Project Documentation

Overview

Quantum Circuit Simulator is a software tool that allows users to design and simulate (evaluate) quantum circuits. The user can create circuits by adding quantum gates, configure qubits, and visualize the simulation results.

The software uses quantum gate illustration such as PauliX, PauliY, PauliZ, Hadamard, and CNOT gates, and simulates the results of these gates on qubits.

The GUI presents evaluation of each gate, and import to, export from file for visulization.

The software does not simulate a complete quantum computer with noise models or quantum correlation. It does not perform optimization of quantum circuits. The current version only supports basic quantum gates, not complex operations or multi qubit gates beyond CNOT.

Features

1. Basic Quantum Components such as Pauli-X (NOT), Pauli-Y, Pauli-Z, Hadamard (H), and CNOT gates.
2. Graphical User Interface (GUI) for users to design and simulate quantum circuits. The interface should allow users to add, move, connect, and remove quantum gates and qubits.
3. Allow users to save and load quantum circuits to and from files, preserving their structure and configurations.
4. Quantum State Calculations to compute and display the final state vector of the quantum circuit after the gates are applied.
5. Ready-made Quantum Circuits: Include several pre-built circuits (e.g., Bell state generation, quantum teleportation) to demonstrate the program's functionality.

Software structure

Overall Architecture:

The software follows an object-oriented design pattern with the following components:

Quantum Circuit Class: Represents a collection of qubits and gates.

Gate Class: An abstract class for quantum gates, with specific gate implementations like Pauli-X, Pauli-Y, Pauli-Z, Hadamard, and CNOT.

Simulator Class: Handles the simulation of the quantum circuit and computes the final state vector after applying the gates.

File I/O: Includes functionality for reading and writing circuits to/from JSON files.

In addition there are classes for visual components, that are VisualQubit, VisualGate, VisualCNOT, Button, PlaceholderGate and Result.

Class Relationships Diagram:

QuantumCircuit is the main controller that holds a collection of gates and qubits.

Gate is an abstract class, with subclasses like PauliX, PauliY, PauliZ, H, and CNOT, which implement specific quantum operations.

Simulator interacts with QuantumCircuit to perform the simulation and calculate the results.

FileIO handles loading and saving circuits from/to in JSON format.

Interfaces to External Libraries:

SFML (Simple and Fast Multimedia Library): SFML is used for graphical user interface components, window management, and event handling.

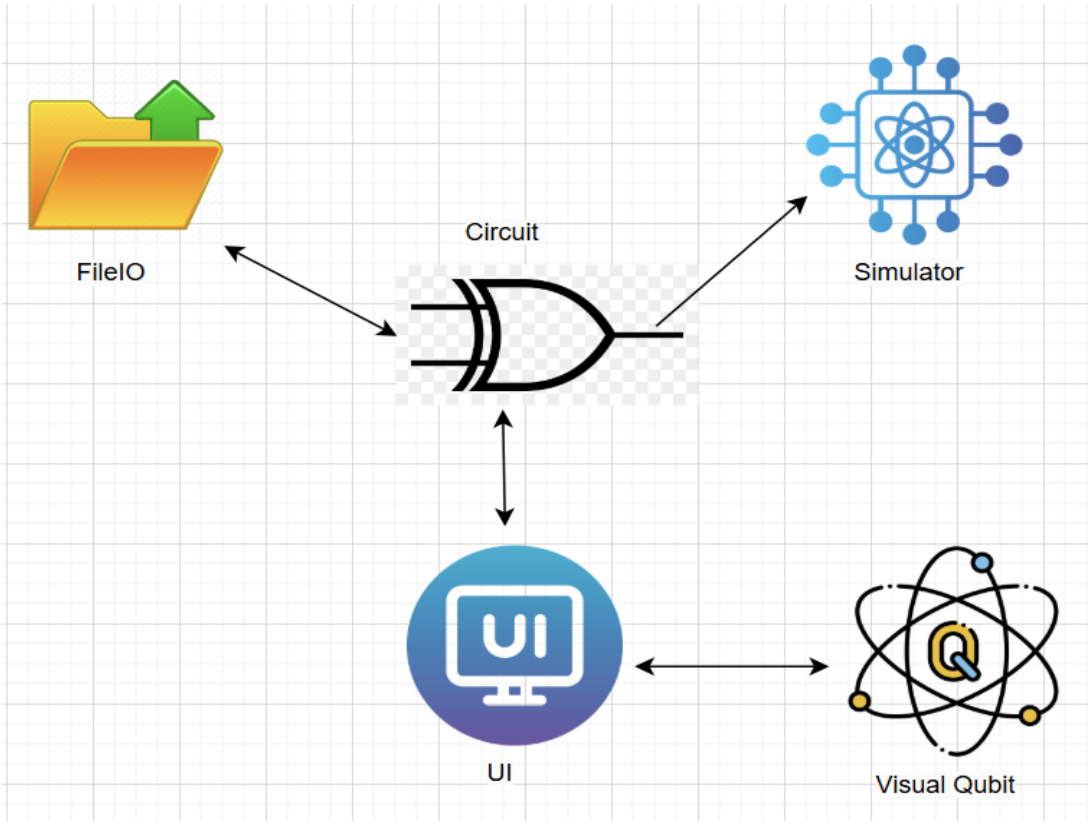
SFML-Graphics: Used for rendering 2D graphics such as drawing gates and qubits.

