How to Create Kodu Tutorial Levels

I should start with a warning: creating a tutorial in Kodu is difficult. It involves directly editing the level’s Xml file and is very intolerant of errors. Over time we’ll hopefully be able to add features to Kodu to make this easier, but for now it’s not for the faint of heart.

Still with me? Good. Let’s get started. First, an overview of how tutorials work and an introduction into some terminology:

# Overview

### Modes

At any given time, a user working in Kodu will be in one of several dozen *modes*. Each mode is a combination of where in the app the user is, whether they are using keyboard or gamepad input, what tool they are using, etc. As the user performs actions the current mode changes. The idea of modes is important to tutorials because we need to be able to guide the user to the correct mode for performing each step of the tutorial. For instance, in order to add an apple to the world the user needs to be editing the world and using the object tool. Part of the tutorial system in Kodu is a navigation system that will guide the user from wherever they currently are to the proper mode for the next step of the tutorial. As a tutorial writer you need to be aware of this system because you will be specifying what mode you want the system to guide the user to each step along the way.

### Steps

A tutorial consists of a sequence of *steps*. Steps can be *modal* or *non-modal*. A modal step is one where information is displayed to the user and they must click “continue” to move on. During the display of a modal step, the user cannot interact with the rest of Kodu in any way. These appear to the user as a full screen dialog. Non-modal steps display context sensitive instructions to the user while the user is interacting with Kodu. These appear to the user in the white bar across the top of the screen.

All steps will be able to specify an optional “target mode”. This will be the mode that the step needs the user to navigate to before the tutorial can continue. The user will be guided to the correct mode via the navigation system mentioned above.

### Modal Steps

Modal steps are displayed to the user as something like a standard dialog box that pops up in the center of the screen and displays textual information to the user. The user then clicks on *Continue* to move on to the next step of the tutorial. In general every tutorial should start with one of these to explain what the tutorial is all about. These are also good for reinforcing concepts just learned.

It’s entirely possible to create some tutorials that consist solely of modal steps. Tutorials of this type will be more like guided tours rather than interactive lessons.

### Non-modal Steps

Non-modal steps are displayed using the top 20% of the screen. This gives us 3-4 lines of text to display to the user. The top line is generally used to display the goal of what we’re asking the user to do. For instance “Add an Apple”. The next lines are then used as needed to guide the user through this task. So even if the user gets off the path and the navigation system kicks in, the top line will still show what the higher level goal of the task is.

For a simple tutorial I expect the high level goals to be something like:

Tutorial: Teach Kodu how to find and eat an apple

Goals:

1. Add an apple
2. Add a Kodu
3. Program Kodu to move to the apple.
4. Test (run)
5. Program Kodu to eat the apple when bumped
6. Test (run)

Each of these goals could consist of several steps but these would be the high level goals we are displaying to the user. There would also be modal steps interspersed introducing the tutorial, new concepts and finally congratulating the user on completing the tutorial.

Unlike modal steps which are completed when the user presses *Continue*, non-modal steps will trigger their completion based on some change in world state. Examples include when an Apple exists in the world or the cursor is moved to the apple(edit mode) or when an apple is eaten (run mode).

# The Gory Details

The tutorial steps we’ve been talking about live in one the world’s Xml file as a series of entries called tutorialSteps. In order to add or change these we need to be able to edit that file directly. You can edit these in NotePad or any other text editor. I prefer to use ConText which is a freeware text editor that does nice formatting for Xml files. VisualStudio also works well as an editor for Xml files.

### Finding the World File

Ok, now we need to get into the details of how you would go about creating a tutorial. The first thing you need to do is to be able to find and edit the Xml file where the tutorial information lives. When you save a world in Kodu, we actually write out 4 files. The one you care about gets stored in Documents\SavedGames\Boku\Player1\Content\Xml\Levels\MyWorlds. You can use Windows Explorer to go there and see the files in the folder. (For those of you that are trivia buffs, “Boku” was the original name for Kodu but had to be changed because it was already being used by another company. Digging into the Kodu files you will still see occasional references to the old name.)

