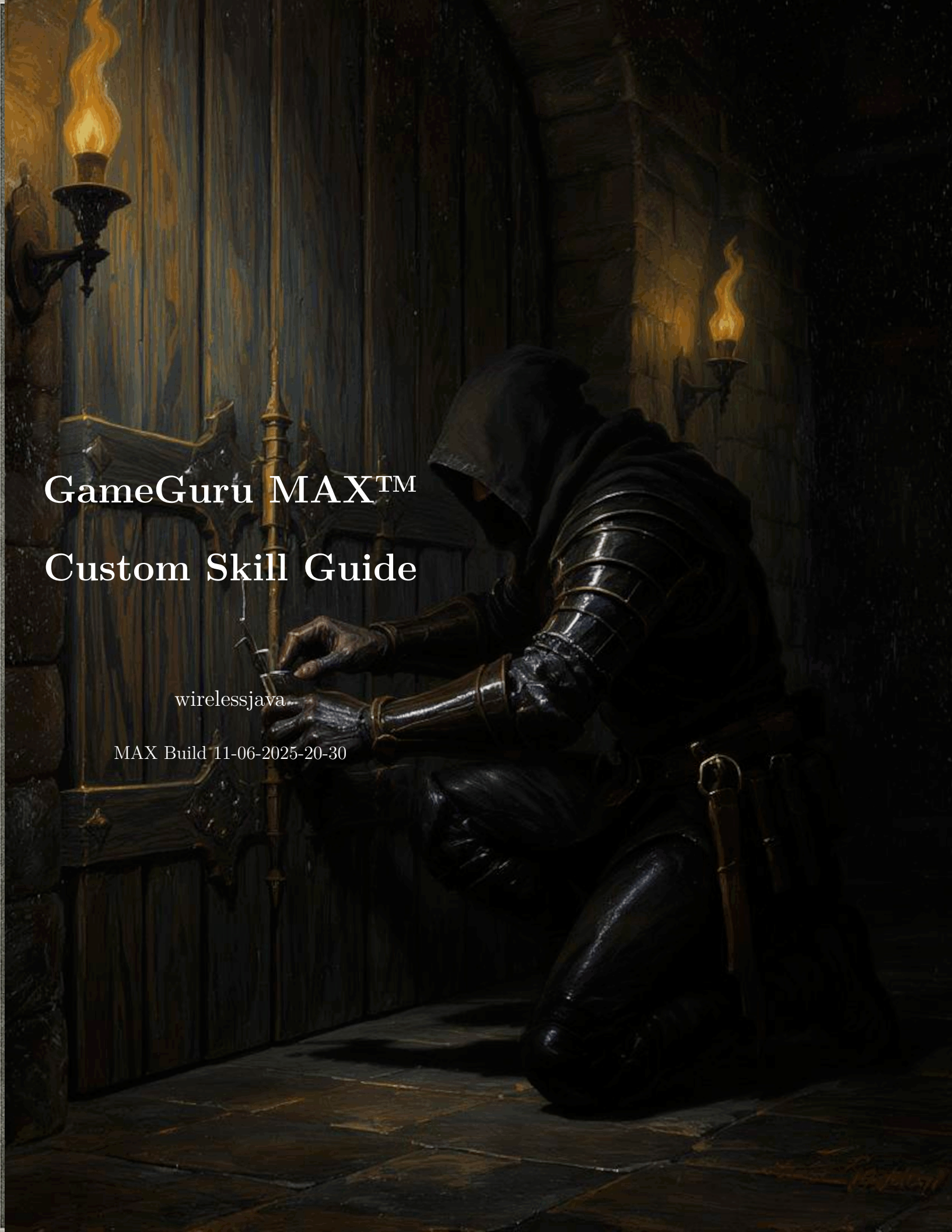

GameGuru MAX™ Custom Skill Guide

wirelessjava

MAX Build 11-06-2025-20-30

Contents

0.1 Guide: Creating a Lockpicking Skill in GameGuru MAX

In this guide, we'll walk through creating a simple lockpicking mechanic using custom behaviors and a global skill variable in GameGuru MAX.