SFML-Window: Manages window creation and user input events (mouse clicks, window events).

Eigen: Used for matrix operations (quantum state vector manipulations and gate operations).

nlohmann/json: Used for reading and writing circuit data in JSON format.

tinyfiledialogs: Used for helping file selection and saving to the system.



Picture 1: The architecture diagram

1. Simulator
   - Definition: Executes the quantum circuit to compute the final quantum state. This component applies gate operations to simulate quantum mechanics.

   - Responsibilities:
     a. Take the circuit as input and process its logic.
     b. Generate the final state vector using linear algebra.
     c. Simulate quantum mechanics accurately using libraries like Eigen.

2. UI (User Interface)
   - Definition: Provides an interface for users to interact with the simulator. Allows users to build circuits visually, simulate them, and view results.

   - Responsibilities:
     a. Render the circuit visually using SFML (Simple and Fast Multimedia Library).
     b. Handle user inputs like adding gates, connecting qubits, and managing simulation.

3. VisualQubit
    - Definition: Represents the visual depiction of a single qubit within the UI. It stores information about the qubit's state and associated gates.
    - Responsibilities:
        a. Render the qubit and its gates visually in the UI.
        b. Handle interactions with the qubit, such as adding gates and updating its state.
        c. Handle user interaction for setting or modifying the state of the qubit.

4. FileIo
    - Definition: Handles input and output of quantum circuit data. It allows saving quantum circuits to files in JSON format and reading circuits from files to reconstruct their structure and logic.
    - Responsibilities:
        a. Serialize and deserialize the quantum circuit.
        b. Ensure compatibility with the JSON format.
5. Circuit
    - Definition: Executes the quantum circuit to compute the final quantum state. This component applies gate operations to simulate quantum mechanics.
    - Responsibilities:
        a. Take the circuit as input and process its logic.
        c. Generate the final state vector using linear algebra.
        d. Simulate quantum mechanics accurately using libraries like Eigen.

Instructions for building and using the software

Note on external libraries

Some of the libraries have dependencies that need to be installed on your computer (at least on Linux). The dependencies can be installed with "sudo apt update && sudo apt install <dependency1> <dependency2> <...>"

SFML

- freetype
- x11
- xrandr
- udev
- opengl
- flac
- ogg

- vorbis
- vorbisenc
- vorbisfile
- openal
- pthread

Nlohmann/json

- nlohmann-json3-dev
- libjsoncpp-dev

Tinyfiledialogs

If you want a graphical file selection, you need one of these:

- applescript
- kdialog
- yad
- Xdialog
- zenity
- (or matedialog or shellementary or qarma)

The developers used zenity successfully.

