



3DNES SCRIPTING MANUAL

Contents

1.	Basic Concepts.....	1
2.	System Overview	3
3.	APIs In Detail.....	4
3.1	Utility classes.....	4
3.2	ScriptManager	5
3.3	Frame Manager.....	5
3.4	Nespad3D.....	6
3.5	2DShape	6
3.6	3DShape	7
3.7	SettingManager	8
4.	Examples	9
4.1	Super Mario Bros – UI	9
4.2	Legend Of Zelda – UI.....	10
4.3	Rotation effect.....	11
4.4	Day-night effect	11

1. Basic Concepts

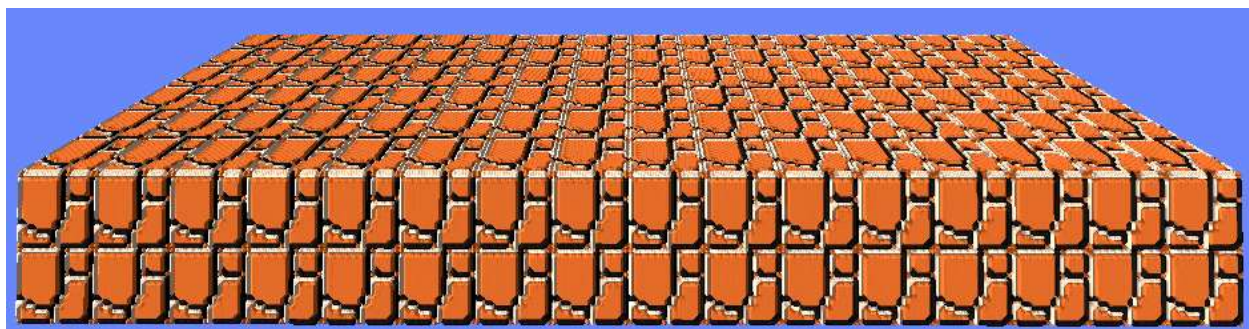
There are concepts of 3DNes that you should get familiar with:

- Shape: in-game graphic object that can be perceived by human. For example in Super Mario Bros, Mario and Luigi are shapes, mushrooms are shapes, bullets are shapes, pipes are shapes ...

- 2DShape: the 2d representation of shape, that's what players can see in the original game. At technical level, it's more or less equivalent to meta-sprite concept.



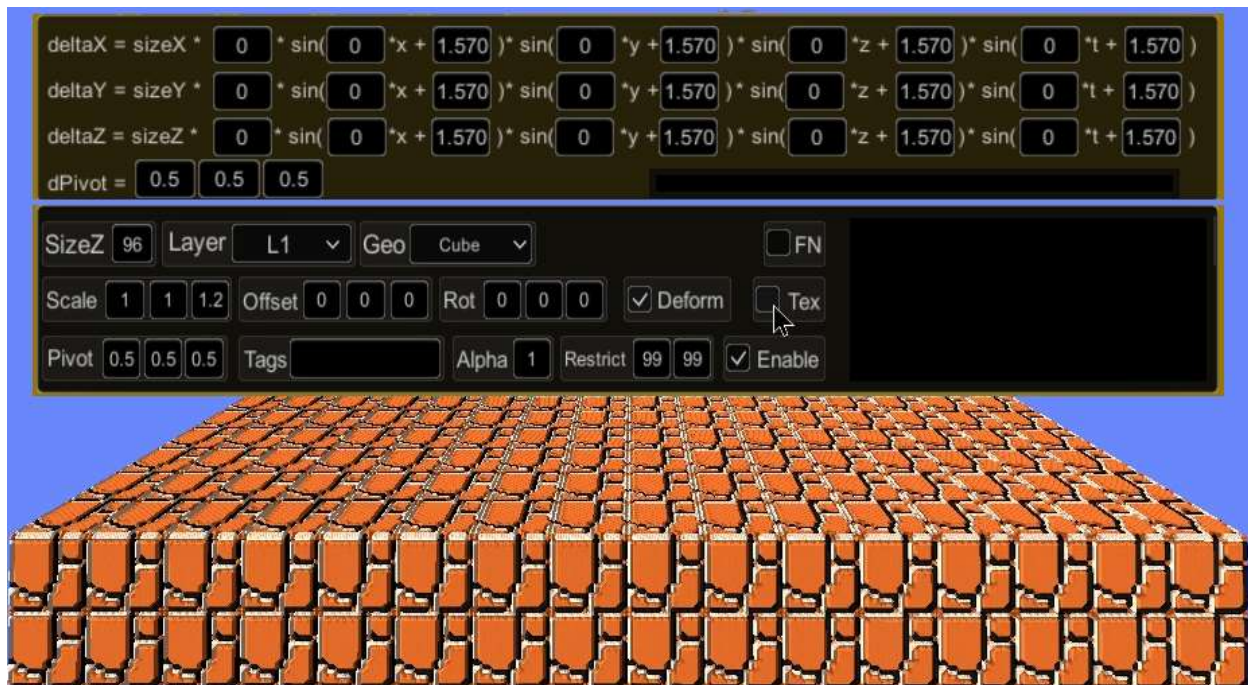
- 3DShape: the real shape in 3d world that 3DNes will create and render while emulating the game.