- Setting up Global Variable
- Adding to HUD
- Modifying an existing Behaviour

0.1.1 Step 1: Creating the Lockpicking Skill as a Global User Variable

Before we dive into scripting our lockpicking system, we first need to create the global user variable that will store the lockpicking skill. We'll set this up within the GameGuru MAX HUD Editor.

In my setup I have used the Built in RPG HUD screens, so from the Game Storyboard click the **Add RPG HUD Screens** button. This will add amongst others the Character HUD Screen which makes sense to add our skill variables, to start at least. You can do this from scratch or add to an exsisting HUD screen.

Tick View Advanced Settings checkbox and enable almost every available option. Doing so will ensure that you see all the options covered in this guide and will save you the confusion of wondering why some features appear missing.

Follow these simple steps:

Navigate to your Game Storyboard.

Open the HUD Editor by double clicking on **HUD Screen 3** which is the Character Stats screen (Or the one you want to edit):

On the left under readouts, Add a User Defined Global (1st in list).

On the right hand panel of the new Global variable, in the box under **Custom Value** change it from **0** to **MyLockPicking** and press the ENTER button very hard. This will add the new variable which can now be used from anywhere in the game. (The Save/Load system will automatically detect this new variable and persist accross levels and game saves)

Initially set the default value to 0, indicating no lockpicking skill at the start.

Figure 1: Character stats created by Add RPG HUD Screens button

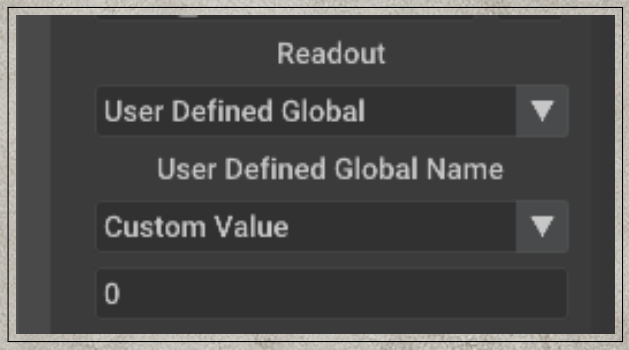

Figure 2: MyLockPicking will appear in the list of other variables, ie MyMoney, MyXP...

Position this readout clearly on the HUD—typically near other skill or stat readouts. Add a Text line along side so the player knows its the LockiPicking Skill.

Confirm the changes by returning to the storyboard and testing your HUD in-game.

Why a Global User Variable?

Using a global user variable allows your lockpicking skill to be:

- Easily accessible from any script or behavior.
- Persistent, meaning it maintains its state across different levels and save/load cycles.

This forms the backbone for a robust, scalable RPG skill system.

0.1.2 Step 2: Selecting and Modifying the Door Behaviour

Select the door object in your game scene. The standard Door asset already has the Door Rotate behaviour attached, and can be dragged in from the list of standard assets. Make sure the **Is Unlocked** is unticked, ie locked at the start.

Figure 3: Locked Door in scene editor

In the Object Tools Panel, open the Behaviour dropdown.

Double-click the default behaviour: Door Rotate.

It's good practice to duplicate and rename built-in behaviors to avoid accidentally overwriting official scripts.

0.1.3 Step 3: Creating a New Custom Behaviour

Click New Behaviour.

Figure 4: Let the Editor create the new script

Name the new behavior exactly: **door_rotate_lockpick** GameGuru MAX automatically generates the necessary script file named door_rotate_lockpick.lua.

TIP: Follow this exact naming convention—GameGuru MAX uses it to link scripts and behaviors seamlessly.

0.1.4 Step 4: Locating and Editing the New Script

Navigate to your project's scriptbank **user** directory. This should be located in your project folder, i.e:

```
Documents/GameGuruApps/MyProject/Files/scriptbank/user
```

The newly created script will be inside the user folder, named **door_rotate_lockpick.lua**

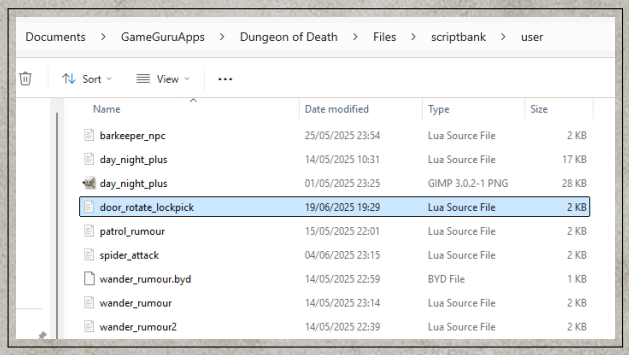

Figure 5: Editor created new script in projects scriptbank

and now look like this in the editor

Use your favorite IDE or text editor to open it. A recommended IDE is Visual Studio Code (VSCode) but even Notepad works perfectly fine.

