Quantum Circuit Simulator

Software Design Documentation

SE 360 Advances In Software Development

Mert Can ÇIKLA

November 29, 2013

Contents

1 Introduction

Quantum Circuit Simulator is an application for Android capable of simulating essential gates used in fields of quantum computation and quantum information processing. The application aims to provide a way to simulate and see how a quantum calculation actually takes place.

1.1 Goals and Objectives

1.1.1 Ease of Use

First of all application is fairly easy to use with a clean and simple GUI. Drag/drop which is an intuitive action on a touch screen will be implemented with the framework included in Android API 11.

1.1.2 Functionality

This project will be able to simulate behaviour of any quantum gate or circuit that acts on up to 8 qubits or possibly less depending on screen resolution of the device running.

1.1.3 Speed

This application which has matrix multiplications which is a heavy calculation at its core, will be able to run smootly on any mobile device.

1.2 Target Users and Devices

Students studying mathematics, physics or computer engineering is the main targeted user group of this application however it might be a useful tool for researchers and professionals aswell. Mobile devices that are running Android 3.0 or above such as tablets or cellphones with a screen larger than 5 inches are the targeted devices of this project.

2 User Interface

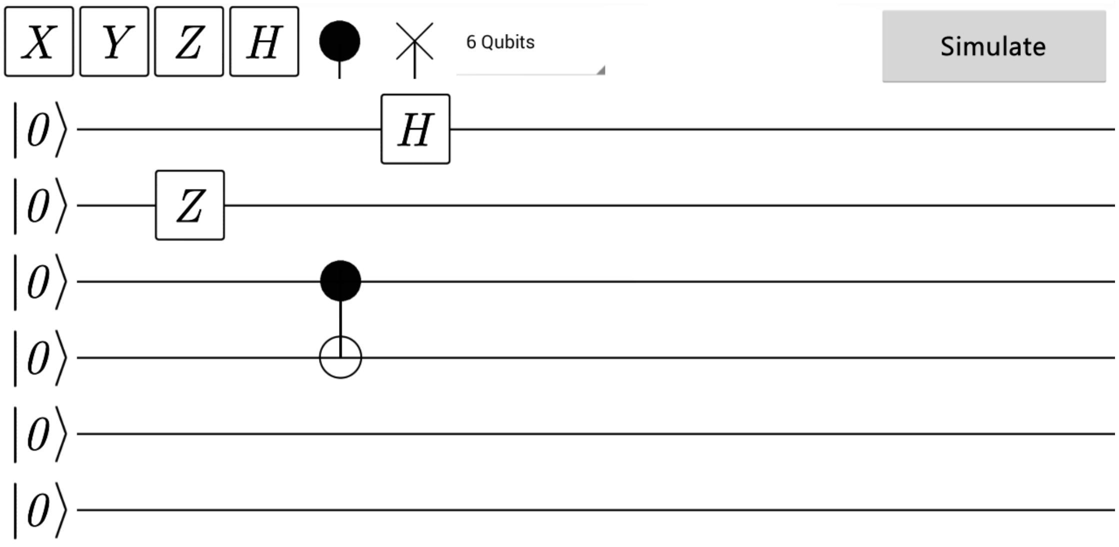


Figure 1: Sample User Interface

The user interface will be featured in fullscreen mode and defined in a single XML file. There is a single screen to be displayed with a result window popping up after simulation.

2.1 Toolbar

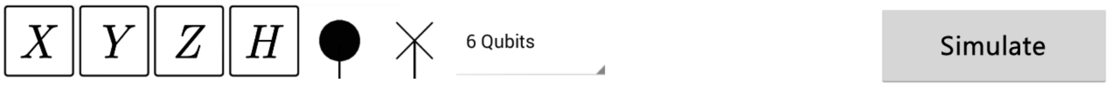


Figure 2: Sample Toolbar

The toolbar is to be defined as a horizontal LinearLayout object specified by the Android API. The toolbar holds quantum gates, spinner for number of qubit selection and the simulate button. A sample is shown in Figure 2.