- Pattern: the most important concept of 3DNes. A pattern is a set of unique tiles, it defines a shape cluster constructed by that set of tiles. For example in Super Mario Bros, Mario and Luigi are in the same cluster, they are basically the same with different color palettes, all the vertical pipes with different heights are in the same cluster – they are all created by the same set of unique tiles, all the clouds are also in the same cluster.



- 3DPattern: contains a set of parameter defining how shape cluster linked to the given pattern will be modeled and rendered in 3d space. By default a pattern only has a 3DPattern, it means that a 2DShape will be converted to a 3DShape. But in some cases, a pattern may have more than one 3DPattern and therefore a 2DShape linked to this pattern will be converted into a set of 3DShapes.



2. System Overview

Scripts are pieces of Lua code that can be added to dynamically adjust emulation settings and 3DShape parameters in the emulation process. It give 3DNes a limitless possibility to further personalize 3DShape and to create different animations and effects.

A script has a unique name as the identifier, a list of tag, *Enable* property and the code itself. Script tags will be matched with 3DPattern tags to pair script and 3DPattern. If a script is paired with a 3DPattern then it is paired with every 3DShape of that pattern cluster. Therefore in every frame 3DShapes of that cluster are objects to modify of this script code. * is a special tag that matches all tags.

To hook itself into emulation process, a scripts must implement one or some of four following special functions.

- **Start:** after loading the rom or the save state, if a script is enable and error-free, its **Start** function will be executed.
- **End:** right before finishing the emulation process of a given game, if a script is enable and error-free, its **End** function will be executed.
- **Update:** after every frame emulation, if a script is enable and error-free, its **Update** function will be executed. This function should be used to query emulation state and render buffer.



- **UpdateS:** after every frame emulation, if a script is enable and error-free, for each shape in render buffer that is paired with it, its **UpdateS** function will be executed once. In this execution context global variables **shape** and **shape2D** will refers to the respective 3DShape and 2DShape. This function should be used to adjust parameters of some specific 3DPatterns.

In return scripts can access and modify emulation state and render buffer via various global variables:

- variable *Script* of type **ScriptManager**
- variable *Frame* of type **FrameManager**
- variable *Setting* of type **SettingManager**
- variable *Light* - shortcut of *Setting.Light*
- variable *Layer* - shortcut of *Setting.Layer*
- variable *Clipping* - shortcut of *Setting.Clipping*
- variable *frame* - the frame counter, shortcut of *Frame.frameCounter*
- variable *time* – current game time in second, shortcut of *Frame.time*
- variable *shape* of type **Pattern3D**
- variable *shape2D* of type **Shape2D**
- variable *gamepad*, *gamepad1*, *gamepad2* of type **Nespad3D**

Two variables *shape* and *shape2D* are only determinist in **UpdateS** scope. Variables *gamepad1* and *gamepad2* report the state of respective gamepads, variable *gamepad*'s API will return true if either *gamepad1*'s or *gamepad2*'s corresponding API return true.