VSCode offers syntax highlighting and error-checking, which can significantly speed up development.

0.1.5 Step 5: Copying Original Door Rotate Logic

Find the original door_rotate.lua script in your Steam folder:

```
C:/Program Files (x86)/Steam/steamapps/common/GameGuru MAX/Files/scriptbank/objects/door_rotate.lua
```

Copy the entire contents of this file.

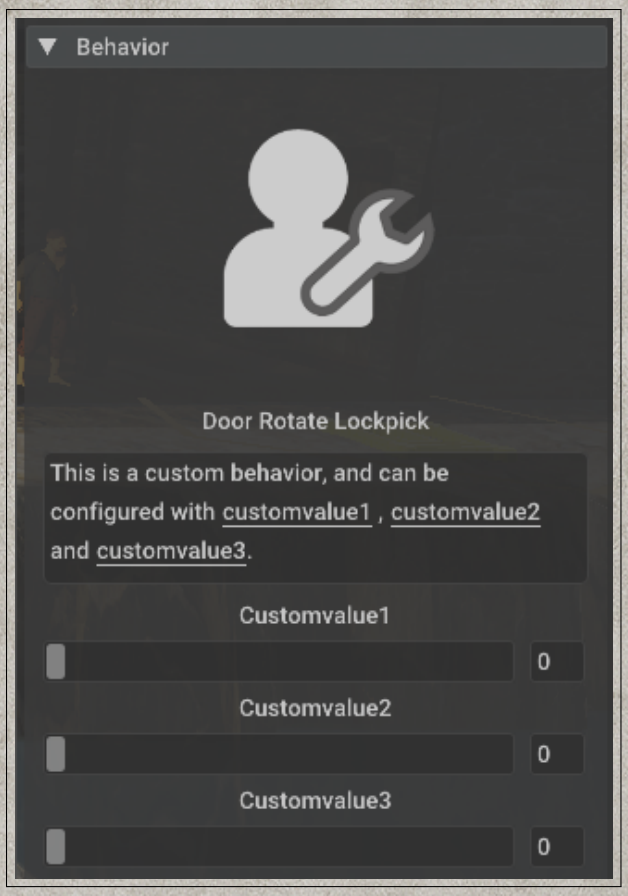

Figure 6: Default script created by the editor

Paste it into your new door_rotate_lockpick.lua file, overwriting the default code initially provided by GameGuru MAX.

Save the script.

0.1.6 Step 6: Saving and Restarting GameGuru MAX

Save your game in the Storyboard.

Restart GameGuru MAX for changes to fully take effect. After restarting, select your door again and verify it now displays your new Door Rotate Lockpick behavior.

Frequent restarts after changing scripts help ensure MAX registers all script changes correctly.

Figure 7: VSCode, industry standard code IDE, and Free

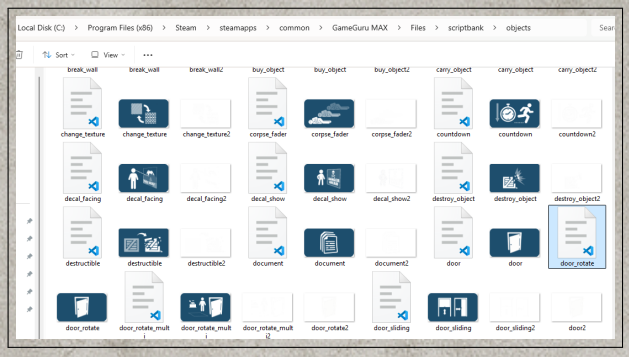

Figure 8: Original script in Steam Folder

0.1.7 Step 7: Testing the Copied Behaviour

Test the level to ensure the copied Door Rotate Lockpick behavior functions exactly like the original Door Rotate behavior before modifications.

