Canberk KOSE - 18/05/2022

OOP PROJECT

LOGIC SIMULATOR

Project Goals:

- Designing a Logic Simulator by using Object Oriented Programming techniques.

Implementation:

Firstly, classes and their use cases are going to be explained in this section.

<u>Object Class</u>

Every sprite in the Logic Simulator is derived from this class. Apart from the attributes in project pdf, every object has its own name as a string and locked object position if the sprite is a locked object. Polymorphism is used in this class for to draw sprites to the screen -which is called “window” in SFML library-. Also for to set Wire objects’ coordinates.

This class also has attribute for holding the rectangle border of a sprite to be appeared red when the sprite is selected.

This class has a friend class called “Simulator” which is designed to hold the linked list and events for the Logic Simulator as it is going to be explained in the next sections.

<u>Wire Class</u>

This class is derived from Object class. It has attributes for its coordinates and pins that the wire is connected to. These coordinates are used for displaying the wires to the screen and changing its color to red when the wire is selected.

When a wire is placed on the screen then it is guaranteed that it is connected to viable pins of the other sprites. This class has this pin pointer attributes for to handle placing,deleting and calculation of the “logic” when the simulator is started.

Pin Class

This class is used for connecting sprites to each other in order to calculate the logic. Every logic gate has its own number of pins. This class has attributes for holding the state of the pin, pin type (output or input), pin position for to handle drawing wires correctly, wires pointers for handling deletion of objects and calculation of the logic and lastly other pin pointers that this pin is connected to for again handling deletion of objects and calculation of the logic.

Logic Element Class

This class is derived from the Object class. This is the base class for all the logic gates as it can be understood from its name. This class has attributes for number of the pins on a logic element, an array object for pins on a logic element.

There is an enum called pinPos which is the offsets of the pins on a logic element. This enum is used for setting the pins' position so that a wire can be drawn accordingly.

There is a "calculateOutput()" method in this class which is polymorphic. Every logic gate is going to have its own calculations so making this method polymorphic makes the code more readable and adjustable when necessary.

There is a method called "directOutputPinResultToConnectedPins()" which is used for calculation of logic. Every logic element has this function because when the calculation is done and the output pin value is set, this value needs to be passed to the connected logic elements. This method takes care of this part of the communication of the logic elements.

And Gate Class

This class is derived from Logic Element class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method handles the logic for an AND GATE. It checks if the input pins are set different from high impedance. If so, AND logic operation is done and the output pin is set according to the result. After the output is set, output value is directed to the connected pins via the method called "directOutputPinResultToConnectedPins()" which was described in Logic Element Class section.

Clock Class

This class is derived from Logic Element Class. It has attribute called "clock" whose type is defined in SFML library. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method checks elapsed time, if the elapsed time is bigger than 1 seconds (1Hz), it sets its output pin according to the previous output pin value. If output pin value is "0" then it is going to be "1" next and vice versa. After the output pin value is set, this value is directed to the other connected pins via the method called "directOutputPinResultToConnectedPins()" which was described in Logic Element Class section.

Dff Class

This is the class for D Flip-Flop. This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method handles D Flip Flop logic. It checks if its two input pins are different from high impedance. If so, it checks if the clock is "logic 1". If so, the input "Data" pin (index 0) value is directed to the first output pin. Also, the negated value of the input "Data" pin is directed to second output pin. After all the output pins values' are set, these values are directed to the other connected pins via the method called "directOutputPinResultToConnectedPins()" which was described in Logic Element Class section.

Gnd Class

This is the class for Ground (logic-0). This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method handles directing "logic 0" to all connected pins by iterating over all connected pins.

Led Class

This is the class for LED element. This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

There is only one pin in a LED element. calculateOutput() method checks this pin state and alters its sprite texture according to it. When the pin state is “logic 1”, the led is red which means it is on. Otherwise it is white which means the pin state is “logic 0”.

NotGate Class

This is the class for NOT gate element. This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method checks if the input pin state is different from high impedance. If so, the output pin state is set to the negated value of the input pin.

