Course Name: Software Engineering

Course Number and Section: 14:332:152

VirtualLogicLabs Technical Document

Demo 1

Github: https://github.com/SagarPhanda/VirtualLogicLabs

Date Submitted: March 31st, 2018

Group #10:
Sagar Phanda, Khalid Akash, Dhruvik Patel, Vikas Khan, Joe Cella, Yiwen Tao

Virtual Logic Labs Technical Documentation

What is Unity and why was it used for this project?

This project is built on top of the Unity Engine framework (Version 2017.3.1f1). The Unity Engine is a graphics/game library that can be interfaced with the .NET C# language, or Javascript. For this particular project, we have solely focused on implementing our program in C#. Unity's main job is to render the 2D or 3D models on to the screen (for this case, its 2D) and facilitate a platform to detect interactions between different "GameObjects".

Everything in Unity is based around the concept of 'GameObjects' which have properties or components that describe the GameObject, such as the Transform component, which contains information about the GameObjects position, rotation, and scale on the screen. Each GameObject can take in user made "Scripts" implemented as classes from the Classical Object Model. They all 'extend' or 'inherit' from Unity's own GameObject class called 'MonoBehaviour' which allows the programmer to immediately access information about the GameObject, such as all the components belonging to the GameObject, and allows Unity to add it to it's callback systems. Unity's callback system allows us to do powerful, things, such as detect collisions between two GameObjects, detect user inputs such as mouse click. In addition to the advantage of using these GameObject constructs, we can also utilize Unity's multithreading system, called Coroutines, that allowed us to do computationally heavy tasks/or animations without freezing up the main graphical thread of the application.

Unity is an event-driven system, what this means is that there is an inner loop, called the main Update loop which changes the components that each GameObject has during each frame that the loop is called. This allows programmers to dynamically take user inputs, react to changes to the overall system, and update the system based on predefined programmed instructions. What this also means is that it is the programmer's responsibility to ensure that any algorithm taking advantage of this system is efficient so that it doesn't affect the user experience as all computation must be done before moving on to the next frame/loop.

In addition to Unity's powerful libraries, we also heavily utilized the powerful .NET framework libraries that come in with C#. We heavily utilized built in data structures, such as: Dictionaries (Hash Table), Linked Lists, Array Lists, and Arrays in general.

There were several reasons we decided to use Unity over other engines. Our project is a visual, simulation based project. Many visual libraries are built on C++ due to its speed and optimization capabilities, however, due to the scope and experience of our members on this project, we wanted to really focus on our mission itself, to create an accurate simulation of Digital Logic laboratories. We wanted a language that was powerful, but also was easily buildable/testable to different target operating systems and or platforms. After a significant amount of research, we decided that Unity would be the best framework to work under. It made use of several different languages, including C# which is easily compilable to different platforms due to it's Virtual Machine system and JIT compiler (Similar to Java). It also frees the programmers worry about garbage collection, which was important to us. In addition to the language of choice, Unity came out on top due to its extensive documentation, support, popularity amongst independent developers, and most importantly, the ease of building and testing for different platforms.

Login and Users' database

We did not emphasize on our login and user database in the demo. It was coded but not fully implemented into our demo project.

The user's field table in our database has 3 attributes: User ID (UID), username and password. UID is our primary key. It automatically increments as new user is created. We have two different role of users: professors(admin) and students(normal users). Since we are implementing our DLD lab, which does not have too many professor, we assume there are 10 professors in total and they are previously set as UID 1 to 10. Students' accounts are created by the professor, so their IDs go from 10.

Entering the correct professor credentials will direct the user to admin subsystem. At the meantime, It grabs a copy of user's list and show it with a table by default. It has the following function:

Create User:
Professors can create user by pressing "Create User" button. After entering the credentials and hit OK, it will passes them to the cloud database and save it there. It will be granted a unique UID.

Delete User:
Professors can click the trash can icon to delete the according user. It will prompt a confirmation window to let user double check. However, professors cannot delete professors. Deleted user is no longer in the database and it cannot be used for next login.

Reset Password:
Users can reset their password by using reset password function. It directs user to another screen, which asks user to enter its username, old password, new password and confirm password. After finishing, new credentials will be updated accordingly on the database.

Conceptual Overview of Our Lab System

Logic Nodes:

Digital Logic Design, in the most basic form, is based on two states for every logic circuit input and output, a logic high (usually 5V), and a logic low (ground). Every Digital Logic device has this "State" property on it's input and output nodes. It was important to make a system to easily detect these, and create a reliable interaction between different nodes within the system. Using this idea, we created a GameObject called conceptually "Logic Node" that have several properties. We represent a Logic Node graphically with a small circle that has three different colors, Green for Logic High, Red for Logic Low, and White for Neutral. These states are kept by each Logic Node as an integer that is predefined statically. Outside of testing scenarios, every single Logic Node is a child of a GameObject that implements an interface called Logic Device. We will expand upon this further down in the document.