Kodu uses GUIDs for file names which means that they’re pretty much unreadable by human beings. Notice that there are 2 kinds of files in this folder. Xml files and DDS files. The DDS files are the thumbnail images for each of the levels. But it’s the Xml files that we care about. Because the file name is a GUID rather than something human readable we need a way to figure out what file it is we want to edit. The best way I’ve found to get to the correct file is to edit and save the world I want to use as a tutorial in Kodu and then sort the folder in Windows Explorer by DateModified. The world you just saved should be at the top of the list and show a modified time of just a few seconds ago. For starters I recommend loading Tutorial #1 in Kodu and then saving it with your own name. Once done you should see the file at the top of the list with the latest modification date. Go ahead and open the Xml file in your text editor and let’s look behind the curtain.

### Editing the World File

At the top of the file you should see something like this:

<?xml version="1.0"?>

<XmlWorldData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<id>2aaa2189-1824-4c98-9ac1-c4e647411f0f</id>

<name>My First Tutorial</name>

<description>This is the first tutorial.

In this tutorial you will learn how to add a Kodu &lt;kodu&gt; to your world and program him to respond to user input.</description>

<creator>scoy</creator>

<lastWriteTime>2011-09-29T23:27:25.7941523Z</lastWriteTime>

<BokuVersion>1.2.1.0</BokuVersion>

…

As you can see, I changed the name to My First Tutrial when I saved the file. The top of the file contains information about the camera and other world settings for this level. It also has information pointing to the terrain data used for the level. Scroll down about 150 lines and find the beginning of the tutorial section. This is indicated by the tag <tutorialSteps>.

Within the tutorialSteps section you will find a sequence of Steps. For levels with no tutorial all you will see is

<tutorialSteps>  
</tutorialSteps>

As I mentioned in the overview, a tutorial is represented as a sequence of steps. Here’s where you, the tutorial writer, get to define the steps for your tutorial. Let’s look at the first step for Tutorial #1:

<Step>

<Comment>how-to text for tutorials</Comment>

<DisplayMode>Modal</DisplayMode>

<Exclusion>None</Exclusion>

<TargetModeGamepad>MainMenu</TargetModeGamepad>

<TargetModeMouse>MainMenu</TargetModeMouse>

<GoalText>First Tutorial</GoalText>

<GamepadText>Welcome to the first Kodu Game Lab tutorial.

During the tutorial you will see a bar at the top of the screen with instructions for you to follow. These instructions will guide you through the actions needed to complete the tutorial.</GamepadText>

<MouseText>Welcome to the first Kodu Game Lab tutorial.

During the tutorial you will see a bar at the top of the screen with instructions for you to follow. These instructions will guide you through the actions needed to complete the tutorial.</MouseText>

</Step>

### Step Internals

As you can see, each step is made up of several fields of information. Let’s go through them one at a time.

**Comment** – This is strictly for the tutorial writer’s benefit. When Kodu loads and save the level files, any Xml-specific comments are stripped out. This field gives you a place to put notes to yourself about what this particular step is supposed to be doing. Inside Kodu it is not used for anything so you can leave it out if you choose.

**DisplayMode** – This determines whether this step is a Modal Step or a Non-Modal Step. The valid values are **Modal** and **NonModal**.

**Exclusion** – This is a bit of an advanced feature. A well written tutorial should work equally well when run using either a gamepad or keyboard/mouse input. There are, however, times when performing some task takes more steps using one input than the other. The **Exclusion** field allows you to specify that a step should be skipped depending on the input being used. The valid values are **None**, **Mouse**, and **Gamepad**. Most of the time you will want to use **None**. This means that the step is to run regardless of whether the user is using a gamepad or keyboard/mouse for input. If this is set to **Mouse** and the user is using mouse input the step will be skipped. If set to **Gamepad** and the user is using a gamepad for input, then it will be skipped.

**TargetModeGamepad, TargetModeMouse** – As explained in the description of modes, each step in the tutorial can have a target mode. Realistically these are only needed for non-modal steps. The target mode is the mode that you want the user guided to before the step is activated. For modal steps any value here is ignored. Each step has both a Gamepad and a Mouse version of the TargetMode. This is because the user will be in a different mode depending on the input device they are using. This also allows the user to switch input devices mid-stream without problems. Here is the list of valid modes, most of them should be easy to figure out (I hope):

InGame, // Running the game.

MainMenu,

OptionsMenu,

HelpScreens,

LoadLevelMenu,

CommunityMenu,

SharingHub, // Xbox only.

SharingSession, // Xbox only.

HomeMenu,

Programming,

ProgrammingHelp,

//

// Mouse modes

//

MouseHome,

MousePlay,

MouseCameraMove,

MouseEditObject,

MouseEditObjectFocus, // Object is in focus.

