true STATE

finite state machine

MANUAL

VERSION 2.0.0

PIXELATED POPE

Contents

QUICK START GUIDE

1. Add the system to your project

Not all scripts need to be added, although I recommend adding them all. Bare minimum, you need to add the following:

truestate_system_init, truestate_create_state, truestate_set_default, truestate_switch

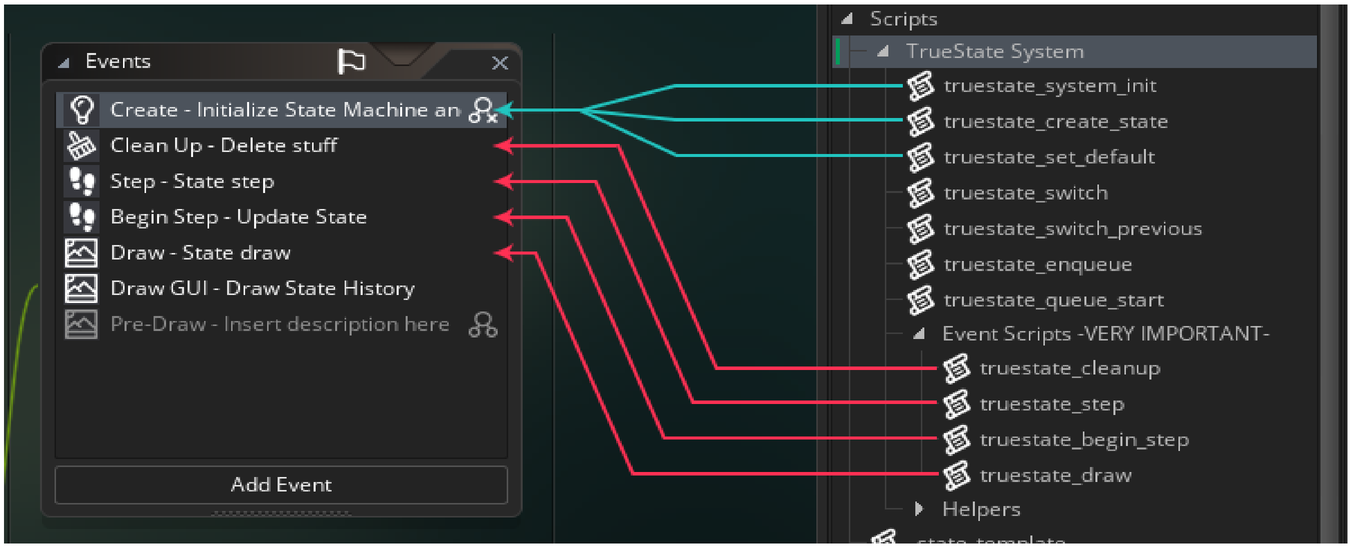
And all the scripts in the Event Scripts folder

truestate_step, truestate_draw, truestate_begin_step, truestate_cleanup

While not necessary, I also recommend adding the **_state_template** script, at least until you get the hang of how to manage your state scripts.

2. Setup your object

Call each event script **in the event that it is named for**.



It's alright to have other code in those events as well.

3. Init the system

In your create event, call **truestate_system_init** with no arguments. This will setup all the necessary variables and data structures for the system. This should only be done once for every object that will be using the system.

4. Define your states

First, create a new enum that you will be using to manage the indexes of your scripts. Then, using **truestate_create_state**, build the library of states this object will need. Of course, this library can be expanded at any time in the future, so start small and build little by little.

5. Set your default state

call **truestate_set_default** to set your default state. This will be the first state your object starts in, and if you attempt to switch to a non-existent state, the object will switch to the default instead.

6. Build your states

Starting with the template, define the step event and draw event code for this state. Use true**state_switch**() to change from one state to the next.

Congratulations, you have your first TrueState enabled object!

STATE SCRIPT STRUCTURE

Each state script is broken up into several sections using a switch statement. A switch statement isn't strictly necessary (you could re-write the script to use if/else checks if desired). The possible values for argument0 (which represent each possible section of code in a state script) are as follows:

TRUESTATE_NEW
This block of code will run once every time the state changes. It could be ran just before the step or draw event depending on when your instance was created. It will also be ran when you initially set the state as the default state in the create event of your object. So make sure to define any variables that are required by this block of code above your state definitions in your create event.

