\*

Logic Circuit Simulation Tool for Students

COMPUTER SCIENCE A-LEVEL NON-EXAM ASSESSMENT

Ruben Doyle-Morgan (HCACP)

2019

Contents

[Analysis 3](#_Toc7786198)

[Background 3](#_Toc7786199)

[Interview with D.F, Computer Science Student, Year 12 – Primary Client 3](#_Toc7786200)

[Observation of existing Logic gate simulators 4](#_Toc7786201)

[Plans and Objectives 7](#_Toc7786202)

[Documented Design 8](#_Toc7786203)

[Program Functions 9](#_Toc7786204)

[Class Functions 10](#_Toc7786205)

[Menu Class 10](#_Toc7786206)

[Source Class (Input) 10](#_Toc7786207)

[Output Class 10](#_Toc7786208)

[Gate Class 10](#_Toc7786209)

[Wire Class 11](#_Toc7786210)

[Gate Logic 12](#_Toc7786211)

[Technical Solution 14](#_Toc7786212)

[Prototype: Text-based Logic Gate Simulator 14](#_Toc7786213)

[Main Program: Interactive Logic Gate Simulator & Revision Tool 19](#_Toc7786214)

[Visual representation of Mesh and Sub-Mesh 21](#_Toc7786215)

[Drawing and Connecting Wires 22](#_Toc7786216)

[Save function 23](#_Toc7786217)

[System Testing 24](#_Toc7786218)

[Constructing and Simulating Logic Circuits 24](#_Toc7786219)

[Half Adder 24](#_Toc7786220)

[Full Adder 25](#_Toc7786221)

[2-Bit Multiplier 26](#_Toc7786222)

[3-Bit Multiplier 28](#_Toc7786223)

[Utility Testing 32](#_Toc7786224)

[Clear Editor 32](#_Toc7786225)

[Item Hints 32](#_Toc7786226)

[Save to file 33](#_Toc7786227)

[Load from file 33](#_Toc7786228)

[Videos of the Logic Gate Simulator 34](#_Toc7786229)

[Demonstration of Program 34](#_Toc7786230)

[Logic gate testing video 34](#_Toc7786231)

[Evaluation 35](#_Toc7786232)

[Summary of Objectives met 35](#_Toc7786233)

[Possible Expansions of the program 36](#_Toc7786234)

[Conclusion 36](#_Toc7786235)

[Appendix 37](#_Toc7786236)

[[Prototype] Text-Based Logic Gate Simulator (Code) 37](#_Toc7786237)

[[Main Program] Interactive Logic Gate Simulator (Code) 40](#_Toc7786238)

# Analysis

## Background

Boolean logic is a some-what simple yet hugely important branch of Computer Science which is taught in both the GCSE and A-Level variants of the subject alike. Although most students may grasp the concept of how individual logic gates work fairly quickly, many find it hard to gather an understand on how logic circuits are integrated into most, if not all the technology they use every day, be it their phone, calculator or burglar alarm at home. Despite the importance of this topic of the subject, there do not appear to be many tools to help gain a better understanding of how logic gates are used in unison and their importance in everyday appliances. A lot of circuit builder simulators available online are either low quality (top google result of a gate simulator from “Academo” is very basic and only allows one output), or really expensive as they are for commercial use (logic.ly is well made, yet costs $59 for a student to use, or $599 for a classroom).

My goal is to create a program that not only allows users to build and simulate whatever circuit they desire, but also can teach them about how they work and give examples of circuits that power certain technologies and devices, with loadable pre-sets of said circuits. With a tool like this at hand, computer science students would be able to gain a strong understanding over how Boolean logic circuits function and how they are used relatively quickly, increasing quality of life for both them and their teacher. Students could also use the program anytime they need to revise logic circuits for their exams.

## Interview with D.F, Computer Science Student, Year 12 – Primary Client

**How well do you understand logic circuits and Boolean algebra so far? (and their functionality)**

DF: I understand how logic gates work, but to be honest I’m not sure how they are actually used in the real world. I’m ok at simplifying Boolean algebra but I struggle with the more complex problems.

**Would it help if you were able to have a program visualise Boolean algebra for you?**

DF: Yeah that would help a lot actually, I’m finding it hard to visualise complex problems.

**Do you think a program that allows you to build and experiment with logic circuits, either from scratch or from devices that already exist e.g. a full adder in a calculator, would help you gain a better understanding of Boolean logic in general?**

DF: Yes, it would be nice to get to interact with logic circuits myself. Writing outcome tables in my exercise book doesn’t help me that much, as I find it hard to tell what’s going on in more complex circuits. It would also be nice to see examples of how logic gates are used in the real world.

**Do you think this program would also benefit others in your class?**

DF: Yeah, other people in my class struggle with Boolean algebra too.

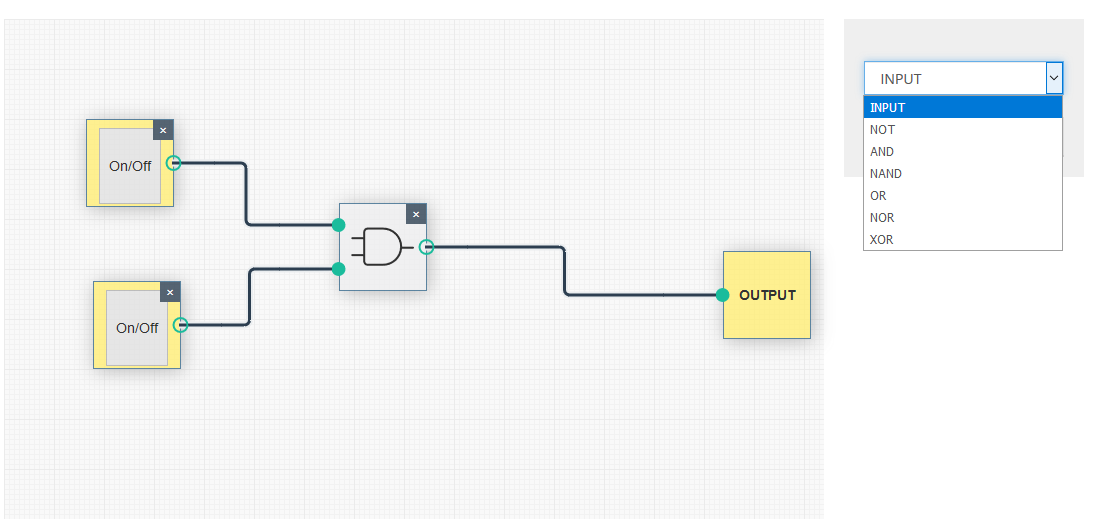
**Would you agree that a tutorial mode option that gives explanations of early topics (like binary adding) as well as harder relevant topics could be a useful feature of the program?**

DF: Yes, it would be useful to students who are just starting Computer Science, they would be able to gain an understanding of Boolean algebra and logic gates a lot faster.

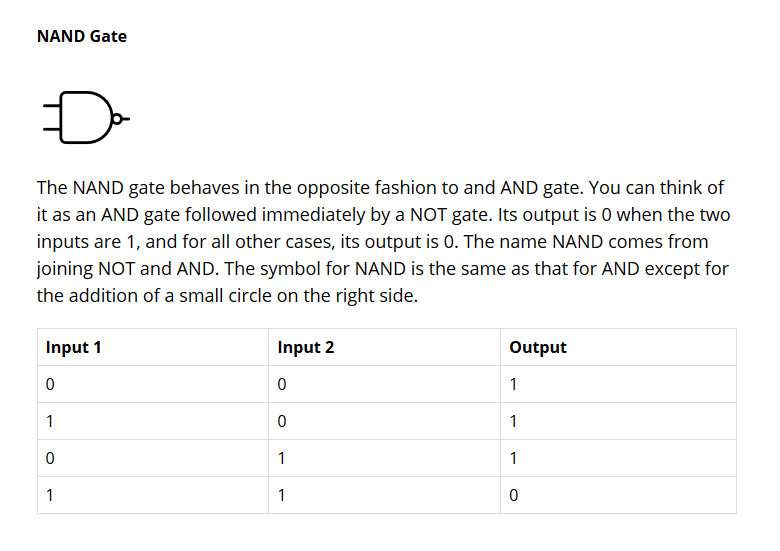
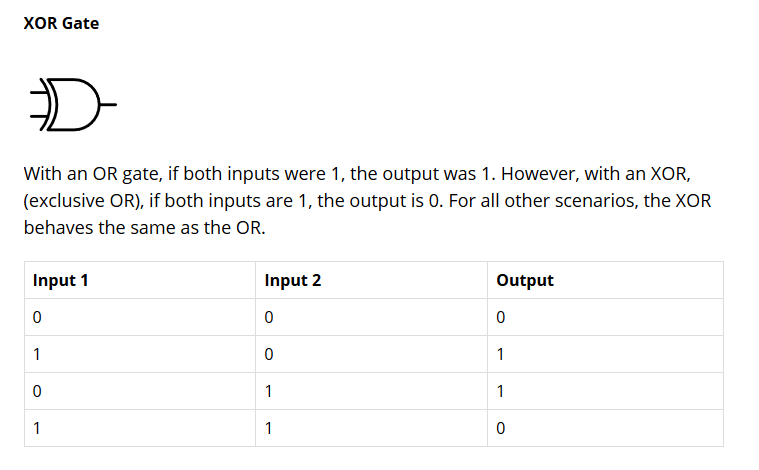
## Observation of existing Logic gate simulators

### “Logic Gate Simulator” by Academo

<https://academo.org/demos/logic-gate-simulator/>

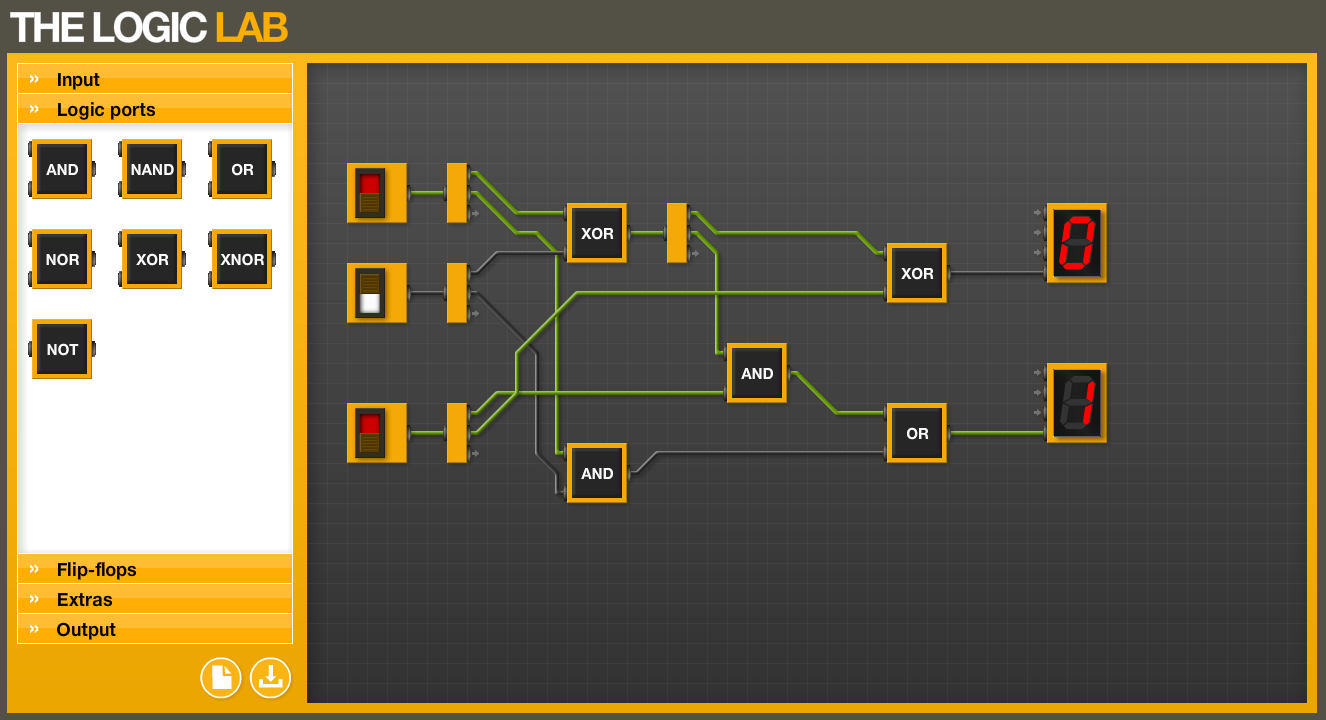


This editor is the top result on Google from searching “logic gate simulator”. It is good for simple circuits, but a big downside is that you can only have one output, therefore it is impossible to make more functional circuits like a Full Adder for instance. Components glow yellow when they are activated, which is a nice indicator to give the user a better idea to what’s going on. The website also has descriptions for each logic gate, which is useful for students that are unfamiliar with how they work.



### “The Logic Lab” by neuroproductions

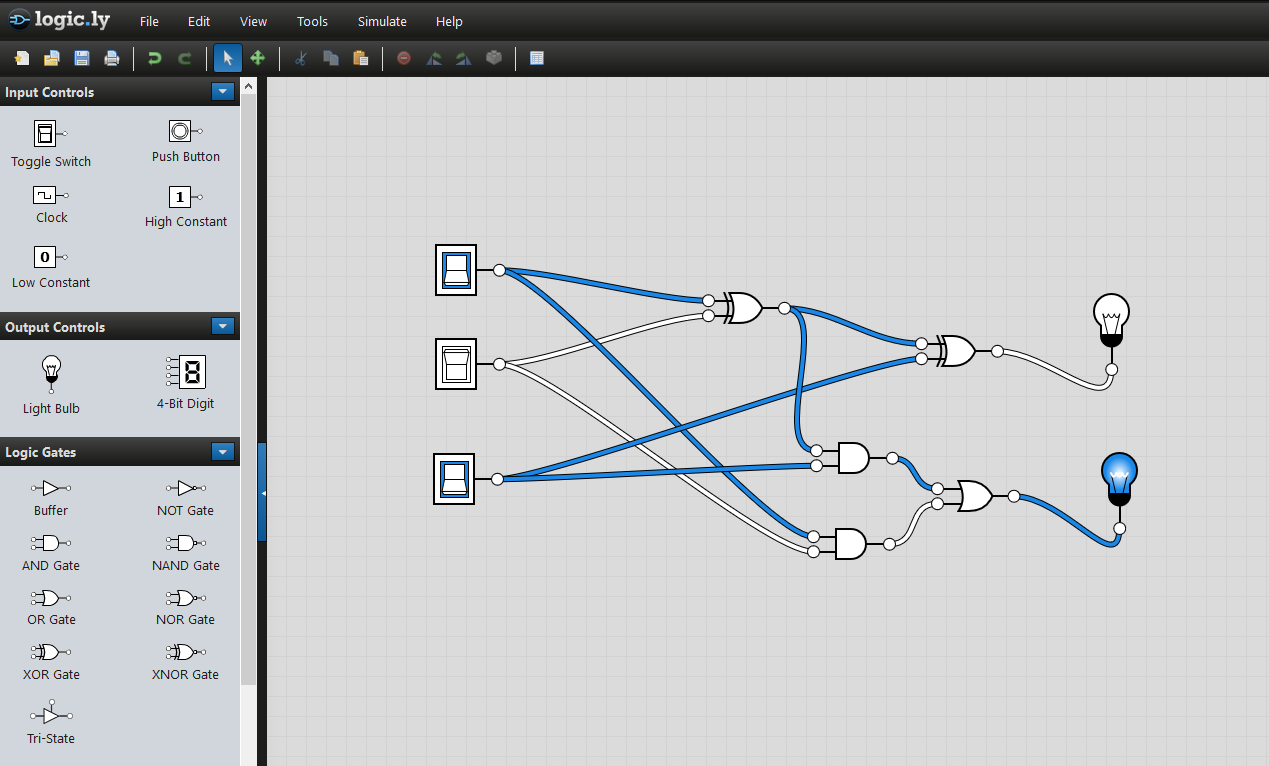
http://www.neuroproductions.be/logic-lab/



Full Adder circuit created on the Logic Lab

The Logic Lab is a more complex simulator than the one by Academo, allowing not only multiple outputs, but variants for inputs and outputs such as 4-bit denary displays and buttons for pulses. It also comes with flip-flops. An annoying downside to this program is that nodes cannot be shared, forcing the user to use the 1 to 3 connector part whenever they want to have multiple connections. This makes building circuits awkward and a bit tedious. There is also no information provided on how the different components function, thus making the program difficult and confusing to use for a novice Computer Science student.

### Logic.ly by Bowler Hat LLC

https://logic.ly/demo

Full Adder circuit created on the logic.ly demo

Logic.ly is a very sophisticated simulator which gives a lot of freedom to the user to do what they want by giving them many components to work with along with tips to guide the on how certain things work. Unlike the other simulators, circuits can be saved and loaded, which is useful as students can work on large scale circuits over time if they need to, or could receive files to load from their teacher for a lesson. The interface is simple and easy to understand and look at. This software is by far the best out of the three, which is to be expected with a $59 fee for an individual copy alone. Alternatively, the teacher could get a classroom package for $599, which allows 20 students to use the program at once. There is also a campus package for $1299 which allows unlimited users within the site for the program. This is a pretty big downside for logic.ly, as the computer science department would need a lot of funding from the school to be able to use this program.

## Plans and Objectives

My goal is to make a program that has good functionality and convenience for students, without the downside of requiring funding to utilise. All of these simulators have flaws. I can use that to make sure my program does not have these flaws, so it can be as helpful as possible for the students that use it.

I intend for the program to be just as useful to new computer science students as to students nearing their exams, therefore to achieve this I plan to:

* Add an option to toggle a tutorial mode, which will run the student through how different logic gates, circuits and components function if the setting is enabled.
* Keep the design of the program clean and simple, so that it is not confusing to interpret, yet still functional for students that want to make complex circuits on.
* Add informational tip icons for components that the user can use, in case they forget or want to learn how it works, but do not want to use tutorial mode.
* Include pre-sets of useful circuits like full adders, along with other circuits that tend to come up on exams like flip-flops.
* Include an option to write logic circuits in Boolean algebra, for students that would like to see how they compare.

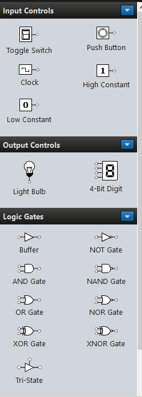
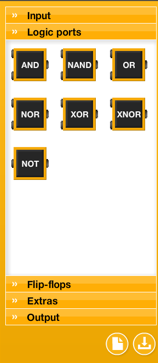
In order to ensure development goes smoothly I have decided to aim for the program to reach these objectives:

1. Able to create any possible 1-bit logic circuit with the standard logic gates (AND, OR, NOT, XOR, NAND, NOR, XNOR).
2. Have a decently large library or pre-made circuits that are commonly used, such as a full adder circuit (with the ability to combine said circuit into an array, with hints on how to do so), bit multiplier circuits, etc.
3. An option to enable a guide to teach the user how different logic components and systems work and are used, a feature intended for students who are newer to Computer Science.
4. A hint icon on each item that can be triggered to give a quick summary on how it works/how it’s used.
5. A clean, simplistic interface that is not confusing, yet still pleasing to look at and functional.

