Design Manual

Architecture Overview

The application is structured around the main `MainPanel` class, which extends `JPanel`. It orchestrates the overall layout and interactions between the `PalettePanel` and `GridPanel`.

- `App`: Serves as the main entry point for the application.
- `MenuBar`: Contains menu items for `Save` and `Open` operations
- `MainPanel`: Handles mouse events and interactions between the PalettePanel and GridPanel
- `PalettePanel`: Displays useable LogicGates and information.
- `PaletteComponent`: Represents a draggable component in the `PalettePanel` with properties such as `logicGate` and `bounds`. It includes a `draw()` method for rendering.
- `GridPanel`: Handles grid components and wires using relative coordinate system
- `GridLogicHandler`: Handles the logic related to the grid, including interactions with logic gates and wires.
- `LogicGate`: An abstract class extended by specific gate types like `ANDGate`. It provides essential methods like `draw()`, `contains()`, and `output()` to handle logic gate functionalities.

Each of these components works together to provide a comprehensive digital logic simulation environment.

Design Patterns

Singleton

The `PalettePanel`, `GridPanel`, `GridImporter`, and `GridExporter` all utilize the Singleton pattern.
There are no cases with the application that there would need to be more than a single version of these classes. For example when the `GridExporter` needs to export the current

grid layout to an XML file, export is called on the single instance of `GridPanel`. The Singleton allows for easy interfacing and synchronization between the classes.

Component Descriptions

App

At the top of the architectural hierarchy is the App class. This initializes the window with MainPanel and MenuBar objects.

MenuBar

The MenuBar contains commands for saving and loading circuits. There are also menu options for exiting the application, and menu options for an `About` page and `Documentation` page with a hyperlink.

MainPanel

The MainPanel contains the logic for handling mouse operations (may be refactored out to a MouseHandler class). This class manages the interaction between the `PalettePanel` and `GridPanel`, such as drag and dropping new logic gate items from the `PalettePanel` into the `GridPanel`.

PalettePanel

The `PalettePanel` displays `PaletteComponent`s which each house a `LogicGate` component. Each of the default gates are shown on the lefthand size of the application. When a component is hovered, the component colors the background of the component gray to make it clear to the user.

GridPanel

The GridPanel displays a grid and the current view of the logic gates and wires. In future updates, a `GridLogicHandler` class will be added that calculates outcomes based on the circuit on the grid.
When dragging a component around the GridPanel, components snap to the grid.