After the output pin value is set, this value is directed to the other connected pins via the method called “directOutputPinResultToConnectedPins()” which was described in Logic Element Class section.

ORGate Class

This is the class for OR gate element. This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method check if the input pin states are different from high impedance. If so, pin states are summed up and if the result is equal and bigger than integer “1”, the output pin is set to “logic 1”, else to “logic 0”.

After the output pin value is set, this value is directed to the other connected pins via the method called “directOutputPinResultToConnectedPins()” which was described in Logic Element Class section.

Vdd Class

This is the class for Vdd element. This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method works like Gnd Class calculateOutput() method. It iterates over all connected pins and sets their state to “logic 1”.

XOR Class

This is the class for XOR gate element. This class is derived from the Logic Element Class. When an object instance is created, texture of it is set by the method in Object class.

calculateOutput() method checks if the input pins states are different from high impedance. If so, it checks if the input pins are equal to each other or not. Output is set to “logic 1” if the input pin states are not equal to each other. Else, it is set to “logic 0”.

After the output pin value is set, this value is directed to the other connected pins via the method called “directOutputPinResultToConnectedPins()” which was described in Logic Element Class section.

Simulator Class

This class includes some event handling methods and the linked list for the sprites. Simulator class attributes are: flags for drawing wire and simulating, simulation clock, the SFML window and head pointer of the linked list. A Simulator Class object instance is created once at the main(). On creation, the toolbar logic elements and simulation buttons are created via the methods “createNewObject” and “createButtonToolBar”.

Some methods are going to be explained next in order to get the gist of their purpose.

- Method for appending an object instance to the linked list.
- Method for deleting an object from the linked list and destructing it.
- Method for drawing object instances to the window by traversing the linked list. (traverseAndDrawAllObjects)
- Method for making “selected” attribute false for all the linked list elements. (resetSelectedObjects)
- Method for creating new Object Class instance according to the arguments passed to the method like object name, coordinates and if the element is going to be “locked” or not. After creating the object instance, sprite positions and pins of the object is set. Lastly, the new created object is appended to the linked list. (createNewObject)

- Method for creating a new wire object instance and appending it to the linked list. (createNewWire)
- Method for getting the locked object position. This method is called when drag&dropping finish. It is used to return the locked object to its predefined coordinates. (getLockedObjectPositionByName)
- Method for getting wire Object instance pointer by traversing the linked list and checking if the wire Object instance's second pin is connected to a pin or not. This method is used to update wires' coordinates while drawing wires and checking if the wire can be drawn or not. (getWireObjectPointer)
- Method for getting the pin pointer of an Object class instance. It iterates over all the pins of the Logic Element Class instance, gets the pins' coordinates and checks if the mouse coordinates and the pin's coordinates are the same. If so, the pin pointer is returned. Else null pointer is returned. This method is used for checking if the user wants to draw a wire or not on mousePressed event. (getPinPointerOfObj)
- Method for getting the pin pointer matched by the mouse coordinates and pin position by traversing the linked list. Its logic is similar to getPinPointerOfObj() method. This method is used to check if a wire can be drawn or not.
- Method for checking if the wire can be drawn or not by controlling if a valid pin is targeted by mouse or not. It also checks the pins' type. Because it will be not logical to connect two OUTPUT pins to each other. (chechIfWireCanBeDrawn)
- Method for setting pins' "connectedTo" and "wires" attribute to nullptr. This method is used when deletion of a wire occurs. These attributes needs to be set to nullptr because there will not be a wire connecting the pins together after deletion. (onWireDeleteHandleConnectedTo)

Now, events are going to be explained:

onMousePressed Event

Firstly, we check if the “LEFT” mouse button is pressed or not. This is the only event we handle in onMousePressed.

We traverse the list and check if an object is clicked. If not, we set all objects “selected” to boolean “false”.

If an object is clicked, we check if this object is an toolbar object or not.

