Design

# Input, output, process and storage

The user experience (UX) is an important aspect of modern design and I have considered from the start the interactions users will expect to find in the LGS interface, this includes common conventions and unique methods.

Input:

* Left click on UI objects, this is the main interaction of the program is will be controlled by mainly the mouse.
* scroll wheel, this is for zooming in and out for the program.
* mouse move, the program will use the mouse position to determen what the program should do.
* Button click, this is for each access of linking methods to an action by the user.
* Drop down menu, gives a compact and easy solution for different avinuos for the user to do something.
* textbox, the user can input a string and the program will then run a function when textbox contain changes so that the program is dynamic.
* Read Json file, allows for files and storage of setup of the program to be added to the program.

Process:

* Mouse move, the program is dynamic so it will react to each input you do, this requires mouse move to update the program each time.
* dynamic and trackable objects on a canvas, the data behind the program needs to match the information on the screen, this is why data binding is needed for the data to be in sync with each element of the program.
* dynamic and trackable UI Elements on a canvas, because the canvas doesn’t use a grid-based system the program should work for an infinite value of inputs.
* Transfer object between canvases, the switch between different areas of the program needs to be seamless and memory efficient.
* Save and read files, loading and creating a file needs to be easy and secure, it needs to be repeatable and check for edits to the file.
* Scaling canvases, for the zooming in and out the rescaling should change everything in the canvas and for the program to adjust to the new size so that every method works to the new scale.
* Translating canvases, Visually the canvas needs to move but also each method should recognise the change of location.

Here I consider both the physical storage requirements of the project and the objects produced.

Storage:

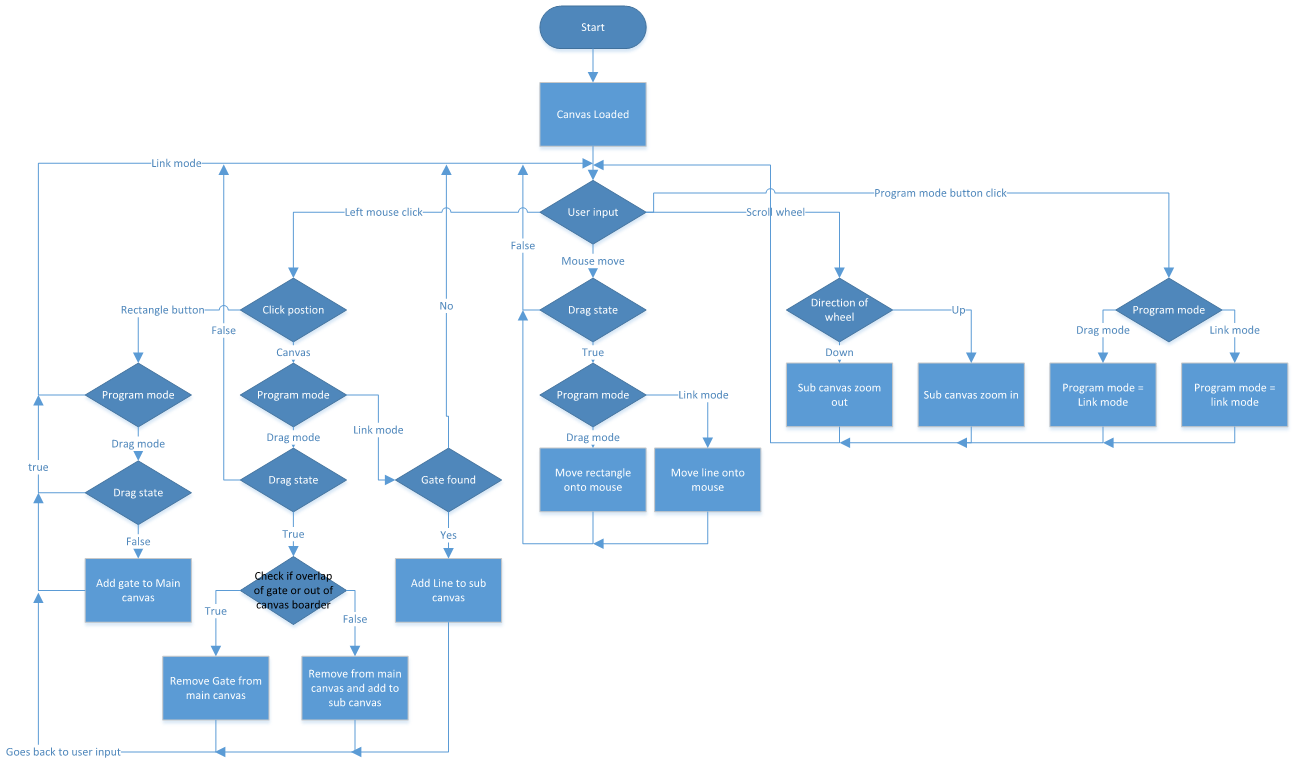
* In a worst case scenario the LGS can manage 272 gates, with 1kb per gate, even with overhead the program will be require less than \_\_\_\_.
* The program will need to be stored on the device from which it is used, there is currently no online functionality planned. The program will be distributed as an executable, the source code will only be provided to the client.
* List of objects, all data is group together for easy access and readability.
* c# variables, require understandable names and declaring the variable should be in the lowest level that is possible.
* Json Files, this should match the format of the OOP class so that there is easy decode and creating, security shouldn’t be a problem but should be added for difficulty to editing of the file.