For each Logic Node, it is incredibly important to determine if it is positionally overlapping with another Logic Node, analogous to a digital logic component connecting to another digital logic component via physical contact. This is only detected when a collider component in the shape of the GameObject's collision perimeter is added to the Logic Node GameObject on object instantiation. Whenever this overlap happens between two Logic Nodes, a collision is detected by the Unity engine, and a callback function called OnTriggerEnter() is called, which notifies the programmer to react to this collision. For Logic Nodes, we notify the object that a collision has been detected recently and keep note of the Logic Node that collided. The Logic Node object does not immediately react to any collisions as the user may be actively moving the Logic Node's position. Upon reaching the next Update loop, responsibility of how to change the Logic State is given to the owning device of the Logic Node.

Devices:

Every device in this system is implemented as a Device interface that implements a few functions. The most relevant one right now is the ReactToLogic() function. What this allows us to do is it lets every logic node access it's owning device's specific logic configuration without code duplication and let's every device handle it's logic data structures in anyway that it wants. Once the ReactToLogic() function is called, all the Logic Nodes that the device owns, and have two important things checked: their Logic States, and the states for any colliding Logic Nodes.

Usually, most devices handle inputs and outputs during Logic computation in the following way:
The device's input logic are never set to a specific state, but rather they keep their state's neutral to ensure that their colliding nodes aren't influenced by their states. This is particularly important when a device is colliding with the Protoboard device as the logic calculation on a set of rows/columns in the Protoboard's Logic Nodes are based on a priority system. The priority system prioritizes a logic low, then a logic high, and finally a logic neutral. If an input Logic Node on a colliding device is set to low, and the Protoboard's set of Logic Nodes are requested to change to a logic of high, it will refuse to change as it detects that a colliding node has a state of low.
The device's output are always set to the state that it needs to be, to let the output Logic Node communicate to it's colliding Logic Node to request the owning device to change states. This is the pattern followed for all devices outside of special cases such as the Protoboard.

Typically, devices are movable (with exceptions), and the device's position based on the mouse position are controlled by the OnMouseClick() and OnMouseDrag() callback functions from Unity. Here, the current position of the device, and the offset from the device and the mouse position is calculated to move the device to the correct mouse position.

Protoboard -

The Protoboard acts as both an input and output device on all of it's Logic Nodes. A crucial data structure for the Protoboard is the hash table, due to the way the data is structured, and the speed of the retrieval of data. As specific rows, and specific columns of Logic Nodes have the relationship of representing one Logic Nodes, they need to represented in a way where a list of Logic Nodes is retrieved for a specific column/row request. A Hash Table is the perfect data structure for this as a key can be assigned to every set of related nodes, and a List (Array) data structure of Logic Node GameObjects can be assigned as the value for the key value pair. During the ReactToLogic() function, the relevant list of of Logic Nodes can be received by knowing the calling Logic Node's key in a time complexity of O(1). As mentioned earlier, a priority system is used to update the list of Logic Node's state as a set must all have the same state. All colliding Logic Node's with the set are checked for their Logic States, and based on a priority system, the set as a whole is assigned one logic state. The priority system assigns the logic low first, logic high second, and assigns logic neutral last. The protoboard is an immovable device as for the protoboard to be clickable, it would need to have a Box

Collider component for the mouse input callbacks to be registered. However, since the Logic Nodes contained inside it also have Colliders, the Unity engine has a difficult time distinguishing which GameObject is colliding with which other GameObject. We decided to remove the movable functionality from the protoboard due to this.

Chips (74LS00 (NAND), 74LS04 (INVERTER), 74LS08 (AND), 74LS32 (OR)) -

The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.

LED -

The LED is an important movable device, and similar to the 'chips', they can be snapped. The LED takes in two inputs by detecting collision on both of it's 'legs'. If the shorter leg's collided Logic Node has a state of logic low and the longer leg's collided Logic Node has a state of logic high, then the LED has a state of being "On". This also means that the sprite of the LED is modified to show that it is emitting a light source. In every other situation, the LED is in the state of being "Off" and has a sprite that reflects that. The importance of the LED doesn't only come from being a good debugging device for the user, but also is important in a technical aspect as it is used to check the input and output states of the overall circuit of the lab. This will be further expanded on.

Switch -

The switch is another important movable device, similar to the LED. It contains three Logic Nodes, two of which are inputs, and one of which is an output. It can be toggled up or down, and will prioritize the output to reflect the top, or bottom Logic Node input. Similar to the LED, they play an important role when analyzing the built in circuit as they can be designated as an overall input to the system.

Wires -

Wires are generated using the 'w' key on the keyboard, or can be accessed via the dropdown menu. After initializing them, a click on a Logic Node is listened for on a callback, after the first click, the Wire 'Line' is rendered to follow the mouse from the specified Logic Node. Every click after ward creates an inflection point for the wire, if the clicked point is not a Logic Node, in which another 'Line' is rendered from that point to the mouse. Only when a Logic Node is clicked is the wire sequence is done executing within the Wire object's Update function, and two new Logic Nodes are created at each end of the wire. The wire follows a similar priority system to that of the Protoboard by analyzing the colliding Logic Nodes of both ends of the wire.

