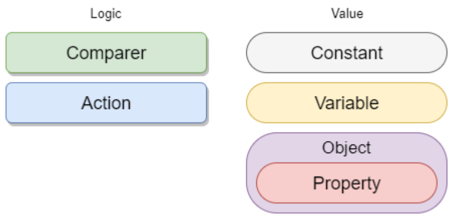
LOGIC - OVERVIEW

This document gives you the overview for creating logic with the GOTO Logic system. More detailed information can be found in the other documents, tutorials and scripting reference.

COMPARER & ACTION

Simplest form of logic is a single **comparer** & **action** pair.

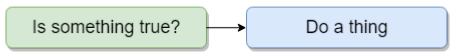
- If X is true, do a thing Y.



CONDITION & SEQUENCE

It is also possible to stack these logic blocks.

- When stacked, **comparers** form a **condition**.
- When stacked, **actions** form a **sequence**.

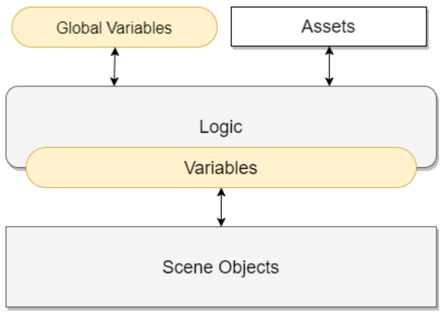


VARIABLES

Logic is nothing without data. Data flows in and out of logic through **variables**.

Variable has double meaning. It can be a final storage or pass-through.

Access to scene objects always happens via variable, however access to assets can be direct.



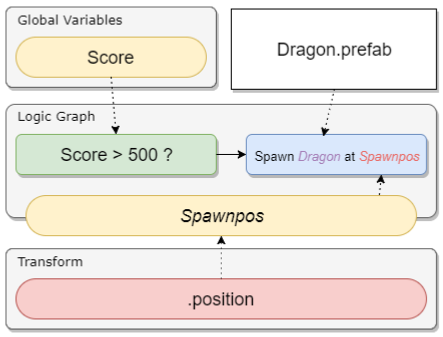
PUTTING IT ALL TOGETHER

Here is a logic graph that uses the basic features.

If a global score is bigger than 500, a dragon is instantiated in the position of a `Transform` in a scene.

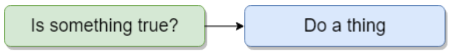
The comparer compares global variable (`Score`) into a constant (`500`).

The action instantiates asset (`Dragon.prefab`) to a position defined by a variable (`Spawnpos`). The variable itself is pass-through and is connected to a `Transform` components `.position` property.



ACTION SEQUENCE

Action Sequence is a simplistic **Logic Graph**. In fact, it is so simple that it does not require a graph editor. All the editing happens in the Unity inspector.



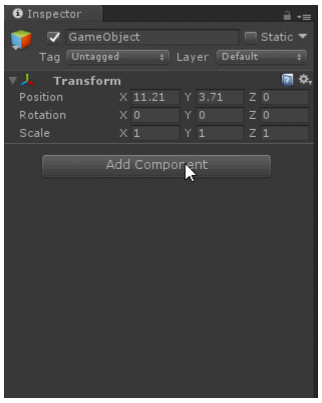
In other words, it is a single **sequence** of **actions** combined with an optional trigger **condition**. Condition means one or many **comparers** combined.

When you need something more complex, consider using a graph instead.

GETTING STARTED

Create or select a game object from the scene and add **Logic Player** component from the *Add Component* button.

Click the *Create New* button and choose *Sequence* from the dropdown.



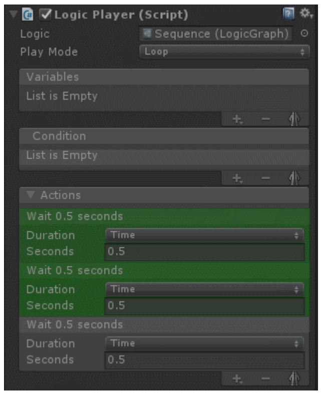
Inspector
GameObject Static
Tag Untagged Layer Default
Transform
Position X 11.21 Y 3.71 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

WHAT IS IT?

It’s a list of actions, started from top to bottom, triggered by passing the (optional) condition.

Next action doesn't wait for previous action to finish, unless explicitly forced by a `Wait` block. All actions are guaranteed to finish, before next iteration of the sequence can start.

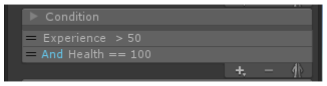
Action Sequence is an asset. To execute it, you always need a Logic Player Component.