0.1.8 Step 8: Modifying the Script (Basic Setup)

We'll carefully change function names to match our new script file **door_rotate_lockpick.lua** Function names must start with the name of the script file. (so replaceing door_rotate with door_rotate_lockpick)

Find these functions and update their names:

Change:

```
function door_rotate_properties(e,
    promptdisplay, prompttext, range,
    door_type, opening_style,
    initialstate)
```

to:

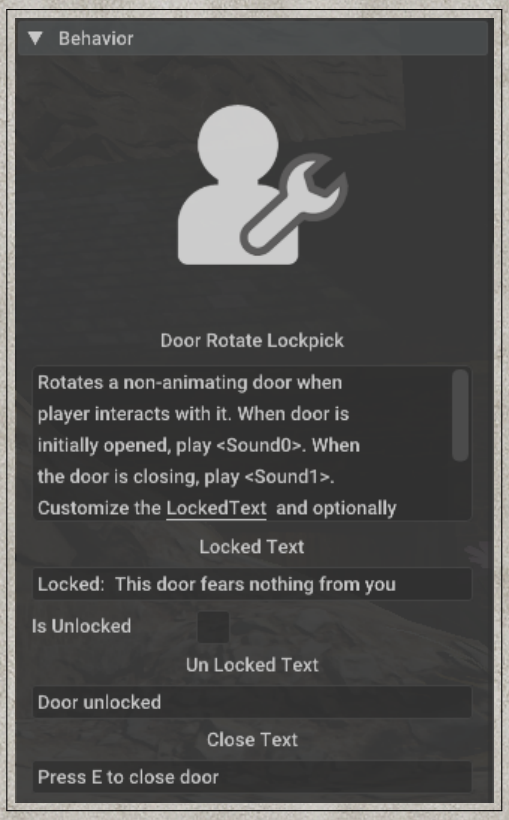

Figure 9: Looks like the old Door Rotate, fill in missing values

```
function door_rotate_lockpick_properties(
    e, promptdisplay, prompttext, range,
    door_type, opening_style,
    initialstate)
```

Change:

```
function door_rotate_init_name(e)
```

to:

```
function door_rotate_lockpick_init_name(e
    )
```

Change:

```
function door_rotate_main(e)
```

to:

```
function door_rotate_lockpick_main(e)
```

Save, then restart GameGuru MAX again, and test thoroughly to confirm all is working.

Test after each step to isolate issues easily if something goes wrong.

0.1.9 Step 9: Implementing the Lockpicking Skill Variable

Previously, you've created a global variable called MyLockPicking in the HUD system. We'll integrate this into our script.

Locate the main function (door_rotate_lockpick_main) and add the following simplified logic for incrementing your lockpicking skill when the "E" key is pressed:

Towards the top of the script you will see many local variables being declared, add our new lockpicking variable after the last in the list

```
local lockpicking = "MyLockPicking"
```

Add this initialisation block at the end of the **door_rotate_lockpick_init_name** function

```
if _G["g_UserGlobal['"..lockpicking.."']"
    ] == nil then
    _G["g_UserGlobal['"..lockpicking.."']
        "] = 0
end
```

And finally add the main logic at around line 168, just after this line:

```
if (PlayerDist < door.door_range and tEnt
    [e] ~= 0 and GetEntityVisibility(e)
    == 1) or allowautoopenremotely == 1
    then
```

```
if g_KeyPressE == 1 and not keyPressed
    then
    keyPressed = true
    _G["g_UserGlobal['"..lockpicking.."']
        "] = _G["g_UserGlobal['"..
        lockpicking.."']"] + 1

    Prompt("Lockpick attempt: " .. _G["
        g_UserGlobal['"..lockpicking.."']
        "])

    -- Check if lockpick skill reached 10
        (Hardcoded for demonstration
        purposes)
    if _G["g_UserGlobal['"..lockpicking..
        "']"] >= 10 then
```

```
        door.isunlocked = true
        Prompt("Door is now unlocked!")
        PlaySound(e, 0)
    end
elseif g_KeyPressE == 0 then
    keyPressed = false
end
```

0.1.10 How this works:

Every time the player presses ”E” near a locked door, the lockpick skill increases by 1.