Magnifying Glass-

The magnifying glass allows the user to access information about components that may be useful while conducting the lab, such as datasheets, instructions, and animations. This information can be uncovered by hovering the magnifying glass over the object of interest (Logic Chips, LED, Switch, Wires, etc). When this occurs, a collision is detected between the magnifying glass and the object of interest, which triggers a function in the *MagnifierBehavior* script. This function passes a collision parameter which provides information on the collision, allowing the corresponding information to be displayed. When the magnifying glass exits a collision with an object, a different function in the *MagnifierBehavior* script executes, which in turn hides the corresponding information.

Trash -

Since an unlimited number of components can be generated by the user, there must be a system of deletion in order to prevent unnecessary pile up of unwanted components. The trash feature allows the user to delete components by dragging and dropping the component to be deleted over the trash icon. Once a component is hovering over the trash icon, the resulting collision triggers a function that alerts the user by allowing the trash icon to glow. As the trash icon is glowing, an update loop checks if the user has let go of the left mouse button. If so, then the component is destroyed and memory is freed.

Power Supply -

The power supply device is unique as it does nothing with the ReactToLogic() interface even though it implements it. This device's only job is to continually set it's Logic Node's to logic high, and logic low through the Update loop. This way, even if there is a mechanism that changes it's states (it will be discussed further down), the Power Supply will force it's logic state to its proper value and permeate the value through the collision system through the rest of the circuit.

Prelab / Postlab -

For the prelabs and postlabs, the user is asked to enter inputs in the respective fields of the k-map. Once those inputs are entered, the user hits the "Check" button. For the text fields, the program restricts inputs to single digit integers. This minimizes any issues that could be caused by inappropriate spacing or text input in the fields. When the user hits the "Check" button, they system checks the user's inputs, which are all stored in variables, compared to the answers that should have been inputted. If the answers inputted and the correct answers are the same, a message appears saying and it goes onto the next lab within 5 seconds. If the user inputs an incorrect answer, a message will display to the user informing them to try again.

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire

- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

CheckerTagScript Class

Namespace
: global

Base Types

- MonoBehaviour

Syntax

```
public class CheckerTagScript : MonoBehaviour
```

Properties

Name	Value	Summary
Type	string	

Methods

Name	Value	Summary
GetCollidingObject()	GameObject	
isSnapped()	bool	

Search Types...

global Namespace

Class Types

Class	Summary
ANDGate	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
CheckerTagScript	
Equipment	Equipment object that holds information about all Equipment available to use and lists them on a Dropdown menu. Allows the user to create new equipment from clicking the Dropdown.
EquipmentTests	
GradingCONSTANTS	
InputChecker	
IntegrationTesting	

Class	Summary
INVGate	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be ‘snapped’ to 14 other nodes, typically this means a collision between all of the chip’s Logic Nodes and the Protoboard’s Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node’s collided Logic Node’s position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node’s states are taken, and the output is set.
Kmap	
LabOneGrader	
LEDScript	The LED is an important movable device, and similar to the ‘chips’, they can be snapped. The LED takes in two inputs by detecting collision on both of it’s ‘legs’. If the shorter leg’s collided Logic Node has a state of logic low and the longer leg’s collided Logic Node has a state of logic high, then the LED has a state of being “On”. This also means that the sprite of the LED is modified to show that it is emitting a light source. In every other situation, the LED is in the state of being “Off” and has a sprite that reflects that. The importance of the LED doesn’t only come from being a good debugging device for the user, but also is important in a technical aspect as it is used to check the input and output states of the overall circuit of the lab. This will be further expanded on.
LogicManager	
LogicNode	We represent a Logic Node graphically with a small circle that has three different colors, Green for Logic High, Red for Logic Low, and White for Neutral. These states are kept by each Logic Node as an integer that is predefined statically. Outside of testing scenarios, every single Logic Node is a child of a GameObject that implements an interface called Logic Device. We will expand upon this further down in the document. For each Logic Node, it is incredibly important to determine if it is positionally overlapping with another Logic Node, analogous to a digital logic component connecting to another digital logic component via physical contact.This is only detected when a collider component in the shape of the GameObject’s collision perimeter is added to the Logic Node GameObject on object instantiation. Whenever this overlap happens between two Logic Nodes, a collision is detected by the Unity engine, and a callback function called OnTriggerEnter() is called, which notifies the programmer to react to this collision.For Logic Nodes, we notify the object that a collision has been detected recently and keep note of the Logic Node that collided.The Logic Node object does not immediately react to any collisions as the user may be actively moving the Logic Node’s position. Upon reaching the next Update loop, responsibility of how to change the Logic State is given to the owning device of the Logic Node.
Login	
MagnifierBehavior	
	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be ‘snapped’ to 14 other nodes, typically this means a collision between all of the chip’s Logic Nodes and the Protoboard’s Logic Nodes are detected simultaneously. Once the chip detect that