Logic Player (Script)
Logic Sequence (LogicGraph)
Play Mode Loop
Variables
List is Empty
Condition
List is Empty
Actions
Wait 0.5 seconds
Duration Time
Seconds 0.5
Wait 0.5 seconds
Duration Time
Seconds 0.5
Wait 0.5 seconds
Duration Time
Seconds 0.5

PLAYBACK CONDITION

Comparers are key building block for the Core system. They can be added to the Action Sequence as a condition for the execution start.



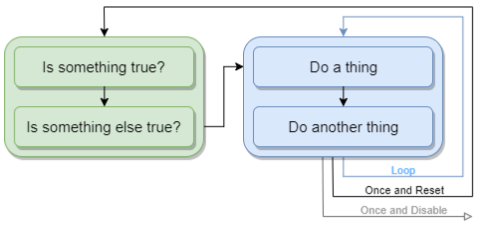
Condition
Experience > 50
And Health == 100

One can use multiple comparers to create it's playback condition (example: `Experience > 100 AND Health > 200`).

If the condition if left empty, the sequence will start automatically when the game object is enabled/created.

PLAYMODE

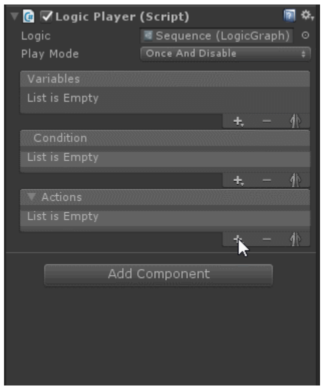
- `Once and Disable` = Sequence plays once and then the playback is disabled.
- `Once and Reset` = Sequence plays once, resets it's state and is ready to play again. Same as `Loop`, but playback condition is checked on every round.
- `Loop` = Sequence plays and loops indefinitely. Playback condition is checked only on first iteration.



ADDING AND REMOVING ACTIONS

Actions are added to the sequence by pressing the plus sign on the bottom right. This will open up menu with all actions currently in the project, separated into different categories.

Removing actions can be done by first selecting the action from the list, and then clicking the minus button on the bottom right.

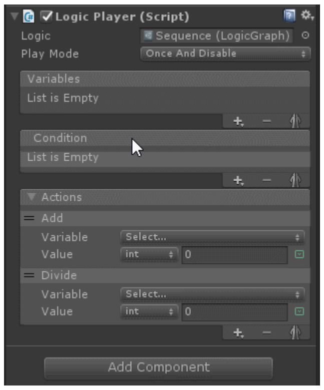


Logic Player (Script)
Logic: Sequence (LogicGraph)
Play Mode: Once And Disable
Variables
List is Empty
Condition
List is Empty
Actions
List is Empty
Add Component

HIDE AND SHOW ACTION PROPERTIES

Hiding properties will give better overview of the whole sequence. Most of the labels state what they are doing.

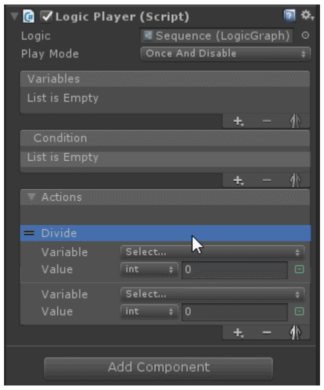
For example “Wait 10 seconds”, “Play Open Door Animation”, which gives a good overview of what the sequence doing.



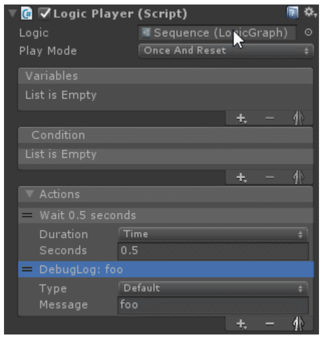
Logic Player (Script)
Logic: Sequence (LogicGraph)
Play Mode: Once And Disable
Variables
List is Empty
Condition
List is Empty
Actions
Add
Variable: Select...
Value: int 0
Divide
Variable: Select...
Value: int 0
Add Component

REORGANIZE ACTIONS

Actions can be dragged up and down on the list. This will change the playback order for the sequence.


Logic Player (Script)
Logic: Sequence (LogicGraph)
Play Mode: Once And Disable
Variables
List is Empty
Condition
List is Empty
Actions
Divide
Variable: Select...
Value: int 0
Variable: Select...
Value: int 0

CONVERTING TO GRAPH


Logic Player (Script)
Logic: Sequence (LogicGraph)
Play Mode: Once And Reset
Variables
List is Empty
Condition
List is Empty
Actions
Wait 0.5 seconds
Duration: Time
Seconds: 0.5
DebugLog: foo
Type: Default
Message: foo

In GOTO Logic, there is only single logic asset type: The Logic Graph.