How to compile the program

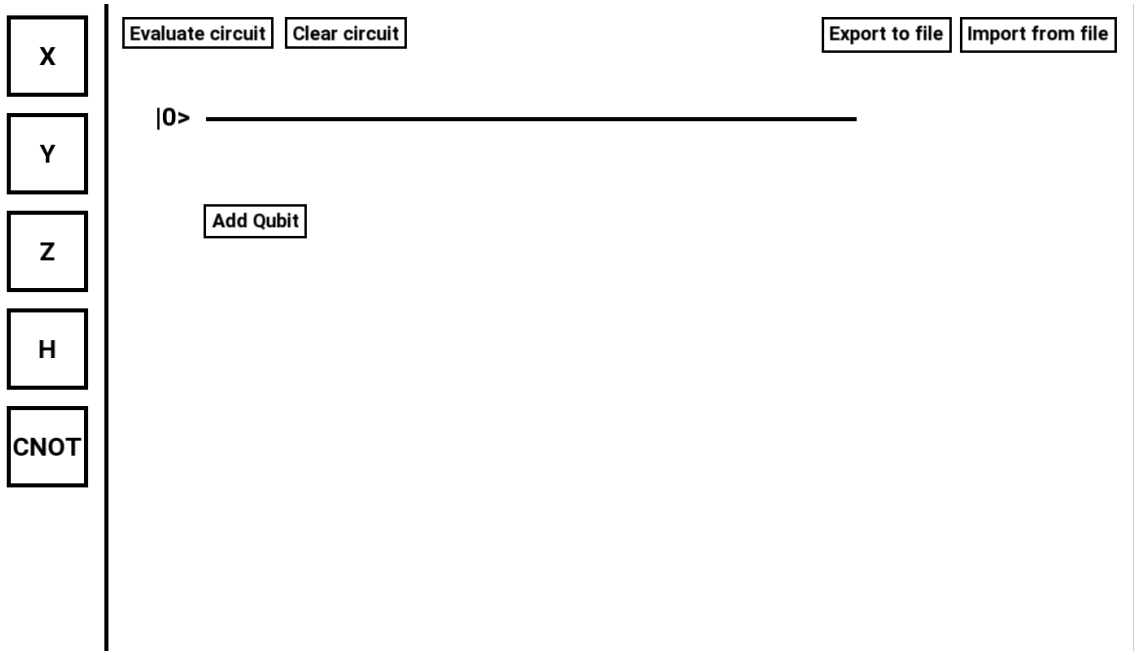
1. Initialize submodules
   git submodule update --init –recursive

2. Create a build directory
   mkdir build && cd build

3. Run CMake to configure the project
   cmake ..

4. Build the project
   Make

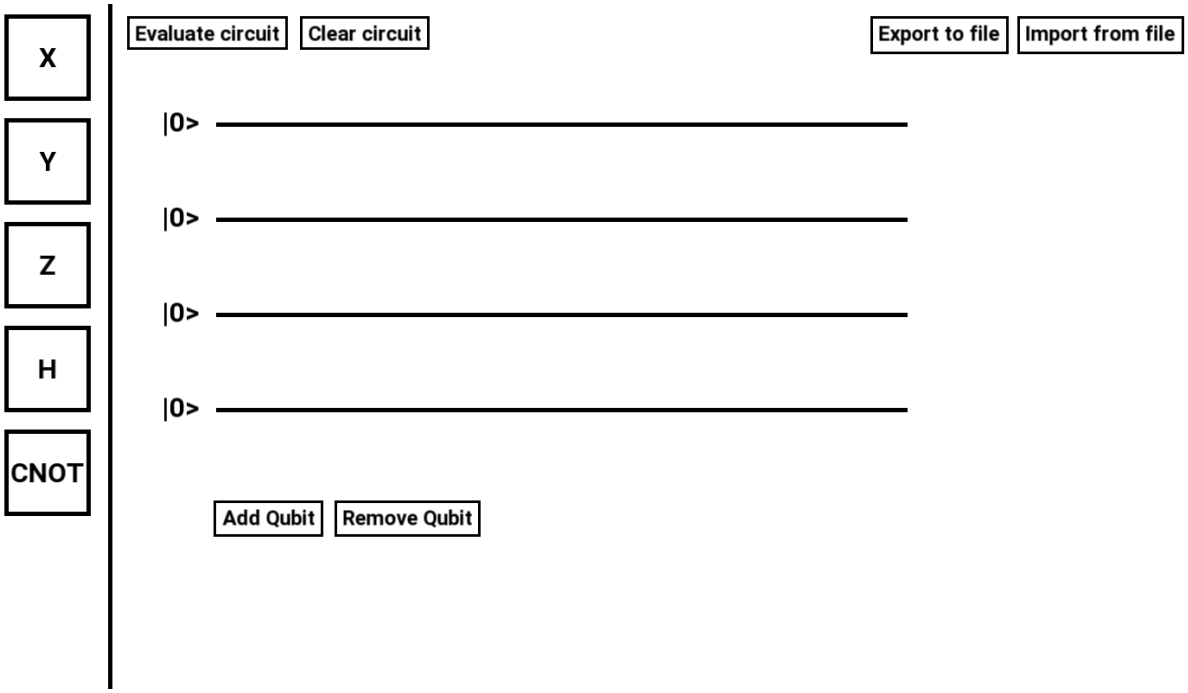
How to use the program



Picture 2: The UI when the program is opened

Add and remove qubits

Initially the window contains only one qubit. Below it is a button called “Add qubit”. Pressing the button adds one qubit below the first one and moves the button down. Another button called “Remove qubit” appears next to the button. By pressing that the down-most qubit and its gates are removed from the window. If the there is only one qubit left, “Remove qubit” button disappears. The maximum number of qubits in the window is four.