Class	Summary
NANDGate	all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
ORGate	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
PostLab1	
PowerSupplyScript	
PreLab1	
ProtoboardObject	The Protoboard acts as both an input and output device on all of it's Logic Nodes. A crucial data structure for the Protoboard is the hash table, due to the way the data is structured, and the speed of the retrieval of data. As specific rows, and specific columns of Logic Nodes have the relationship of representing one Logic Nodes, they need to represented in a way where a list of Logic Nodes is retrieved for a specific column/row request. A Hash Table is the perfect data structure for this as a key can be assigned to every set of related nodes, and a List (Array) data structure of Logic Node GameObjects can be assigned as the value for the key value pair. During the ReactToLogic() function, the relevant list of of Logic Nodes can be received by knowing the calling Logic Node's key in a time complexity of O(1). As mentioned earlier, a priority system is used to update the list of Logic Node's state as a set must all have the same state. All colliding Logic Node's with the set are checked for their Logic States, and based on a priority system, the set as a whole is assigned one logic state. The priority system assigns the logic low first, logic high second, and assigns logic neutral last. The protoboard is an immovable device as for the protoboard to be clickable, it would need to have a Box Collider component for the mouse input callbacks to be registered. However, since the Logic Nodes contained inside it also have Colliders, the Unity engine has a difficult time distinguishing which GameObject is colliding with which other GameObject. We decided to remove the movable functionality from the protoboard due to this.
SquareboxObject	
StudentSubsystem	
Switch	The switch is another important movable device, similar to the LED. It contains three Logic Nodes, two of which are inputs, and one of which is an output. It can be toggled up or down, and will prioritize the output to reflect the top, or bottom Logic Node input. Similar to the LED, they play an important role when analyzing the built in circuit as they can be designated as an overall input to the

Class	Summary
Toast	system.
TrashBehavior	Facilitates deletion of GameObjects that represent equipments in the Virtual Logic Lab.
TruthTable	
Wire	Wires are generated using the ‘w’ key on the keyboard, or can be accessed via the dropdown menu. After initializing them, a click on a Logic Node is listened for on a callback, after the first click, the Wire ‘Line’ is rendered to follow the mouse from the specified Logic Node. Every click after ward creates an inflection point for the wire, if the clicked point is not a Logic Node, in which another ‘Line’ is rendered from that point to the mouse. Only when a Logic Node is clicked is the wire sequence is done executing within the Wire object’s Update function, and two new Logic Nodes are created at each end of the wire. The wire follows a similar priority system to that of the Protoboard by analyzing the colliding Logic Nodes of both ends of the wire.
WireInflection	

Enum Types

Enum	Summary
LOGIC	
SOURCE	

Interface Types

Interface	Summary
LogicInterface	

On This Page

Search Types...

- Namespaces
- global

global Namespace

Class Types

Class	Summary
ANDGate	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
CheckerTagScript	
Equipment	Equipment object that holds information about all Equipment available to use and lists them on a Dropdown menu. Allows the user to create new equipment from clicking the Dropdown.
EquipmentTests	
GradingCONSTANTS	
InputChecker	
IntegrationTesting	

Class	Summary
INVGate	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
Kmap	
LabOneGrader	
LEDScript	The LED is an important movable device, and similar to the 'chips', they can be snapped. The LED takes in two inputs by detecting collision on both of it's 'legs'. If the shorter leg's collided Logic Node has a state of logic low and the longer leg's collided Logic Node has a state of logic high, then the LED has a state of being "On". This also means that the sprite of the LED is modified to show that it is emitting a light source. In every other situation, the LED is in the state of being "Off" and has a sprite that reflects that. The importance of the LED doesn't only come from being a good debugging device for the user, but also is important in a technical aspect as it is used to check the input and output states of the overall circuit of the lab. This will be further expanded on.
LogicManager	
LogicNode	We represent a Logic Node graphically with a small circle that has three different colors, Green for Logic High, Red for Logic Low, and White for Neutral. These states are kept by each Logic Node as an integer that is predefined statically. Outside of testing scenarios, every single Logic Node is a child of a GameObject that implements an interface called Logic Device. We will expand upon this further down in the document. For each Logic Node, it is incredibly important to determine if it is positionally overlapping with another Logic Node, analogous to a digital logic component connecting to another digital logic component via physical contact.This is only detected when a collider component in the shape of the GameObject's collision perimeter is added to the Logic Node GameObject on object instantiation. Whenever this overlap happens between two Logic Nodes, a collision is detected by the Unity engine, and a callback function called OnTriggerEnter() is called, which notifies the programmer to react to this collision.For Logic Nodes, we notify the object that a collision has been detected recently and keep note of the Logic Node that collided.The Logic Node object does not immediately react to any collisions as the user may be actively moving the Logic Node's position. Upon reaching the next Update loop, responsibility of how to change the Logic State is given to the owning device of the Logic Node.
Login	
MagnifierBehavior	
	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that