# Documented Design

The application will be developed using Python3. Classes will be used to define the interactive objects. I have decided to use the PyGame module to design and utilise a GUI for the program, as it is commonly used to design games with graphics on a UI it should prove to be a good tool to create this program. The program will be using .png images to draw the objects, and will be able to save and load data using text files. Its main function will be a logic circuit building tool, where the user can drag desired logic gates from a side menu for use in whatever logic circuit they’re making.

I have decided to use the logic.ly editor as the main influence to the design of my program as it is the most ascetically pleasing editor, while still being simple and easy to understand.

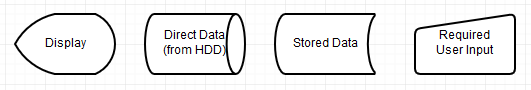
The editor menus for both logic.ly and The Logic Lab use tabs/subsections to divide the object components into groups. Although this may make navigating the menu easier, I will most likely not use tabs in my menu as I am only planning on adding the basics in terms of logic circuit components.

As stated in the opening paragraph I intend to carry over the save/load function from the prototype program. This will be more challenging however since a lot more data will be required to recreate a circuit identical to the one saved to a file (e.g. object position, state [for inputs], etc will be required). Text files should still be suitable for storing this data.

For the object models I intend to create my own designs for them with the program paint.net, as it has suitable drawing tools for making sprite objects. I plan to model them based on their diagram symbols. I will most likely need to create separate images for each possible state for the gates, as I want there to be a visual change when a signal is given to one of the gate ports. This could be possible with one image for the gate, however it would be difficult and tedious to code these effects, so I will probably go with making multiple sprites for each gate. None of the existing programs I’ve seen have dynamic graphics for their gates based on their individual ports so this idea was not influenced. I just felt that it would benefit students by making it easier to follow what’s going on in a logic circuit, especially larger scale ones like multipliers.

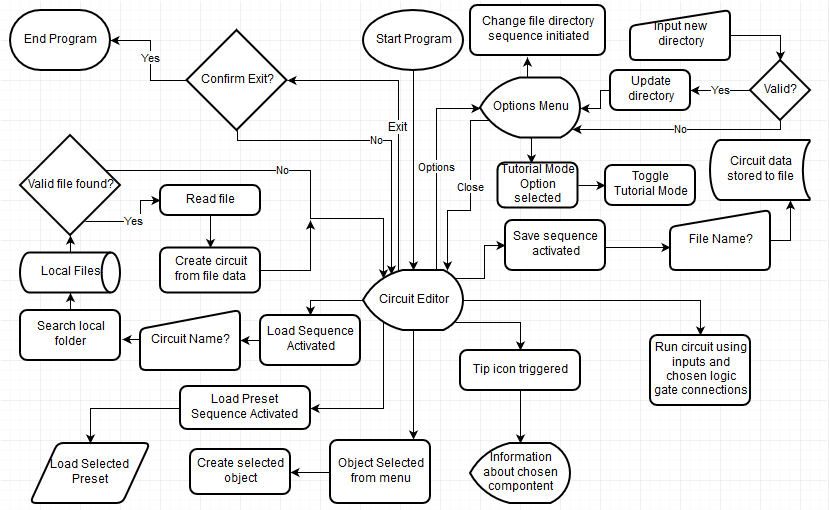
## Program Functions

In order to get a better understanding with what I want to integrate into this program I created a flow chart of its intended functions.



Key:

This flow chart shows most of the functions that I intend to integrate into the program.



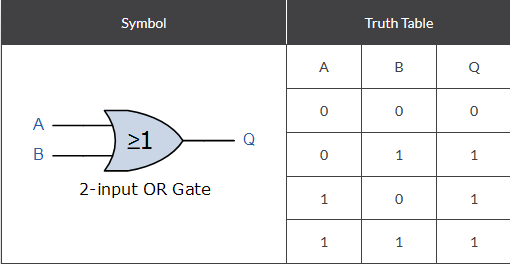
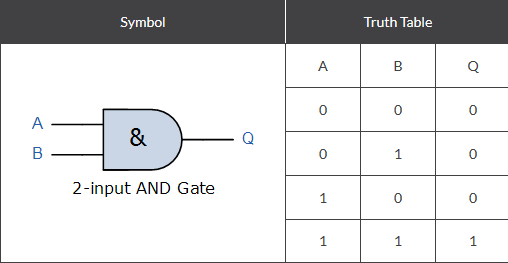
The program may not end up including all of these functions, but it should include a vast majority. I’m still not entirely sure if a tutorial mode will be a necessary feature, as the program may well be easy to understand on its own, along with object hints. An options menu may also not be necessary, as extra functions of the program such as saving and loading may be able to fit on the menu.

## Class Functions

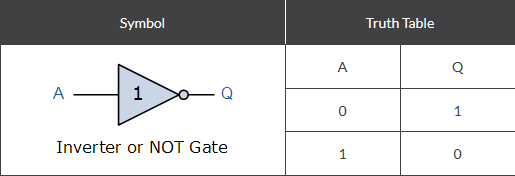
|  |  |
| --- | --- |
| Menu Class | |
| **Initial** | * Creates empty lists for holding each type of menu object. * Appends images to the self.decal list, then appends each decal to the object list along with a rectangle drawn with its resolution. These objects include the buttons, logic gates, the inputs and output object and item hints. |
| **DrawItems** | Sets the positions for buttons and input/output objects. For loop automatically sets positions based on the self.gap variable determined in the initialisation. If statement handles which circuit hint is shown based on what the user has clicked. Objects are then “blitted” onto the editor for the user to see. This Function is run every frame. |
| Source Class (Input) | |
| **Initial** | * State is set to 0, facade is set based on the objects initial state and “skin” variables. Skin variable was intended for different variants for the object but was not implemented. * Main mesh rectangle is created using the resolution of the decal using get\_rect(). * Coordinate variable is set to its predetermined startx, starty values. Handling variables for wires and dragging are created and set to their default state. |
| **DrawMesh** | Four more rectangles are drawn around its connection points in its sides using the size and position of the main mesh. These are used for connecting wires. |
| **UpdateFacade** | Decal is updated to match the object’s state value (for when the user toggles it). This function is run every frame. |
| Output Class | |
| **Initial** | * State is set to 0, facade is set based on the objects initial state and “skin” variables. Skin variable was intended for different variants for the object but was not implemented. * Main mesh rectangle is created using the resolution of the decal using get\_rect(). * Coordinate variable is set to its predetermined startx, starty values. Handling variables for wires and dragging are created and set to their default state. |
| **DrawMesh** | Four more rectangles are drawn around its connection points in its sides using the size and position of the main mesh. These are used for connecting wires. |
| **inRead** | Reads the state value of the wire it is connected to and sets its own value to it. |
| **UpdateFacade** | Decal is updated to match the object’s state value (for when the user toggles it). This function is run every frame. |
| Gate Class | |
| Initial | * Type of gate is predefined in parsing. * Input values are set to 0 by default. * Façade is set based on the self.name string, mesh rectangle is then drawn with the decal. * Starting coordinates are predefined based on its location on the menu. * Handling (drag) variable is set to None. “inform” connections are set to None initially. |
| ioCycle | * Determines output value based on its input values. In1 and in2 values are set to the signal value of the wires connected to them. If there is no connection is stays as 0. The logic function is then called to determine the output value. * Finally, the updateFaçade function is called.   This function is run every frame. |
| DrawSubmesh | Rectangles are drawn around its connection points and are bound to the main mesh. They are used for connecting wires to the gate. |
| UpdateFaçade | Decal is updated to match the object’s state value (for when the user toggles it). This function is run every frame. |
| logic | * Depending on the type of gate, input values are funneled to the correct if statement and then calculated using the logic for that gate. Each possible combination of inputs is tested to assign the correct state value so that the graphic can be dynamic. * The functionally will finally return the calculated output value, which is handled by the ioCycle function. |
| Wire Class | |
| Initial | * When a wire is created it must be given values for its input connection (where it gets its signal from), its output connection (where it sends its signal to) and its start position. * If it is drawn from an input (which gives out a signal) its in connection is assigned to that input, whereas its output connection is set to None until it is attached to an input node. * If it is drawn from an input node then its output connection is assigned to that object and its input connection is set to None. * As indexes are used to transmit signals to and from objects, a variable is needed to clarify which object type this is an index for. This is done by the ioStatus variable. If it is True then it is connected to an input object. If it is False it is connected to an output object. If it is none it is connected to neither (only gates). * Colour value is set to grey (off) by default. * Position variable is used to remember which side of the input/output object a wire should be connected to. This is set to none if its only connections are with gates. * The fromsource variable handles the prevention of joining an input with an output, so that the ioStatus handler does not malfunction. It is set to True when the wire is connected to an input. |
| on | Wire signal is set to 1 and colour is set to blue. |
| off | Wire signal is set to 0 and colour is set to grey. |
| UpdateWire | * If the wire has been placed this function will first check the ioStatus handler, then will determine two sets of coordinates to draw the wire from depending on its saved indexes. These coordinates are pulled using the sub-meshes of the objects the wire is attached to, allowing the wire to be drawn specifically on the nodes of the objects. This function is run every frame, making the wire remained attached to its assigned objects while they are repositioned by the user. * Secondly, this function will check the ioStatus handler again to check whether the wire should be getting a signal from an Input or a logic gate. It then sets the wires signal to that object’s output signal. This happens whether the wire has been placed or not. * Lastly the wire is drawn with all the properties specified by the earlier lines of the function (with a width of 5 pixels)\*   This function is run every frame. |

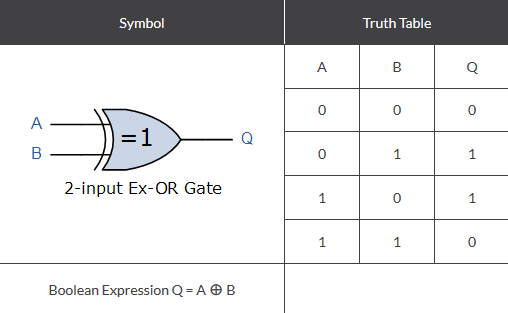
## Gate Logic

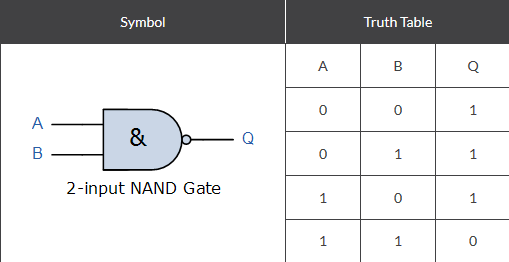
As the program will let users create and simulate logic circuits, it will need to functions to compare inputs values to determine an output value based on the type of gate. Truth tables will help a lot when programming functions for this.

****The **AND** gate simply multiplies the two input values together. Since multiplying by zero will always give zero, the only combination for an output of 1 is both inputs being 1.

The **OR** gate adds bits together. The only way it can output zero is if it adds two zeros, therefore both inputs would have to be 0.

****The **NOT** gate only takes one input. Its output value is the inverse of its input value.

The **XOR** gate (**Exclusive** **OR**) compares its two inputs. If there is a difference the output will be 1, otherwise it will be 0. When multiple bits are used in an input port, XORing with 0 will return the same string of bits, however XORing with 1 will return a string of bits inverse to the inputted string.

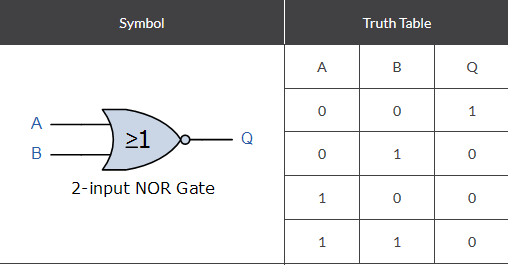
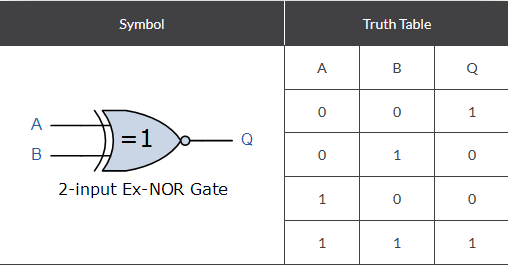


The **NAND** gate works the same as the AND gate, except its output is inversed. It is the equivalent of using a NOT gate on the output of an AND gate.

The **NOR** gate is similar to the NAND gate, except it is the inverse of the OR gate.

The **XNOR** gate is the inverse of the XOR gate. It compares its input values, if they are identical it will output a value of 1, if there is a difference it will output 0.

Each gate will be contained in a class, with each type needing its own function within said class. This will be essential for simulating logic circuits within the program. The code for the logic gate class functions will imitate these truth tables, they will also be used for the items hints for logic gates to tell the user how each gate functions.



Images from ElectronicsTutorials

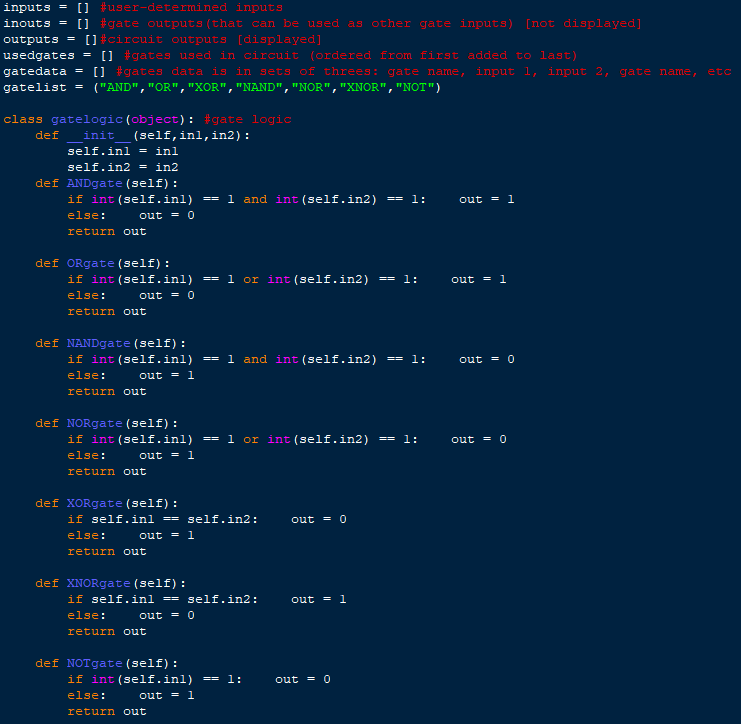
Source Material - https://www.electronics-tutorials.ws/boolean/bool\_7.html

# Technical Solution

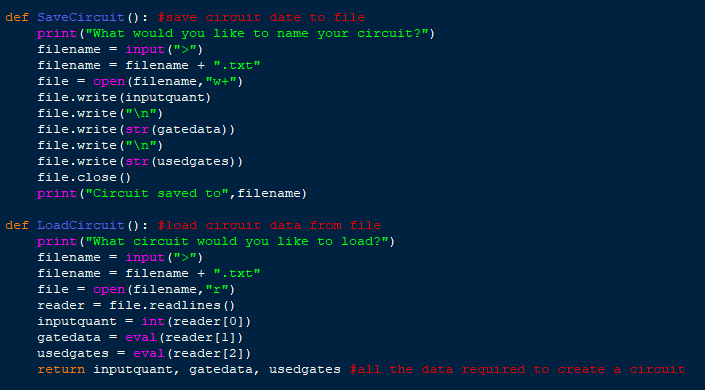
Before writing up the actual program, I programmed a test version of a text-based logic gate builder/simulator. This was to give me a gasp on how the interactive program should be coded before I made a start on it.

## Prototype: Text-based Logic Gate Simulator

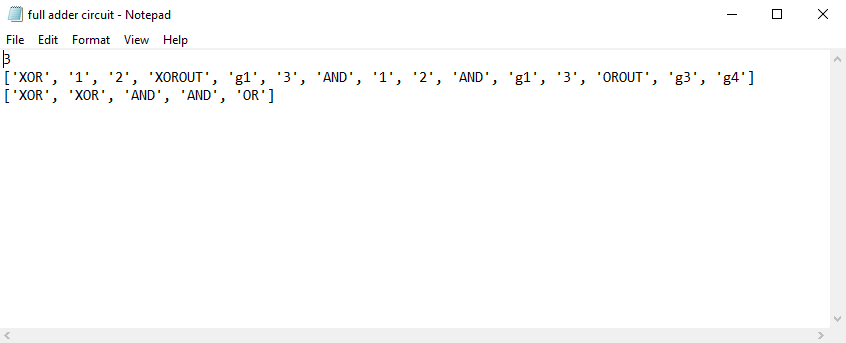
The text-based version had its gates as functions stored in a class, which were called whenever a gate needed to be read when the program was run. Most of the data is stored as strings and integers inside lists, such as the data used for each gate to function (the type of gate it is and where its input values are coming from), values being treated as the output of a circuit, etc. Both of those can be seen here:



This program also has the functionality to save and load circuits. It does this by saving and loading the data from the inputquant, gatedata and usedgates with text files. These values are all that are needed to create a circuit. This is the code used for saving and loading:

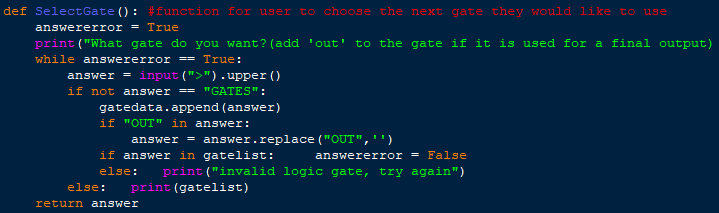


The data is saved to and loaded from a text file. This is an example of what the data looks like:

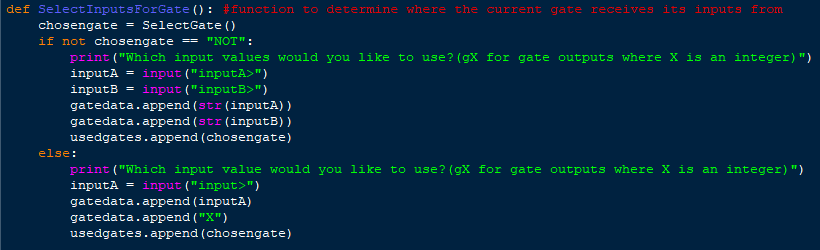


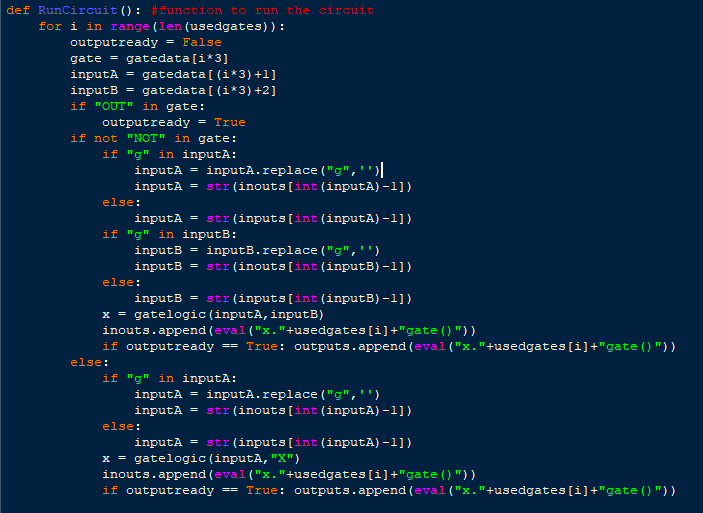
The top value is the number of inputs in the circuit, line 2 is gate data (each gate uses three values: gate property, first input location, second input location), line three is the types of gates used listed in order.