Output:

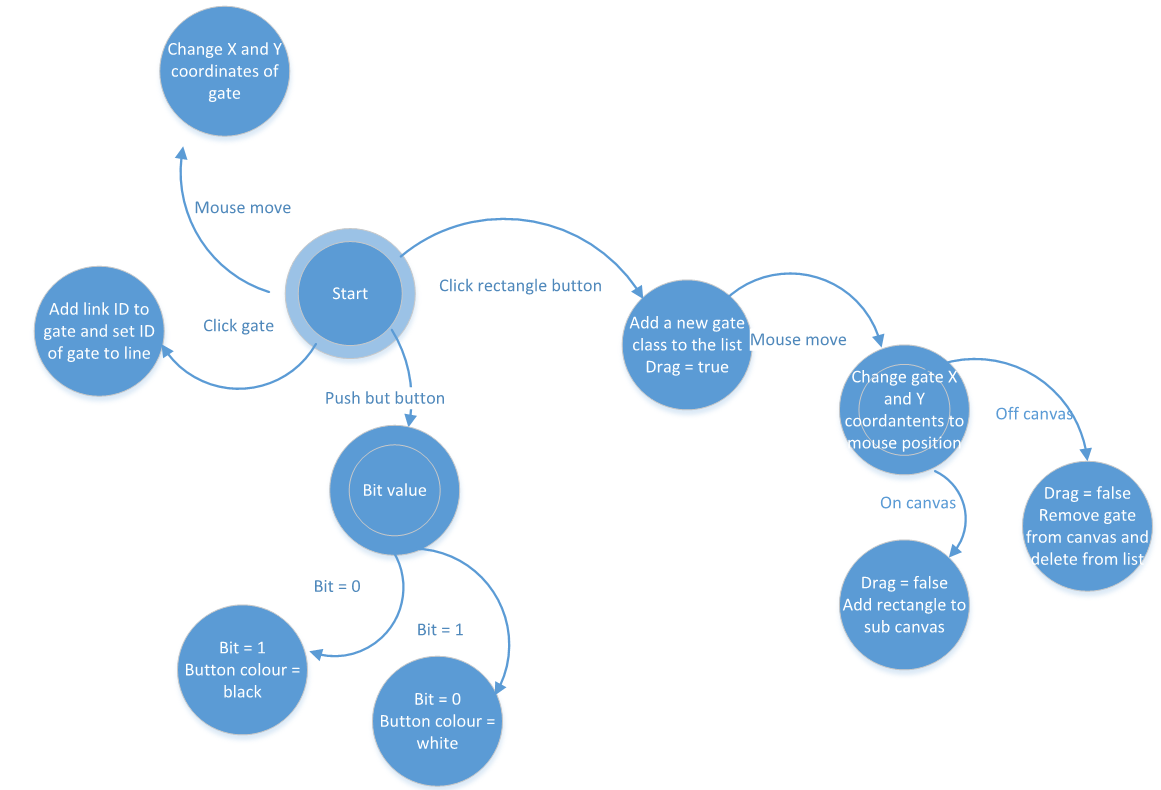
* Graphical UI, using the WPF format for the basic UI objects so that development is easy and readable due to large amount of resources and popularity.
* colour coded action, for easy understanding of the GUI for the user to follow the program.
* Create Json file, formatting should match the program data.

