Note: For the Lua API documentation (a complete list of the Lua functions and tables that interact with the engine) please see the `doc` folder packaged in your TombEngine download.

Contents:

1. Obelisk Puzzle

2. Sequence Switch Puzzle

3. Infinite Enemy Random Respawn

1. Obelisk Puzzle

In TR4, Lara can use three turn switches to rotate obelisks to point toward an object she wishes to attain. In TEN, the hardcoded behavior for these objects was removed - instead you can use LUA to create a variation of this puzzle that suits your needs.

Tip: An easy way to get the correct positions or rotations to reference in your script is to load your level with the object in the appropriate spot, then using the print() command. For example `print(object:GetRotation())` will give you the current rotation of `object`.

In this example, we "disable" the pulley by shifting it down so Lara can't use it. It won't be set to its rightful position until Lara gets the obelisks in the correct position. The level needs the following objects and LUA names:

- Three turnswitches: turn_switch_a, turn_switch_b, turn_switch_c
- Three obelisks: obelisk_a, obelisk_b, obelisk_c
- One pulley: pulley

To solve the puzzle with this code:

- obelisk_a needs to face east
- obelisk_b needs to face west
- obelisk_c needs to face south

```
local Util = require("Util")
Util.ShortenTENCalls()

--this boolean prevents pulley raising in OnControlPhase from happening more than
once
puzzleSolved = false

--at the start, lower pulley into the ground to "disable" it
LevelFuncs.OnStart = function ()
    local pulley = GetMoveableByName("pulley")
    local pos = pulley:GetPosition()
        pos.y = pos.y + 500
        print(pos.y)
    pulley:SetPosition(pos)
end


--oncontrolphase executes every frame. check for changes in turn switch rotation
here
LevelFuncs.OnControlPhase = function()
    --get objects
    local switchA = GetMoveableByName("turn_switch_a")
    local switchB = GetMoveableByName("turn_switch_b")
    local switchC = GetMoveableByName("turn_switch_c")
    local obeliskA = GetMoveableByName("obelisk_a")
    local obeliskB = GetMoveableByName("obelisk_b")
    local obeliskC = GetMoveableByName("obelisk_c")
    local pulley = GetMoveableByName("pulley")
    --make obelisk rotation match turn switch rotation
    obeliskA:SetRotation(Rotation.new(0, switchA:GetRotation().y, 0))
    obeliskB:SetRotation(Rotation.new(0, switchB:GetRotation().y, 0))
    obeliskC:SetRotation(Rotation.new(0, switchC:GetRotation().y, 0))
    --check of obelisks are in correct orientations, raise pulley back up to allow
player to use it
    --Rotation angles can be positive or negative in TEN, so multiple checks are
included.
    if puzzleSolved == false and (obeliskA:GetRotation().y == 0 or
obeliskA:GetRotation().y == 360) and (obeliskB:GetRotation().y == 180 or
obeliskB:GetRotation().y == -180) and (obeliskC:GetRotation().y == 90 or
obeliskC:GetRotation().y == -270) then
        puzzleSolved = true
        local pos = pulley:GetPosition()
        pos.y = pos.y - 500
        print(pos.y)
        pulley:SetPosition(pos)
    end
end
```

2. Sequence Switch Puzzle

In TR4, Lara finds three switches that open different doors based on the order she pressed them in. As with above, you can use LUA to reconstruct this kind of puzzle in TEN according to your needs.

In this example, three basic wall switches are used. After Lara pulls them all down, the code checks what sequence was used, and whether that opens one of the three doors. In any case the switches are destroyed + recreated in order to be usable again. (Without legacy triggers, switches can't reset themselves normally.) The level needs the following objects and LUA names:

