Project Report

Introduction

The objective of this project is to create a Digital Logic Simulator tailored for educational use. Its purpose is to facilitate students' comprehension and exploration of digital logic circuits. Leveraging Java Swing, the software will extend an existing basic paint application framework to allow users to construct, visualize, and simulate circuits incorporating various digital logic gates (such as AND, OR, NOT, XOR, etc.). The simulator will offer an easy-to-use interface for gate placement, wire connections, and the simulation of logical operations.

Background

This project takes heavy influence from Logism. While far simpler in its aims, it hopes to utilize the many useful features and interface designs whilst also providing an alternative approach to some aspects of the interface that some users may find frustrating. Features such as wire's auto snapping to the grid will not be included such that users may better organize their designs. It will not feature the multitude of gates which Logism offers, however by including the basic logic gates it will provide users to make their own (adders, multiplexers, etc.).

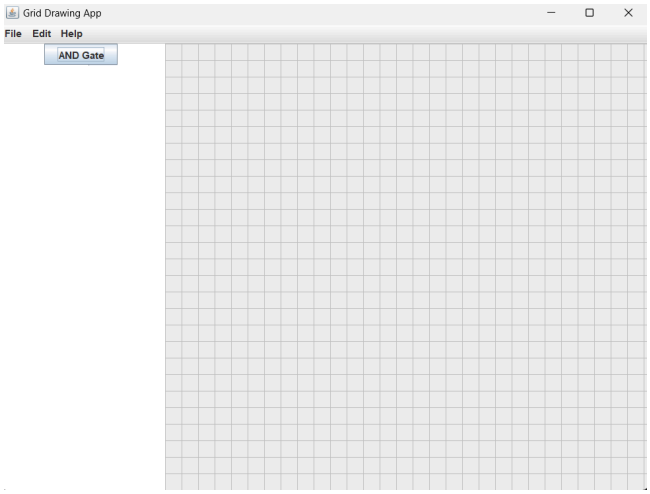
Methodology

Tbd … (at this stage it will be discussed in software design)

Implementation Details

Phase 1, UI Design:

This is an initial drawing of what the end product software should resemble. Part of usability is familiarity, such is that users should be expected to better understand software which is similar to that which they have used before. Menu options such as file and edit are included in their usual spaces. This program need not be complex with its design as its operations are fairly simple.



The functionality of the design should be simple. A user clicks their desired gate, and it is instanced in the workspace, after which, a user may drag it to their desired location. They will then likely want to add “wires” to connect any gates they may have in the workspace. These wires should follow the grid such that they can only exist along discrete coordinates. Such is that gates should only be allowed to be on the grid as well. From this design, by adding connection points to the gates and storing the end points of a wire, we can easily determine whether a wire is connected to a gate. This wire should then change color to demonstrate this or any other potential change to its state(in the case of simulation).

Let’s examine possible scenarios where this design needs to be improved upon. The grid exists in finite space, such that it may be too small for some designs. A zoom feature should be implemented that expands or deflates the grid to allow for easier editing. Additionally, if all gates are instanced in the same portion of the workspace, they may disrupt any current design; such that the addition of a designated area for them to be dragged from may be beneficial. Other problems could include a lack of a discrete field for input/output widgets (input/output widgets may behave similarly to a gate with this implementation, however they may seem that they belong to a different category to a user), the lack of any feedback to show that circuit is currently in simulation mode, or the fact that it is currently not possible to rotate gates to be in a more visually understandable position. These issues are user stories, some of which are potential issues observed in Logism, and that this implementation corrects.

At this stage, the largest existing class is AppInterface. It is far too large for the things it does and is better suited separated into other classes. Its primary duty, currently, is to create the JPanels, however these are better done in their own respective classes. Ideally, there will be two classes which extend from AppInterface, board and selectionMenu. Both of these should be implemented as singletons, as neither needs more than one instance at any point in time(implementing it as such isn’t strictly necessary but it will help ME not make mistakes and accidentally reinstance the workspace). Gates will implement an interface “Gate” which will handle the methods to get the gate logic and any other data associated with a gate. Wire is currently a class, but will be an interface which utilizes the state pattern as wires can have multiple states. Gates

Testing and Evaluation

Tbd..

Results and discussion
Tbd..
Conclusion
Tbd..

User Manual

…

Software Design

(This is highly subject to change, class diagrams will be added once I have a better feel for what the structure will end up being).

References and Appendices

Reference code for the UI design -
https://chat.openai.com/share/722f972e-5ca3-4c97-931b-b88eb341fcc2
https://chat.openai.com/share/722f972e-5ca3-4c97-931b-b88eb341fcc2

The introduction -
https://chat.openai.com/share/7eb3a40b-a7e7-408b-9445-4af2f1148f73