# Diagrams

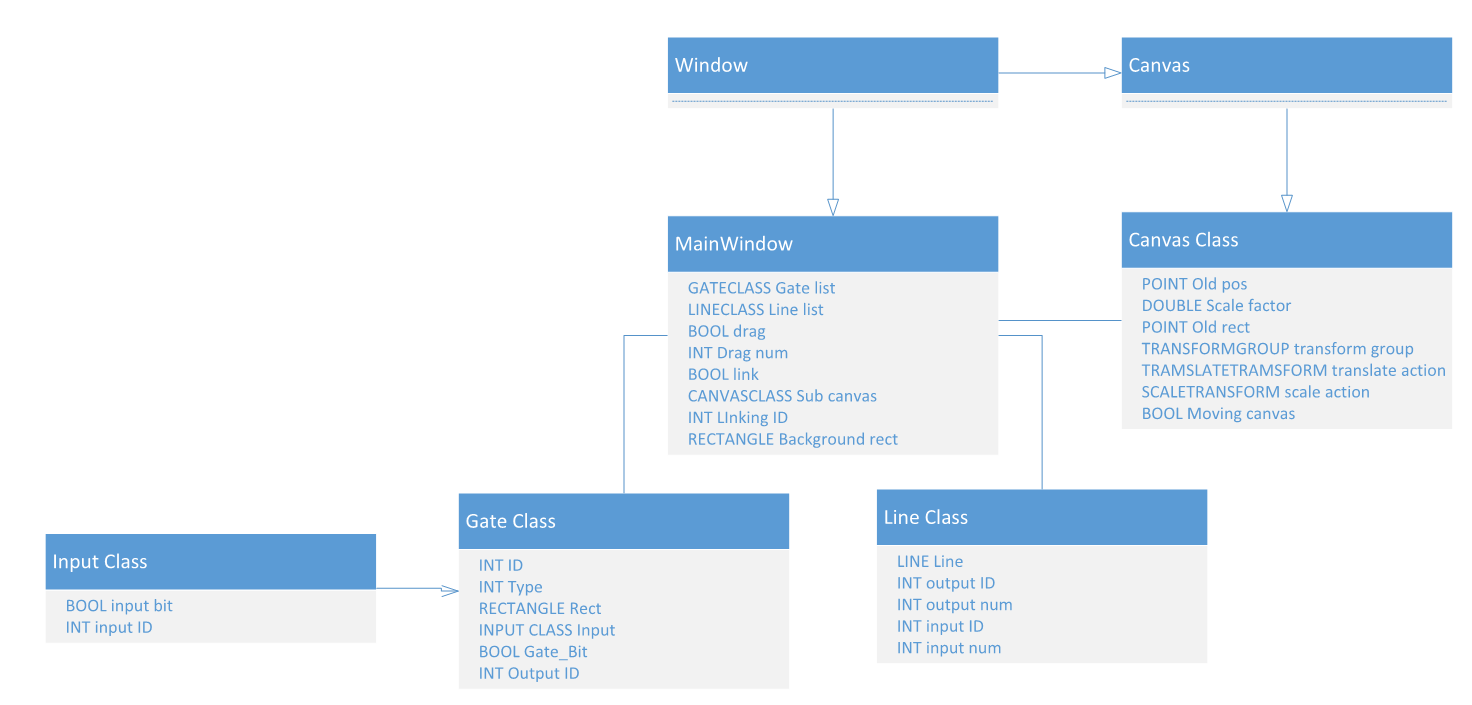
Diagrams: System flow chart, data flow diagram (this will be helpful), class diagram(Might as well)

