

****

**Top Row - Color Palette:**

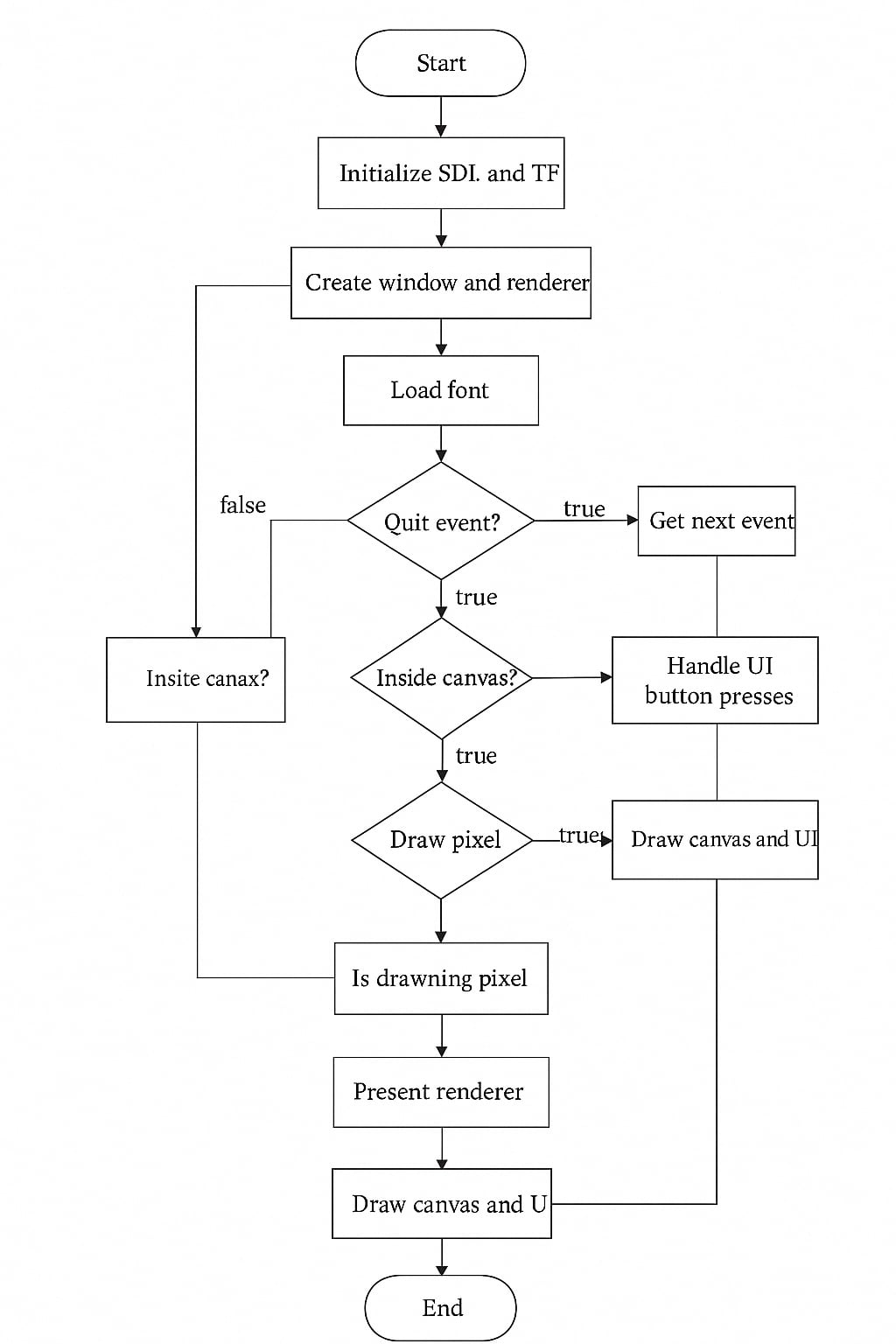
**A set of color squares used to select the drawing color:**

* Primary colors: Red, Blue, Yellow
* Bright colors: Magenta, Cyan, Green
* Shades: White, Black, Gray, Brown, Maroon, Light gray
* Additional vibrant colors: Purple, Orange, Sky blue, Olive green

**Second Row - Tools and Controls:**

* S, M, L: Represent brush or pen sizes (Small, Medium, Large)
* Save / Load / Export: For saving, loading, and exporting the diagram
* U and R: "Undo" and "Redo"
* Grid: Toggles a grid overlay (useful for aligning elements)
* Eraser: Tool for erasing parts of the drawing

**XII. System Flowchart**



**XIII. References**

[1] SDL2 Documentation. Available: https://wiki.libsdl.org/

[2] Stroustrup, B., "The C++ Programming Language," 4th Edition, Addison-Wesley, 2013.

[3] Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F., "Computer Graphics: Principles and Practice," 3rd Edition, Addison-Wesley, 2013.

[4] C Programming Language. Brian W. Kernighan and Dennis M. Ritchie.

[5] Pixil Art (www.pixilart.com)