Built-in Lua variables are the wrappers of C# objects. To access a field or a property, we use the normal syntax:

Variable.FieldName

But to invoke a method, we use the following syntax

Variable:MethodName()

For example:

Setting.BgColor = false; Setting.Save();

3. APIs In Detail

3.1 Utility classes

public struct IntVector2 {public int x, y;}

```

public struct Vector3 { public float x, y, z;}
public class PVector3 {public int x, y, z;}
public class PVector4 {public float x, y, z, w;}
public class PMatrix4x4 { public float m00, m01,m02, m03, m10, m11,m12, m13, m20, m21,m22, m23, m30,
m31,m32, m33;}
public class Color {public float r, g, b, a;}
public enum ZLayer { UNIDENTIFIED, UI, L1, L2, L3, L4, COUNT };
public enum Geo3DType
{
    Default,
    HCYLINDER,
    VCYLINDER,
    CUBE,
    HALFHCYLINDER,
    HALFVCYLINDER,
};
public enum SelectionMode { Shape3D, Tile };
public enum Alignment { FRONT, CENTER, BACK };
public enum RenderAlgo { Greedy, Marching };

```

3.2 ScriptManager

Provide APIs to access to scripts and read, write game memory.

```

public class ScriptManager
{
    Script Get(string name); // get script with name
    byte ReadMem(ushort address); // read from nes memory
    void WriteMem(ushort address, byte value); // write to nes memory
}

public class Script
{
    string Name { get; } // script name
    bool Enable { get; set; } // if the script is allowed to run
    bool Error { get; } // if there is any error in script code – auto detect by 3DNes
}

```

3.3 Frame Manager

Provide APIs to access to shapes in current frame buffer.

```

public class FrameManager

```

```
{
    List<Pattern3D> GetShapesWithTag(string tag); // return a list of shape having the tag "tag" in the current frame buffer
    Pattern3D GetShapeWithTag(string tag); // return the first shape having the tag "tag" in the current frame buffer
    Pattern3D GetShape(int index); // return the index-th 3DShape in the current frame buffer
    int shapeCount { get; } // return the 3DShape count in the current frame buffer
    int frameCounter { get; } // return the current frame counter
    float time { get; } // return time in second of current frame buffer
}
```

3.4 Nespad3D

Provides APIs to access gamepad state.

```
public class Nespad3D
{
    bool UpPressed { get; }
    bool DownPressed { get; }
    bool LeftPressed { get; }
    bool RightPressed { get; }
    bool SelectPressed { get; }
    bool StartPressed { get; }
    bool APressed { get; }
    bool BPressed { get; }
    bool RotLeftPressed { get; }
    bool RotRightPressed { get; }
    bool RotUpPressed { get; }
    bool RotDownPressed { get; }
    bool ZoomInPressed { get; }
    bool ZoomOutPressed { get; }
}
```

3.5 2DShape

Provide APIs to access properties of a 2DShape

```
public class Shape
{
    int Count { get; } // the number of shape3D linked with this 2dshape
    Pattern3D Shape3D(int index = 0); // get the index-th shape3D linked with this 2DShape
    IntVector2 TStart { get; } // start (top left) position in tile unit (8 pixels)
    IntVector2 TEnd { get; } // end position (bottom right) position in tile unit
}
```

```

IntVector2 TSize { get; } // shape size in tile
IntVector2 Start { get; } // start (top left) position in pixel
IntVector2 End; // end position (bottom right) position in pixel
IntVector2 Size { get; } // shape size in pixel
bool Bg { get; } // true if created by background tiles
bool Hiden { get; } // true if one of one of its sprite tiles is hidden
int Palette { get; } // palette index
int TileCount { get; } // tile count
bool Completed { get; } // true if the shape has every tile in the pattern tile set
bool Inside { get; } // true if the shape is completely located inside the screen
bool AtBorder { get; } // true if shape lies on a screen border
int Age { get; } // the current amount of time - in frame unit - that the linked pattern has at least one linked shape
in output frame
}