When creating a circuit in the program, the user is prompted to define the properties of each gate the would like to use. First prompt is the type of gate, “out” should be added to the end of the type of gate if the output value of the gate is desired to be an output of the circuit.

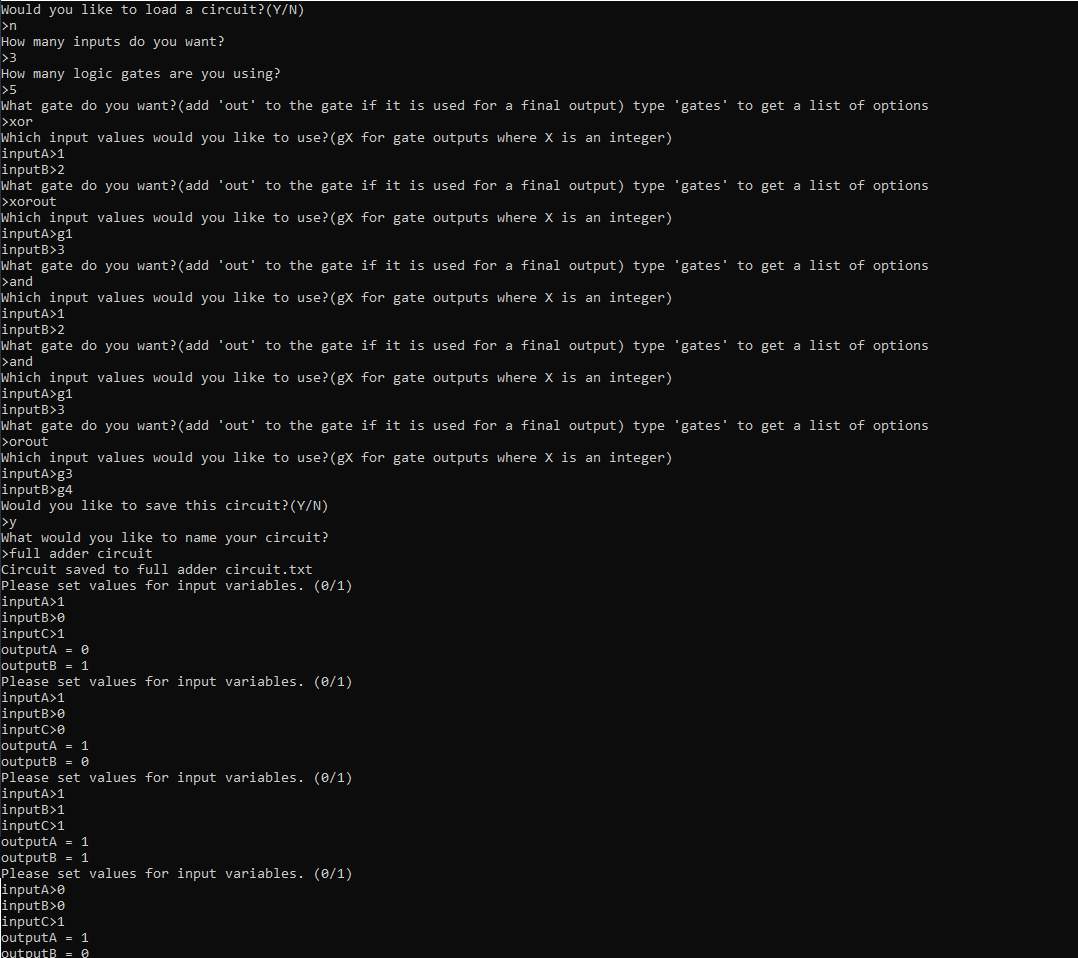


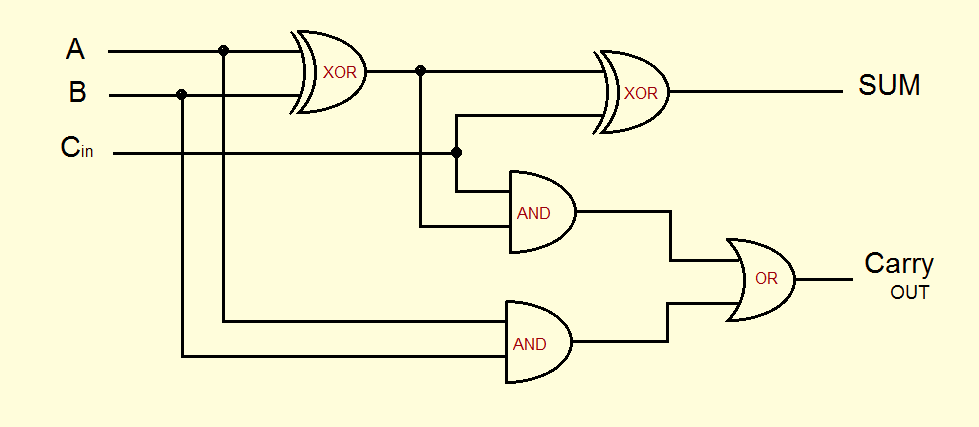
After the type of gate is defined, the input locations are prompted from the user. A normal integer is used for inputs of the circuit, for gate outputs an integer with a “g” infront is used (the integer represents which gate is used, in terms of the order they were created in).



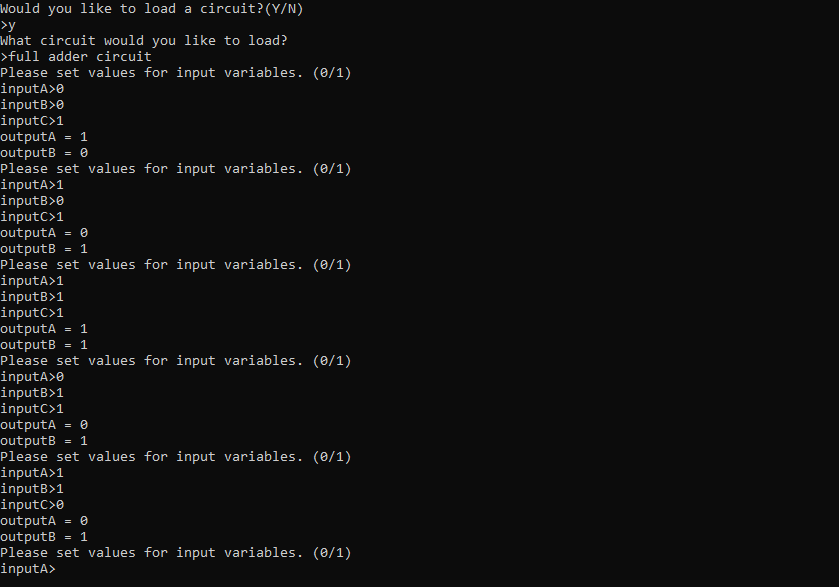
This process is repeated for every gate the user inputted to use. After that the circuit has been completed, and is ready for running. The user is prompted to determine the values for each input to the circuit (0/1). Afterwards the circuit will run with the function RunCircuit() and will display the outputs to the circuit when it has finished. This is the code for running the circuit:

Example of creating a full adder circuit with the program:



**Reference diagram**

Example of utilising the LoadCircuit functionality:



**Advantages of the text-based program:**

* Functional: Can be used to create and simulate logic gates that have 26 inputs/outputs or less (since they are assigned a letter)
* Useful: If the program were to come with premade circuits that are relevant to their course (such as adder circuits and multiplier circuits), it could prove useful as it could easily be setup to simulate those circuits which could help for revision.
* Convenient: As it only uses raw python, it can be run on any computer with an interpreter using python3.

**Disadvantages of the text-based program:**

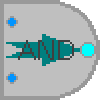
* Difficult: When allowing students to test this program, they reported that it was unclear and confusing when trying to use it. I will make sure to eliminate this issue in the interface program by making it simple and easy to use, along with a tutorial if users decide they need it.
* Inconvenient: When making a large circuit, even if you are looking at a diagram it can be hard to remember which gate is assigned to which number. For instance, if there are 10 gates and you want to use the 6th one you may accidently assign the wrong gate. This shouldn’t be a problem on the interface version, as circuits will be purely visual.
* Inconvenient: Although the program allows the user to keep re-inputting the input variables of the circuit, it can be annoying to have to enter each one individually every time if only small changes wish to be made to the set of inputs. In the interactive version the circuit inputs will be able to be toggled on and off in real-time, eliminating this issue.

## Main Program: Interactive Logic Gate Simulator & Revision Tool

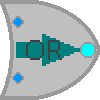
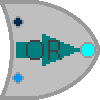
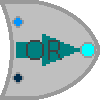
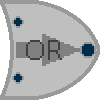
As the logic gates are now being portrayed on an interface, I needed to create images for each object that could be used in a circuit. Each object needed a different image for each possible state. I created the images using Paint.net. All gates are 100 by 100 pixels.

Logic Gates have four possible states (excluding the NOT gate):

The two lights on the left represent the gate’s input values.

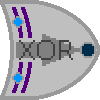
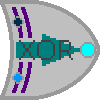
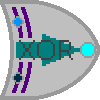
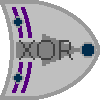


The larger light on the right represents the gate’s output value.

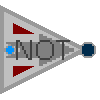
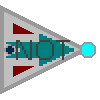


Centre arrow will glow when the gate is outputting a signal.

Models are based off each gate’s circuit diagram symbol.

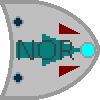


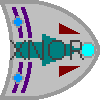
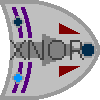
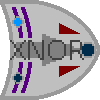
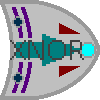
Purple lines indicate an exclusive gate. Normally on circuit diagrams an extra curve is on the outside of the symbol, but I wanted all the gates to be the same size.



Red streaks indicate inverter gate. Reasoning is the same as the exclusive indicator. I did not want to change the size of the model.







Other images used the program:





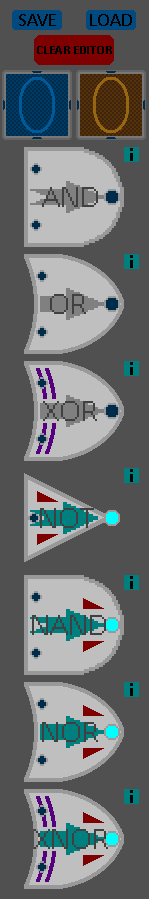




Toggle Inputs and Circuit Outputs have two possible states. The inputs will be able to be toggled with a right-click.





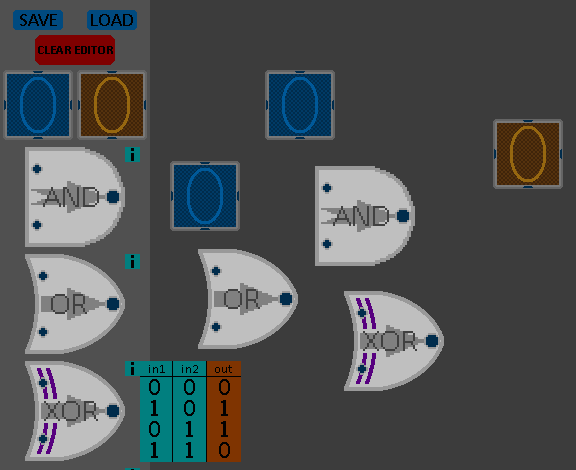
This is the editor menu (seen on the right), which is located on the side of the application. Items are dragged from the side onto the editor for use.

The save button will prompt the user for a name to give the save file, and then save the data of the circuit on the board to a text file with the given name. If for some reason the program is unable to save the circuit, it will return “Save Failed”, although all possible circuits should be saveable.

The load button will prompt the user to give a name of a file to load. If the file with the given name is not found or the file loaded is not valid the program will return “Load Failed”. If a valid file is selected the circuit will be loaded onto the program, which will return “Load Successful”. Clicking the load button will clear the editor regardless of whether the load was successful or not. Pre-made circuits, such as a 2-bit full adder and a 2-bit multiplier, come with the program and can be loaded with this function.

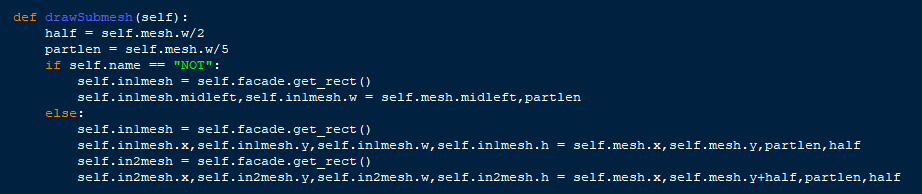
The information icons next to the gates will provide a truth table for the gate it is next to when clicked by the user. This can be seen next to the XOR gate below.

**Example of objects that were created from the menu:**



As each object has its own functions and needed the ability to be reproduced from the menu, they are made with classes. There’s a class for each type of object used in the program (gates, inputs, wires, etc). The class has everything needed for the object to function properly, such as its state and its connections, along with its class functions. There is a list for each type of object, and the menu creates instances of each object by appending the object’s class to the corresponding list.

To be able to draw the decal of the object in place and detect when the user is trying to drag the object around, an invisible rectangle or “mesh” is used. The size and position of this mesh is stored in its class. Sub-meshes are drawn in class functions to define where on the object the user needs to click to create and attach wires from it. Example of the sub-mesh function for the logic gate class:

As each gate (other than the NOT gate) has two inputs and one output, a sub-mesh needs to be created on each relevant section of the gate for these ports.

The sub-mesh is only used for creating and attaching wires, and moves with the main mesh.

## Visual representation of Mesh and Sub-Mesh



**Purple: Main Mesh**

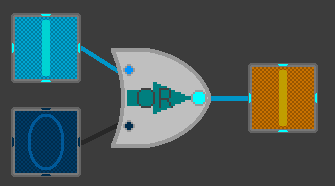
**Blue: Input Sub-Mesh**

**Orange: Output Sub-Mesh**



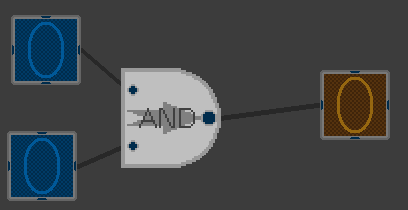
Input meshes receive wire signals and output meshes determine wire signals. Main meshes are used for dragging the object around. If the cursor is clicked on a sub-mesh a wire will be drawn from the centre of that sub-mesh if possible. Infinite wires can come from an output but only one wire can be attached to an input. If the cursor is clicked (and held) on the main mesh only the object will be dragged. Objects can be dragged while wires are attached to them.

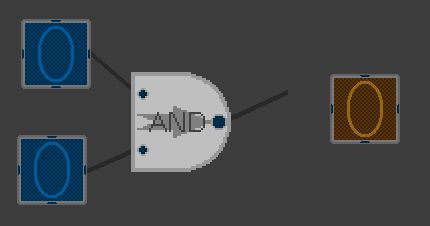
**Example of a connected logic circuit:**

Gates calculate whether they should output a signal or not based on their input signals using their ioCycle class function, which is executed for each frame of the program. This allows the gates to function based on the user-decided states of the connected circuit inputs in real-time.

## Drawing and Connecting Wires

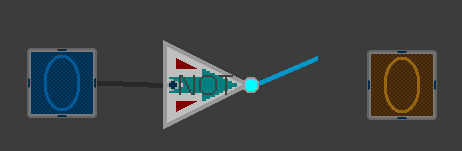
To draw a wire in the program the user must click on a valid sub-mesh that can support a new wire to be attached to it. If valid, a wire will appear. It will be attached to the object clicked and the mouse cursor. The wire should then be attached to a sub-mesh of opposite polarity to the first one selected, for example a gate output should be connected to a circuit output since it is supposed to receive a signal. It cannot be connected to another gate output. If the user clicks on a valid sub-mesh to place the other end of the wire, it will join to that sub-mesh and stay connected. This can be seen in the screenshots below.





Wires can be drawn from both input meshes and output meshes. Gates cannot be connected to themselves and circuit inputs cannot be connected to circuit outputs.

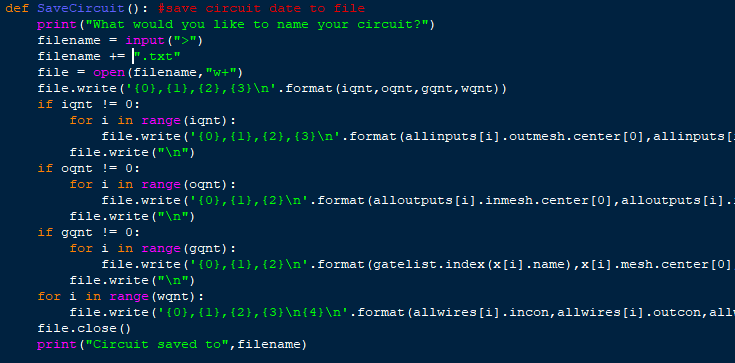
Wires that are being drawn can be cancelled by either right-clicking or by hitting the escape key. Wires will glow light blue if they have an active signal. It will also glow light blue if the wire is being drawn from an active signal output as it is already reviving a signal. This can be seen from the image below.

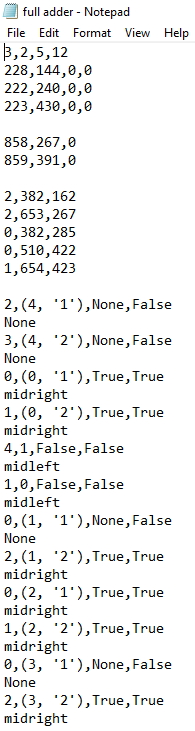


Unlimited wires can come from a signal output, but only one can enter a signal input. Circuit outputs can also only have one wire connected to it.

Wires will continue to stay connected to objects while they are moved around the map. They will always be layered underneath the other objects on the map, so if there are a large amount of wires being used they will not obstruct any of the circuit components.

## Save function

The save function writes all the information needed to re-create a logic circuit to a text file. This information can include position, name, wire connections, etc.

This is an example of what a save file looks like. Each object type is separated by a break.