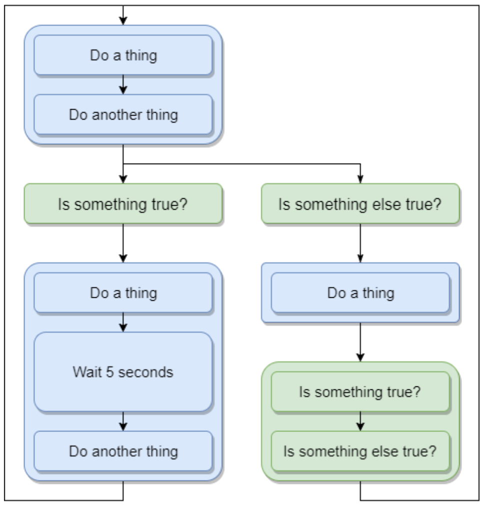
Action Sequence is actually a Logic Graph asset with a special flag. The flag means, that the logic is simple enough to be edited straight in the Unity inspector. Graphs without the flag always need to be edited via the separate editor window.

If you have an existing Action Sequence and feel like expanding it into something more complex, it is possible to convert it into a graph from the Logic Player inspector context menu.

LOGIC GRAPH

Logic Graph is a reusable state machine asset that defines game logic.

To execute a Logic Graph, you need a **Logic Player** component.

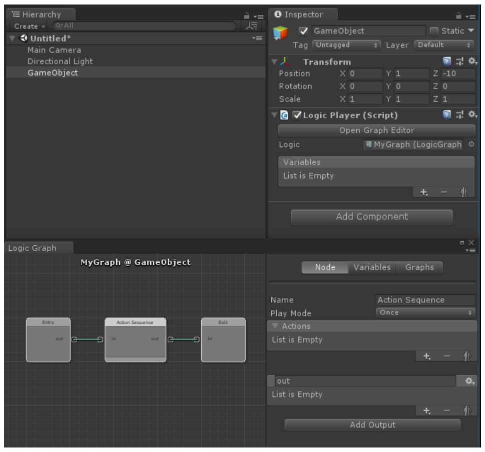


GETTING STARTED

To get started:

1. Add a Logic Player component to your game object
2. Click the *Create New* button and choose *Graph* from the dropdown
3. Choose a location to store your Logic Graph asset

After creating the asset, Logic Graph editor window will open up automatically


Hierarchy

Untitled*
Main Camera
Directional Light
GameObject

Inspector

GameObject
Tag Untagged Layer Default

Transform
Position X 0 Y 1 Z -10
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Logic Player (Script)
Open Graph Editor
Logic MyGraph (LogicGraph
Variables
List is Empty
Add Component

Logic Graph
MyGraph @ GameObject
Node Variables Graphs
Name Action Sequence
Play Mode Once
Actions
List is Empty
out
List is Empty
Add Output

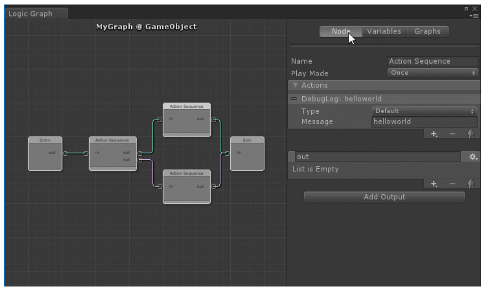
LOGIC GRAPH EDITOR

Editor is split into two views. Left side is called *Graph View* and right side is called *Node Inspector*.

Graph View shows all logic **nodes** the graph contains, and connections between those nodes. In this view you can create, delete and rearrange nodes and connections.

Node Inspector shows properties of the currently selected node. Right side view has tabs called *Node*, *Variables* and *Graphs*.

When a state is selected from node view, its content can be modified in the *Node* tab. This allows renaming the node, editing


Logic Graph
MyGraph @ GameObject
Node Variables Graphs
Name Action Sequence
Play Mode Once
Actions
DebugLog: helloworld
Type Default
Message helloworld
out
List is Empty
Add Output

sequence in it, and conditions for transitioning into another node.

Variables tab shows all graphs **variables**. Those same variables can be used in all states of the graph, and are exposed and overridable in the scene using the Logic Player component.