MousePaths,

MouseTerrainPaint,

MouseTerrainUpDown,

MouseTerrainSmoothLevel,

MouseTerrainSpikeyHilly,

MouseWater,

MouseDeleteObjects,

MouseWorldSettings,

//

// Gamepad modes

//

// ToolMenu\* is just when the user is in the tool menu. Actual

// tool modes are below. These should not be used as targets but

// are included just to give a complete set. These will show up

// when the Tutorial Debug Mode is turned on as the mode you are

// in when using the gamepad and navigating through the tool menu.

ToolMenuRunSim,

ToolMenuHomeMenu,

ToolMenuEditObject,

ToolMenuTerrainPaint,

ToolMenuTerrainUpDown,

ToolMenuTerrainSmoothLevel,

ToolMenuTerrainSpikeyHilly,

ToolMenuWater,

ToolMenuDeleteObjects,

ToolMenuWorldSettings,

GamepadEditObject,

GamepadEditObjectFocus,

GamepadTerrainPaint,

GamepadTerrainUpDown,

GamepadTerrainSmoothLevel,

GamepadTerrainSpikeyHilly,

GamepadWater,

GamepadDeleteObjects,

GamepadWorldSettings,

// Edit related modes shared by gamepad and mouse.

MaterialPicker,

BrushPicker,

WaterPicker,

AddItem,

AddItemHelp,

ObjectTweak,

A final caution about the TargetMode values. The system is not at all tolerant of typos. If you get the spelling or capitalization wrong then the level will not load in Kodu. I’ve had this happen to me. I’ll make some changes to the tutorial steps and then try and test what I’ve done in Kodu but the world no longer appears in the Load Level Menu. Inevitably what has happened is that I’ve mistyped a TargetMode value and so Kodu ends up thinking that the file is malformed and doesn’t show it in the menu. The only thing you can do is to go back and very carefully verify the values you’ve typed. Because of this I also find it much easier to just add one or two steps at a time and test the tutorial often. This way you at least have fewer possible places to look for the error.

***Debugging*** *– One of the hardest parts of debugging a tutorial is understanding which mode the user is currently in and knowing which mode you want the user to transition to. To help make this a little more obvious we’ve added a tutorial debug mode. To enable the debug mode go to Options menu off of the Main Menu. Near the bottom of the list will be the option to turn on the Tutorial mode debugging. Once turned on this will display extra information on the screen including the current mode and the target mode. This information is not formatted nicely but it is very helpful. If you find that the text is hard to read because it covers the help overlay text you can temporarily turn off the help overlay text via the Options menu or by hitting F2.*

**GoalText** – This is the text displayed to the user on the first line of a NonModal dialog. This is ignored for a Modal dialog. In general this should be the higher level goal talked about earlier.

**GamePadText, MouseText** – This is the detailed text give to the user. In some cases the text will be identical but in most cases we want to be able to customize what the user is being told depending on the input device they are using. In a Modal step this is the text that will be displayed in the dialog in the center of the screen. In a NonModal step this is the text that will be displayed under the **GoalText**. Note that for a NonModal step you don’t really have a lot of space so try and keep these fairly short. You should also test you tutorial when running at lower resolutions (800x600) to ensure that everything still fits.

The final field is exclusive to NonModal steps. This is called the **CompletionTest**. For a Modal step we know we’re done and can move on when the user clicks “Continue”. For the NonModal steps we need some way to know when that part of the task is done and we should move one. That’s where the CompletionTests come into play. Here’s an example of a NonModal step with a CompletionTest. In this case we want the user to add a Kodu to the world:

<Step>

<Comment>Add the Kodu</Comment>

<DisplayMode>NonModal</DisplayMode>

<Exclusion>None</Exclusion>

<TargetModeGamepad>AddItem</TargetModeGamepad>

<TargetModeMouse>AddItem</TargetModeMouse>

<GoalText>Adding Kodu &lt;kodu&gt; (Step 1/4)</GoalText>

<GamepadText>Select the Kodu &lt;kodu&gt; and press &lt;A&gt;</GamepadText>

<MouseText>Left click &lt;leftmouse&gt; on the Kodu &lt;kodu&gt;</MouseText>

<CompletionTest Name="BotCount" Args="Boku 1" />

</Step>

Note that the TargetModes are both AddItem. When this step is activated, if the user is in any mode other than AddItme the navigation system will guide them through the steps needed to get to AddItem. At that point the text from the tutorial mode will be displayed and the system will start looking for the CompletionTest to be satisfied. This particular test says that the step is considered complete when a single Boku (Kodu) exists in the world. The completion test uses the internal names for all the characters. There’s a complete list at the end of this document.