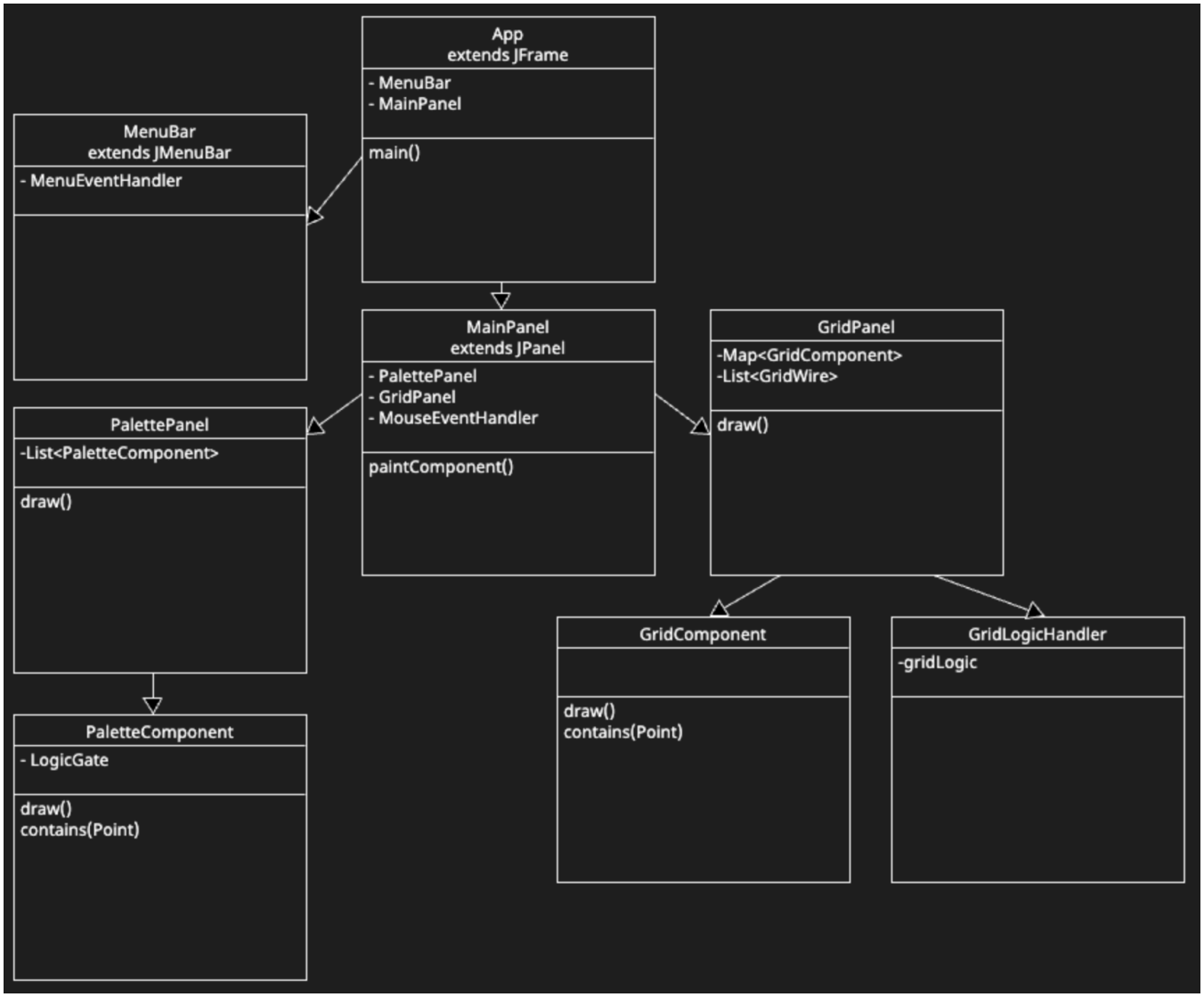
GridLogicHandler

Whenever a wire is added to the grid, if the wire connects some output to some input, the wire creates a connection between two components. This creates a Directed Acyclic Graph (DAG). The `GridLogicHandler` guarantees that connected components create a DAG without any cycles. If all the conditions are met, the input signals are processed and create an output.

LogicGate

Logic Gates are implemented as an abstract class which is extended by each gate such as ANDGate, ORGate, and XORGate. Each logic gate have associated image files in `resources/logicsim/gates`.

Diagrams



Standards and Conventions

The project adheres to Java's camelCase naming convention, with classes beginning with uppercase letters, methods as verbs, and objects as nouns.

Project Report

Introduction

The digital logic simulator project aims to create a user-friendly, interactive environment for building and testing digital circuits. The primary goal is to address usability issues encountered in existing simulators, like Logisim, and incorporate their successful features such as a grid layout, panning, and zooming.

Background

Logisim, a key reference for this project, offers valuable lessons in user experience and functionality. However, its wiring system presented significant usability challenges, prompting the development of this simulator. By improving the user interface and streamlining the wiring process, this project seeks to enhance the digital circuit design experience.

Methodology

The project adopted an agile development methodology, with iterative testing and refinement. Java Swing was used for the graphical user interface, ensuring cross-platform compatibility and ease of use.

Implementation Details

The simulator is being implemented in phases, starting with a large-scale skeleton overview of the project. Each item is being added in incremental steps. For example, the grid is first being created statically without zooming and panning, and will be expanded to include these features.

Testing and Evaluation

Testing was conducted informally through debugging sessions.

Results and Discussion

The first week took a lot of trial and error learning about Java Swing, brainstorming the architecture of the project, and working with ChatGPT effectively.
In the second week, much of the project was rewritten to manually handle the transition of drag and drop objects from the PalettePanel into the GridPanel, instead of using the TransferHandler interface and multiple JPanels in a Border style layout.
In the third week, the gate display was updated to use images instead of drawing using Java Swing graphics. In addition, resizing functionality was added to the Grid.

Conclusion

References and Appendices

- Logisim software
- Java Swing documentation
- ChatGPT
- Logic Gate images taken from https://www.categories.acsl.org/wiki/index.php?title=Digital_Electronics

User Manual

Setup

Launch the Application by running the App.java file `/src/main/java/logicsim/App.java`.

Functionality

Clicking and dragging a component allows you to add the Logic Gate to the Grid on the right-hand side. Each logic gate will be able to be rotated, as well as have its inputs or outputs negated. Each component will snap to the grid when dragging, and a visual shows the current position of the dragged object.
Objects placed on the grid are able to be moved with another drag and drop operation.

Future Functionality

Modifying Components

Rotation and editing the `not` state of each of the inputs and outputs for the object will be added.

Wiring

Click and drag from the input-end or output-end of any component to connect wires between components.