```

3.6 3DShape

Provide APIs to access to properties of a 3DShape.

```

public class Pattern3D
{
    Shape Shape2D { get; } // the corresponding 2dshape
    int Index { get; } // its order in its linked 2dshape's 3dshape list
    ZLayer Layer { get; set; } // shape layer
    bool UI { get; set; } // return true if Layer is UI, otherwise return false, setting it to true corresponds setting
    Layer to ZLayer.UI
    PVector3 Scale { get; }
    PVector3 Rot { get; }
    PVector3 Offset { get; }
    PVector3 Pivot { get; } // the pivot of scale and rotation operation
    float Alpha { get; set; } // shape transparency
    PVector3 DeformAmp { get; }
    PVector3 DeformPivot { get; }
    PMatrix4x4 DeformSpeed { get; }
    PMatrix4x4 DeformOffset { get; }
    bool Enable { get; set; } // if true, the shape will be rendered
    bool Sp { get; } // return true if this is a sprite shape
    bool Bg { get; } // return true if this is a background shape
    bool CastShadow { get; set; } // shape will cast shadow or not
    bool ReceiveShadow { get; set; } // shape will receive shadow or not
    Vector3 OriPos { get; } // the original shape position, calculated based on BottomLeft, TopRight and Pivot
    Vector3 BottomLeft { get; } // the original bottom left coordination in pixel
}

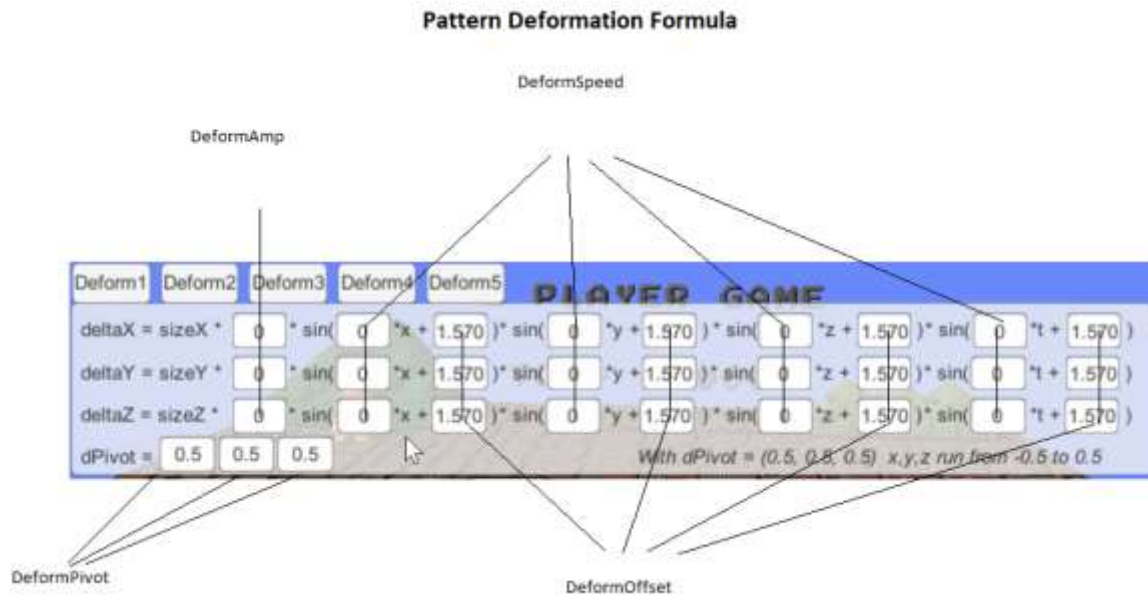
```



```

Vector3 TopRight { get; } // the original top right coordination in pixel
Vector3 GetDeform(Vector3 pos); // return the deform vector at "pos" position based on Deform parameters
and formula
bool ContainsTag(string tag); // return true if shape contains the tag
}

```



3.7 SettingManager

Provide APIs to access setting parameters.

```

public class SettingManager
{
    public SelectionMode SelectionMode { get; set; }
    public class LightConfig
    {
        public Vector3 Direction { get; set; }
        public Color Color { get; set; }
        public float Intensity { get; set; }
    }
    LightConfig Light { get; }
    bool BgColor { get; set; } //if true, use the game background color else use CustomBgColor
    Color CustomBgColor { get; set; }
    PVector4 Clipping { get; } //amount of clipped pixel at four borders Left, Bottom, Right, Top
    bool MapEnable { get; set; } // show or hide the mini map
    public class LayerP
    {