* The first line is the quantities of each object type (inputs, outputs, gates, wires). The program needs this information to know how many lines to read for each object type when loading from the file.
* The first data block after line 0 is for circuit inputs. Each line is for one object. The first two values are co-ordinates to mark their position. The other two is data for the appearance of the input.
* The second data block is for circuit outputs. The values are similar to input objects, except they do not have data of whether they were on or off, since that will be determined from its connections when the circuit is loaded again.
* The third data block is for gates. The first value represents the gate they are, the number being the index in the list of possible gates in the program. The other two values are co-ordinates.
* The fourth data block is for wires. Each wire uses two lines. The first value represents the object it’s getting its signal from. The second item and represents the object it’s transferring its signal to. It only uses this information to determine where the end of it should be attached to. If the object is a list the wire is feeding into a gate, else it is feeding into a circuit output. The first object of the list represents the index of the list of created gates from which it attached to. The second is the specific input it is attached to, ‘1’ is input 1, ‘2’ is input 2. If it is a single value then the object is an output and the number represents the index for the output it is connected to. The third item is information on whether it’s connected to a circuit input, circuit output or neither (True = input, False = output, None = neither). A circuit input cannot connect to a circuit output. The fourth item is information on whether it’s be getting its signal from a circuit input or not. The fifth item (on the second line) is the side on the circuit input that the wire is coming out from. This will be None if the wire is not coming from a circuit input.

With this information the load function can recreate the circuit that was saved

# System Testing

In order to test the effectiveness of the program I will be creating and test running various logic circuits of varying complexity by changing the inputs. If the circuit provides expected results then the test will have been successful. I will also be testing the save, load and clear options in a utility test. I will record two videos of the program, one for a demonstration of the program’s functions and one for logic gate testing. These videos will be accessible via QR codes that will be included at the end of the tests.

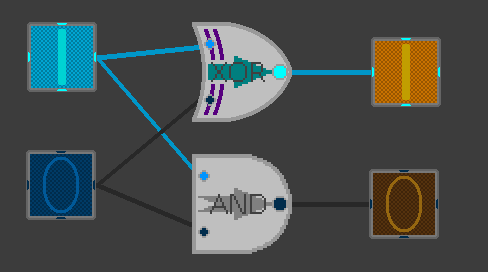
## Constructing and Simulating Logic Circuits

### Half Adder

Binary: 0 + 0 = 0 carry 0

Denary: 0 + 0 = 0

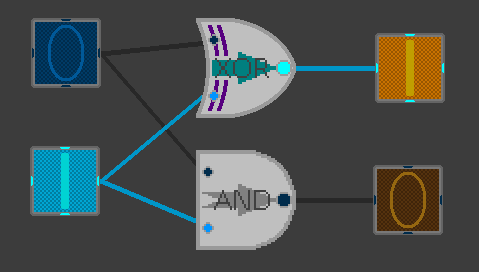
Expected Result.



Binary: 1 + 0 = 1 carry 0

Denary: 1 + 0 = 1

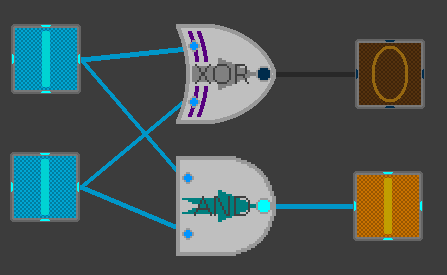
Expected Result.



Binary: 0 + 1 = 1 carry 0

Denary: 0 + 1 = 1

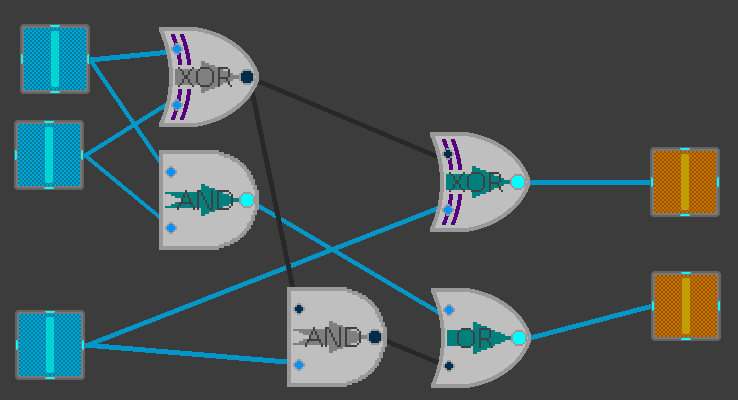
Expected Result.

Binary: 1 + 1 = 0 carry 1

Denary: 1 + 1 = 2

Expected Result.

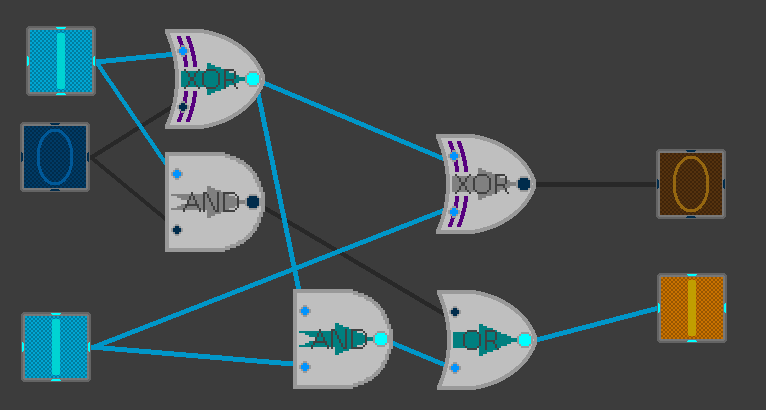
### Full Adder



Binary: 1+1+1(carry) = 1 carry 1

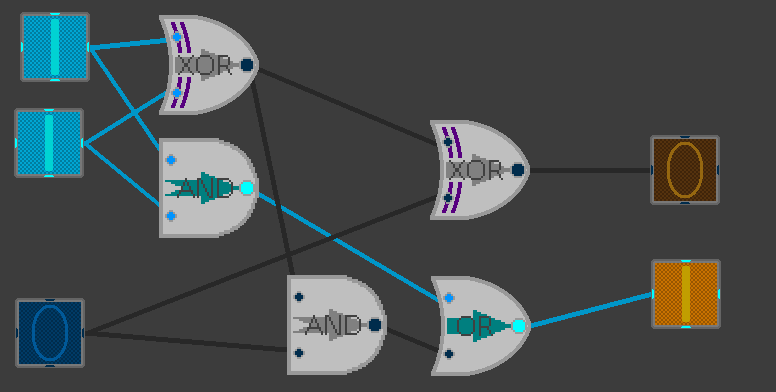
Denary: 1+1+1 = 3

Expected Result

Binary: 1+0+1(carry) = 0 carry 1

Denary: 1+0+1 = 2

Expected Result



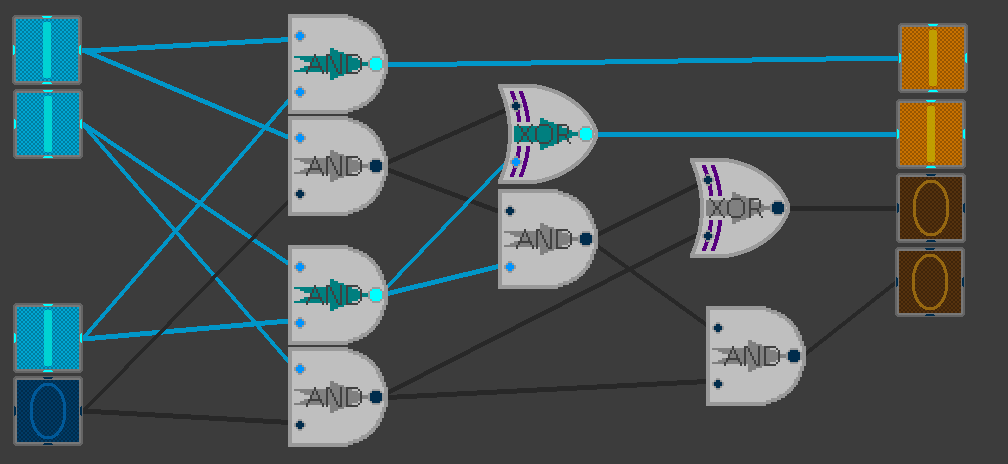
Binary: 1+1+0(carry) = 0 carry 1

Denary: 1+1+0 = 2

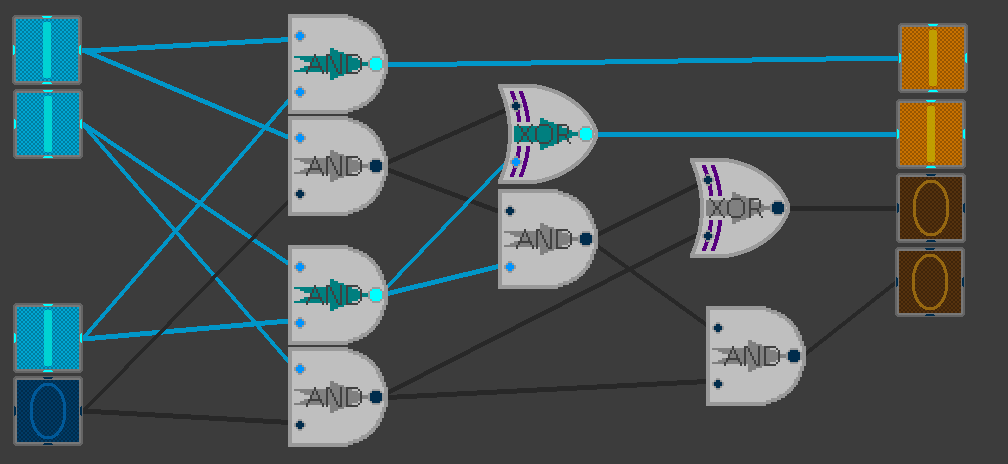
Expected Result

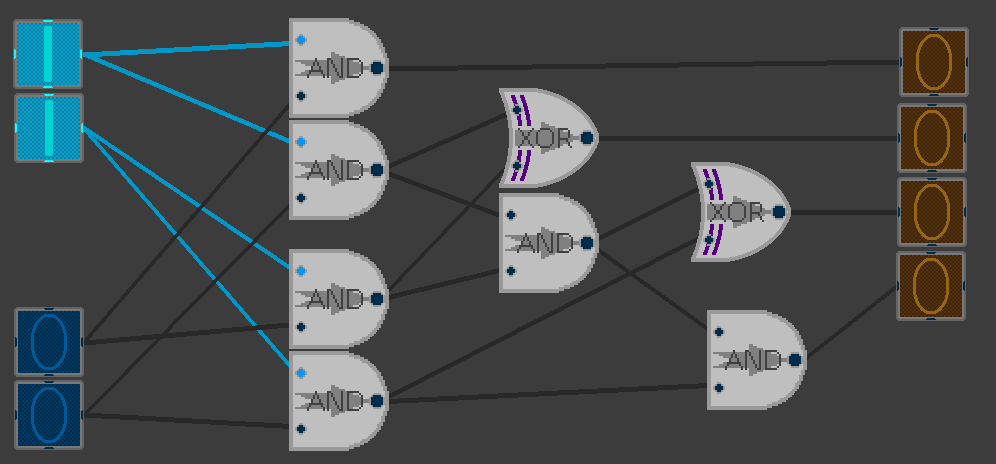
### 2-Bit Multiplier

Binary: 10 . 11 = 0110 Denary: 2 x 3 = 6 Status: Correct

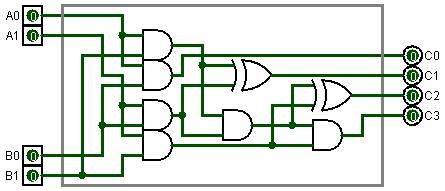
Binary: 11 . 11 = 1001 Denary: 3 x 3 = 9 Status: Correct

Binary: 11 . 10 = 1100 Denary: 3 x 2 = 6 Status: Correct

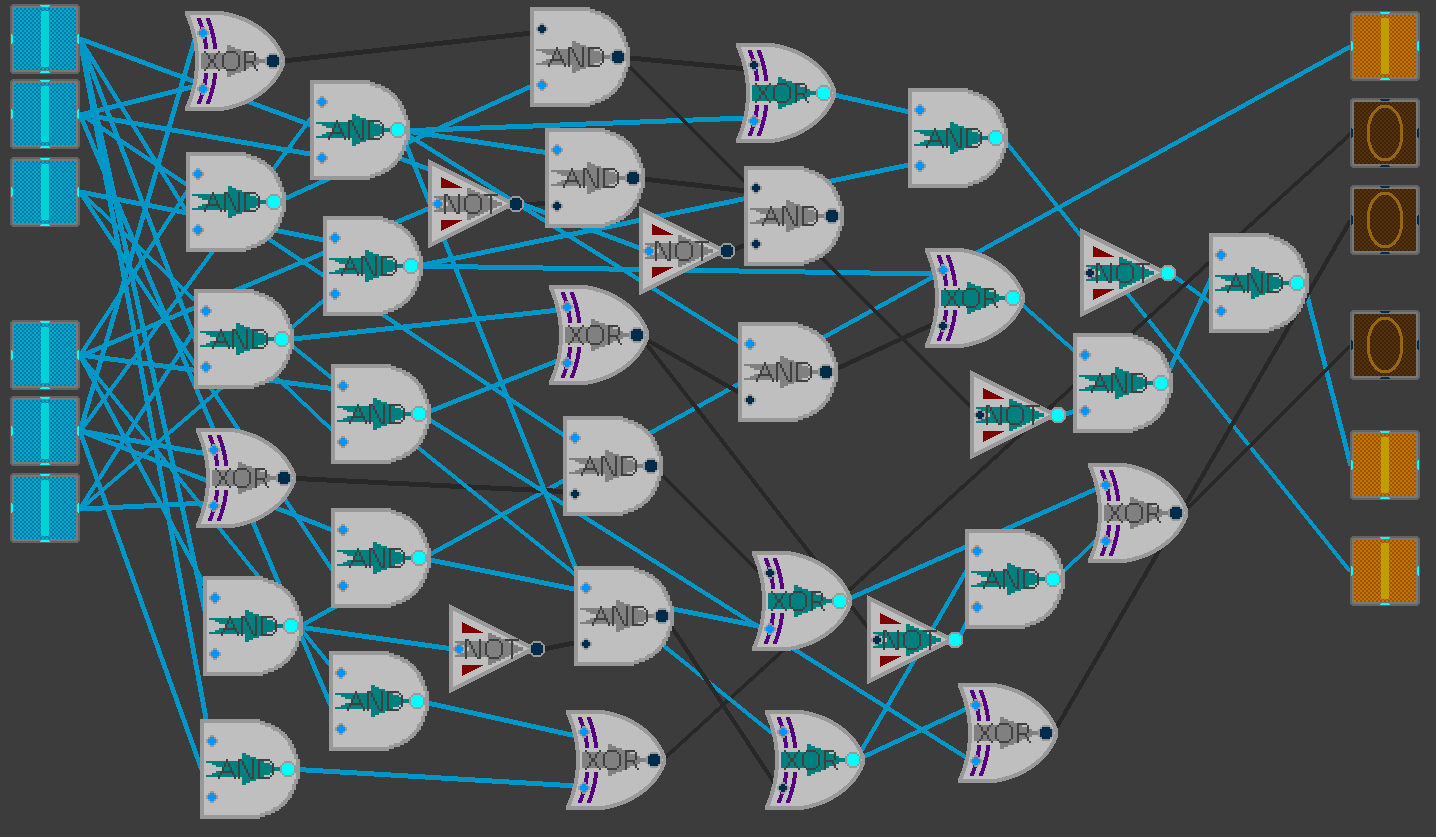
2-Bit Multiplier (continued)

Binary: 11 . 01 = 0011 Denary: 3 x 1 = 3 Status: Correct

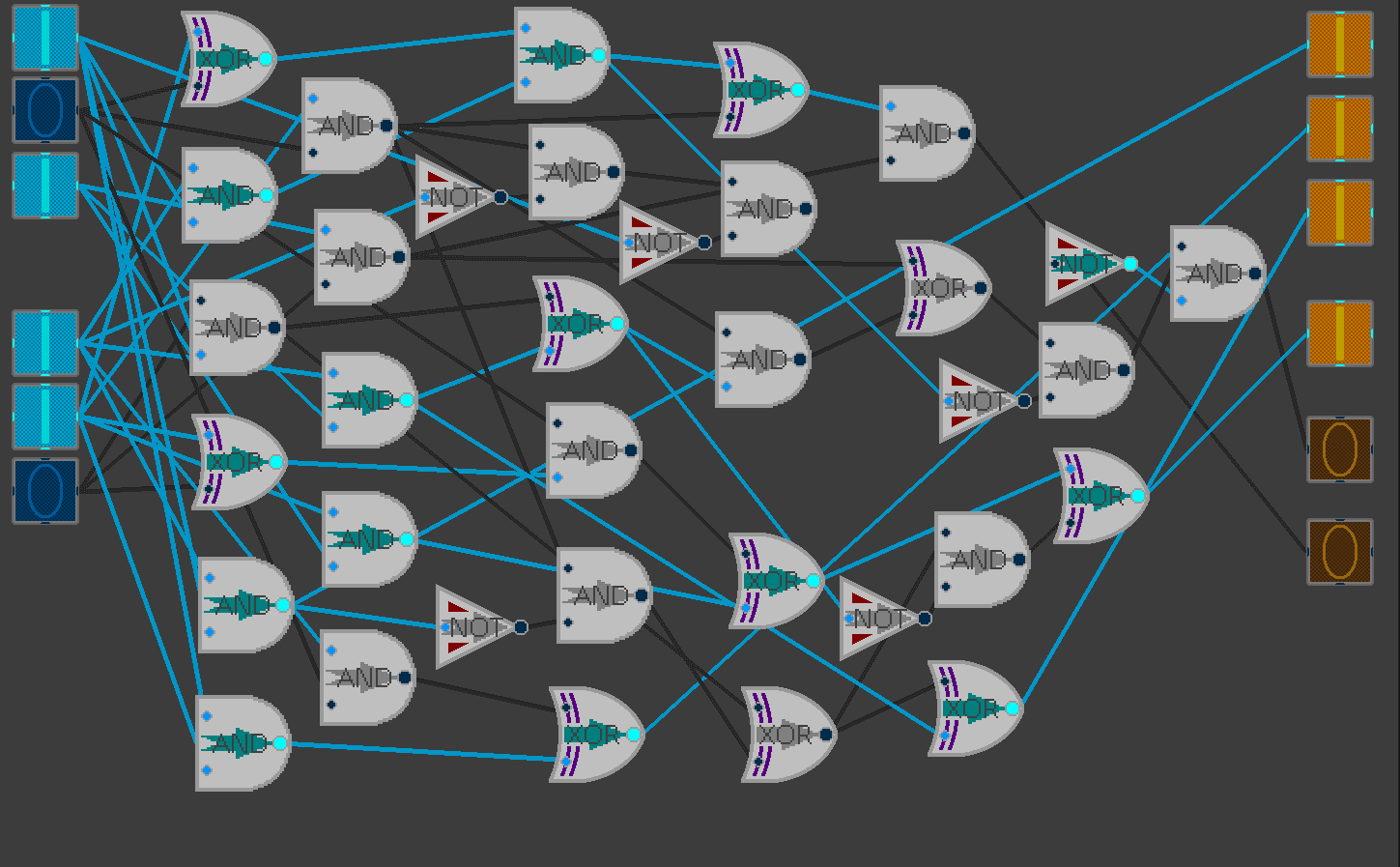
Binary: 11 . 00 = 0000 Denary: 3 x 0 = 0 Status: Correct