TRUESTATE_STEP
This block of code will be called once each step. Pretty straight forward.

TRUESTATE_DRAW
Similar to TRUESTATE_STEP, will be called one each step in a draw event.

TRUESTATE_FINAL
This block of code will be called once right before the state switches to a new state. It's a good place to clean up structures you may have created specifically for the current state. It's also called in the cleanup event when the instance is destroyed.

VARIABLE REFERENCE

When initializing the state machine on an object using **truestate_system_init**, many new instance variables are created to keep track of necessary data for the system to function. This section will detail them.

Engine Variables

These variables are created primarily for the system itself, and should not be altered manually.

Truestate_current_state *(real, enum)*
Keeps track of the current state. This will be the value of the state enum you used to define your state.

truestate_default_state *(*real*, enum)*
Set by truestate_set_default(), holds the value of the default state to switch to if truestate_switch is not passed an argument, or is passed an invalid state.

truestate_next_state *(*real*, enum)*
Set by truestate_switch(), holds the value of the next state to switch to at the end of this step.

truestate_previous_state *real, enum*
Holds the enum for the previous state. Can be useful if you need to know which state preceeded the current state.

truestate_state_script *(real, script id)*
Holds the id of the current state script

truestate_switch_locked *(boolean)*
Set by truestate_switch. If set to true, all other state switches will be ignored. For more details, see *truestate_switch* script reference.

truestate_stack_locked *(boolean)*
Used by truestate_switch_previous. Prevents adding the current state to the state stack when returning to a previous state.

truestate_reset_state *{boolean)*
Used to allow the system to switch to a state it is already in; restarting the state as if it was just switched to.

truestate_in_queue *(Boolean)*
Tells the system whether a queued series of states is currently being ran. If so, any state switch that tells the object to switch to a state not in the queue will be ignored in favor of continuing the queue uninterrupted.

Engine Data Structures

The system is built using several data structures. These structures are all cleaned up in the truestate_cleanup() script, and their contents probably should not be altered manually.

truestate_map *(ds_map <enum, script>)*
A map that holds the script to run for each state created.

truestate_names *(ds_map <enum, string>)*
Serves as a reference for what states are called. If you do not provide a name upon state creation, the name of the script assigned to the state will be used.

truestate_stack *(ds_stack <enum>)*
The "history" of all state changes. Holds the enum values of each state that has been used since the last time the history was cleared.

truestate_queue *(ds_queue <enum>)*
Holds the enum for all states queued up as part of a state queue.

truestate_vars *(ds_map <variable>)*
This ds_map can be utilized in a wide variety of ways. Its core purpose is to allow you to keep track of values that will persist between steps but will only be relevant to the current state and -in some cases- the beginning of the next state.

For example, say you have a charge attack that requires the player to hold button down for a set amount of time for different effects. Instead of creating a new "charge_hold_time" variable, you just use truestate_vars[? “charge time”].

It's a bit strange at first, but every time you go back to your create event to initialize a variable that only one state cares about, ask yourself if you could just use state_vars instead.

Useful Variable(s)

These variables are provided for you to use, ideally within a state script.

truestate_timer *(real)*
This variable counts up from 0 for each step that is spent in the current state. Very useful for timing things within a state.

TRUESTATE SCRIPT REFERENCE

The system comes with several scripts that can be useful for managing your object’s states. In this section we will run through each and detail their usage.

System Scripts

In most cases, these scripts should only be called once per object in a specific event. They should not be used inside a state script.

truestate_system_init
Arguments: none
Return: N/A
Initializes the true state system for the object. Should be called in the *create event* before any other TrueState scripts. Should only ever be called once per object, else a memory leak will be created.

truestate_create_state
Arguments: StateEnum <*real, enum*>*,* Script <*real, script id*>**,** *StateName <string, optional: defaults to script name>*
Return: N/A
Creates a new state for the current object to use. Should only be called in the *create event* after truestate_system_init. Pass the enum for the state, and the script to execute for that state’s step and draw event. Optionally, you can give the state a name for debugging purposes. The name will default to the name of the script passed. The StateEnum should be a unique value per object. Using the same enum with a different script will overwrite the previously defined state.