Class	Summary
NANDGate	all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
ORGate	The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.
PostLab1	
PowerSupplyScript	
PreLab1	
ProtoboardObject	The Protoboard acts as both an input and output device on all of it's Logic Nodes. A crucial data structure for the Protoboard is the hash table, due to the way the data is structured, and the speed of the retrieval of data. As specific rows, and specific columns of Logic Nodes have the relationship of representing one Logic Nodes, they need to represented in a way where a list of Logic Nodes is retrieved for a specific column/row request. A Hash Table is the perfect data structure for this as a key can be assigned to every set of related nodes, and a List (Array) data structure of Logic Node GameObjects can be assigned as the value for the key value pair. During the ReactToLogic() function, the relevant list of of Logic Nodes can be received by knowing the calling Logic Node's key in a time complexity of O(1). As mentioned earlier, a priority system is used to update the list of Logic Node's state as a set must all have the same state. All colliding Logic Node's with the set are checked for their Logic States, and based on a priority system, the set as a whole is assigned one logic state. The priority system assigns the logic low first, logic high second, and assigns logic neutral last. The protoboard is an immovable device as for the protoboard to be clickable, it would need to have a Box Collider component for the mouse input callbacks to be registered. However, since the Logic Nodes contained inside it also have Colliders, the Unity engine has a difficult time distinguishing which GameObject is colliding with which other GameObject. We decided to remove the movable functionality from the protoboard due to this.
SquareboxObject	
StudentSubsystem	
Switch	The switch is another important movable device, similar to the LED. It contains three Logic Nodes, two of which are inputs, and one of which is an output. It can be toggled up or down, and will prioritize the output to reflect the top, or bottom Logic Node input. Similar to the LED, they play an important role when analyzing the built in circuit as they can be designated as an overall input to the

Class	Summary
Toast	system.
TrashBehavior	Facilitates deletion of GameObjects that represent equipments in the Virtual Logic Lab.
TruthTable	
Wire	Wires are generated using the ‘w’ key on the keyboard, or can be accessed via the dropdown menu. After initializing them, a click on a Logic Node is listened for on a callback, after the first click, the Wire ‘Line’ is rendered to follow the mouse from the specified Logic Node. Every click after ward creates an inflection point for the wire, if the clicked point is not a Logic Node, in which another ‘Line’ is rendered from that point to the mouse. Only when a Logic Node is clicked is the wire sequence is done executing within the Wire object’s Update function, and two new Logic Nodes are created at each end of the wire. The wire follows a similar priority system to that of the Protoboard by analyzing the colliding Logic Nodes of both ends of the wire.
WireInflection	

Enum Types

Enum	Summary
LOGIC	
SOURCE	

Interface Types

Interface	Summary
LogicInterface	

Generated by Wyam

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior

- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

ANDGate Class

Summary

The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.

Namespace
: global

Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class ANDGate : MonoBehaviour, LogicInterface
```

Fields

Name	Constant Value	Summary
LOGIC_DEVICE_ID	74LS08_AND_NODE_	*static*

Methods

Name	Value	Summary
GetLogicDictionary ()	Dictionary <TKey, TValue>	

Name	Value	Summary
IsDeviceOn()	bool	Checks if pin 7 and pin 14 is connected to ground and logic high respectively. This is checked whenever a new state change is requested to be reacted to.
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	If the chip is snapped, react to the input logics and set the outputs to the correct states. Otherwise, clear the chips.
SetSnapped(bool)	void	

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior

- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

Equipment Class

Summary

Equipment object that holds information about all Equipment available to use and lists them on a Dropdown menu. Allows the user to create new equipment from clicking the Dropdown.

Namespace
  global
Base Types

- MonoBehaviour

Syntax

```
public class Equipment : MonoBehaviour
```

Fields

Name	Constant Value	Summary
dropDown		
equipmentNames		

Methods

Name	Value	Summary
CallBackWithParameter(int)	void	Call back from clicking the Equipment Dropdown menu
Start()	void	Lists the available equipment to the Dropdown menu

---

Search Types...

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

EquipmentTests Class

Namespace
    global
Base Types

- object

Syntax

```
public class EquipmentTests
```

Methods

Name	Value	Summary
ANDTest()	IEnumerator	
INVTest()	IEnumerator	
LEDTest()	IEnumerator	
Magnifier()	IEnumerator	
NANDTest()	IEnumerator	
ORTest()	IEnumerator	
PowerSupplyTest()	IEnumerator	
ProtoboardTest()	IEnumerator	
SwitchTest()	IEnumerator	
Trash()	IEnumerator	

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

GradingCONSTANTS Class

Namespace
    global
Base Types

- object

Syntax

```
public static class GradingCONSTANTS
```

Fields

Name	Constant Value	Summary
INPUT		*static*
OUTPUT		*static*

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

InputChecker Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class InputChecker : MonoBehaviour
```

Fields

Name	Constant Value	Summary
mainMenu		

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE

- Interface Types
- LogicInterface

IntegrationTesting Class

Namespace
    global
Base Types

  - object

Syntax

```
public class IntegrationTesting
```

- [TruthTable]
- [Wire]
- [WireInflection]
- Enum Types
- [LOGIC]
- [SOURCE]
- Interface Types
- [LogicInterface]

INVGate Class

Summary

The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.