**Reference diagram**

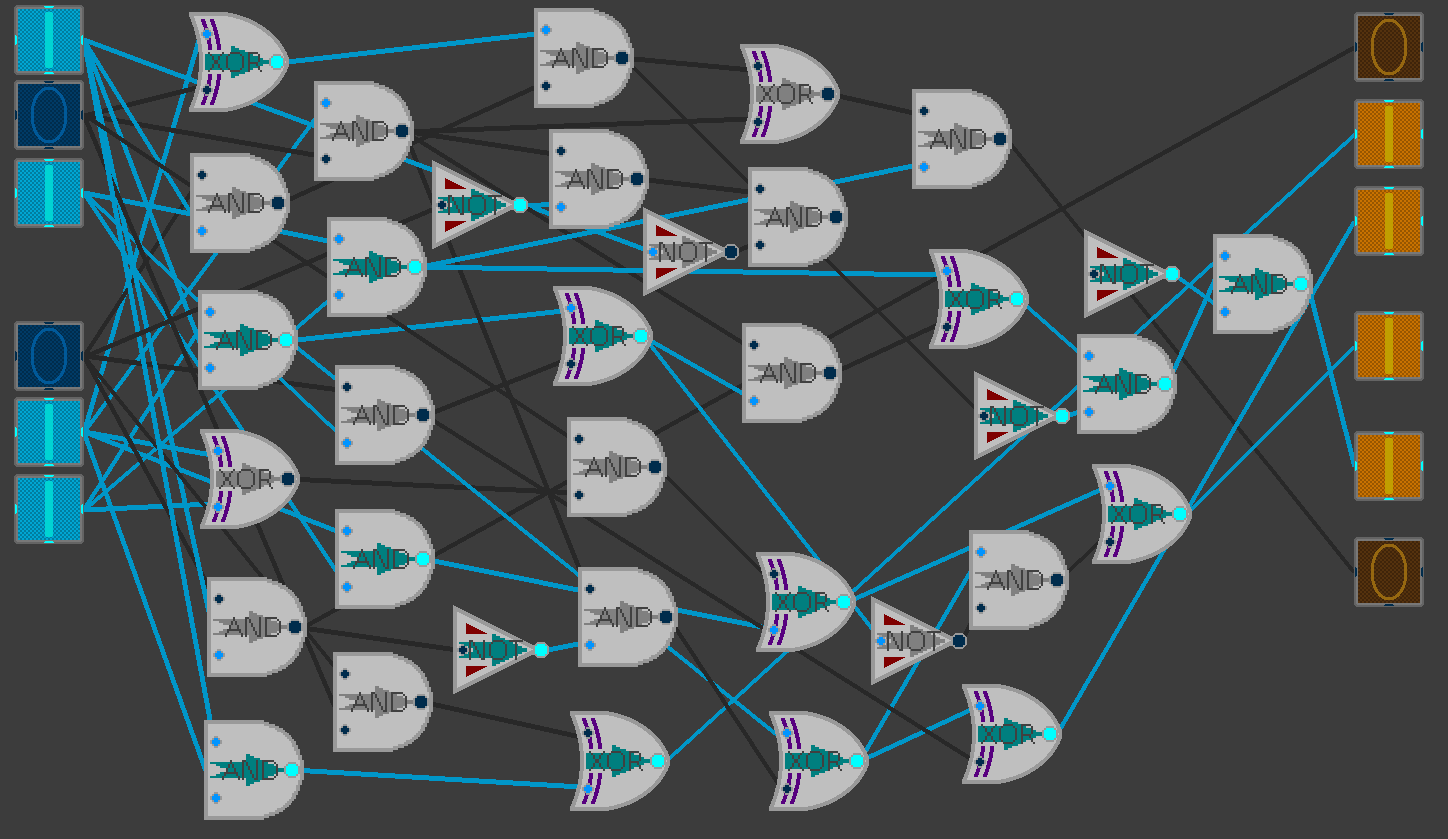
### 3-Bit Multiplier

Binary: 000 . 000 = 000000 Denary: 0 x 0 = 0 Status: Correct

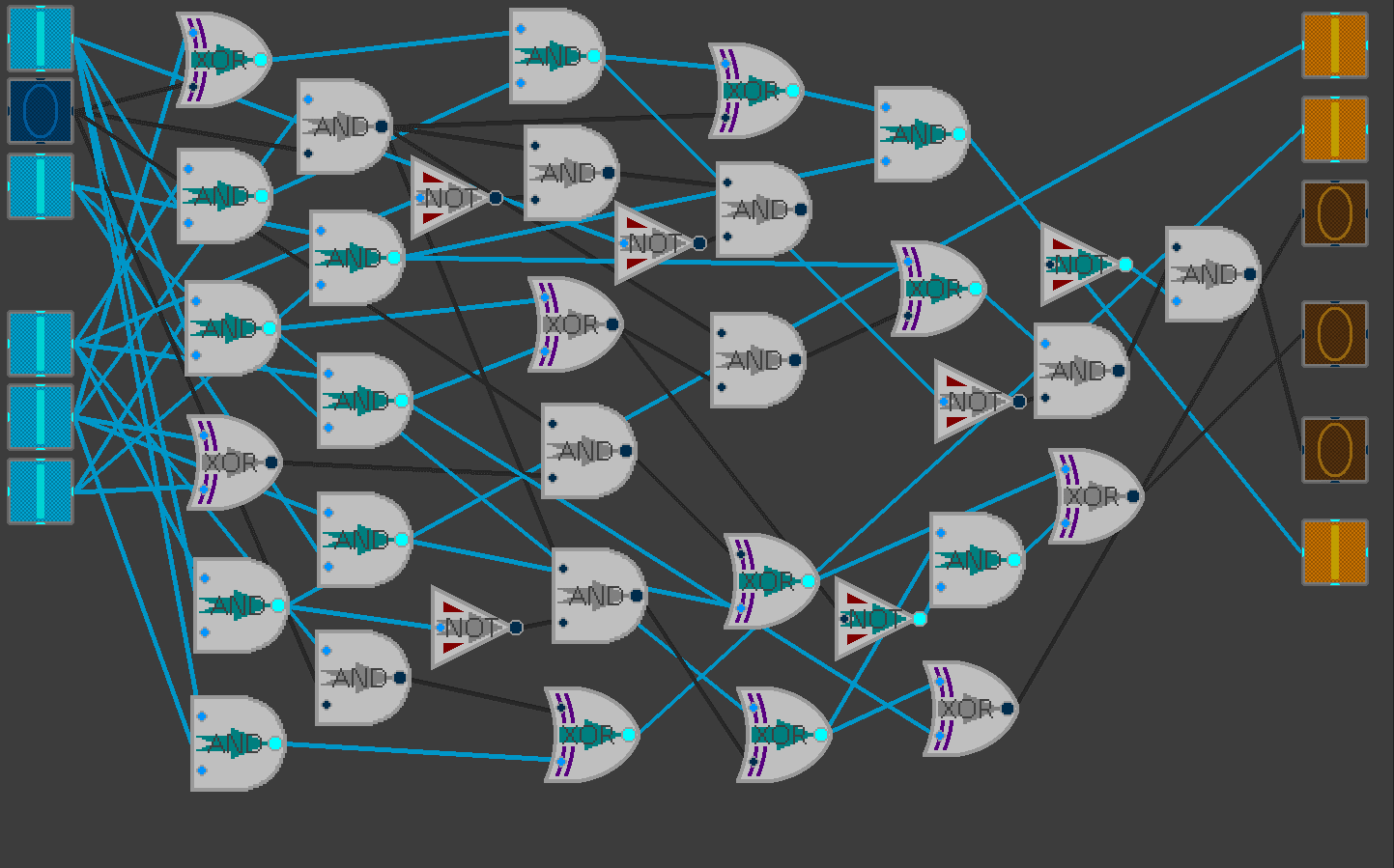
Binary: 111 . 111 = 110001 Denary: 7 x 7 = 49 Status: Correct

3-Bit Multiplier (continued)

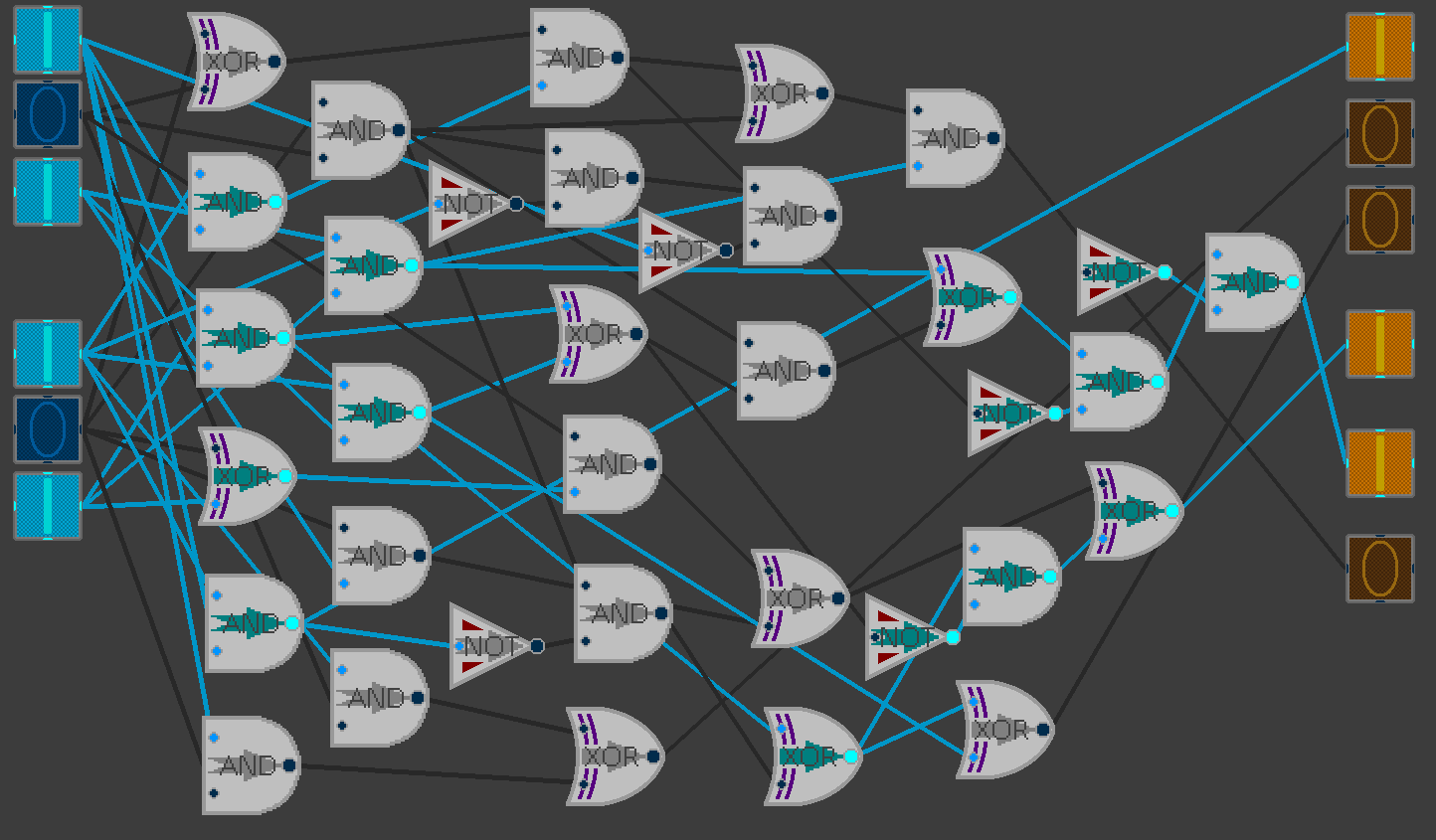
Binary: 101 . 011 = 001111 Denary: 5 x 3 = 15 Status: Correct



Binary: 101 . 110 = 011110 Denary: 5 x 6 = 30 Status: Correct

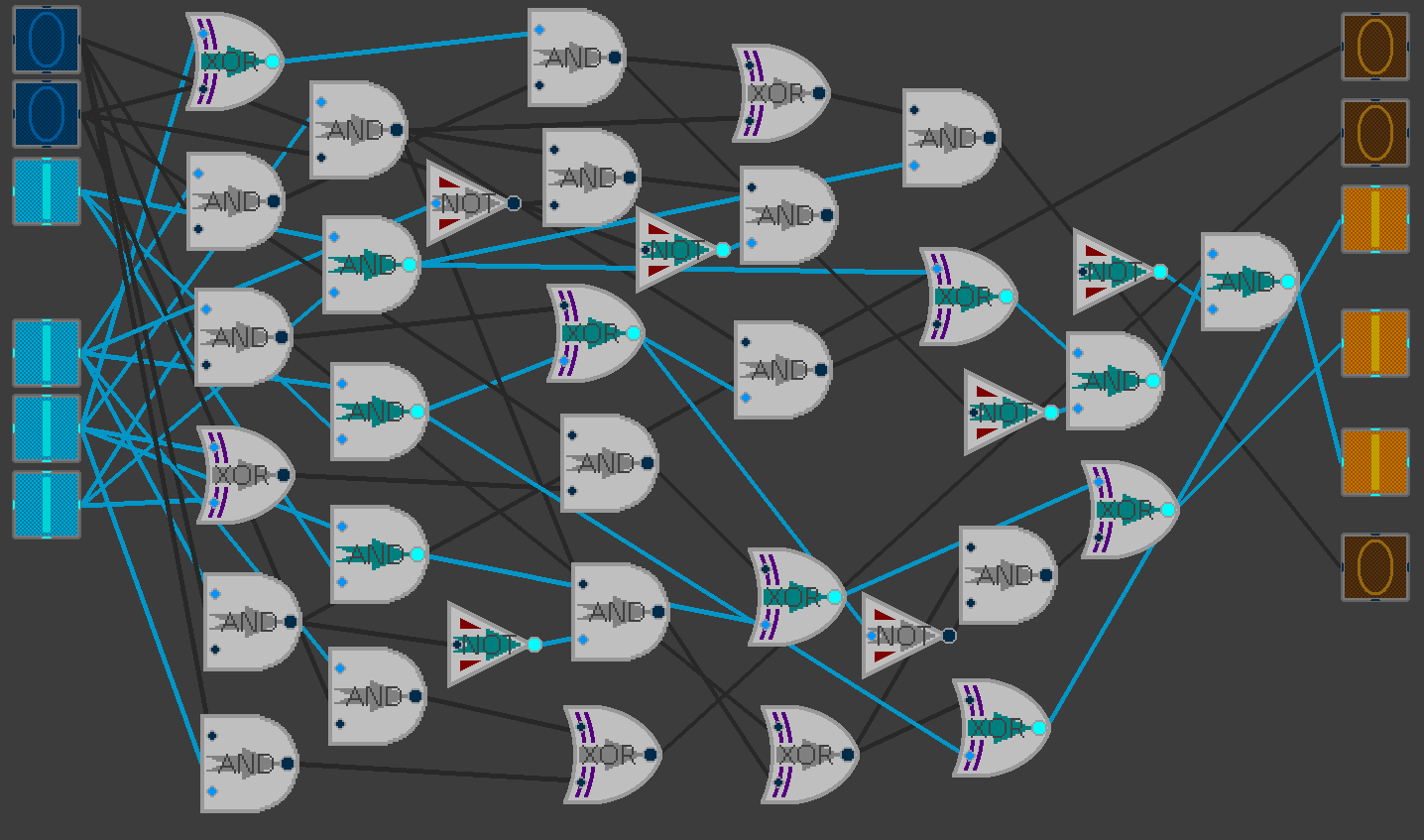
3-Bit Multiplier (continued)

Binary: 101 . 111 = 100011 Denary: 5 x 7 = 35 Status: Correct

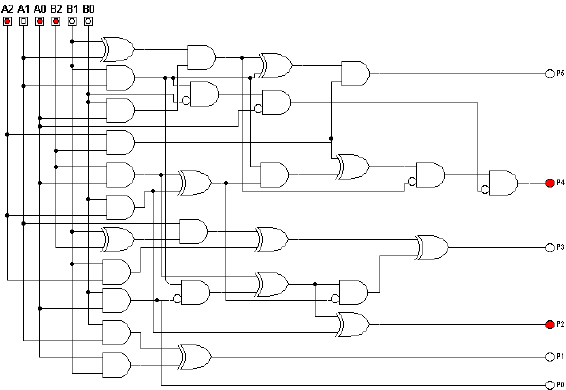


Binary: 101 . 101 = 011001 Denary: 5 x 5 = 25 Status: Correct

3-Bit Multiplier (continued)