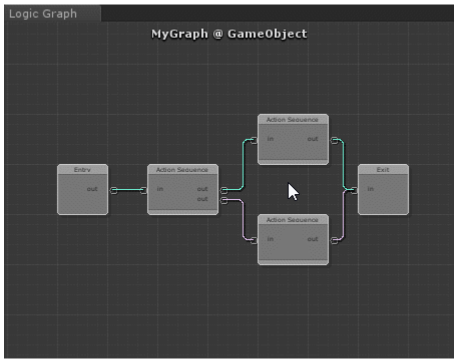
Graph tab shows all LogicGraphs in the project for easier navigation between them.

CONTROLS

Graph View can be panned around by dragging with right mouse button, or with left mouse while holding `Alt` key. Zoomed in and out with mouse wheel, or by double clicking on node or empty space.

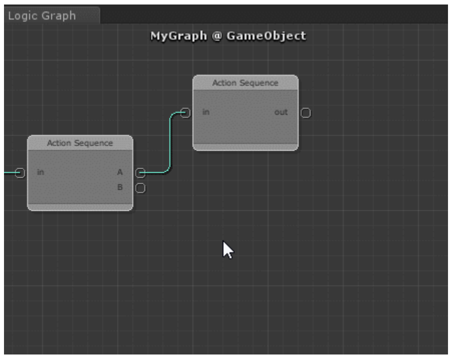
Graph View is zoomed in and out with mouse wheel, or by double clicking on node or empty space.

Pressing `F` key will frame the view to contain all selected nodes.



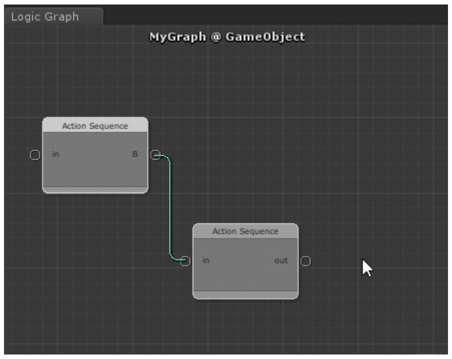
CREATING AND REMOVING NODES

New states can be created by dragging from right side connection box of a node to a empty space. States can be moved around by dragging from the title bar. Right mouse click opens up the context menu for deleting or duplicating the state.



DUPLICATE, COPY & PASTE NODES

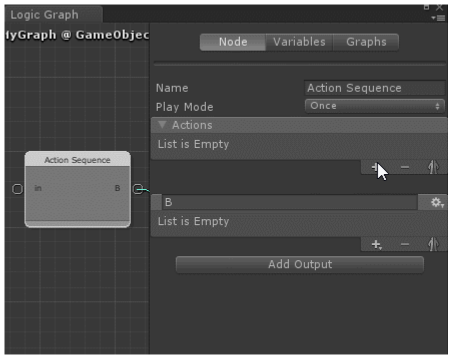
Selected nodes can be copied between graphs through context menu or by using keyboard commands (`ctrl+c`, `ctrl+v`, `ctrl+d`)



ACTIONS INSIDE STATE

Each node contains an list of **actions** to play when inside a state. This works in the same way as the **Action Sequence (/Manual/Page?**

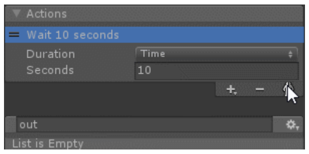
name=ActionSequence&category=wiki\Logic) and has all the same features.



Name Action Sequence
Play Mode Once
Actions
List is Empty
B
List is Empty

DUPLICATE ACTIONS

Actions can be duplicated by selecting one from the list and pressing the duplicate icon on bottom right of the action list.

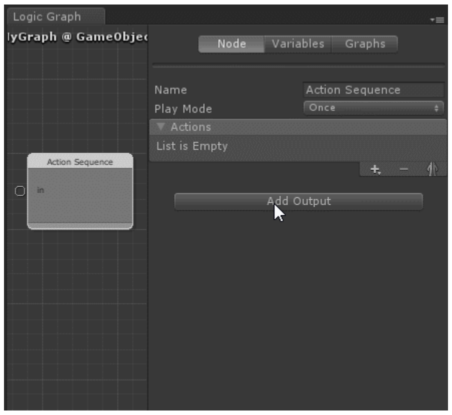


Wait 10 seconds
Duration Time
Seconds 10
out
List is Empty

COMPARERS

Comparers are used to check if we should transition to another state. These could be that player has pressed a keyboard button, hit a trigger collider in the scene, or some variable is over a certain value. One or more comparers batched together form a **condition**.

State can have multiple conditions, and those can be seen on the Node view as output slots. You can connect the states together by setting where playback should transition to when the condition is true.