System Flow Chart

Data Flow Diagram



Class Diagram



# Human Computer interaction

HCI: XAML code and the event handlers. Talk about how I used Microsoft word and how their HCI works, talk about the User Interface.

My project is heavily dependent on the GUI and how the user interacts with it. This is why I’ve done rigorous testing and take ideas from other programs. It’s followed an iterative cycle of development that has allowed me to create an effective interface.

My research into HCI from the testing and examining other programs I came to the conclusion that a drag drop system works best and as little movement of the mouse for the user was desirable. In my research I looked into different types of output. I came to the conclusion that sound cues are overly accepted and more of an annoyance. Colour coded actions let the user know something has works or is working and therefore should be added and now incorporated into my design.

For my questioning I used my previous prototype version with different set up for how the GUI would works and then tweaked or changed it for the end product. The questions I asked are in the table below, while there is only 2 option per question, the question asked were up for interpretation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Question | Option one | Vote for one | Option two | Vote for two |
| Should there be a drag drop or click-click system for UI object | When interaction with UI objects you just need to click and hold then release when done with it | 4 | When interaction with UI object you just click once and release then the UI object will follow you until you click again | 2 |
| Should there be a border around the gate | For a border | 2 | Against a border | 2 |
| Should a new rectangle spawn in the middle of the canvas or on your mouse | Spawn on your mouse courser | 4 | Spawn already on the canvas | 0 |

As the program needed be as natural for the user to use, I needed to anticipate the user actions that they would take. As these actions would most likely be based around programs, they use every day I examined the Microsoft office software for the actions that should take place.

It is important that the program is lag free and efficient as possible so that the user experience isn’t hindered. I did research into wither it’s faster to sort then binary search or just using hash tables and rehash. I came to the conclusion that hash table would be move adapted to the problem but not a vital part required as it’s going to be around twenty objects being linear searched.

The intuitions interfaces by the Microsoft products is what’s influenced me the most. They are widely liked and used around the word. This is why they made them a good choice to base mine off. The Colour difference between background and usable space and the border around everything when hovering over it is what really stand out for the HCI. Similarly, I looked at Paint and the tops up display with all the functions and settings and choose similarly. While these all have great features, I also changed them to be updated for modern technology. Because monitor sizes have become wider (16:9 instead of 3:4) having the setting on the side of the screen makes it more space efficient than on the top. And instead of greying out of adding a border to a highlighted object I went from black to red when highlighted. This made it stand out more and easily noticeable but had its draw backs for colour blind people.

# Algorithms

Algorithms: UI\_object search, Gate methods, gate order method, file creation and file load, validation of gates, pseudocode.

The backbone of the program is set around the GUI and how you interact with it. This is why many mechanics will be based around multiple event handlers. The main actions are called under their respective class and then sub methods will be done in the variable class. This allows the structure for program info to stay central and independent methods to be designed for the class to sort out.