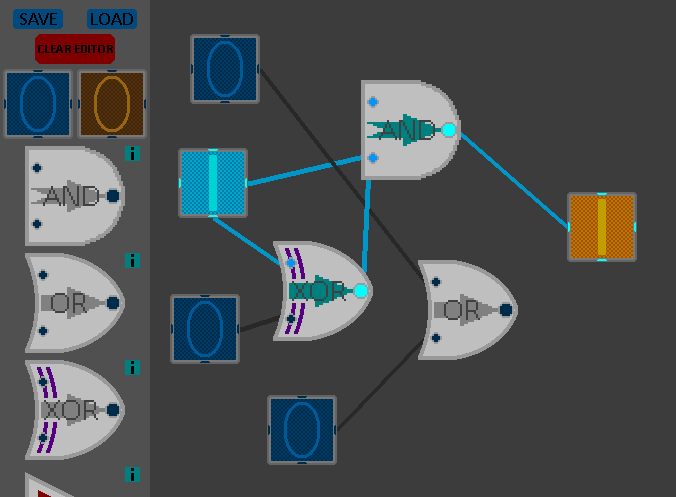


Binary: 100 . 111 = 011100 Denary: 4 x 7 = 28 Status: Correct

**Reference Diagram**

## Utility Testing

### Clear Editor

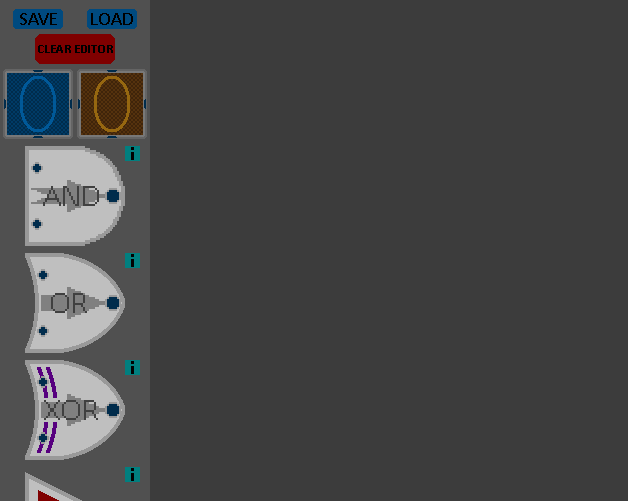


First screen capture: Before Clearing

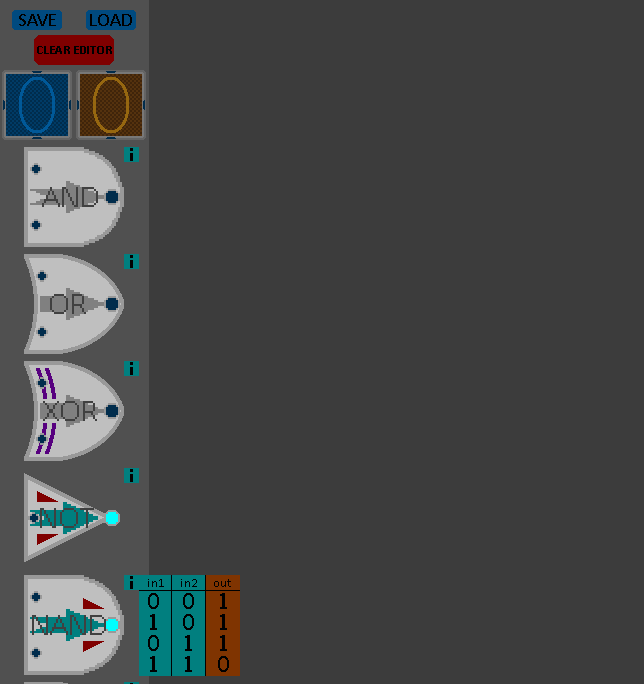
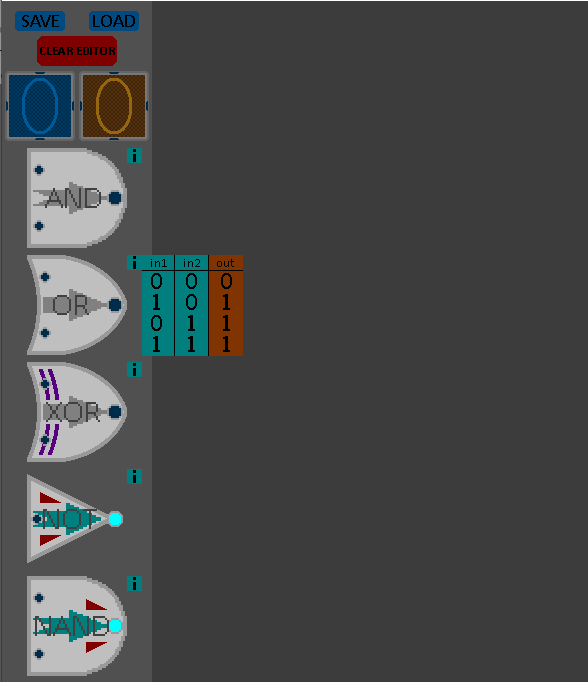
Second screen capture: After Clearing

**Expected Result, Functional.**

After clearing the editor, the user can start building a circuit again as normal.



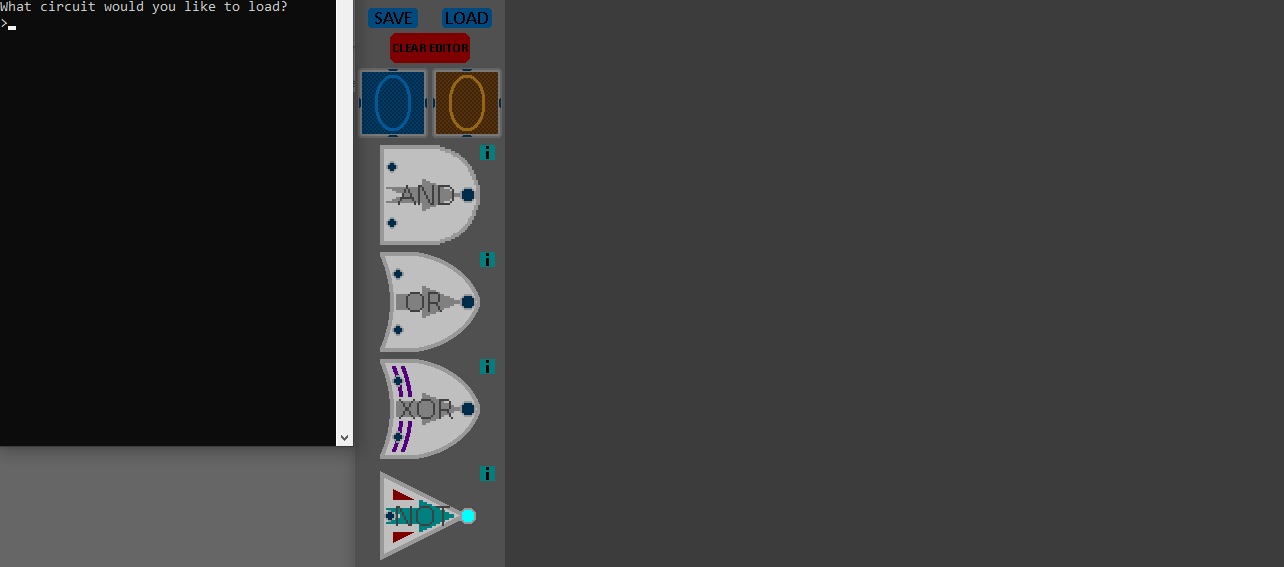
### Item Hints

****Actions in order: OR gate hint pressed, NAND gate hint pressed, NAND gate hint pressed.

**Expected Results, Functional.**

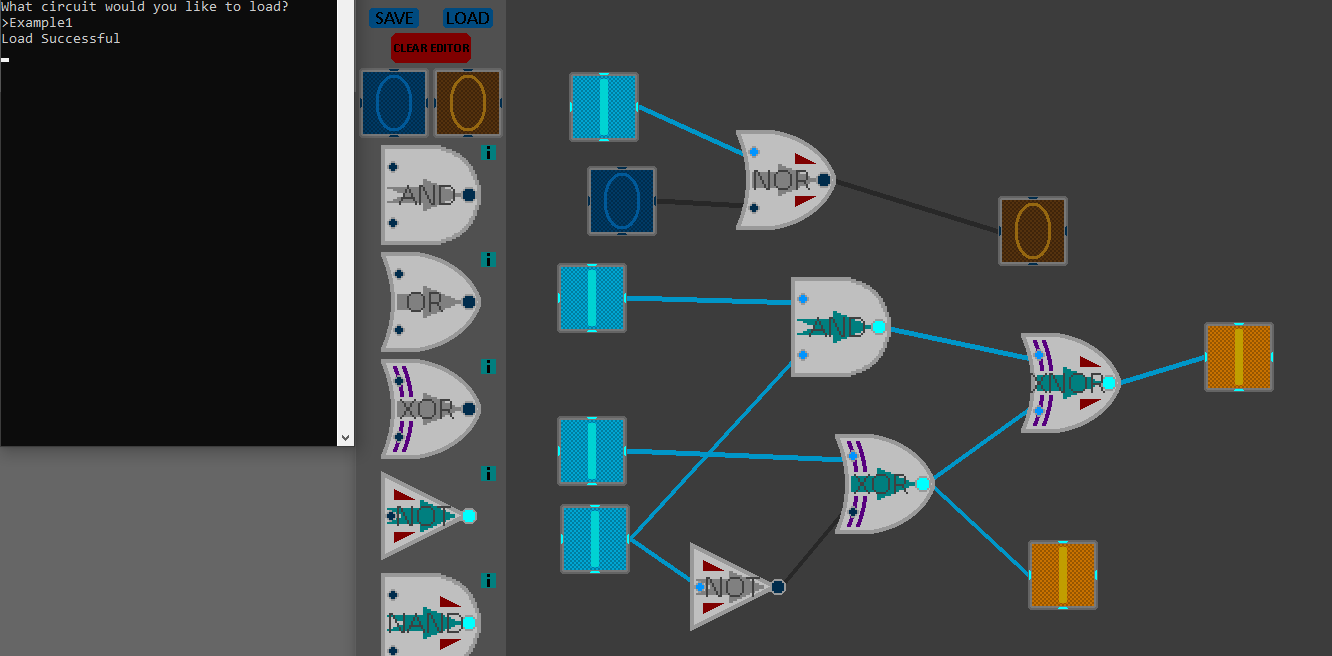
The item hints can be used anytime. Clicking a hint will show a truth table for the corresponding logic gate. Clicking the same hint will hide the truth table, whereas clicking a different hint will hide the first table and show the new one.

### Save to file

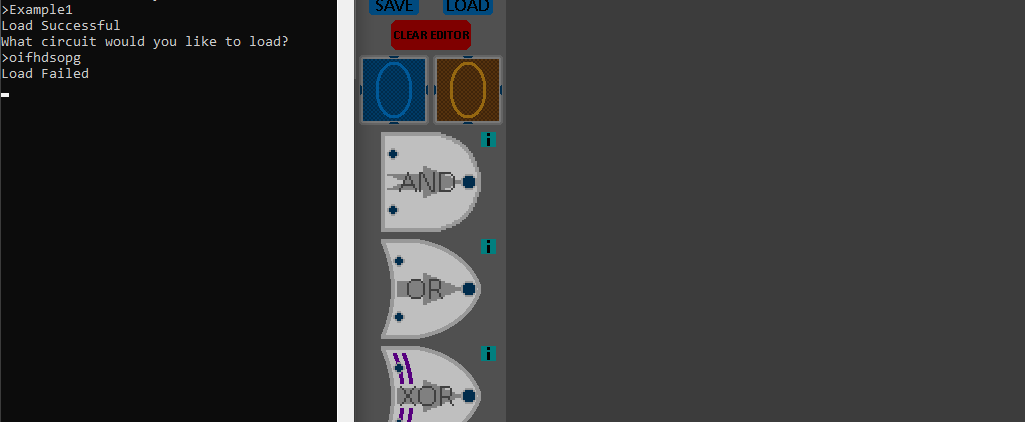
File saved with given name using the python console. Save file on the right.

### Load from file

Starting with a clear board.

****

Loading saved circuit from earlier by inputting the file name. Input states are saved (whether they’re on or off).

****

Example of trying to load a file that doesn’t exist. Same result will happen if a file is selected that cannot be mapped into a circuit for any reason.

Pre-set circuit files can be downloaded along with the app. Loading them with this function will map the premade circuit for the user.

**Save and Load functions work as intended.**

## Videos of the Logic Gate Simulator

### Demonstration of Program

I recorded a video on YouTube of me using and demonstrating the program. It can be accessed by using the QR code. It explains how to use the program, along with demonstrating the utilities available in the menu. [https://www.youtube.com/watch?v=mpJgVgraLPI]

0:03 – Creating items from the menu

0:16 – Connecting wires + Item dragging

0:30 – Wire Clearing

1:05 – Clear Editor

1:22 – Creating a Circuit

2:19 – Save Function

2:38 – Load Function

3:04 – Item hint demonstration

### Logic gate testing video

This video is of the logic circuit testing, which is what the screenshots from earlier were demonstrating. Video can be accessed using the QR code below. [https://www.youtube.com/watch?v=Z9cuewdKjoo]



0:06 – Half Adder

0:22 – Full Adder

0:42 – 2-Bit Multiplier

1:06 – 3-Bit Multiplier

# Evaluation

## Summary of Objectives met

**Objective 1: Able to create any possible logic circuit with the standard logic gates (2 inputs, AND, OR, NOT, XOR, NAND, NOR, XNOR).**

* The program enables the user to create any logic circuit they desire. As there is no limit to how many objects and connections can be made, the only limit is the user’s computer and the size of the board. However the gates can be stacked, therefore any circuit could be compressed. This objective has been achieved.

**Objective 2: Have a decently large library or pre-made circuits that are commonly used, such as a full adder circuit (with the ability to combine said circuit into an array, with hints on how to do so), bit multiplier circuits, etc.**

* Although the quantity of circuit presets could be larger, it comes with all the logic circuits students will have to know at A-level, excluding the flip-flop circuits as a clock object was not implemented into the program. These circuits can be accessed via the load function. This objective has been achieved.

**Objective 3: An option to enable a guide to teach the user how different logic components and systems work and are used, a feature intended for students who are newer to Computer Science.**

* Although I originally planned to have a “tutorial mode” or guide in the program to help out beginners, I did not follow through with it. After implementing the truth table hints I decided that the program was easy enough to follow on its own already, and so decided that a tutorial mode would be unnecessary and scrapped the idea.

This objective was not met.

**Objective 4: A hint icon on each item that can be triggered to give a quick summary on how it works/how it’s used.**

* As evidenced in the Utility testing section, hint icons were a success. Each logic gate has its own hint icon, and when clicked by the user will provide a truth table for that gate, showing how the gate processes its values to give an output. They are very effective at teaching/reminding the user how the logic gates work.

This objective has been achieved.

**Objective 5: A clean, simplistic interface that is not confusing, yet still pleasing to look at and functional.**

* The editor design remained plain and simple, adorning a dark grey design for the menu and circuit board. Everything on the menu is functional, as evidenced in the Utility testing section, and each gate has a hint icon if the user has any problems understanding how the logic gates work. My primary client reported that they like the design of the program, especially the colour scheme as it they prefer dark coloured programs over “blindingly bright” white ones, therefore meeting the requirement of being pleasing to look at. This was my intention of using a dark colour scheme for the program.
* Logic gates and wires use dynamic graphics, therefore making it much easier for the user to follow how a logic circuit works.

This objective has been achieved.

## Possible Expansions of the program

* More circuit objects could be added, such as clocks and pulse inputs/formers for flip flop circuits (circuit memory), all that would be needed is a new class and class list for the new items. This would make the program more useful to more people and make it a better studying tool. Premade circuit library would also be expanded as more important circuits will be available to make.
* An option could be added to compress circuits into small blocks that have the same amount of inputs and outputs (nodes) as the circuit had, allowing for adder circuits or flip-flop circuits to be used in chains. This would allow many more circuits to be created with ease, expanding the usability of the program and could even be used as a revision tool by university students. Premade circuit library would also be expanded as more important circuits will be available to make.
* The option to label circuit components could be a useful design feature as it that could be used with the premade circuits to be clearer (e.g. carry-in/out bits could be labelled on a full adder circuit) or could just convenient for the user when making their own circuits.

## Conclusion

This program could be expanded to be even more useful in the future, as much more complex circuits could be created and simulated with ease if the right components were added. I would release the simulator as an open source program, available to utilise for study, commercial or personal use, whichever the user desires to use it for. This would make it a decent alternative to the logic.ly program that I investigated in my Analysis, as it would have similar depth to that program while remaining to be free to use, whereas logic.ly costs a fair amount of money to use.

Overall I’d say my logic gate simulator was a success, as it achieved my main objectives and surpassed my expectations. It is a functional tool for studying and working with logic gates, proving to be a dependable simulator to how logic circuits would function while used commercially. It has a lot of potential to be expanded and become an even more useful utility for anyone to use, be it a university student, professor or even a computer engineer.

# Appendix

## [Prototype] Text-Based Logic Gate Simulator (Code)

1. **import** string
2. alph = list(string.ascii\_uppercase)
3. inputs = [] #user-determined inputs
4. inouts = [] #gate outputs(that can be used as other gate inputs) [not displayed]
5. outputs = []#circuit outputs [displayed]
6. usedgates = [] #gates used in circuit (ordered from first added to last)
7. gatedata = [] #gates data is in sets of threes: gate name, input 1, input 2, gate name, etc
8. gates = ("AND","OR","XOR","NAND","NOR","XNOR","NOT")
10. **class** gatelogic(object): #gate logic
11. **def** \_\_init\_\_(self,in1,in2):
12. self.in1 = in1
13. self.in2 = in2
14. **def** ANDgate(self):
15. **if** int(self.in1) == 1 **and** int(self.in2) == 1:   SS out = 1
16. **else**:    out = 0
17. **return** out
19. **def** ORgate(self):
20. **if** int(self.in1) == 1 **or** int(self.in2) == 1:    out = 1
21. **else**:    out = 0
22. **return** out
24. **def** NANDgate(self):
25. **if** int(self.in1) == 1 **and** int(self.in2) == 1:    out = 0
26. **else**:    out = 1
27. **return** out
29. **def** NORgate(self):
30. **if** int(self.in1) == 1 **or** int(self.in2) == 1:    out = 0
31. **else**:    out = 1
32. **return** out
34. **def** XORgate(self):
35. **if** self.in1 == self.in2:    out = 0
36. **else**:    out = 1
37. **return** out
39. **def** XNORgate(self):
40. **if** self.in1 == self.in2:    out = 1
41. **else**:    out = 0
42. **return** out
44. **def** NOTgate(self):
45. **if** int(self.in1) == 1:    out = 0
46. **else**:    out = 1
47. **return** out
49. **def** SelectGate(): #function for user to choose the next gate they would like to use
50. answererror = True
51. **print**("What gate do you want?(add 'out' to the gate if it is used for a final output) type 'gates' to get a list of options")
52. **while** answererror == True:
53. answer = input(">").upper()
54. **if** **not** answer == "GATES":
55. gatedata.append(answer)
56. **if** "OUT" **in** answer:
57. answer = answer.replace("OUT",'')
58. **if** answer **in** gates:     answererror = False
59. **else**:   **print**("invalid logic gate, try again")
60. **else**:   **print**(gates)
61. **return** answer
63. **def** SelectInputsForGate(): #function to determine where the current gate receives its inputs from
64. chosengate = SelectGate()
65. **if** **not** chosengate == "NOT":
66. **print**("Which input values would you like to use?(gX for gate outputs where X is an integer)")
67. inputA = input("inputA>")
68. inputB = input("inputB>")
69. gatedata.append(str(inputA))
70. gatedata.append(str(inputB))
71. usedgates.append(chosengate)
72. **else**:
73. **print**("Which input value would you like to use?(gX for gate outputs where X is an integer)")
74. inputA = input("input>")
75. gatedata.append(inputA)
76. gatedata.append("X")
77. usedgates.append(chosengate)
79. **def** InputQuant(): #user chooses how many inputs the circuit has
80. **print**("How many inputs do you want?")
81. inputamount = input(">")
82. **while** **not** int(inputamount) **in** range(1,27):
83. **print**("Invalid value, try again.")
84. inputamount = input(">")
85. **return** inputamount
87. **def** RunCircuit(): #function to run the circuit
88. **for** i **in** range(len(usedgates)):
89. outputready = False
90. gate = gatedata[i\*3]
91. inputA = gatedata[(i\*3)+1]
92. inputB = gatedata[(i\*3)+2]
93. **if** "OUT" **in** gate:
94. outputready = True
95. **if** **not** "NOT" **in** gate:
96. **if** "g" **in** inputA:
97. inputA = inputA.replace("g",'')
98. inputA = str(inouts[int(inputA)-1])
99. **else**:
100. inputA = str(inputs[int(inputA)-1])
101. **if** "g" **in** inputB:
102. inputB = inputB.replace("g",'')
103. inputB = str(inouts[int(inputB)-1])
104. **else**:
105. inputB = str(inputs[int(inputB)-1])
106. x = gatelogic(inputA,inputB)
107. inouts.append(eval("x."+usedgates[i]+"gate()"))
108. **if** outputready == True: outputs.append(eval("x."+usedgates[i]+"gate()"))
109. **else**:
110. **if** "g" **in** inputA:
111. inputA = inputA.replace("g",'')
112. inputA = str(inouts[int(inputA)-1])
113. **else**:
114. inputA = str(inputs[int(inputA)-1])
115. x = gatelogic(inputA,"X")
116. inouts.append(eval("x."+usedgates[i]+"gate()"))
117. **if** outputready == True: outputs.append(eval("x."+usedgates[i]+"gate()"))
119. **def** AskAmounts(): #user determines state of inputs
120. **print**("How many logic gates are you using?")
121. gatecount = int(input(">"))
122. **for** i **in** range(gatecount):
123. SelectInputsForGate()