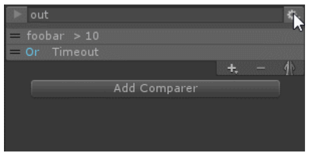


Name Action Sequence
Play Mode Once
Actions
List is Empty

DUPLICATE, COPY & PASTE COMPARERS

Single comparer can be duplicated the same way as actions.

Entire condition (list of one or more comparers), can be copied between nodes and graphs using the context menu on the top-right corner of list.



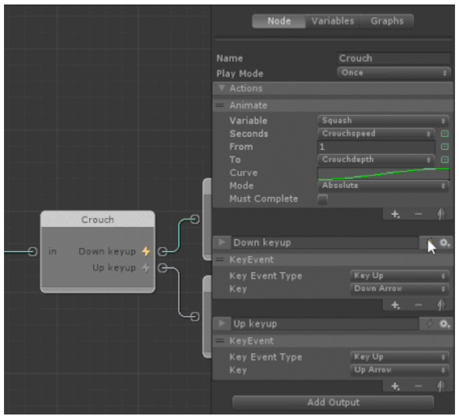
out
foobar > 10
Or Timeout

INTERRUPTIBLE OUTPUTS

Condition is a output slot of the node. By default the output can be used only after all the **actions** of the node have finished.

However, an output can be made to interrupt by enabling the yellow lightning icon button.

When interruptible output triggers, all the actions currently running are interrupted and the remaining actions that haven't started yet, are executed and interruped immediately.



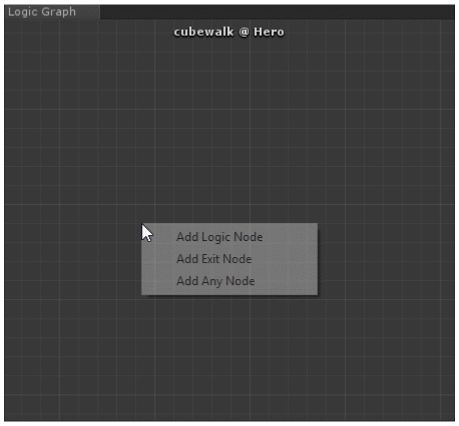
Name Crouch
Play Mode Once
Actions
Animate
Variable Squash
Seconds Crouchspeed
From 1
To Crouchdepth
Curve
Mode Absolute
Must Complete
Down keyup
KeyEvent
Key Event Type Key Up
Key Down Arrow
Up keyup
KeyEvent
Key Event Type Key Up
Key Up Arrow

ANY NODE

Any is a special node mainly for event handling.

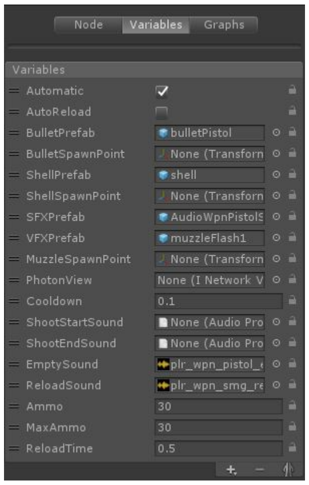
The outputs of Any node will trigger regardless of which node is currently active.

Any node is always empty and doesn't have any **actions**.



VARIABLE TAB

You can add, remove and rearrange variables on the list. Variables are used inside the graph, but can also be exposed and overriden through the Logic Player component.



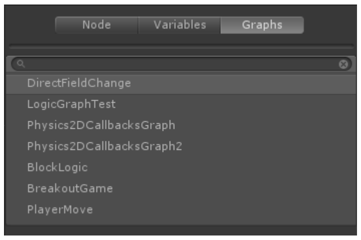
Variables

Automatic	✓
AutoReload	
BulletPrefab	bulletPistol
BulletSpawnPoint	None (Transforn
ShellPrefab	shell
ShellSpawnPoint	None (Transforn
SFXPrefab	AudioWpnPistolS
VFXPrefab	muzzleFlash1
MuzzleSpawnPoint	None (Transforn
PhotonView	None (I Network V
Cooldown	0.1
ShootStartSound	None (Audio Pro
ShootEndSound	None (Audio Pro
EmptySound	plr_wpn_pistol_(
ReloadSound	plr_wpn_smg_re
Ammo	30
MaxAmmo	30
ReloadTime	0.5

GRAPHS TAB

List of all Logic Graph assets on the project.

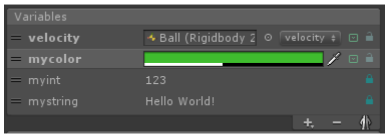
Selecting a graph from the list will open the graph, and also ping the asset on the project view.