Rectangle detection algorithm

There are many ways that can be done for detection algorithm. While some are better than others, they all depend on a case by case situation. The 3 ways are: sorting by 2D array and then binary search, Geometric hashing and linear search. 2D array and binary search would be the best if the list is extremely large, Hashing would be good as a middle ground and linear search is good for a small list. The problem with hashing and 2D array is due to the list always changing. The size is never fixed and this means the array will need to be resorted each time a new element is added. For the hash table it’s not so bad you will only need to rehash it after a few rectangles have been added but if 1 is removed from the hash it will all need to be rehashed. This is due to if there is a collision and the object is moved up the list and then if an object between where it should be and where it is, is removed then the hash table will be corrupted. Mathematically the most gates that can fit on the canvas is 272. This means at maximum it will only need to do 272 checks to find the object it’s looking for. This is why linear search is an acceptable solution. However, if development was for professional use to make large logic gate sets for CPU design or chip design where they have billions of transistors then 1 of the other solutions will be required. If I had to guess their use case of the software then it would be 2D array due as they would make the design and execute it once everything is created.

Sorted with 2D array (bubble sort)

for i in range arr.count

for x in range arr.count-i-1

if arr[x].X > arr[x+1].X

swap arr[x] and arr[x+1]

else if arr[x].X == arr[x+1]

if arr[x].Y > arr[x+1].Y

swap arr[x] and arr[x+1]

binary search 2D array

q=0

r=arr.length-1

mid = (r-q)/2 + q

while found == false or q-r>1

if arr[mid] == find

found = true

else if arr[mid] > find

r = mid

else

q = mid

mid = (r-q)/2 + q

If sorting into hash table

hashtable[arr.length\*1.2] = [-1, … , -1

For i in range arr.length

Num = arr[i].X ^ arr[i].Y + arr[i].Y ^ arr[i].X

Output = num mod arr.length\*1.2

While run == true

If hashtable[output + z] == -1

Hashtable[putput + z] = i

Run = false

Else

Z = Z + 1

Run = true

Searching the hash table

Num = pos.X ^ pos.Y + pos.Y ^ pos.X

Output = num mod arr.length\*1.2

While run == true

If hashtable[output+i] == Find

Return output + i

Else If hashtable[output+i] == -1

Return -1

Else If output + i == arr.count-1

i = 0

Else

i = i + 1

Linear search

Run = true

Count = -1

Find = INPUT

WHILE Run == TRUE AND Count != RectList.count

Count = Count + 1

IF RectList[Count].ID == Find

Run = FALSE

IF Run==TRUE

Return -1

RETURN Count

Gate methods

The gates are the main functional part of the program that user cares about. As the gate had its own class and all the variables needed it were in it the method for the output would be in the class. I had 2 ways of doing it, just have 1 base class that stores all the methods for each gate and then when needed to work out the output would use a switch. Or make 8 children classes that inherit the base class and override the method and when need the output just call the method. Both have their pros and cons and even now I don’t know which the best way is.

Order of gate method

When the program runs you want it to execute in the order from the start to the end. Due to being able to have infinite number of start positions and any number of end positions with gates in-between working out the sequence is important. This is due to the input for a gate may not be calculated yet and just be null. This will cause problems but also desired if you start at 2 different locations but merge to make 1 exit point.

In my prototype I calculated the order of execution by working out the maximum number of gates it would take to reach that gate. This works well for the prototype as it meant no gate is left without an invalid or not calculated input. It was also easy to code as it meant all you had to do was trace through each input until it found the exit and replace a variable if it was greater. There were a few problems with this method. 1 if there were no exit it would go in an endless loop and never be executable (or only be executed once if you add an error check). 2 if you wanted a circuit to loop it was impossible to have an infinitely long list with repeating gate order. While you could just say go to this point in the list again if you had 2 loops running at the same time but one was length 5 and the other was length 7 when the length 5 has ended then the length 7 would be reset. There was also a problem with if you wanted the starting gates to all start at once instead of the longest chain you couldn’t do it.

This brings me onto my new method. Instead of calculating the order at the start the program will go through the start gates all at once and then send the output to every gate. It will then remove all the gates and add the gates input gates into the list. This method removes all the problems that the other algorithm had. This system also works better for the program due to the easier coding of the simulation. It means real time I can make the changes to each gate and pause it when at the end of calculation. This is why making all the start gates start at once desirable due to being able to see the progress of the circuit as the message goes down the wire.