Currently we support five different completion tests for building tutorials. Each one is specified by its name and an argument string.

**BotCount** –Allows the counting of specified bot types.

Args:

1. bot type, uses standard bot name
2. count, number of this bot type to look for (optional, default is any)
3. min max delta (optional, modifies count "2 min" triggers on a minimum of 2

"2 max" triggers on a maximum of 2

"2 delta" triggers on a change of 2)

Examples:

* "Fruit" -- triggers if any apple exists.
* " Fruit 2" -- triggers if exactly 2 apples exist.
* " Fruit 4 min" -- triggers if 4 or more apples exist.
* " Fruit 3 delta" -- triggers when 3 apples have been added.

**Timer** – Simple timed delay. Useful for adding a quick pause in the tutorial.

Args:

1. # of seconds to delay

**Kode** – Allows testing for specific Kode sequences when programming.

Args:

1. (optional) page n - specifies the page to look on. Otherwise all pages are checked.
2. (optional) reflex n - specifies the reflex to test. Otherwise all reflexes are tested. (Reflex is internal Kodu-speak for a programming rule)
3. (optional) indent n - specifies the required indent level of the reflex. Otherwise all levels are valid.
4. list of 0 or more tile IDs. (the easiest way to figure out the tile IDs is to program what you want in Kodu and then look in the Stuff directory for the file corresponding to the Xml file you are editing. This file contains all the info about the characters and their programming. From this you can find the tile IDs. You can also look at the document ProgrammingTiles.doc which should be posted to the community web site along with this document)

Examples:

* "sensor.eyes" -- triggers if the "see" sensor is used anywhere in this bot's programming.
* "sensor.eyes filter.fruit actuator.movement selector.towardclosest" -- triggers when "see apple move toward" is programmed.
* "reflex 1 actuator.movement selector.wander" -- triggers when "move wander" is programmed in the first refex.
* "page 2 reflex 3 actuator.switchtask modifier.taskc -- triggers when the 3rd reflex of page 2 is "switch page3".
* "reflex 2 indent 1 actuator.movement selector.wander" -- triggers when "move wander" is programmed and indented 1 level.

**Terra** – Allows testing to detect the usage of terrain tools.

As with other tests, having more than one arg indicates that you want each of the triggers to fire, not just one.

Args:

1. paint -- triggers when terrain material is changed.
2. add -- triggers when terrain is added
3. delete -- triggers when terrain is deleted
4. raise -- triggers when terrain is raised
5. lower -- triggers when terrain is lowered
6. smooth -- triggers when terrain is smoothed
7. water -- triggers when water is added

**ModeChange** – Allows testing for mode change.

Args:

1. List of 1 or more modes to test for. The test triggers when the user transitions to a mode not on the list. Generally this is used to detect when the user has left a mode. For instance it can be used to give info to the user about the material picker and will trigger once the user makes a selection and leaves the material picker.

# Putting it All Together

So now you should have the information you need to create your own tutorials in Kodu. As I mentioned at the outset, it’s not going to be an easy task. What I’ve found works best is to start simple and plan out what you want to do ahead of time. As you add the steps to your tutorial test them often. Note that when you save your Xml file, Kodu will not automatically reload that version into the Load Level Menu. You must exit the menu and re-enter it. It is at this time if the world disappears that you know that your latest changes have caused a problem. Take it slow and work incrementally. Also, feel free to examine and steal from the existing tutorials.

# Internal Character Names

These are the names that the characters are referred to internally. You must use these names when testing for the existence of these characters.

Balloonbot – the balloon

bigyucca1 – the large tree

boku – Kodu

bullet

castle

cloud

coin

cruisemissile – missiles can’t be created by users. They only appear when shot.

drum

factory

fastbot – the cycle

floatbot – the boat

flyfish – the flying fish

fruit – the apple

heart

hut

jet

light

mine

puck

pushpad

rock

saucer

soccerball

sputnik

star

stick

subbot – the submarine

swimfish – the swimming fish

terracannon – the cannon

turtle

windblimp – the blimp

wisp

yucca1 – the smallest tree

yucca2 – the second smallest tree

yucca3 – the second largest tree