DirectFieldChange
LogicGraphTest
Physics2DCallbacksGraph
Physics2DCallbacksGraph2
BlockLogic
BreakoutGame
PlayerMove

VARIABLES

Value and **variable** are the two concepts to handle data in GOTO Logic system.



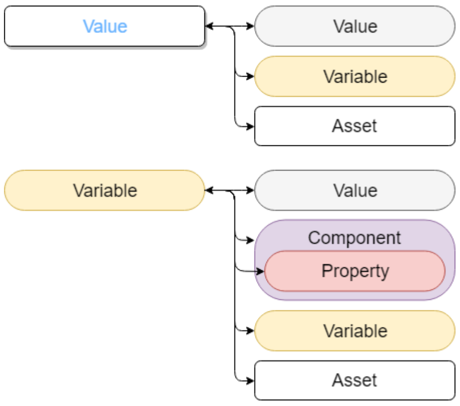
Variables

velocity	Ball (Rigidbody 2 · velocity
mycolor	
myint	123
mystring	Hello World!

Actions like `Instantiate` or `Set` are meaningless, unless there is something to instantiate or something to set value of.

VALUE SOURCES

Whenever **action** or **comparer** needs to input or output data, it exposes a value field.



Value field can refer to:

- Value (`500`)
- Variable (`score`)
- Asset (`Dragon.prefab`)

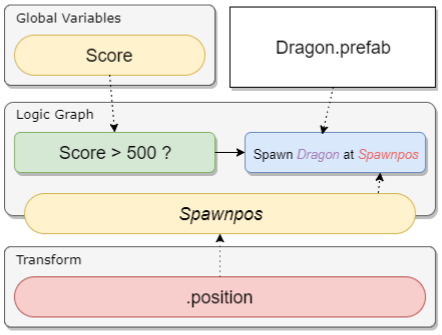
Whenever **Logic Graph** needs to input or output data, it exposes a variable.

Variable can refer to:

- Value (`500`)
- Object (`Rigidbody`)
- Property (`Rigidbody.velocity`)
- Asset (`Dragon.prefab`)
- Another variable

EXAMPLE

Here is a Logic Graph that uses most of the features.

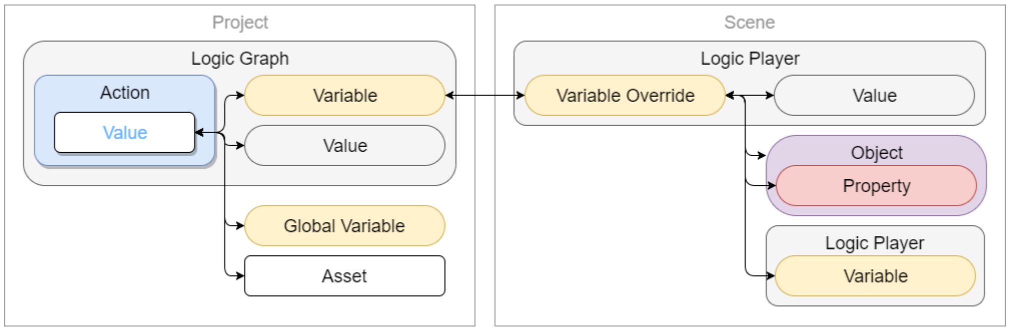


If a global score is bigger than 500, a dragon is instantiated in the position of a `Transform` in a scene.

The comparer compares global variable (`Score`) into a constant (`500`).

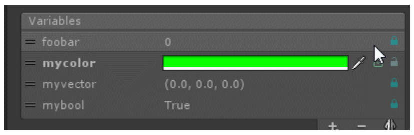
The action instantiates asset (`Dragon.prefab`) to a position defined by a variable (`Spawnpos`). The variable itself is **pass-through** and is connected to a `Transform` components `.position` property.

PROJECT VS. SCENE



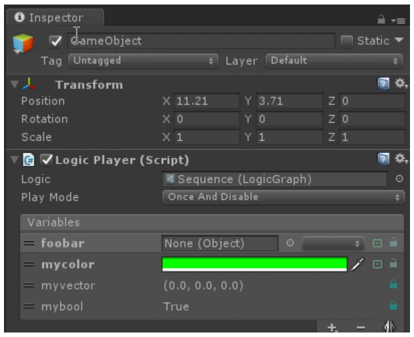
LINKING THINGS

Most simple way to use a variable is to simply input a value straight up (Example: 123). That said, often you want to link the variable to other things, too.