For x in range input\_list.count

If input\_list[x].active = true

Active\_Gate.add(input\_list[x].Input\_ID)

For x in range Active\_gate.count

Bool Duplicate = false

For (I=x+1) in range Active\_Gate.count

If Active\_Gate[i] = Active\_gate[x]

Duplicate = true

If duplicate = true

Active\_gate.removeat(x)

X= x-1

File creation

There are 2 different file types that I could use, JSON and BinaryFormatter. JSON was a good choice because of its interlanguage capabilities. It’s also excellent for readability but lacks in the security part. For BinaryFormatter it’s secure and hard to edit but lacks the readability. It’s smaller in file size and will be better for large sets of data. For testing JSON is the clear choice but as the user won’t be messing with the file and doesn’t need to the smaller file size of BinaryFormatter is the better choice.

Because this part is largely language dependent, I created each method in C#.

JSON File creation

List<data> \_data = new List<data>();

\_data.Add(new data()

{

//class variables

});

//converts the array into a format that JSON can use

string json = JsonConvert.SerializeObject(\_data.ToArray());

//uses the string file and write it into a document

System.IO.File.WriteAllText(@"D:\path.txt", json);

JSON File reading

//Read file and puts it in r

using (StreamReader r = new StreamReader("file.json"))

{

//Puts everything into a string

string json = r.ReadToEnd();

//puts everything back into list of the class

List<Item> items = JsonConvert.DeserializeObject<List<Item>>(json);

}

BinaryFormatter Creation

//FileStream reads the file

FileStream fs = new FileStream("File.dat", FileMode.Create);

BinaryFormatter bf = new BinaryFormatter();

// bf converts the file into binary format

bf.Serialize(fs, list);

//This ends the creation

fs.Close();

BinaryFormatter reader

List<Class\_data> items = new List<Class\_data>();

//opens the file and splits it into each class of the list

var bformatter = new BinaryFormatter();

using (Stream stream = File.Open("File.dat", FileMode.Open))

{

//Reformat the file to fit the list of objects

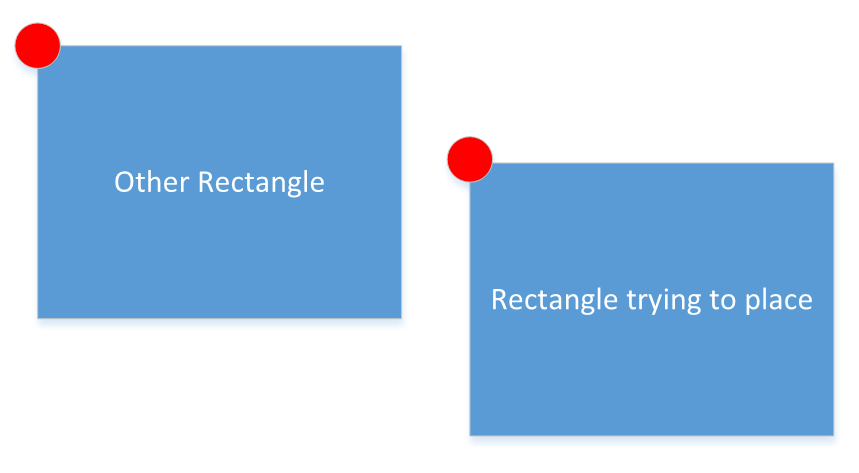
items = (List<Class\_data>)bformatter.Deserialize(stream); }