Picture 3: Circuit with the maximum number of qubits added

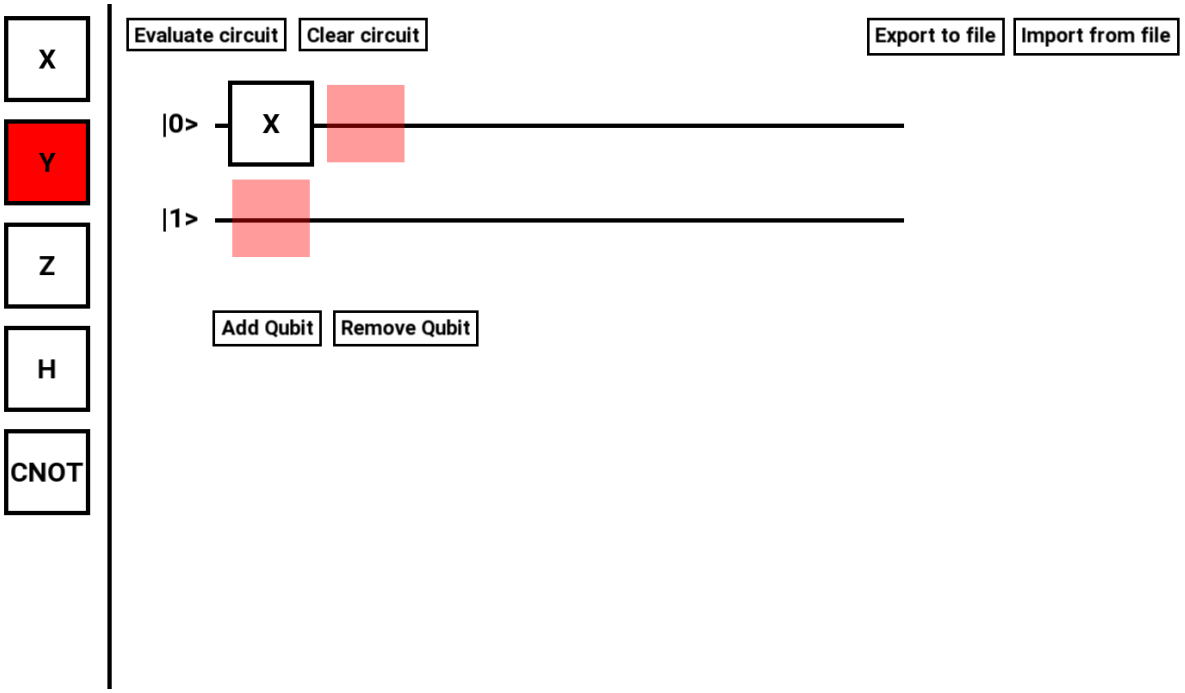
Switch initial state of the qubit

Next to the qubit is a text “|0>” which indicates that the initial state of the qubit is 0. By clicking the text it changes to “|1>” which indicates the initial state is now 1. By clicking the text again, it changes back to 0. The state is unique to the qubit, so there can be qubits with both initial stages.

Add gates to qubits

Single-qubit gates (X, Y, Z, H)

The gate selection can be found from the left side of the window. Clicking one of the gates changes the background colour of the gate red, indicating that the gate is selected. A placeholder “gate” with slightly transparent red background appears to every visible qubit that has room for new gates. Gate can be unselected by clicking the same gate again or selecting another gate. Clicking one placeholder adds a gate of selected type to the qubit and unselects the gate. If one gate is selected again, the placeholder in the qubit that now has gates has moved to the right to the next free “slot”.