Variables
foobar 0
mycolor
myvector (0.0, 0.0, 0.0)
mybool True

This can be achieved by clicking the green icon next to the variable field.



Inspector
GameObject Static
Tag Untagged Layer Default
Transform
Position X 11.21 Y 3.71 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Logic Player (Script)
Logic Sequence (LogicGraph)
Play Mode Once And Disable
Variables
foobar None (Object)
mycolor
myvector (0.0, 0.0, 0.0)
mybool True

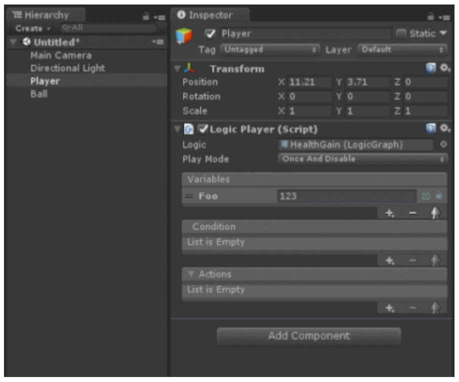
If you want to link to an `Object`, just set the field as *Value* (clicking the green icon) and drag in any `Object` (Example: `RigidBody`).

DRAG'N'DROP AUTO VARIABLE

Creating a new variable can also be achieved with single drag'n'drop.

You can drag any object straight into variable list.

Note: You can drag'n'drop into the logic graph editor window too!

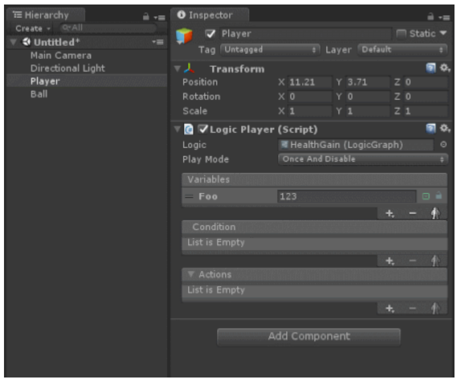


Hierarchy
Create
Untitled*
Main Camera
Directional Light
Player
Ball

Inspector
Player Static
Tag Untagged Layer Default
Transform
Position X 11.21 Y 3.71 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Logic Player (Script)
Logic HealthGain (LogicGraph)
Play Mode Once And Disable
Variables
Foo 123
Condition
List is Empty
Actions
List is Empty
Add Component

You can also drag any object straight into the value field to create a new variable and a connect it to the field.

Note: You can drag'n'drop into the logic graph editor window too!

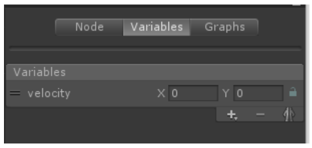


Hierarchy
Create
Untitled*
Main Camera
Directional Light
Player
Ball

Inspector
Player Static
Tag Untagged Layer Default
Transform
Position X 11.21 Y 3.71 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Logic Player (Script)
Logic HealthGain (LogicGraph)
Play Mode Once And Disable
Variables
Foo 123
Condition
List is Empty
Actions
List is Empty
Add Component

LOGIC GRAPH VARIABLES

When defined in the Logic Graph, a variable is internal to the graph and can't be seen from the outside.



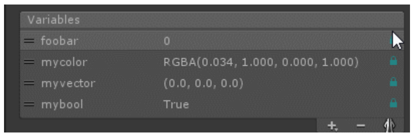
Variables
velocity X 0 Y 0

Linking variables from the Logic Graph to the outside world:

1. Designer defines a variable in the *Variables* tab of the logic graph editor.
2. Variable is exposed in the Logic Player component inspector, clicking the lock icon.
3. Exposed variable can now be linked to other things like object, property or even another variable.

Here you see the `velocity` variable connected to the `Rigidbody2D.velocity` of the `GameObject`.

VARIABLE LOCK



Variables
foobar 0
mycolor RGBA(0.034, 1.000, 0.000, 1.000)
myvector (0.0, 0.0, 0.0)
mybool True

The lock icon next to variable field defines availability.

- If the lock is closed, the variable is only used internally by the graph
- If the lock is open, the variable is exposed and linkable externally

Only full Logic Graph can have internal variables. For an Action Sequence, there are no locks available as all variables are public.

CODERS

Here is how value fields are handled from the coder point of view.

When defined this way, instead of classic C# field, the designer gets the power to link the value to other things.