Once the skill reaches 10, the door automatically unlocks.

0.1.11 Step 10: Final Testing and Adjustments

Thoroughly test in-game to confirm your Lockpicking Skill behaves as expected.

Adjust skill thresholds or prompts for game balance and player experience.

Additional Insights for Users:

This lockpicking system can easily expand into a deeper RPG mechanic, including skill checks, random chance, and even mini-games.

Integrating skill progression into broader RPG systems provides players with rewarding character development.

Tips and Best Practices:

Regularly comment your Lua scripts, especially when they grow complex. Clear comments save tremendous time later.

Frequently test small changes incrementally—avoiding the common pitfall of extensive troubleshooting. Dont forget to test Standalone Build as well.

0.1.12 Learning LUA

One of the best ways to deepen your scripting skills in GameGuru MAX is to study existing scripts included in the scriptbank folder. By carefully examining these pre-made Lua scripts, you can understand the logic, structure, and common conventions used by experienced game developers. When you encounter challenges or have questions, engaging with the GameGuru MAX community on Discord can be extremely beneficial.

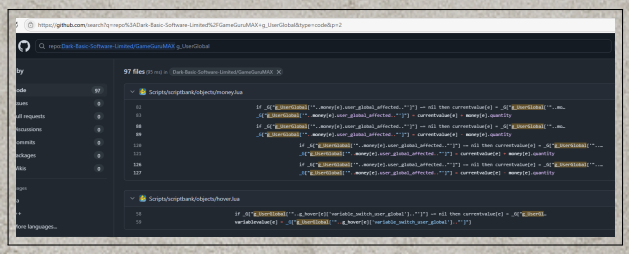

Figure 10: GameGuru MAX GitHub repo

0.1.13 Small Inprovements

In GameGuru MAX, you can expose a variable in the script so it becomes editable in the Behaviour panel in the Object Properties. This means you won't need to hardcode values such as the 10 for difficulty.

There are a few steps to make it appear in the readout and then thread it through to our logic

At the very top there is a list of DESCRIPTION lines as comments, add the following line to the end of that block to make it appear in the Behaviour's readout.

```
-- DESCRIPTION: [LOCKPICK_DIFFICULTY
    =10(1,100)] Skill level required to
    unlock the door
```

Add another local variable after the MyLockPicking one we added earlier

```
local lockpick_difficulty = 10
```

Next add the new **lockpick_difficulty** variable as a parameter to the function

```
function door_rotate_lockpick_properties(
    e, promptdisplay, prompttext, range,
    door_type, opening_style,
    initialstate, lockpick_difficulty)
```

Also at the end of that function, after **opening_style** and before the keyword **END**

```
door.lockpick_difficulty =
    lockpick_difficulty or 10
```

this will add the variable to the Doors **table**, and make it available on that object later

Scroll down a little and find the **door_rotate_lockpick_init_name** func-

tion. After **opening_style = opening_style,** add this line

```
lockpick_difficulty = lockpick_difficulty
    ,
```

Then finally, at last, in the main code we can swap out the hardcoded value of **10** to **door.lockpick_difficulty** and tweak the formula slightly

```
        -- Simple difficulty: player must
            reach DOUBLE the door
           difficulty to unlock
           if _G["g_UserGlobal['"..
               lockpicking.."']"] >= (
               door.lockpick_difficulty
               * 2) then
```

Once you get the hand of it, its fairly straight forward. Repetition is the key.