Namespace
: global

Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class INVGate : MonoBehaviour, LogicInterface
```

Fields

Name	Constant Value	Summary
LOGIC_DEVICE_ID	74LS04_INV_NODE_	*static*

Methods

Name	Value	Summary
GetLogicDictionary ()	Dictionary <TKey, TValue>	

Name	Value	Summary
IsDeviceOn()	bool	Checks if pin 7 and pin 14 is connected to ground and logic high respectively. This is checked whenever a new state change is requested to be reacted to.
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	If the chip is snapped, react to the input logics and set the outputs to the correct states. Otherwise, clear the chips.
SetSnapped(bool)	void	

Generated by Wyam

Search Types...

Kmap Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class Kmap : MonoBehaviour
```

Fields

Name	Constant Value Summary
button	
inputfield0000	
inputfield0001	
inputfield0010	
inputfield0011	
inputfield0100	
inputfield0101	
inputfield0110	
inputfield0111	
inputfield1000	
inputfield1001	
inputfield1010	
inputfield1011	
inputfield1100	
inputfield1101	
inputfield1110	
inputfield1111	
text	

Search Types...

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

LabOneGrader Class

Namespace
: global

Base Types

- MonoBehaviour

Syntax

```
public class LabOneGrader : MonoBehaviour
```

Fields

Name	Constant Value	Summary
Finish		

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior

- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

LEDScript Class

Summary

The LED is an important movable device, and similar to the ‘chips’, they can be snapped. The LED takes in two inputs by detecting collision on both of it’s ‘legs’. If the shorter leg’s collided Logic Node has a state of logic low and the longer leg’s collided Logic Node has a state of logic high, then the LED has a state of being “On”. This also means that the sprite of the LED is modified to show that it is emitting a light source. In every other situation, the LED is in the state of being “Off” and has a sprite that reflects that. The importance of the LED doesn’t only come from being a good debugging device for the user, but also is important in a technical aspect as it is used to check the input and output states of the overall circuit of the lab. This will be further expanded on.

Namespace
    global
Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class LEDScript : MonoBehaviour, LogicInterface
```

Fields

Name	Constant Value	Summary
SNAPPED		

Methods

Name	Value	Summary
GetLEDNodeGnd ()	GameObject	
GetLEDNodeVCC ()	GameObject	

Name	Value	Summary
isLEDON()	bool	Checks if the LED is in the on state, meaning that the ground node is connected to a ground state, and the vcc node is connected to a logic high.
OnMouseUp()	void	Checks if the chip is snapped when the Mouse click is released to snap it into position.
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	If the chip is snapped, react to the input logics and set the outputs to the correct states. Otherwise, clear the chips.

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

LogicManager Class

Namespace
  global
Base Types

  - MonoBehaviour

Syntax

```
public class LogicManager : MonoBehaviour
```

Methods

Name	Value Summary
AddGameObject(GameObject)	void
AddGameObject(List<GameObject>)	void
RemoveGameObject(GameObject)	void
RemoveGameObject(List<GameObject>)	void
ResetAllLogic()	void

- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

LogicNode Class

Summary

We represent a Logic Node graphically with a small circle that has three different colors, Green for Logic High, Red for Logic Low, and White for Neutral. These states are kept by each Logic Node as an integer that is predefined statically. Outside of testing scenarios, every single Logic Node is a child of a GameObject that implements an interface called Logic Device. We will expand upon this further down in the document. For each Logic Node, it is incredibly important to determine if it is positionally overlapping with another Logic Node, analogous to a digital logic component connecting to another digital logic component via physical contact.This is only detected when a collider component in the shape of the GameObject's collision perimeter is added to the Logic Node GameObject on object instantiation. Whenever this overlap happens between two Logic Nodes, a collision is detected by the Unity engine, and a callback function called OnTriggerEnter() is called, which notifies the programmer to react to this collision.For Logic Nodes, we notify the object that a collision has been detected recently and keep note of the Logic Node that collided.The Logic Node object does not immediately react to any collisions as the user may be actively moving the Logic Node's position. Upon reaching the next Update loop, responsibility of how to change the Logic State is given to the owning device of the Logic Node.

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class LogicNode : MonoBehaviour
```

Fields

Name	Constant Value	Summary
logic_state		

Methods

Name	Value	Summary
GetCollidingNode()	GameObject	
GetLogicNode()	GameObject	

Name	Value	Summary
GetLogicState()	int	
RequestStateChange (int)	void	
SetLogicState(int)	void	Sets logic state of this particular component. logic_id MUST be set before this method is called Accepted values are LOGIC.HIGH(int = 10) and LOGIC.LOW(int = -10)
SetLogicState WithoutNotification (int)	void	Sets logic state of this particular component. logic_id MUST be set before this method is called Accepted values are LOGIC.HIGH(int = 10) and LOGIC.LOW(int = -10) DOES NOT set '{RecentStateChange}' that gets checked in the update() method
SetOwningDevice (LogicInterface)	void	

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

Login Class

Namespace

global

Base Types

- MonoBehaviour

Syntax

```
public class Login : MonoBehaviour
```

Fields

Name	Constant Value Summary
BtnExit	
BtnLogin	
BtnResetPassword	
InputPassword	
InputUsername	