truestate_set_default
Arguments: StateEnum <*real, enum*>
Sets the default state for the object. Should be called in the *create event* after you’ve created all of your desired states. This will be the state that the object starts in, and also the state that the object will default to if you attempt to switch to a non-existent state.

truestate_step
Arguments: none
Return: N/A
Runs the step portion of the state script. Should only be called in the object's step event, and only once.

truestate_draw
Arguments: none
Return: N/A
Runs the draw portion of the state script. Should only be called in the object's primary draw event of choice, and only once.

truestate_begin_step
Arguments: none
Return: N/A
Changes the state for the next step event or updates the truestate_timer if the state stays the same. Should only be called once per object.

truestate_cleanup
Arguments: none
Return: N/A
Destroys all data structures crated as part of the system. Should be called in the *cleanup event*, and should only be called once.

State Flow Control Scripts

These scripts should be used inside of a state script, and can be used to control how the object moves from one state to another.

truestate_switch
Arguments: StateEnum <*real, enum*>, LockSwitch <Boolean, *optional: defaults to false*>
Return: N/A
Use this script to change from the current state to another state. Should only be used in a state script, ideally the step portion. The current state script will continue as normal until the *draw gui end* event is ran. The lock argument will prevent further truestate_switch() calls from overwriting the locked switch. For example, say you have a state with a check for if the object's hp has dropped to zero, and if so switch to State.death. But in that same state a few lines below it, you check to see if the player took damage recently, and if so go to State.knock_back. Once you've determined that you should go to the death state, you wouldn't want the switch to knock_back to happen. So locking the switch prevents that. A locked switch will also interrupt a queue that is currently being executed. The lock is removed upon the completion of the step when the state switch actually occurs.

truestate_switch_previous
Arguments: none
Return: N/A
Operates very similarly to *state_switch* but will always return to the state that was ran previous to the current state. This is useful when you have a state that can be accessed from multiple states. For example, say you have an attack state. Your player can press the attack button when the character is standing or running. Once the attack animation finishes, you switch back to the previous step, and if the player was walking when they attacked, they'll return to walking, otherwise they will return to standing.

truestate_reset_current_state
Arguments: none
Return: N/A
Operates very similarly to *state_switch* but will "switch" to the state that is already running; resetting state_timer back to 0, and setting state_new back to true. An example of why you might want this is for a double jump. Instead of having a separate state for the second jump, you could just reset the current jump state.

truestate_enqueue
Arguments: StateEnum <*real, enum*>, any number of state enums…
Return: N/A
Adds the desired states to the queue. Any number of states may be added to the queue with a single call of truestate_enqueue.

truestate_ queue_start
Arguments: none
Return: N/A
Starts the state queue. Once started, the queue will be ran from the first state added to the last. While the queue is being executed any and all state_switch calls will go to the next state in the queue regardless of the passed argument. With the exception of state switches which "lock" the switch. These will empty the queue, and switch to the state indicated in the locked switch.

truestate_clear_history
Arguments: none
Return: N/A
Every time you change states, that state is added to the ds_stack *state_stack*. This is used by state_switch_previous, but can also be useful for debugging purposes. It is recommended that you clear this history occasionally to prevent it from getting enormous. I recommend calling this script when state_new == true in your default state.

Debug Scripts
I've included a few scripts that can help you debug your states.

truestate_get_name
Arguments: StateEnum <*real, enum*>
Return: string
Sometimes it is useful to print the current state to the screen somewhere so you can watch the flow of your object's states. Use this script to get the name of that state in a more consumable string format.

truestate_draw_current
Arguments: x <*real*>, y <*real*>
Return: none
Will draw the current state's name followed by the current value of the state_timer in parentheses at the given coordinate. A nice shortcut for debugging purposes.

truestate_state_exists
Arguments: StateEnum <*real, enum*>
Return: boolean
Allows you to test to see if an object has a state with the provided StateEnum. For example, in a very complicated system, you might share state scripts among many different type of objects. For example, many of your characters,

npcs, and enemies will have a “stand” state where they just hold still, so why duplicate that code all over the place? But some of your characters might have charming “idle animations” that NPCs or Enemies might not have. In the stand state script, you could check if this object has an idle state before trying to switch to it. Probably won’t be used too often, and requires that you set up the values of your state enum in a specific way to make sure states that share a value won’t give you a false positive.