126. **def** SaveCircuit(): #save circuit date to file
127. **print**("What would you like to name your circuit?")
128. filename = input(">")
129. filename = filename + ".txt"
130. file = open(filename,"w+")
131. file.write(inputquant)
132. file.write("\n")
133. file.write(str(gatedata))
134. file.write("\n")
135. file.write(str(usedgates))
136. file.close()
137. **print**("Circuit saved to",filename)
139. **def** LoadCircuit(): #load circuit data from file
140. **print**("What circuit would you like to load?")
141. filename = input(">")
142. filename = filename + ".txt"
143. file = open(filename,"r")
144. reader = file.readlines()
145. inputquant = int(reader[0])
146. gatedata = eval(reader[1])
147. usedgates = eval(reader[2])
148. **return** inputquant, gatedata, usedgates #all the data required to create a circuit

151. **print**("Would you like to load a circuit?(Y/N)")
152. askload = input(">").upper()
153. **if** askload == "Y":
154. inputquant, gatedata, usedgates = LoadCircuit()
155. **else**:
156. inputquant = InputQuant()
157. AskAmounts()
158. **print**("Would you like to save this circuit?(Y/N)")
159. askload = input(">").upper()
160. **if** askload == "Y":
161. SaveCircuit()
162. **while** 1 == 1:
163. **print**("Please set values for input variables. (0/1)")
164. **for** i **in** range(int(inputquant)):
165. inputs.append(input("input"+alph[i]+">"))
166. RunCircuit()
167. **for** i **in** range(len(outputs)):
168. **print**("output"+alph[i]+" = "+str(outputs[i]))
170. outputs, inouts, inputs = [],[],[]

## [Main Program] Interactive Logic Gate Simulator (Code)

1. **import** pygame as pg
3. pg.init()
4. pg.font.init()
6. res = [1600,900]
7. fps = 144
9. win = pg.display.set\_mode(res)
10. pg.display.set\_caption("Logic Gate Simulator [Experimental Build]")
11. clock = pg.time.Clock()
13. IMand = [pg.image.load('images/andgate/andgate0.png'), pg.image.load('images/andgate/andgate1.png'), pg.image.load('images/andgate/andgate2.png'), pg.image.load('images/andgate/andgate3.png')]
14. IMor = [pg.image.load('images/orgate/orgate0.png'), pg.image.load('images/orgate/orgate1.png'), pg.image.load('images/orgate/orgate2.png'), pg.image.load('images/orgate/orgate3.png')]
15. IMxor = [pg.image.load('images/xorgate/xorgate0.png'), pg.image.load('images/xorgate/xorgate1.png'), pg.image.load('images/xorgate/xorgate2.png'), pg.image.load('images/xorgate/xorgate3.png')]
16. IMnand = [pg.image.load('images/nandgate/nandgate0.png'), pg.image.load('images/nandgate/nandgate1.png'), pg.image.load('images/nandgate/nandgate2.png'), pg.image.load('images/nandgate/nandgate3.png')]
17. IMnor = [pg.image.load('images/norgate/norgate0.png'), pg.image.load('images/norgate/norgate1.png'), pg.image.load('images/norgate/norgate2.png'), pg.image.load('images/norgate/norgate3.png')]
18. IMxnor = [pg.image.load('images/xnorgate/xnorgate0.png'), pg.image.load('images/xnorgate/xnorgate1.png'), pg.image.load('images/xnorgate/xnorgate2.png'), pg.image.load('images/xnorgate/xnorgate3.png')]
19. IMnot = [pg.image.load('images/notgate/notgate0.png'), pg.image.load('images/notgate/notgate1.png')]
21. IMin0 = [pg.image.load('images/in0/in0.png'),pg.image.load('images/in0/in1.png')]
22. IMout0 = [pg.image.load('images/out0/out0.png'),pg.image.load('images/out0/out1.png')]
24. hint = [pg.image.load('images/hint/hint0.png'),pg.image.load('images/hint/hint1.png'),pg.image.load('images/hint/hint2.png'),pg.image.load('images/hint/hint3.png'),
25. pg.image.load('images/hint/hint4.png'),pg.image.load('images/hint/hint5.png'),pg.image.load('images/hint/hint6.png')]
27. save = pg.image.load('images/save.png')
28. load = pg.image.load('images/load.png')
29. clear = pg.image.load('images/clear.png')
30. info = pg.image.load('images/info.png')
32. gatelist = ("AND","OR","XOR","NOT","NAND","NOR","XNOR")
34. **class** menu:
35. **def** \_\_init\_\_(self):
36. self.display = 0
37. self.gap = 7
38. self.object = []
39. self.decal = []
40. self.hintdecal = []
41. self.hints = []
42. self.hint = None
43. self.infoicons = []
44. self.infodecal = info
45. self.decal.append(IMin0[0].convert\_alpha())
46. self.object.append(self.decal[0].get\_rect())
47. self.decal.append(IMout0[0].convert\_alpha())
48. self.object.append(self.decal[1].get\_rect())
49. **for** i **in** range(len(gatelist)):
50. self.decal.append(eval("IM"+gatelist[i].lower()+"[0]").convert\_alpha())
51. self.object.append(self.decal[i+2].get\_rect())
52. self.infoicons.append(self.infodecal.get\_rect())
53. self.hintdecal.append(hint[i])
54. self.hints.append(self.hintdecal[i].get\_rect())
55. self.decal.append(save.convert\_alpha())
56. self.object.append(self.decal[len(self.decal)-1].get\_rect())
57. self.decal.append(load.convert\_alpha())
58. self.object.append(self.decal[len(self.decal)-1].get\_rect())
59. self.decal.append(clear.convert\_alpha())
60. self.object.append(self.decal[len(self.decal)-1].get\_rect())
62. **def** DrawItems(self):
63. self.object[0].center = (38,105)
64. self.object[1].center = (112,105)
65. self.object[9].center = (38,20)
66. self.object[10].center = (112,20)
67. self.object[11].center = (75,50)
68. **for** i **in** range(len(gatelist)):
69. self.object[i+2].midtop = (75,self.object[i+1].midbottom[1] + self.gap)
70. self.infoicons[i].topleft = self.object[i+2].topright
71. **if** self.hint != None:
72. **if** self.hint == i:
73. self.hints[i].topleft = self.infoicons[i].topright
74. win.blit(self.hintdecal[i],self.hints[i])
75. **for** i **in** range(len(self.object)):
76. win.blit(self.decal[i],self.object[i])
77. **for** i **in** range(len(self.infoicons)):
78. win.blit(self.infodecal,self.infoicons[i])
80. **class** source: #circuit input
81. **def** \_\_init\_\_(self,startx,starty,skin=0,state=0):
82. self.skin = skin
83. self.state = state
84. self.facade = eval("IMin"+str(skin)+"["+str(self.state)+"]").convert\_alpha()
85. self.startx = startx
86. self.starty = starty
87. self.outmesh = self.facade.get\_rect()
88. self.outmesh.center = (self.startx,self.starty)
89. self.wirefrom = None
90. self.drag = False
92. **def** DrawMesh(self):
93. partlen = self.outmesh.w/4
94. self.mesh = self.facade.get\_rect()
95. self.mesh.w,self.mesh.h,self.mesh.center = self.outmesh.w-partlen,self.outmesh.h-partlen,self.outmesh.center
96. **def** UpdateFacade(self):
97. self.facade = eval("IMin"+str(self.skin)+"["+str(self.state)+"]").convert\_alpha()
99. **class** output: #circuit output
100. **def** \_\_init\_\_(self,startx,starty,skin=0,state=0):
101. self.infrom = None
102. self.skin = skin
103. self.state = state
104. self.facade = eval("IMout"+str(skin)+"[0]").convert\_alpha()
105. self.startx = startx
106. self.starty = starty
107. self.inmesh = self.facade.get\_rect()
108. self.inmesh.center = (self.startx,self.starty)
109. self.wirefrom = None
110. self.drag = False
112. **def** DrawMesh(self):
113. partlen = self.inmesh.w/4
114. self.mesh = self.facade.get\_rect()
115. self.mesh.w,self.mesh.h,self.mesh.center = self.inmesh.w-partlen,self.inmesh.h-partlen,self.inmesh.center
117. **def** inRead(self):
118. **if** **not** self.infrom == None:
119. **for** i **in** range(wqnt):
120. **if** allwires[i].ID == self.infrom:
121. self.state = allwires[i].signal
123. **def** UpdateFacade(self):
124. self.inRead()
125. self.facade = eval("IMout"+str(self.skin)+"["+str(self.state)+"]").convert\_alpha()

128. **class** gate: #logic gate
129. **def** \_\_init\_\_(self,name,startx,starty,in1from=None,in2from=None):
130. self.name = name
131. self.in1 = 0
132. self.in2 = 0
133. self.state = 0
134. self.facade = eval("IM"+self.name.lower()+"[0]").convert\_alpha()
135. self.mesh = self.facade.get\_rect()
136. self.startx = startx
137. self.starty = starty
138. self.mesh.center = (self.startx,self.starty)
139. self.drag = False
140. self.in1from = in1from
141. self.in2from = in2from
142. self.iomesh = self.facade.get\_rect()
143. self.ID = None
145. **def** ioCycle(self): #calculate output value from input values
146. **if** **not** self.in1from == None:
147. **for** i **in** range(wqnt):
148. **if** allwires[i].ID == self.in1from:
149. self.in1 = allwires[i].signal
150. **if** **not** self.in2from == None:
151. **for** i **in** range(wqnt):
152. **if** allwires[i].ID == self.in2from:
153. self.in2 = allwires[i].signal
154. self.output = self.logic()
155. self.UpdateFacade()

158. **def** DrawSubmesh(self):
159. half = self.mesh.w/2
160. partlen = self.mesh.w/5
161. **if** self.name == "NOT":
162. self.in1mesh = self.facade.get\_rect()
163. self.in1mesh.midleft,self.in1mesh.w = self.mesh.midleft,partlen
164. **else**:
165. self.in1mesh = self.facade.get\_rect()
166. self.in1mesh.x,self.in1mesh.y,self.in1mesh.w,self.in1mesh.h = self.mesh.x,self.mesh.y,partlen,half
167. self.in2mesh = self.facade.get\_rect()
168. self.in2mesh.x,self.in2mesh.y,self.in2mesh.w,self.in2mesh.h = self.mesh.x,self.mesh.y+half,partlen,half
170. self.iomesh.midleft,self.iomesh.w = self.mesh.midleft,partlen
172. **def** UpdateFacade(self):
173. self.facade = eval("IM"+self.name.lower()+"["+str(self.state)+"]").convert\_alpha() #e.g. IMand[0]
175. **def** logic(self): #gate logic
176. **if** self.name == "AND":
177. **if** self.in1 == 1 **and** self.in2 == 1:
178. out = 1
179. self.state = 3
180. **elif** self.in1 == 1:
181. self.state = 1
182. **elif** self.in2 == 1:
183. self.state = 2
184. **else**:
185. self.state = 0
186. **if** **not** (self.in1 == 1 **and** self.in2 == 1):
187. out = 0
189. **if** self.name == "OR":
190. **if** self.in1 == 1 **or** self.in2 == 1:
191. out = 1
192. **if** self.in1 == 1 **and** self.in2 == 1:
193. self.state = 3
194. **elif** self.in1 == 1:
195. self.state = 1
196. **elif** self.in2 == 1:
197. self.state = 2
198. **else**:
199. out = 0
200. self.state = 0
202. **if** self.name == "NAND":
203. **if** self.in1 == 1 **and** self.in2 == 1:
204. out = 0
205. self.state = 3
206. **elif** self.in1 == 1:
207. self.state = 1
208. **elif** self.in2 == 1:
209. self.state = 2
210. **else**:
211. self.state = 0
212. **if** **not** (self.in1 == 1 **and** self.in2 == 1):
213. out = 1
215. **if** self.name == "NOR":
216. **if** self.in1 == 1 **or** self.in2 == 1:
217. out = 0
218. **if** self.in1 == 1 **and** self.in2 == 1:
219. self.state = 3
220. **elif** self.in1 == 1:
221. self.state = 1
222. **elif** self.in2 == 1:
223. self.state = 2
224. **else**:
225. out = 1
226. self.state = 0
228. **if** self.name == "XOR":
229. **if** self.in1 == self.in2:    out = 0
230. **if** self.in1 == 1 **and** self.in2 == 1:
231. self.state = 3
232. **elif** self.in1 == 1:
233. self.state = 1
234. out = 1
235. **elif** self.in2 == 1:
236. self.state = 2
237. out = 1
238. **else**:
239. self.state = 0
241. **if** self.name == "XNOR":
242. **if** self.in1 == self.in2:    out = 1
243. **if** self.in1 == 1 **and** self.in2 == 1:
244. self.state = 3
245. **elif** self.in1 == 1:
246. self.state = 1
247. out = 0
248. **elif** self.in2 == 1:
249. self.state = 2
250. out = 0
251. **else**:
252. self.state = 0
254. **if** self.name == "NOT":
255. **if** self.in1 == 1:
256. out = 0
257. self.state = 1
258. **else**:
259. out = 1
260. self.state = 0
261. **return** out
263. **class** wire: #wire
264. **def** \_\_init\_\_(self,incon,outcon,startpos,drawing=False,ID=None):
265. self.incon = incon
266. self.outcon = outcon
267. self.startpos = startpos
268. self.endpos = startpos
269. self.signal = 0
270. self.drawing = drawing
271. self.ID = ID
272. self.ioStatus = None #true->in false->out
273. self.isSet = False
274. self.colour = (40,40,40)
275. self.position = None
276. self.fromsource = False
278. **def** on(self):
279. self.signal = 1
280. self.colour = (0,150,200)
282. **def** off(self):
283. self.signal = 0
284. self.colour = (40,40,40)
286. **def** UpdateWire(self):
287. **if** self.isSet:
288. **if** self.ioStatus == None:
289. **if** self.incon == None: #set wire to gate input
290. self.startpos = allgates[self.incon].iomesh.midleft[0]+95,allgates[self.incon].iomesh.midleft[1]
291. self.endpos = eval("allgates[self.outcon[0]].in"+self.outcon[1]+"mesh.center")
292. **else**: #set wire to gate output
293. self.startpos = eval("allgates[self.outcon[0]].in"+self.outcon[1]+"mesh.center")
294. self.endpos = allgates[self.incon].iomesh.center[0]+80,allgates[self.incon].iomesh.midleft[1]
295. **elif** self.ioStatus == True: #set wire to raw input
296. **for** i **in** range(iqnt):
297. **if** i == self.incon:
298. self.startpos = eval("allinputs[i].outmesh."+self.position)
299. self.endpos = eval("allgates[self.outcon[0]].in"+self.outcon[1]+"mesh.center")
300. **elif** self.fromsource != True: #set wire to output display
301. **for** i **in** range(oqnt):
302. **if** i == self.outcon:
303. self.startpos = allgates[self.incon].iomesh.midleft[0]+95,allgates[self.incon].iomesh.midleft[1]
304. self.endpos = eval("alloutputs[i].inmesh."+self.position)
306. **if** self.incon != None:
307. **if** self.ioStatus != True:
308. **if** allgates[self.incon].output == 1:
309. self.on()
310. **else**:
311. self.off()
312. **else**:
313. **if** allinputs[self.incon].state == 1:
314. self.on()
315. **else**:
316. self.off()
317. **else**:
318. self.off()
319. pg.draw.line(win,self.colour,self.startpos,self.endpos,5)
321. gdrag = None
322. idrag = None
323. odrag = None
324. iqnt = 0 #quantity of input objects
325. oqnt = 0 #quantity of output objects
326. gqnt = 0 #quantity of logic gates
327. wqnt = 0 #quantity of wires
328. inready = False
329. outready = False
330. tempdata = None
332. allgates = []
333. allwires = []
334. allinputs = []
335. alloutputs = []
337. x = allgates
338. m = menu()
340. **def** SaveCircuit(): #save circuit date to file
341. **print**("What would you like to name your circuit?")
342. filename = input(">")
343. filename += ".txt"
344. file = open(filename,"w+")
345. file.write('{0},{1},{2},{3}\n'.format(iqnt,oqnt,gqnt,wqnt))
346. **if** iqnt != 0:
347. **for** i **in** range(iqnt):
348. file.write('{0},{1},{2},{3}\n'.format(allinputs[i].outmesh.center[0],allinputs[i].outmesh.center[1],allinputs[i].skin, allinputs[i].state))
349. file.write("\n")
350. **if** oqnt != 0:
351. **for** i **in** range(oqnt):
352. file.write('{0},{1},{2}\n'.format(alloutputs[i].inmesh.center[0],alloutputs[i].inmesh.center[1],alloutputs[i].skin))
353. file.write("\n")
354. **if** gqnt != 0:
355. **for** i **in** range(gqnt):
356. file.write('{0},{1},{2}\n'.format(gatelist.index(x[i].name),x[i].mesh.center[0],x[i].mesh.center[1]))
357. file.write("\n")
358. **for** i **in** range(wqnt):
359. file.write('{0},{1},{2},{3}\n{4}\n'.format(allwires[i].incon,allwires[i].outcon,allwires[i].ioStatus,allwires[i].fromsource,allwires[i].position))
360. file.close()
361. **print**("Circuit saved to",filename)
363. **def** LoadCircuitData(): #load circuit data from file
364. **print**("What circuit would you like to load?")
365. filename = input(">")
366. filename = filename + ".txt"
367. file = open(filename,"r")
368. reader = file.readlines()
369. i,o,g,w = eval(reader[0])
370. **return** filename,i,o,g,w
372. **def** MapFile(): #map circuit onto editor
373. filename,IN,OUT,GATE,WIRE = LoadCircuitData()
374. file = open(filename,"r")
375. reader = file.readlines()
376. counter = 1
377. temp1,temp2,temp3,temp4,temp5 = 0,0,0,0,0
378. **if** IN != 0:
379. **for** i **in** range(IN): #creating inputs
380. temp1,temp2,temp3,temp4 = eval(reader[counter])
381. allinputs.append(source(temp1,temp2,temp3,temp4))
382. counter += 1
383. counter += 1
384. **if** OUT != 0:
385. **for** i **in** range(OUT): #creating outputs
386. temp1,temp2,temp3 = eval(reader[counter])
387. alloutputs.append(output(temp1,temp2,temp3))
388. counter += 1
389. counter += 1
390. **if** GATE != 0:
391. **for** i **in** range(GATE): #creating gate
392. temp1,temp2,temp3 = eval(reader[counter])
393. x.append(gate(gatelist[temp1],temp2,temp3))
394. x[i].ID = i
395. x[i].ioCycle()
396. x[i].DrawSubmesh()
397. counter += 1
398. **if** WIRE != 0:
399. **for** i **in** range(WIRE): #creating wires
400. counter += 1
401. temp1,temp2,temp3,temp4 = eval(reader[counter])
402. counter+=1
403. temp5 = reader[counter]
404. allwires.append(wire(temp1,temp2,(0,0)))
405. allwires[i].ID = i
406. allwires[i].ioStatus,allwires[i].fromsource,allwires[i].position = temp3,temp4,temp5
407. **if** temp3 != False: #assigning object I/O ports to connected wires
408. **if** temp2[1] == '1':
409. x[temp2[0]].in1from = i
410. **else**:
411. x[temp2[0]].in2from = i
412. **else**:
413. alloutputs[temp2].infrom = i
414. allwires[i].isSet = True
416. **return**(IN,OUT,GATE,WIRE)
418. #mainloop
419. run = True
420. **while** run:
421. win.fill((60,60,60))
422. pg.draw.rect(win,(80,80,80),pg.Rect(0,0,150,res[1]))
423. m.DrawItems()
424. **for** event **in** pg.event.get():
425. **if** event.type == pg.QUIT:
426. run = False
427. **if** event.type == pg.MOUSEBUTTONDOWN:
428. **if** event.button == 3: #right-click
429. **if** inready **or** outready:
430. **for** i **in** range(wqnt):
431. **if** **not** allwires[i].isSet: #cleanup loose wires with right-click
432. wqnt -= 1
433. inready = False
434. outready = False
435. allwires.remove(allwires[i])
436. tempdata = i
437. **for** i **in** range(gqnt):
438. **if** x[i].in1from == tempdata:
439. x[i].in1from = None
440. **if** x[i].in2from == tempdata:
441. x[i].in2from = None
442. **for** i **in** range(oqnt):
443. **if** alloutputs[i].infrom == tempdata:
444. alloutputs[i].infrom = None
445. tempdata = None
446. **else**:
447. **for** i **in** range(iqnt):
448. **if** allinputs[i].outmesh.collidepoint(event.pos):
449. **if** allinputs[i].state == 0:
450. allinputs[i].state = 1
451. **else**:
452. allinputs[i].state = 0
454. **if** event.button == 1: #left-click
455. **if** m.object[9].collidepoint(event.pos): #save button clicked
456. SaveCircuit()
457. **if** m.object[10].collidepoint(event.pos): #load button clicked
458. x.clear()
459. allinputs.clear()
460. alloutputs.clear()
461. allwires.clear()
462. iqnt,oqnt,gqnt,wqnt = 0,0,0,0
463. **try**:
464. iqnt,oqnt,gqnt,wqnt = MapFile()
465. **print**("Load Successful")
466. **except**:
467. **print**("Load Failed")
468. x.clear()
469. allinputs.clear()
470. alloutputs.clear()
471. allwires.clear()
472. iqnt,oqnt,gqnt,wqnt = 0,0,0,0
473. **if** m.object[11].collidepoint(event.pos): #clear editor button clicked
474. x.clear()
475. allinputs.clear()
476. alloutputs.clear()
477. allwires.clear()
478. iqnt,oqnt,gqnt,wqnt = 0,0,0,0
479. **for** i **in** range(len(gatelist)): #logic gate pulled from menu
480. **if** m.object[i+2].collidepoint(event.pos):
481. x.append(gate(gatelist[i],event.pos[0],event.pos[1]))
482. gqnt += 1
483. x[gqnt-1].ID = gqnt-1
484. **if** m.infoicons[i].collidepoint(event.pos):
485. **if** m.hint != i:
486. m.hint = i
487. **else**:
488. m.hint = None
490. **if** m.object[0].collidepoint(event.pos): #input object pulled from menu
491. allinputs.append(source(event.pos[0],event.pos[1]))
492. iqnt += 1
493. **if** m.object[1].collidepoint(event.pos): #output object pulled from menu
494. alloutputs.append(output(event.pos[0],event.pos[1]))
495. oqnt += 1