On This Page

Syntax

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE

- Interface Types
- LogicInterface

MagnifierBehavior Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class MagnifierBehavior : MonoBehaviour
```

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior

- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

NANDGate Class

Summary

The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.

Namespace
    global
Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class NANDGate : MonoBehaviour, LogicInterface
```

Fields

Name	Constant Value	Summary
LOGIC_DEVICE_ID	74LS00_NAND_NODE_	*static*

Methods

Name	Value	Summary
GetLogicDictionary()	Dictionary<TKey, TValue>	

Name	Value	Summary
IsDeviceOn()	bool	Checks if pin 7 and pin 14 is connected to ground and logic high respectively. This is checked whenever a new state change is requested to be reacted to.
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	If the chip is snapped, react to the input logics and set the outputs to the correct states. Otherwise, clear the chips.
SetSnapped(bool)	void	

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior

- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

ORGate Class

Summary

The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detect that all 14 nodes are collided with, and the user lifts the mouse, the OnMouseUp() callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.

Namespace
    global
Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class ORGate : MonoBehaviour, LogicInterface
```

Fields

Name	Constant Value	Summary
LOGIC_DEVICE_ID	74LS32_OR_NODE_	*static*

Methods

Name	Value	Summary
GetLogicDictionary ()	Dictionary <TKey, TValue>	

Name	Value	Summary
IsDeviceOn()	bool	Checks if pin 7 and pin 14 is connected to ground and logic high respectively. This is checked whenever a new state change is requested to be reacted to.
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	If the chip is snapped, react to the input logics and set the outputs to the correct states. Otherwise, clear the chips.
SetSnapped(bool)	void	

Generated by Wyam

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

PostLab1 Class

Namespace
: global

Base Types

- MonoBehaviour

Syntax

```
public class PostLab1 : MonoBehaviour
```

Fields

Name	Constant Value Summary
button	
inputfield0000	
inputfield0001	
inputfield0010	
inputfield0011	
inputfield0100	
inputfield0101	
inputfield0110	
inputfield0111	
inputfield1000	
inputfield1001	
inputfield1010	
inputfield1011	
inputfield1100	
inputfield1101	
inputfield1110	
inputfield1111	
text	

Search Types...

PowerSupplyScript Class

Namespace
: global

Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class PowerSupplyScript : MonoBehaviour, LogicInterface
```

Methods

Name	Value	Summary
GetGndNode()	GameObject	
GetVccNode()	GameObject	
ReactToLogic(GameObject)	void	
ReactToLogic(GameObject, int)	void	

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

PreLab1 Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class PreLab1 : MonoBehaviour
```

Fields

Name	Constant Value Summary
button	
inputfield000	
inputfield001	
inputfield010	
inputfield011	
inputfield100	
inputfield101	
inputfield110	
inputfield111	
text	

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire

- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

ProtoboardObject Class

Summary

The Protoboard acts as both an input and output device on all of it’s Logic Nodes. A crucial data structure for the Protoboard is the hash table, due to the way the data is structured, and the speed of the retrieval of data. As specific rows, and specific columns of Logic Nodes have the relationship of representing one Logic Nodes, they need to represented in a way where a list of Logic Nodes is retrieved for a specific column/row request. A Hash Table is the perfect data structure for this as a key can be assigned to every set of related nodes, and a List (Array) data structure of Logic Node GameObjects can be assigned as the value for the key value pair. During the ReactToLogic() function, the relevant list of of Logic Nodes can be received by knowing the calling Logic Node’s key in a time complexity of O(1). As mentioned earlier, a priority system is used to update the list of Logic Node’s state as a set must all have the same state. All colliding Logic Node’s with the set are checked for their Logic States, and based on a priority system, the set as a whole is assigned one logic state. The priority system assigns the logic low first, logic high second, and assigns logic neutral last. The protoboard is an immovable device as for the protoboard to be clickable, it would need to have a Box Collider component for the mouse input callbacks to be registered. However, since the Logic Nodes contained inside it also have Colliders, the Unity engine has a difficult time distinguishing which GameObject is colliding with which other GameObject. We decided to remove the movable functionality from the protoboard due to this.

Namespace
    global
Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class ProtoboardObject : MonoBehaviour, LogicInterface
```

Methods

Name	Value	Summary
GetGameObjectByID (string)	List<T>	
GetNodeDictionary()	Dictionary <TKey, TValue>	

Name	Value	Summary
ReactToLogic (GameObject)	void	Interface method from LogicInterface.cs that allows the protoboard to react to any changes to its logic nodes. This method is called from OnMouseUp() function, so it regulates mouse toggles
ReactToLogic (GameObject, int)	void	Interface method from LogicInterface.cs that allows the protoboard to react to any changes to its logic nodes. This method is called from OnMouseUp() function, so it regulates mouse toggles

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

SquareboxObject Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class SquareboxObject : MonoBehaviour
```

Methods

Name	Value	Summary
OnTriggerEnter2D(Collider2D)	void	

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

StudentSubsystem Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class StudentSubsystem : MonoBehaviour
```

Fields

Name	Constant Value Summary
Lab1Button	
Lab2Button	
Lab3Button	
SandboxMode	

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire

- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

Switch Class

Summary

The switch is another important movable device, similar to the LED. It contains three Logic Nodes, two of which are inputs, and one of which is an output. It can be toggled up or down, and will prioritize the output to reflect the top, or bottom Logic Node input. Similar to the LED, they play an important role when analyzing the built in circuit as they can be designated as an overall input to the system.