See **Extending Logic (/Manual/Page?name=ExtendingLogic&category=wiki\Logic)** for more examples.

```
// Defining value field
public Value myValue = new Value(typeof(i
nt));

// Writing value
myValue.SetValue(123, variableContext);

// Reading value
int current = myValue.GetValue<int>(varia
bleContext);
```

REUSABILITY

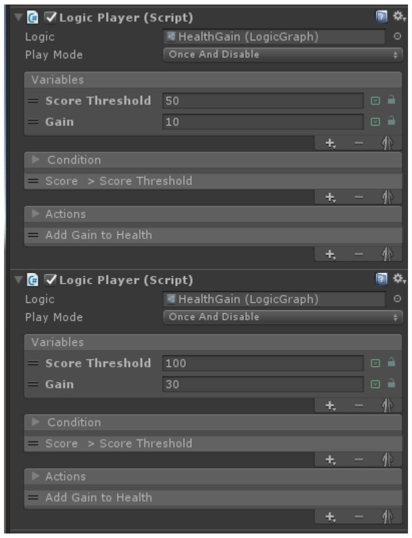
Variables are something that logic asset exposes to the player instance. This means that logic in the Logic Graph asset can be reused in several Logic Player components.

Demonstration of reusability. Here is a logic asset `HealthGain.asset`, used in two different Logic Player components.

Setup:

- Global variables called `Score` and `Health`.
- Local variables called `Score Threshold` and `Gain`.

Logic:



Logic Player (Script)
Logic HealthGain (LogicGraph)
Play Mode Once And Disable
Variables
Score Threshold 50
Gain 10
Condition
Score > Score Threshold
Actions
Add Gain to Health

Logic Player (Script)
Logic HealthGain (LogicGraph)
Play Mode Once And Disable
Variables
Score Threshold 100
Gain 30
Condition
Score > Score Threshold
Actions
Add Gain to Health

- When `Score` goes over `Score Threshold`, the `Gain` is added to `Health`.

The same logic is reused with different parameters:

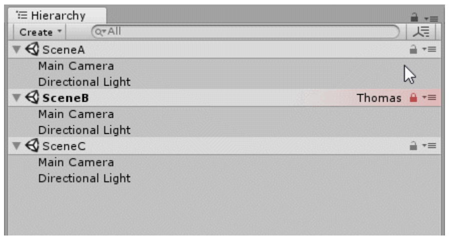
- 1st Player: When `Score` goes over `50`, `Health` gets `+10`.
- 2nd Player: When `Score` goes over `100`, `Health` gets `+30`.

SCENE LOCK

Scene locking service prevents multiple users from working in the same scenes. Soft locking allows unlocking the scene by other users to prevent problems caused by accidental locking, or forgetting to unlock after done working on it. Can be hooked into version control system to automate the process. Can also be hooked into chat software such as Slack to provide out-of-Unity notifications.

OPERATION

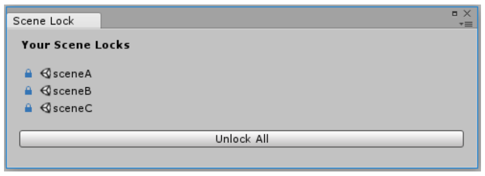
1. Click a grey icon next to the scene name to lock the scene to you. The icon will change color.
2. Other users locks will display as red with the name of the locker. Don't try to edit and submit those scenes.
3. If you've determined it's safe to work on someone else's scene (e.g. the person just forgot to unlock), you can steal the lock by clicking the red lock icon.
4. Once you are done, unlock your scene(s) by clicking the lock once again.



SCENE LOCK WINDOW

Open Scene Lock Window from `GOTO->Windows->Scene Lock` main menu item.

Here you can see all the locks you have open and also close them all at once using the Close All button. You can also close single scenes here by clicking on the individual scene icon.



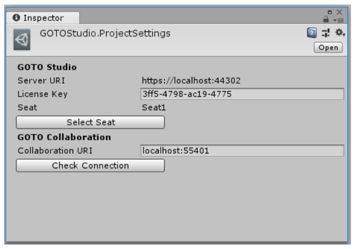
INSTALLING MULTIPLE USERS

Only first user needs to create an account. Rest can just import GOTO Studio and a settings file with license key into their local project.

The settings for your client can be seen in `Assets/GOTOStudio/Settings/GOTOStudio.ProjectSettings.asset`. After the first installation, where you created an GOTO Studio account, you should see your license key here.

Push this file into your source control or manually insert it into everyones local project with GOTO Studio.

Once this is done, each will get the Create Seat window popup. After creating the seat, they too can start locking scenes.



GOTO Studio
Server URI https://localhost:44302
License Key 3ff5-4798-ac19-4775
Seat Seat1
GOTO Collaboration
Collaboration URI localhost:55401