Figure 3: Sample Gate Representation for Pauli-X

Quantum gates will be represented as shown which is the universal standard. This is represented by the class QuantumGate which is an extension of the android class ImageView that makes a copy of itself at start of a drag operation when it is the source, or can be used as a target of the drop operation which causes the source object to be deleted. A sample is shown in Figure 3.

The spinner object will display an array of text objects that on click sets the activeQbits to the corresponding value, activeQbits determine the number of qButton and gateContainer objects on the circuit board.

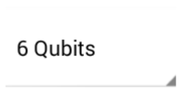


Figure 4: Spinner for number of qubit selection

2.2 Circuit Board

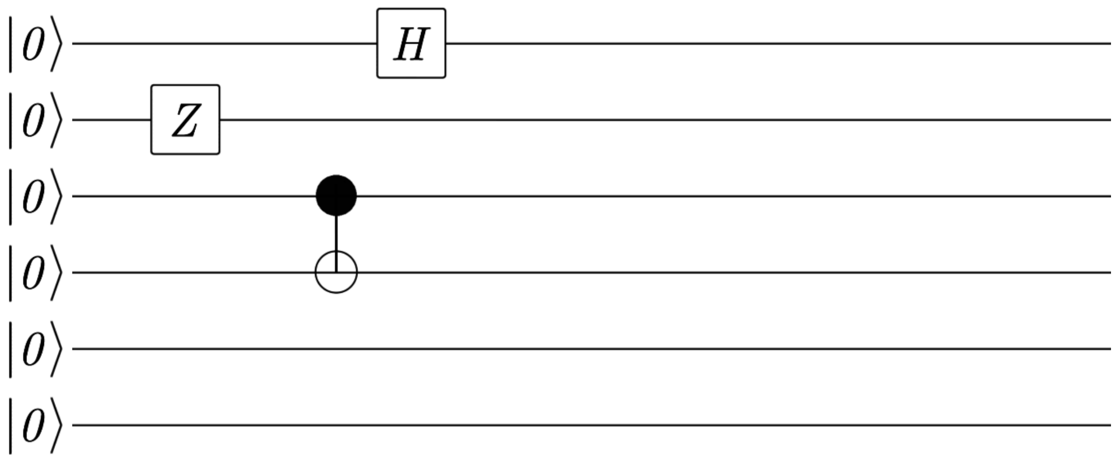


Figure 5: Sample Circuit Board

The circuit board is comprised of the buttons on the left and equal number of horizontally structured GateContainer objects.

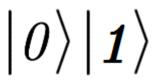
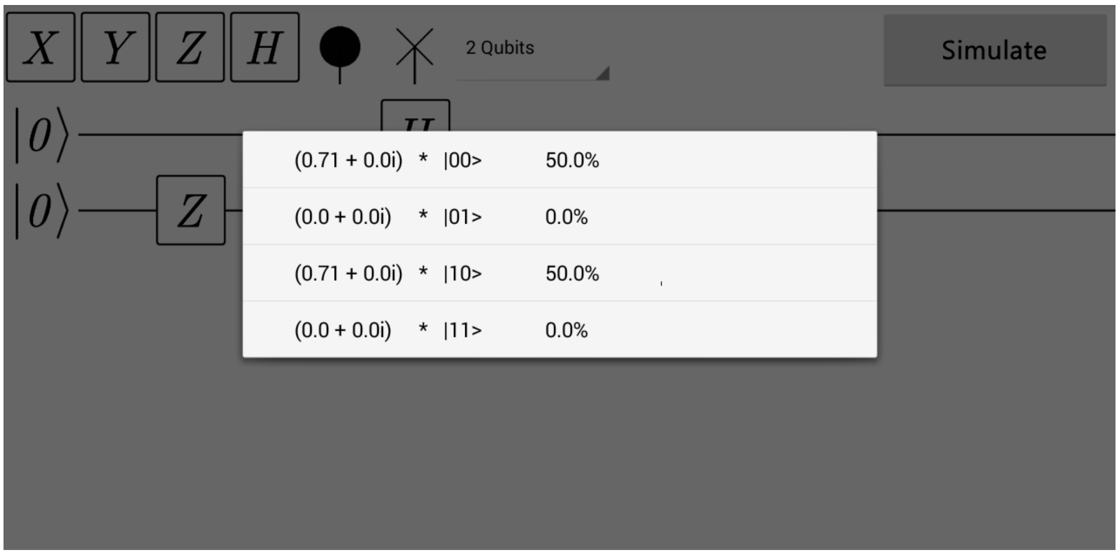


Figure 6: Buttons to change the initial state of a qubit.