Namespace
    global
Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class Switch : MonoBehaviour, LogicInterface
```

Methods

Name	Value	Summary
GetBotNode()	GameObject	
GetMiddleNode()	GameObject	
GetTopNode()	GameObject	
OnMouseUp()	void	
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	Method that reacts to the Logic Node's callback to handle it's Logic State
ToggleSwitch(bool)	void	Method that switches the Sprite and state of the switch from top biased, to bottom biased

---

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire

- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

Toast Class

Namespace
: global

Base Types

- MonoBehaviour

Syntax

```
public class Toast : MonoBehaviour
```

Fields

Name	Constant Value	Summary
Bg		
Instance		*static*
mText		

Methods

Name	Value	Summary
Hide()	void	
Show(string)	void	

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

TrashBehavior Class

Summary

Facilitates deletion of GameObjects that represent equipments in the Virtual Logic Lab.

Namespace
global
Base Types

- MonoBehaviour

Syntax

```
public class TrashBehavior : MonoBehaviour
```

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

TruthTable Class

Namespace
: global

Base Types

- MonoBehaviour

Syntax

```
public class TruthTable : MonoBehaviour
```

Fields

Name	Constant Value	Summary
button		
inputField00		
inputField01		
inputField10		
inputField11		
text		

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire

- WireInflection
- Enum Types
- LOGIC
- SOURCE
- Interface Types
- LogicInterface

Wire Class

Summary

Wires are generated using the ‘w’ key on the keyboard, or can be accessed via the dropdown menu. After initializing them, a click on a Logic Node is listened for on a callback, after the first click, the Wire ‘Line’ is rendered to follow the mouse from the specified Logic Node. Every click after ward creates an inflection point for the wire, if the clicked point is not a Logic Node, in which another ‘Line’ is rendered from that point to the mouse. Only when a Logic Node is clicked is the wire sequence is done executing within the Wire object’s Update function, and two new Logic Nodes are created at each end of the wire. The wire follows a similar priority system to that of the Protoboard by analyzing the colliding Logic Nodes of both ends of the wire.

Namespace
    global
Interfaces

- LogicInterface

Base Types

- MonoBehaviour

Syntax

```
public class Wire : MonoBehaviour, LogicInterface
```

Methods

Name	Value	Summary
ReactToLogic (GameObject)	void	
ReactToLogic (GameObject, int)	void	Reacts to the Logic of the two ends of the wires and sets the overall Logic state of the wire based on a priority system that prioritizes LOGIC.LOW first, LOGIC.HIGH next, and finally LOGIC.INVALID. This method is only usable after activeBeingPlaced is toggled false.
ToggleLineColor ()	void	Callback helper function that toggles line color when the user clicks on the wire

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types
- LOGIC
- SOURCE

- Interface Types
- LogicInterface

WireInflection Class

Namespace
    global
Base Types

- MonoBehaviour

Syntax

```
public class WireInflection : MonoBehaviour
```

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

LOGIC Enum

Namespace
: global

Interfaces

- IComparable
- IFormattable
- IConvertible

Base Types

- object
- ValueType
- Enum

Syntax

```
public enum LOGIC
```

Fields

Name	Constant Value	Summary
HIGH	10	*static*
INVALID	0	*static*
LOW	-10	*static*

On This Page

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

- LOGIC
- SOURCE
- Interface Types
- LogicInterface

SOURCE Enum

Namespace
: global

Interfaces

- IComparable
- IFormattable
- IConvertible

Base Types

- object
- ValueType
- Enum

Syntax

```
public enum SOURCE
```

Fields

Name	Constant Value	Summary
EQUIPMENT	10	*static*
MOUSE	11	*static*

Search Types...

- Namespace
- global
-
- Class Types
- ANDGate
- CheckerTagScript
- Equipment
- EquipmentTests
- GradingCONSTANTS
- InputChecker
- IntegrationTesting
- INVGate
- Kmap
- LabOneGrader
- LEDScript
- LogicManager
- LogicNode
- Login
- MagnifierBehavior
- NANDGate
- ORGate
- PostLab1
- PowerSupplyScript
- PreLab1
- ProtoboardObject
- SquareboxObject
- StudentSubsystem
- Switch
- Toast
- TrashBehavior
- TruthTable
- Wire
- WireInflection
- Enum Types

LogicInterface Interface

Namespace
: global

Implementing Types

- ProtoboardObject
- Wire
- Switch
- ORGate
- INVGate
- NANDGate
- LEDScript
- ANDGate
- PowerSupplyScript

Syntax

```
public interface LogicInterface
```

Methods

Name	Value Summary
ReactToLogic(GameObject)	void
ReactToLogic(GameObject, int)	void