Validation of gates and lines

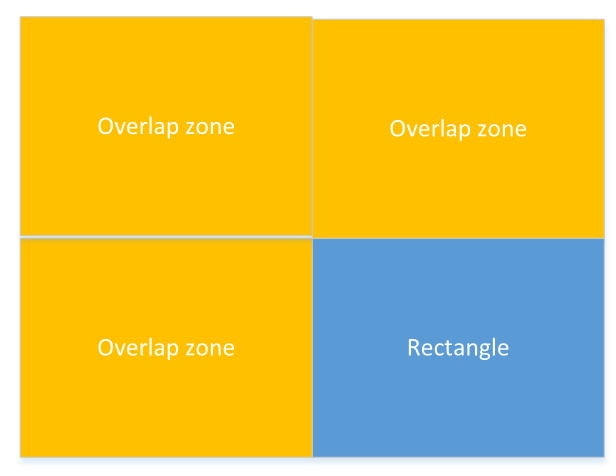
There are lots of checks that are required to make a valid circuit. Some are done while creation is taking place and others are when you try and run the simulation.

Rectangle overlapping

For making sure you don’t have overlapping gates I used the top left position of each rectangle and then some maths to work it out.



Then you just create an area around the rectangle you’re trying to place and see if there is another rectangle with their left point in that zone.



This way you can easily check the area by seeing if the coordinate lands in the rectangle. However, there are different size gates. So, the area that needs to be calculated all needs to take a variable of the size of the gate your comparing it to.

Line validation

The lines had a lot of flexibility in what they can do. I also wanted the user to feel like they can control them like they want. This meant they were required to go to the right input slot and output slot. This just meant that some checks on the mouse position based on the gate needed to take place. As the first click on a gate is always the output this was simple due to only the special gate having more than 1 output slot. For the input all but two had 2 input slots. So, I just had to split the gate height in half and check if it’s above or bellow to connect it. An additional check is required as I needed to see if the slot that was trying to be used was already taken. If they were then the program would just go through each slot starting from the top down and check for an available slot. If there wasn’t then it would be removed.

There was check to make sure that there isn’t a link between the same gate (output can’t go into the input). This just meant I needed to check a new local variable to store the ID of the gate that is being linked and when the line is connected to an output gate that ID didn’t equal the variable.

# Data structure

Data structure: explain the list with all the classes in, Explain the variable in each class

This table stores the acceptable values and description of the variables in the program.

Field: Canvas

This field will store all the variable that class Canvas will use. The Canvas class inherits the library from window Canvas class (The base class in WPF) so a lot of variables are already created.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable description | Data type | length | Validation check | Validation description | Valid data | Erroneous  data |
| Scale factor on a slider but represented as a double | Double | 0 – 2  but increments by 0.0625 from 1 | >0  <= 2 | Because the double is 32 bits I used a slider of 0.0625 which is the 5th bit of a decimal. This means the number will never become impossible to represent in binary double. | 0.9375 | 0.1  0  -1  2.0625 |
| True or false for if a mode is active or not | bool | True/False | None | It only has 2 states | True | Null |

Field: Gate class  
This is the variables for the simulation for each gate that will be stored in a list.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable description | Data type | length | Validation check | Validation description | Valid data | Erroneous  data |
| ID for the gate | Int | >-1 | Check if >-1 | This is just so that it doesn’t break search algorithms | 3 | -1  2147483649 |
| Integer for the type of gate it is | Int | From 0 to 7 | Check if <0 or >7 | There is only 8 gate in the program and by default it will be And Gate(0) so null isn’t acceptable | 0,7 | -1  8 |
| Boolean for if it is a 1 or 0 for the gate output | Bool | True/false | none | Can’t be null | false | null |
| Stores the ID of where the gate is connected to by the output | Int | >-2 | Check that it’s greater than -2 | It can be -1 as null is an acceptable answer but anything lower isn’t allowed | 0 | -2  2147483649 |

Field: Input Class

This class is to break up the gate class so that there can be an array of inputs to save having multiple groups arrays. It also makes the code easier to read as everything is under one name.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable description | Data type | length | Validation check | Validation description | Valid data | Erroneous  data |
| The bit for if the input value is a 1 or zero | Boolean | True/False | none | Can’t be null | true | NULL |
| The ID of the gate that is inputting | int | >-1 | If >-1 | Can be null | 0 | -2  2147483649 |
| Stores the type of input it is | Enum | Enum are strictly typed and only certain input will be valid. This will be based on the Enum class but will contain null, gate, button | Must contain null, gate, button |  | gate | banana |