Buttons will change between the initial states 0 and 1 on click. GateContainers act as container for QuantumGates initially filled w ith invisible objects to be used as drop target locations.

2.3 Result Screen

The result screen is drawn by the simulate method. The method will also pass the The layout is to be defined in a seperate XML file that consists of a single ListView object.



```
(0.71 + 0.0i)  *  |00>     50.0%
(0.0 + 0.0i)   *  |01>     0.0%
(0.71 + 0.0i)  *  |10>     50.0%
(0.0 + 0.0i)   *  |11>     0.0%
```

Figure 7: Example result screen for 2 qubits

3 Drag/Drop

Drag/Drop functionality is to be provided using the framework within Android API. The MainProgram implements interfaces for Touch and Drag Listeners.

3.1 onDrag

The onDrag method is called when a draggable object is dropped. The Method identifies the target and source location of the object and what is the base class of object is. Possible actions are explained on Figure 8.

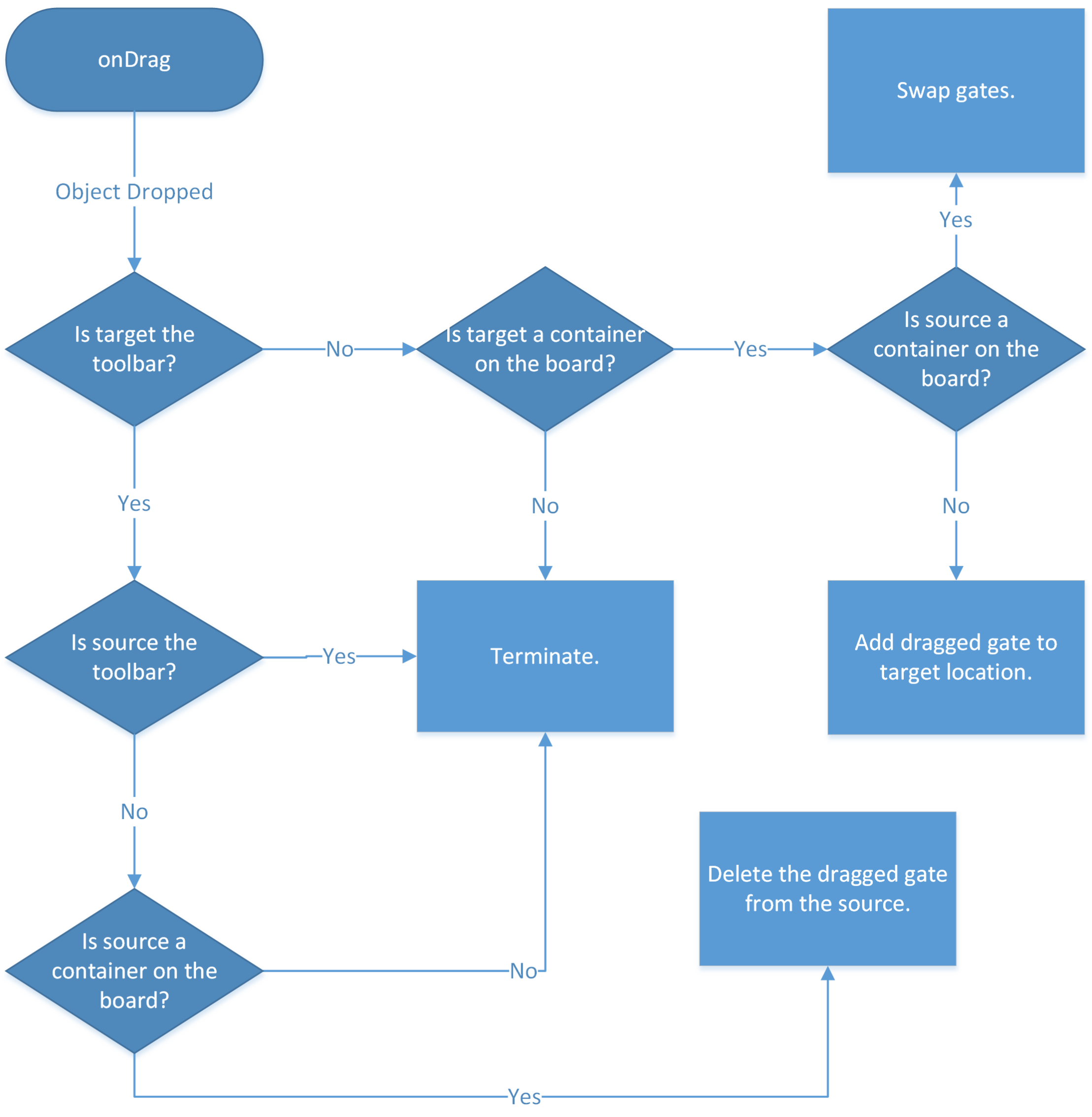
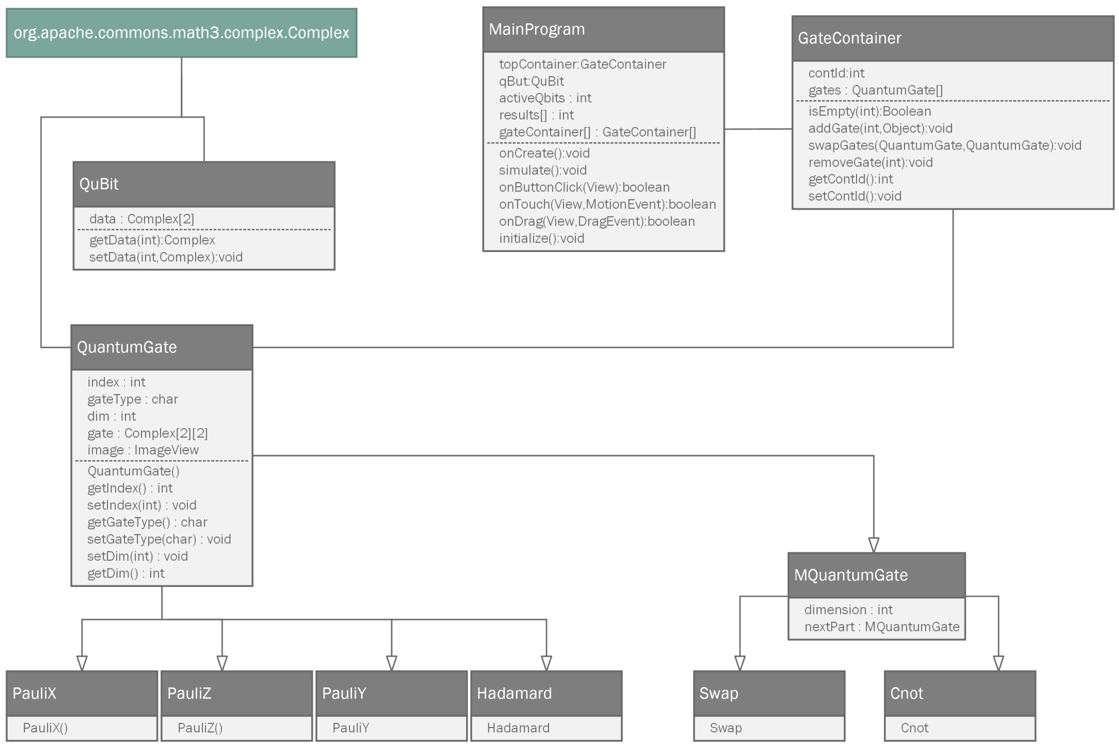

onDrag
Object Dropped
Is target the toolbar?
No
Is target a container on the board?
Yes
Is source a container on the board?
Yes
Swap gates.
Yes
No
No
Is source the toolbar?
Yes
Terminate.
Add dragged gate to target location.
No
Delete the dragged gate from the source.
Is source a container on the board?
No
Yes

Figure 8: Decision diagram for all drag events.