- If the object is a toolbar object, object is checked to be a Button or not. If so, button methods are called. If not, the toolbar object’s “selected” attribute is set to boolean “true” for handling the movement (drag&drop) event for new sprite creation.
- If the object is not a toolbar object, first all objects’ “selected” attribute is set to “false”. Because there should not be two objects that are selected at the same time. Then the user is checked if user wants to draw a wire or not.
    - If user wants to draw a wire create new wire Object and set the drawing wire flag to “true”
    - If user does not want to draw a wire, then user wants to select a sprite to delete. So, set the “selected” attribute to “true”.

mouseMoved Event

First, if the LEFT mouse button is being pressed. If not, do nothing.

Then, check the flag for drawing wire to understand user’s intent.

- If user is drawing a wire, update the wire’s coordinates according to the mouse position.
- If user is not drawing a wire, then user is moving a “locked” toolbar object to create a new sprite. So, check if an object is selected before and check this object if its “locked” attribute is set to “true” for more adjustable code. Then update the sprite’s position according to the mouse position.

mouseButtonReleased Event

First, check if the released mouse button is LEFT or not. If not, do nothing.

Then, check if user is drawing wire or not by controlling the drawing flag.

- If user is drawing a wire, then by releasing the mouse button, user wants to finish drawing a wire. Some controls are made for checking if user drew a valid wire or not.
  - If a valid wire is drawn, set the pins' "connectedTo" and "wires" attributes accordingly.
  - If a valid wire is not drawn, delete the wire object instance which was created before on mouseButtonPressed event.

  Set the flag for drawing to "false".
- If user is not drawing a wire, then the user intents to create a new logic element object. Traverse the linked list and check if an object's "selected" is set to "true" and check if this object is "locked" or not. These all mean that user was moving a toolbar object and released the mouse button to place it to the window. The toolbar object's position is set back to the its locked position and a new object instance is created on mouse coordinates.

keyReleased Event

First, check if the released key is "Delete" key or not. If not, do nothing.

Then, check if an object is previously selected or not. If not, do nothing.

Then, check if the object is an toolbar element or not. If so, do nothing.

Then, the user intents to delete a wire or a logic element.

- If user wants to delete a wire, call onWireDeleteHandleConnectedTo() method for handling the pin attributes.
- If user wants to delete a logic element, then all the wires connected to this logic element must be deleted too. In order to achieve this, iterate over the logic element's pins and while iterating over pins, iterate over every wire that is connected to the pin to process wire deletion.

Then, delete the object.

How it works?

Simulation

When simulation starts, it iterated over the linked list and calls the method that was previously explained calculateOutput() for the objects that are not wire and toolbar object. This process needs to be done a few times because when pin state of a logic element pin is high impedance, the output pin of the logic element is going to be high impedance too. This means that if there is a series of logic elements, the third element needs to wait for the result of the first two logic element in order to calculate its output pin.

Logic Element Pins

When a logic element is created, its pins' state is set to high impedance except for Ground, Clock and Vdd.

Selecting Wire Algorithm

Wire is a 1px wide line. So, the process of setting its "select" to "true" should be simplfied by clicking not only on the wire but also the near of it. In order to calculate how far the clicked coordinates are from the wire, an imaginary triangle is made by the end points of the wire and the clicked coordinates. By knowing all these 3 points' coordinates, triangle area is calculated from Heron's Formula.

To get the distance between the clicked position and wire, the triangle area which is area=(baseLength*height)/2 can be used also. The height here is actually the distance we want to find. We already know the triangle area from the Heron's Formula, and the baseLength is just the wire length.

So the distance can easily be derived from applying these steps but yet there is a problem. If the triangle is an obtuse triangle, then this problem occurs. Because the height can be very short (so the distance we want to find), but the clicked position can be far away from the wire and still select the wire.

In order to handle this problem, an imaginary rectangle, whose diagonal is the wire line, is drawn. If the clicked position is inside the rectangle then we can guarantee that the wire is intended to be selected.

Discussion:

The events can be handled more properly if a new class is designed just for the events.