- Three doors: door_abc, door_acb, door_bca
- Three switches: switch_a, switch_b, switch_c

```lua
local Util = require("Util")
Util.ShortenTENCalls()

--keep track of order of switches in this array
--the goal is: switches[1] stores the first switch pulled, switches[2] stores the
second switch pulled, and so on.
switches = {}

--this variable represents the next index that will be modified in the array.
--Standard convention for Lua is to start with index 1, not 0.
i = 1

--keep track of switch status so we can detect changes: false = pulled up, true =
pulled down
switchAStatus = false
switchBStatus = false
switchCStatus = false

--these variables will remember the starting positions of the switches
positions = {}
rotations = {}
roomNumber = 0

--remember those starting positions using the OnStart function
LevelFuncs.OnStart = function()
    local switchA = GetMoveableByName("switch_a")
    local switchB = GetMoveableByName("switch_b")
    local switchC = GetMoveableByName("switch_c")
    positions[1] = switchA:GetPosition()
    positions[2] = switchB:GetPosition()
    positions[3] = switchC:GetPosition()
    rotations[1] = switchA:GetRotation()
    rotations[2] = switchB:GetRotation()
    rotations[3] = switchC:GetRotation()
    roomNumber = switchC:GetRoom()
end

--oncontrolphase executes every frame. check for changes in switch animations
here.
--for the wall switch animations: 0 is pulled up, 1 is pulled down
LevelFuncs.OnControlPhase = function(delta)
    --get switch objects
    local switchA = GetMoveableByName("switch_a")
    local switchB = GetMoveableByName("switch_b")
    local switchC = GetMoveableByName("switch_c")
    --first check if all three switches have been pulled (i == 4 means switches
array has three elements)
    if i == 4 then
        --open the corresponding door (if any) based on order of switches pulled
        if switches[1] == 'a' and switches[2] == 'b' and switches[3] == 'c' then
            local door = GetMoveableByName("door_abc")
            door:Enable()
        end
        if switches[1] == 'a' and switches[2] == 'c' and switches[3] == 'b' then
            local door = GetMoveableByName("door_acb")
            door:Enable()
        end
        if switches[1] == 'b' and switches[2] == 'c' and switches[3] == 'a' then
            local door = GetMoveableByName("door_bca")
            door:Enable()
        end
        --reset switches + variables
        switches = {}
        i = 1
        switchAStatus = false
        switchBStatus = false
        switchCStatus = false
        switchA:Destroy()
        switchB:Destroy()
        switchC:Destroy()
        Moveable.new(ObjID.SWITCH_TYPE1, "switch_a", positions[1], rotations[1],
roomNumber, 0, 0, 0, 0, {0,0,0,0,0,0})
        Moveable.new(ObjID.SWITCH_TYPE1, "switch_b", positions[2], rotations[2],
roomNumber, 0, 0, 0, 0, {0,0,0,0,0,0})
        Moveable.new(ObjID.SWITCH_TYPE1, "switch_c", positions[3], rotations[3],
roomNumber, 0, 0, 0, 0, {0,0,0,0,0,0})
    else
        --otherwise, see if a switch is pulled down and thus it is ready to be
stored in our array
        if switchA:GetAnim() == 1 and switchAStatus == false then
            switches[i] = 'a'
            switchAStatus = true
            i = i + 1
        end
        if switchB:GetAnim() == 1 and switchBStatus == false then
            switches[i] = 'b'
            switchBStatus = true
            i = i + 1
        end
        if switchC:GetAnim() == 1 and switchCStatus == false then
            switches[i] = 'c'
            switchCStatus = true
            i = i + 1
        end
    end
end
```

3. Infinite Enemy Random Respawn

In this example, we demonstrate the use of a callback function that is used for when a baddy is killed. This script creates a new enemy when the previous one is killed, and will keep doing so indefinitely. The new enemy is randomly spawned between three locations, which are represented by camera targets. (We are using camera targets only because they are a simple object that appears invisible during the game.)

The level needs the following objects and LUA names:

- A GOON2 enemy: baddy1
- Three camera targets: camera_target_1, camera_target_2, camera_target_3

```
local Util = require("Util")
Util.ShortenTENCalls()

--index for current baddy
i = 1
LevelFuncs.OnStart = function()
    local baddy = GetMoveableByName("baddy1")
    baddy:SetOnKilled("OnBaddyKilled")
    baddy:Enable()
end
LevelFuncs.OnBaddyKilled = function()
       --add 1 to our baddy index
       i = i + 1
        --randomly pick a camera target to spawn baddy at
        local cameraTargetIndex = math.random(1, 3)
        local cameraTarget =
GetMoveableByName("camera_target_"..cameraTargetIndex)
        --spawn baddy at camera target's location. Don't forget to assign the
OnKilled event to this one too.
        local newBaddy = Moveable.new(ObjID.GOON2, "baddy"..i,
cameraTarget:GetPosition(), cameraTarget:GetRotation(), cameraTarget:GetRoom(), 0,
0, 100, 0, {0,0,0,0,0,0})
        newBaddy:SetOnKilled("OnBaddyKilled")
        newBaddy:Enable()
end
```