4 Class Descriptions

Classes PauliX, PauliY, PauliZ, Hadamard, Swap and Cnot are simple class representations of the gates. QuBit data values must be set at initialization as well as the gate image resources.

4.1 UML Diagram



4.2 MainProgram

MainProgram is an object with base class Activity and it is executed at startup automatically by Android.

Attributes

- **topContainer : GateContainer**
  Container shown in toolbar which holds all the QuantumGates Pauli-X, Pauli-Y, Pauli-Z, Hadamard, Swap and Cnot. Any child object within must call setOnTouchListener() and setOnDragListener() methods. On drag events, this object is copied for the drag handler to use as reference.

- **qButton : QuBit**
  QuBit object that start with initial value 0 and corresponding image of $|0\rangle$ can change to $|1\rangle$ on click. This object is setup within XML and calls onButtonClick method by android:onClick attribute set within XML.

- **activeQbits : int**
  activeQbits is set to 1 as default and determines the visible qButton and gateContainer objects.

- **results[ ] : ListView**
  This object is filled with TextView populated from the simulate() method. 2^N TextView objects are required for N=activeQubits.

- **gateContainer : GateContainer[20]**
  Container initially filled with invisible QuantumGates that fits the screen. Drag/Drop unto this area is valid. Array size of this object is determined by activeQbits. 20 should be sufficient to fill the screen on any resolution.

Methods

- **onCreate() : void**
  This method is called only by Android at startup. Any initilization and layout drawings are done in this method. Setting up event listeners is also handled here.

- **simulate() : void**
  Calculates the state of each qubit by iterating through the gateContainer. At each step data attribute of corresponding qubit is multiplied with the matrix contained in QuantumGate object's gate matrix. After calculation results are added to an array of TextView's and passed to the result screen for display.

4.3 QuantumGate

QuantumGate class extends ImageView class and is to be used to display quantum gates and store information about it.

Attributes

- **index : int**.
  This attribute containes the location index of the gate. Index is initialized to -1 and can have values between 0 and 20, 0 being the first gate in the gateContainer and 20 last.
- **gateType : char**
  This attribute is used to easily identify the gate type of the object. Set to X for Pauli-X, H for Hadamard etc.
- **gate : Complex[2][2]**
  Square matrix of size 2 that represents gate as the corresponding unitary matrix. Apache Commons java library for complex numbers is used for storing complex numbers.

Methods

- **QuantumGate()**
  Default constructor for the class that initializes values and sets an invisible image for ImageView attribute.

4.4 MQuantumGate

Attributes

- **dim : int**
  Integer object that holds size for the square matrix. Must be a value of 2^N for a gate that applies to N qubits.
- **nextPart : MQuantumGate**
  Holds a reference to the bottom half of a 2 qubit gate. This part will be displayed on the gateContainer under the the one holding the main part with same index.

4.5 QuBit

Attributes

– **data : Complex[2]**.
Stores vector information for the corresponding qubit.

Methods

– **QuBit()**
Initializes data attribute to 0 with array size 2.

4.6 GateContainer

Attributes

– **contId : int**
Identifies the vertical index of gateContainer. Takes values between 0 and 8, 0 for top and 8 for the bottom.

– **gates : QuantumGate[20]**
Holds quantum gate objects to display.

Methods

– **isEmpty(index) : Boolean**
This method returns false if this gateContainer does not have a child at position index. True if a QuantumGate object is stored.

– **addGate(index,Object) : void**
Adds the given object to the position index of this gateContainer.

– **swapGates(QuantumGate,QuantumGate) : void**
Swaps the gates using a temporary QuantumGate object.

– **removeGate(int) : void**
Removes the gate from the given index. Creates a new QuantumGate with no image resource with on drag listener to enable gates to be dropped and added later on.