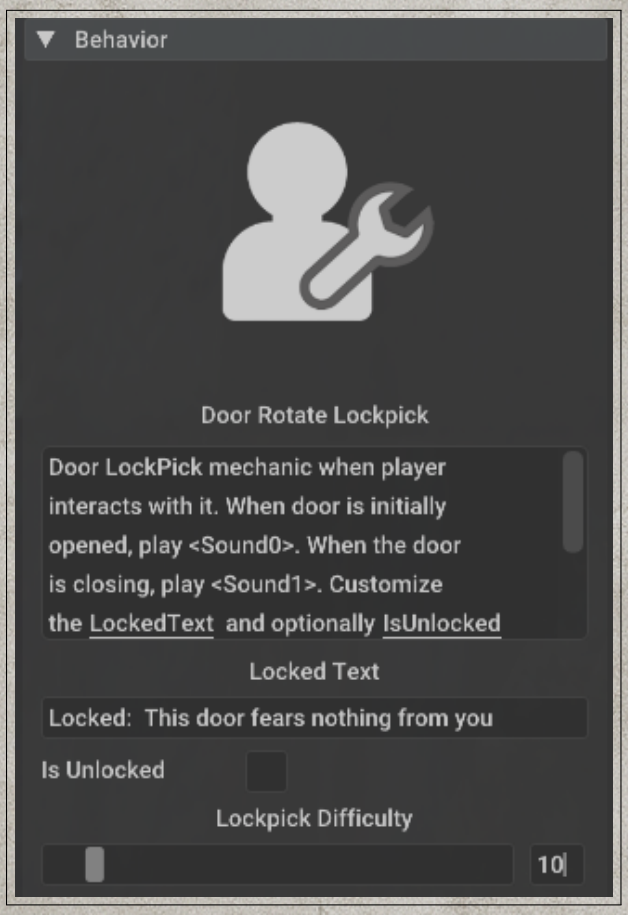

Figure 11: Finished LockPick behaviour with slider for Difficulty

What Next

This basic LockPicking skill demonstration was intentionally kept simple to clearly illustrate the foundational steps involved. However, there are many exciting enhancements you could implement next to expand this mechanic. You might consider adding skill progression systems where the player's lockpicking ability improves over time, or introducing failure consequences such as broken lockpicks or noise alerts that attract nearby enemies. For greater immersion, incorporating mini-game interactions or time-based challenges could provide a richer, more interactive gameplay experience.

Full Source Code

Full Lua script embedded directly(Only works with Adobe Acrobat Reader, not in browser): Click here for Full Lua script

```
// At the very Top
...
-- DESCRIPTION: [@OPENING_STYLE=1(1=Push, 2=Pull)]
-- DESCRIPTION: [LOCKPICK_DIFFICULTY=10(1,100)] Skill level required to unlock the door

...
// Define variable section
...
local defaultDoorRange    = 100
local defaultPromptDisplay= 2
local lockpicking = "MyLockPicking"
local lockpick_difficulty = 10

...
local doorTypesRotation = { 'Auto', 'Manual' }

function door_rotate_lockpick_properties( e, lockedtext, isunlocked, unlockedtext, closetext,
                   door_type, door_range, prompt_display, opening_style, lockpick_difficulty)
...
    door.opening_style = opening_style or 1
    door.lockpick_difficulty = lockpick_difficulty or 10

// End of door_rotate_lockpick_init_name( e, name )
                           opening_style = opening_style,
                           lockpick_difficulty = lockpick_difficulty,
...
// End of door_rotate_lockpick_init_name
...
                           opening_style = opening_style,
                           lockpick_difficulty = lockpick_difficulty,    <-->
                         }
    tEnt[e] = 0
    selectobj[e] = 0

    if _G["g_UserGlobal['"..lockpicking.."']"] == nil then
        _G["g_UserGlobal['"..lockpicking.."']"] = 0

    end

// Main
...

if (PlayerDist < door.door_range and tEnt[e] ~= 0 and GetEntityVisibility(e) == 1)
                                                    or allowautoopenremotely == 1 then

  if g_KeyPressE == 1 and not keyPressed then
      keyPressed = true
      _G["g_UserGlobal['"..lockpicking.."']"] = _G["g_UserGlobal['"..lockpicking.."']"] + 1

      Prompt("Lockpick attempt: " .. _G["g_UserGlobal['"..lockpicking.."']"])

      -- Simple difficulty: player must reach DOUBLE the door difficulty to unlock
        if _G["g_UserGlobal['"..lockpicking.."']"] >= (door.lockpick_difficulty * 2) then
          door.isunlocked = true
          Prompt("Door is now unlocked!")
          PlaySound(e, 0)
      end
  elseif g_KeyPressE == 0 then
      keyPressed = false
  end

  tareweclose = 1
    -- handle door when closed
```