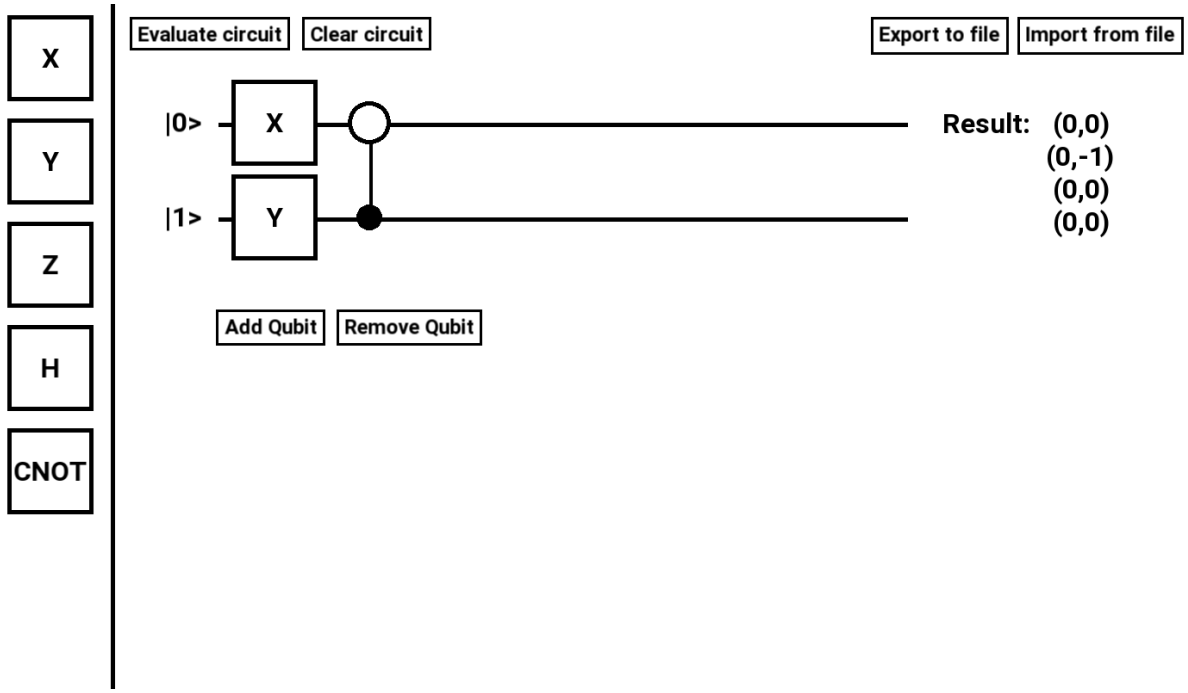
Picture 4: Two-qubit circuit with one X gate and gate Y selected.

Multi-qubit gates (CNOT)

With multi-qubit gates at least two qubits must be visible in order to add it to the qubits. Similarly, as with single-qubit gates, by selecting the gate the placeholder gates appear to the qubits. For CNOT gate, two different placeholder gates must be clicked in order to add the gate to the circuit. The first selected qubit is the control qubit (indicated with a small black dot) and the second one is the target qubit (larger circle with black outlines). The position of the gate is determined based on the right-most placeholder gate's position.

Evaluate circuit

When the desired gates and qubits with initial states are added to the circuit, "Evaluate circuit" button can be clicked in order to get the result of the calculation. The result appears to the right of the window, next to the qubit(s). The same result will be visible until the "Evaluate circuit" button is clicked again.



Picture 5: Evaluated circuit with three gates

Export circuit to file

When the “Export to file” button is clicked, a file dialog is opened (if there is a suitable software for it, otherwise the console is opened). It is possible to name the file to which the circuit will be saved and the location of the file. When the changes are saved, the circuit is saved as JSON file to the specified file in a specified location.

Import circuit from file

When the “Import from file” button is clicked, a file dialog is opened (if there is a suitable software for it, otherwise the console is opened). It is possible to search and select files from the file system of the computer. When the file is opened, the circuit is generated from the file to the window.

Clear circuit

Currently the only way to remove gates from the initial qubit is to clear the circuit or close and reopen the program. By clicking “Clear circuit” button all the gates and all but one qubit is removed, and the initial state of the qubit is set to 0.

Testing

User interface

Since the UI contains minimal logic, it does not have unit tests but is tested manually with every code review.

how the different modules in software were tested, description of the methods and outcomes.

Work log

Team Member	Task	Hours
Sooki	Multi qubit and controlled gates	100
Ha	QuantumCircuit class, read/write file.	100
Emma	Rotation and Pauli gates	100
Henna	UI and integration	100