```



```

    public float Offset {get; set}
    public Alignment Aligned {get; set;}
}
public class LayerConfig
{
    public LayerP this[int index] {get;}
    public ZLayer DefLayer {get; set;}
    public bool Auto {get; set;} // if true, 3dnes will auto assign layer for pattern if possible
}
LayerConfig Layer {get;}
int GraphicQuality {get; set;} // valid values: 0, 1, 2
int VrMode {get; set;} // 0: None, 1: OpenVr, 2: Oculus
int VSync {get; set;} // enable or disable vsync
int FrameRate {get; set;} // the frame buffer count per second if VSync is false
float GameSpeed {get; set;} // from -4 to 4, 0 is normal speed
float Volume {get; set;} // from 0 to 1
RenderAlgo RenderMode {get; set;}
bool Titling {get; set;} // camera titling effect while pressing up, down, left, right button
bool Shadow {get; set;} // enable or disable shadow rendering at global level
bool SamePalette {get; set;} // if true, tiles in a shape must have the same palette
int SpriteTolerance {get; set;} // be default sprite tiles in a shape must be 8px aligned but we could set a tolerance to this constraint using this property
string Save2String(); // save setting to string
void Load(string s); // load setting from string
void Save(); // save setting to the permanent storage
void Load(); // load setting from the permanent storage
}

```

4. Examples

4.1 Super Mario Bros – UI

In Super Mario Bros we want the status bar at the top of the screen to be always visible, and perpendicular to camera view. That can be done by set every involved pattern to layer UI. The issue is when the same shape is positioned in another area, we don't want it in UI layer at all. So the static layer assignment won't work. What we want here is if the shape is high enough (that means it is in the status bar region), its layer should be set to UI. That can be done by a small script tagged “*”:

```

function UpdateS()
    if shape.BottomLeft.y > 216 then
        shape.UI = true

```

```
end
end
```

Because of * tag, it will be applied to every 3Dshape in each frame buffer. If the shape bottom is higher than 216 then it will be move to UI layer.

4.2 Legend Of Zelda – UI



The UI in Legend of Zelda is a little bit different. Normally the UI part is located at the top but when the start button is pressed, the UI will scroll down and show its full content. That means the UI region position isn't fixed. To deal with it, we choose a 3DPattern that only appears at the bottom of UI region and doesn't appear anywhere else, that 3DPattern will be tagged "ui_bottom". Then we add a script with the same tag:

```
function Start()
    saveInt = Light.Intensity
    saveDir = Light.Direction
    Light.Direction = Vector3(0.3, 0.3, 1)
    Light.Intensity = 0.7
end
function Update()
    ui_bottom = Frame:GetShapeWithTag("ui_bottom");
    if ui_bottom then
        bottom = ui_bottom.BottomLeft.y
```

```

else
    bottom = 1000
end
end
function UpdateS()
    if shape.BottomLeft.y >= bottom then
        shape.UI = true
    end
end
function End()
    Light.Intensity = saveInt
    Light.Direction = saveDir
end

```

In function *Start*, we backup light direction and intensity and assign new values for it. In function **Update**, we check if there is a 3DShape with the tag “*ui_bottom*” in the current frame and save its bottom location. In function **UpdateS** that will be paired with all 3DShapes, we check if the shape bottom is higher than our saved value then we move that shape to UI layer. In function **End** we restore original values of light direction and intensity.

4.3 Rotation effect

This is a script tagged “rot”:

```

function UpdateS()
    shape.Rot.y = frame * 2;
end

```

If we want any shape to rotate around Y axis continuously, just assign tag “rot” for it.

4.4 Day-night effect

```

function Update()
    f = frame % 600
    if (f < 200) then
        Light.Intensity = 0.5
    else
        Light.Intensity = 1.2
    end
end
end

```

This script changes the light intensity periodically to emulate the day/night effect.