498. **for** i **in** range(iqnt):
499. **if** allinputs[i].outmesh.collidepoint(event.pos):
500. **if** **not** inready:
501. radius = 15
502. **if** event.pos[0] **in** range(allinputs[i].outmesh.midtop[0]-radius,allinputs[i].outmesh.midtop[0]+radius):
503. **if** event.pos[1] **in** range(allinputs[i].outmesh.midtop[1],allinputs[i].outmesh.midtop[1]+radius):
504. allinputs[i].wirefrom = "midtop"
505. **if** event.pos[0] **in** range(allinputs[i].outmesh.midright[0]-radius,allinputs[i].outmesh.midright[0]):
506. **if** event.pos[1] **in** range(allinputs[i].outmesh.midright[1]-radius,allinputs[i].outmesh.midright[1]+radius):
507. allinputs[i].wirefrom = "midright"
508. **if** event.pos[0] **in** range(allinputs[i].outmesh.midbottom[0]-radius,allinputs[i].outmesh.midbottom[0]+radius):
509. **if** event.pos[1] **in** range(allinputs[i].outmesh.midbottom[1]-15,allinputs[i].outmesh.midbottom[1]):
510. allinputs[i].wirefrom = "midbottom"
511. **if** event.pos[0] **in** range(allinputs[i].outmesh.midleft[0],allinputs[i].outmesh.midleft[0]+radius):
512. **if** event.pos[1] **in** range(allinputs[i].outmesh.midleft[1]-radius,allinputs[i].outmesh.midleft[1]+radius):
513. allinputs[i].wirefrom = "midleft"
514. **if** outready:
515. **if** **not** allinputs[i].wirefrom == None:
516. **if** tempdata != None: #set wire on raw input
517. **if** tempdata[0] == "w":
518. **if** "g" **in** tempdata:
519. allwires[wqnt-1].position = allinputs[i].wirefrom
520. allwires[wqnt-1].ioStatus = True
521. allwires[wqnt-1].incon = i
522. tempdata = None
523. **for** i **in** range(wqnt):
524. **if**  allwires[i].drawing == True:
525. allwires[i].drawing = False
526. allwires[i].isSet = True
527. outready = False
528. **elif** **not** outready:
529. **if** **not** allinputs[i].wirefrom == None: #create new wire from raw input
530. allwires.append(wire(i,None,getattr(allinputs[i].outmesh,allinputs[i].wirefrom),True,wqnt))
531. allwires[wqnt].position = allinputs[i].wirefrom
532. wqnt += 1
533. allwires[wqnt-1].ioStatus = True
534. inready = True
535. tempdata = "w",wqnt-1,"i"
536. allinputs[i].wirefrom = None
537. allwires[wqnt-1].fromsource = True
539. **elif** gdrag == None **and** odrag == None: #being drag input display
540. allinputs[i].drag = True
541. idrag = i
542. mouse\_x, mouse\_y = event.pos
543. offset\_x = allinputs[i].outmesh.x - mouse\_x
544. offset\_y = allinputs[i].outmesh.y - mouse\_y
546. **for** i **in** range(oqnt):
547. **if** alloutputs[i].inmesh.collidepoint(event.pos):
548. **if** **not** outready:
549. radius = 15
550. **if** event.pos[0] **in** range(alloutputs[i].inmesh.midtop[0]-radius,alloutputs[i].inmesh.midtop[0]+radius): #find if user clicked edge of output display
551. **if** event.pos[1] **in** range(alloutputs[i].inmesh.midtop[1],alloutputs[i].inmesh.midtop[1]+radius):
552. alloutputs[i].wirefrom = "midtop"
553. **if** event.pos[0] **in** range(alloutputs[i].inmesh.midright[0]-radius,alloutputs[i].inmesh.midright[0]):
554. **if** event.pos[1] **in** range(alloutputs[i].inmesh.midright[1]-radius,alloutputs[i].inmesh.midright[1]+radius):
555. alloutputs[i].wirefrom = "midright"
556. **if** event.pos[0] **in** range(alloutputs[i].inmesh.midbottom[0]-radius,alloutputs[i].inmesh.midbottom[0]+radius):
557. **if** event.pos[1] **in** range(alloutputs[i].inmesh.midbottom[1]-15,alloutputs[i].inmesh.midbottom[1]):
558. alloutputs[i].wirefrom = "midbottom"
559. **if** event.pos[0] **in** range(alloutputs[i].inmesh.midleft[0],alloutputs[i].inmesh.midleft[0]+radius):
560. **if** event.pos[1] **in** range(alloutputs[i].inmesh.midleft[1]-radius,alloutputs[i].inmesh.midleft[1]+radius):
561. alloutputs[i].wirefrom = "midleft"
562. **if** inready:
563. **if** allwires[wqnt-1].fromsource == False: #set wire on output display
564. **if** **not** alloutputs[i].wirefrom == None **and** alloutputs[i].infrom == None:
565. **if** tempdata != None:
566. **if** tempdata[0] == "w":
567. allwires[wqnt-1].position = alloutputs[i].wirefrom
568. alloutputs[i].infrom = wqnt-1
569. allwires[wqnt-1].outcon = i
570. allwires[wqnt-1].ioStatus = False
571. tempdata = None
572. alloutputs[i].wirefrom = None
573. **for** i **in** range(wqnt):
574. **if** allwires[i].drawing == True:
575. allwires[i].drawing = False
576. allwires[i].isSet = True
577. inready = False
578. **elif** **not** inready:
579. **if** **not** alloutputs[i].wirefrom == None **and** alloutputs[i].infrom == None: #create new wire from raw output
580. allwires.append(wire(None,i,getattr(alloutputs[i].inmesh,alloutputs[i].wirefrom),True,wqnt))
581. alloutputs[i].infrom = wqnt
582. allwires[wqnt].position = alloutputs[i].wirefrom
583. wqnt += 1
584. allwires[wqnt-1].ioStatus = False
585. outready = True
586. tempdata = "w",wqnt-1,"o"
587. alloutputs[i].wirefrom = None
588. **elif** gdrag == None **and** idrag == None: #being drag output display
589. alloutputs[i].drag = True
590. odrag = i
591. mouse\_x, mouse\_y = event.pos
592. offset\_x = alloutputs[i].inmesh.x - mouse\_x
593. offset\_y = alloutputs[i].inmesh.y - mouse\_y
595. **for** i **in** range(gqnt):
596. **if** x[i].mesh.collidepoint(event.pos):
597. **if** **not** x[i].iomesh.collidepoint(event.pos):
598. **if** **not** x[i].iomesh.collidepoint(event.pos[0]-80,event.pos[1]):
599. **if** **not** outready:
600. **if** **not** inready:
601. **if** idrag == None **and** odrag == None:
602. x[i].drag = True
603. gdrag = i
604. mouse\_x, mouse\_y = event.pos
605. offset\_x = x[i].mesh.x - mouse\_x
606. offset\_y = x[i].mesh.y - mouse\_y
607. **else**:
608. **if** **not** outready:
609. **if** **not** inready: #create new wire from gate output
610. allwires.append(wire(x[i].ID,None,(x[i].iomesh.center[0]+80,x[i].iomesh.center[1]),True,wqnt))
611. wqnt += 1
612. inready = True
613. tempdata = "w","g"
614. **if** outready: #set wire on gate output
615. **if** x[i].iomesh.collidepoint(event.pos[0]-80,event.pos[1]):
616. **if** "g" **in** tempdata:
617. **if** i != allwires[wqnt-1].outcon[0]: #isn't trying to connect to itself
618. tempdata = i
619. allwires[wqnt-1].incon = allgates[tempdata].ID
620. tempdata = None
621. **for** i **in** range(wqnt):
622. **if**  allwires[i].drawing == True:
623. allwires[i].drawing = False
624. allwires[i].isSet = True
625. outready = False
626. **elif** "o" **in** tempdata:
627. tempdata = i
628. allwires[wqnt-1].incon = allgates[tempdata].ID
629. tempdata = None
630. **for** i **in** range(wqnt):
631. **if**  allwires[i].drawing == True:
632. allwires[i].drawing = False
633. allwires[i].isSet = True
634. outready = False
635. **elif** inready:
636. **if** **not** outready: #set wire on gate input
637. **if** x[i].in1mesh.collidepoint(event.pos): #is on input 1
638. **if** allwires[wqnt-1].ioStatus == None:
639. **if** i != allwires[wqnt-1].incon: #isn't trying to connect to itself
640. **if** x[i].in1from == None:
641. **if** tempdata != None:
642. **if** tempdata[0] == "w":
643. x[i].in1from = wqnt-1
644. allwires[wqnt-1].outcon = allgates[i].ID,"1"
645. inready = False
646. tempdata = None
647. **for** i **in** range(wqnt):
648. **if** allwires[i].drawing == True:
649. allwires[i].drawing = False
650. allwires[i].isSet = True
651. **else**:
652. **if** x[i].in1from == None:
653. **if** tempdata != None:
654. **if** tempdata[0] == "w":
655. x[i].in1from = wqnt-1
656. allwires[wqnt-1].outcon = allgates[i].ID,"1"
657. inready = False
658. tempdata = None
659. **for** i **in** range(wqnt):
660. **if** allwires[i].drawing == True:
661. allwires[i].drawing = False
662. allwires[i].isSet = True
664. **elif** x[i].in2mesh.collidepoint(event.pos): #is on input 2
665. **if** allwires[wqnt-1].ioStatus == None:
666. **if** i != allwires[wqnt-1].incon: #isn't trying to connect to itself
667. **if** x[i].in2from == None:
668. **if** tempdata != None:
669. **if** tempdata[0] == "w":
670. x[i].in2from = wqnt-1
671. allwires[wqnt-1].outcon = allgates[i].ID,"2"
672. inready = False
673. tempdata = None
674. **for** i **in** range(wqnt):
675. **if** allwires[i].drawing == True:
676. allwires[i].drawing = False
677. allwires[i].isSet = True
678. **else**:
679. **if** x[i].in2from == None:
680. **if** tempdata != None:
681. **if** tempdata[0] == "w":
682. x[i].in2from = wqnt-1
683. allwires[wqnt-1].outcon = allgates[i].ID,"2"
684. inready = False
685. tempdata = None
686. **for** i **in** range(wqnt):
687. **if** allwires[i].drawing == True:
688. allwires[i].drawing = False
689. allwires[i].isSet = True
690. **else**:
691. **if** **not** outready: #create new wire from gate input
692. **try**:
693. **if** x[i].in1mesh.collidepoint(event.pos) **and** x[i].in1from == None:
694. allwires.append(wire(None,(x[i].ID,"1"),(x[i].in1mesh.center[0],x[i].in1mesh.center[1]),True,wqnt))
695. wqnt += 1
696. x[i].in1from = wqnt - 1
697. outready = True
698. tempdata = "w","g"
699. **elif** x[i].in2mesh.collidepoint(event.pos) **and** x[i].in2from == None:
700. allwires.append(wire(None,(x[i].ID,"2"),(x[i].in2mesh.center[0],x[i].in2mesh.center[1]),True,wqnt))
701. wqnt += 1
702. x[i].in2from = wqnt - 1
703. outready = True
704. tempdata = "w","g"
705. **except** AttributeError:
706. **continue**
708. **elif** event.type == pg.MOUSEBUTTONUP:
709. **if** event.button == 1:
710. **try**:
711. **if** gdrag **in** range(gqnt):
712. x[gdrag].drag = False
713. gdrag = None
714. **except** ValueError:
715. **continue**
716. **try**:
717. **if** idrag **in** range(iqnt):
718. allinputs[idrag].drag = False
719. idrag = None
720. **except** ValueError:
721. **continue**
722. **try**:
723. **if** odrag **in** range(oqnt):
724. alloutputs[odrag].drag = False
725. odrag = None
726. **except** ValueError:
727. **continue**
729. **elif** event.type == pg.MOUSEMOTION:
730. **if** inready == True **or** outready == True:
731. **for** i **in** range(wqnt):
732. **if** allwires[i].drawing:
733. allwires[i].endpos = event.pos
735. **try**:
736. **if** gdrag **in** range(gqnt):
737. **if** x[gdrag].drag == True:
738. mouse\_x, mouse\_y = event.pos
739. x[gdrag].mesh.x = mouse\_x + offset\_x
740. x[gdrag].mesh.y = mouse\_y + offset\_y
741. **except** ValueError:
742. **continue**
743. **try**:
744. **if** idrag **in** range(iqnt):
745. **if** allinputs[idrag].drag == True:
746. mouse\_x, mouse\_y = event.pos
747. allinputs[idrag].outmesh.x = mouse\_x + offset\_x
748. allinputs[idrag].outmesh.y = mouse\_y + offset\_y
749. **except** ValueError:**continue**
750. **try**:
751. **if** odrag **in** range(oqnt):
752. **if** alloutputs[odrag].drag == True:
753. mouse\_x, mouse\_y = event.pos
754. alloutputs[odrag].inmesh.x = mouse\_x + offset\_x
755. alloutputs[odrag].inmesh.y = mouse\_y + offset\_y
756. **except** ValueError:**continue**
758. keys = pg.key.get\_pressed()
759. **for** i **in** range(wqnt):
760. allwires[i].UpdateWire()
761. **for** i **in** range(gqnt):
762. win.blit(x[i].facade,x[i].mesh)
763. x[i].ioCycle()
764. allgates[i].DrawSubmesh()
765. **for** i **in** range(oqnt):
766. win.blit(alloutputs[i].facade,alloutputs[i].inmesh)
767. alloutputs[i].DrawMesh()
768. alloutputs[i].UpdateFacade()
769. **for** i **in** range(iqnt):
770. win.blit(allinputs[i].facade,allinputs[i].outmesh)
771. allinputs[i].DrawMesh()
772. allinputs[i].UpdateFacade()
773. **if** keys[pg.K\_f]:
774. iqnt,oqnt,gqnt,wqnt = MapFile()
776. **if** keys[pg.K\_ESCAPE]: #cleanup loose wires with ESC
777. **for** i **in** range(wqnt):
778. **if** **not** allwires[i].isSet:
779. wqnt -= 1
780. inready = False
781. outready = False
782. allwires.remove(allwires[i])
783. tempdata = i
784. **for** i **in** range(gqnt):
785. **if** x[i].in1from == tempdata:
786. x[i].in1from = None
787. **if** x[i].in2from == tempdata:
788. x[i].in2from = None
789. tempdata = None
791. pg.display.flip()
792. clock.tick(fps)
794. pg.quit()