Field: Line Class

This field stores the variables for all the lines and mainly just a way of tracking the path and connections that each gate has.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable description | Data type | length | Validation check | Validation description | Valid data | Erroneous  data |
| ID of the gate the line needs to connect to. | Int | -1> | If >-2 | -1 equals null | 100000 | -100000  2147483649 |
| This stores the output number for the line. The output number is the port from which the line comes out of. | int | 0 or 1 | >-1  <2 | Can’t be out of those ranges | 0,1 | -1 2 |

Field: Main

This field stores the variables for the state of the program and everything that is happening.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable description | Data type | length | Validation check | Validation description | Valid data | Erroneous  data |
| State of if the program is being used right now | bool | True/false | none | Can’t equal NULL | true | NULL |
| The integer value of the ID for the object that is being active | Int | >0 | If-1> | Can’t be a negative number | 0 | -1  2147483649 |
| Boolean for flipping between the 2 states the program can have | Bool | True/False | None | Can’t be null | True | Null |
| This stores what is active and where | Enum | Null, main, sub, linkSub | Strictly typed | Can’t be anything else | linksub | apple |

Field: XAML

The XAML doesn’t have variables but has attributes. There is only really one attribute that can carry over to the main program and that is tag.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable description | Data type | length | Validation check | Validation description | Valid data | Erroneous  data |
| The TAG can store the File location of the vector image that the rectangle that is being added to be stored | String | Strictly typed | Would be very hard to change the value | Hard coded | And\_Gate\_L | cherry |

# Software development model

Before deciding on the LGS as my final project I made a prototype to investigate possible solutions to the problem, this code was quite crude but I learned how to develop many of the modules far further as a result. I restarted with the mind of using OOP as the main goal. While my style and structure would work with console applications it didn’t work amazingly with WPF. This was due to lack of testing and understanding of the frame work of C#. So, I restarted again but knew I could reuse the methods and Xaml file in the old project. I used my research of custom Canvases and moved my methods into an override event handler, this made the code much better.

The overall development model of the project was the spiral SDLC model where you make prototype before you release the real thing. While this model has been inefficient, I don’t think I would’ve got to where I ended up without restarting.

While the project was one model when coding I feel I followed 2 separate models for programming. The project can be broken down into 2 aspects the GUI and the functionality part. For the GUI each feature was almost independent of each other so I could follow a feature driven development approach. This worked really well as it made testing efficient and easy. It also meant code could easily be changed and maintained due to each function not affecting others.

For the functionality part of the project I will follow the agile model but keep it tight. What I mean by this is that I know the functions that need to happen and while I could just code them, I will keep in mind and plan out for how one function works with another. This will hopefully give me enough of a plan but also flexibility for me to follow feature driven development on the small parts.

# File structure and organisation

File structure and organisation: Json for external storage, Window markup file for the gate in vector format.

I use external files two times in my project. One is to store the vector XAML file for the gate image in the rectangle UI object. The file type is Window Markup file which is basically just a text file but readable as XAML by programs. I used Microsoft expression design 4 to make the images with stock parts: line, ellipse and text. Then import it to the XAML vector format.

For storing the setup and loading of files for gate setup I’ve got the option of 2. I could use JSON or .dat/Serializable. While both are hard it depends on what I want it for. If I want security I will use .dat and serializable because it encrypts the data. If I want readability and easy changes then I would use JSON. While both aren’t massively large file types .dat is smaller.

Process for making .dat file:

# Security and integrity of data

Security and Integrity of Data: not really needed but talk about how everything is strictly typed

While my program doesn’t hold personal data and if someone hacks it, it would only affect them, I want to keep the program code and how it works a secret.

# Modular Structure

# Other

Other: diagram for the cycle of gates, How the multi-threading works.