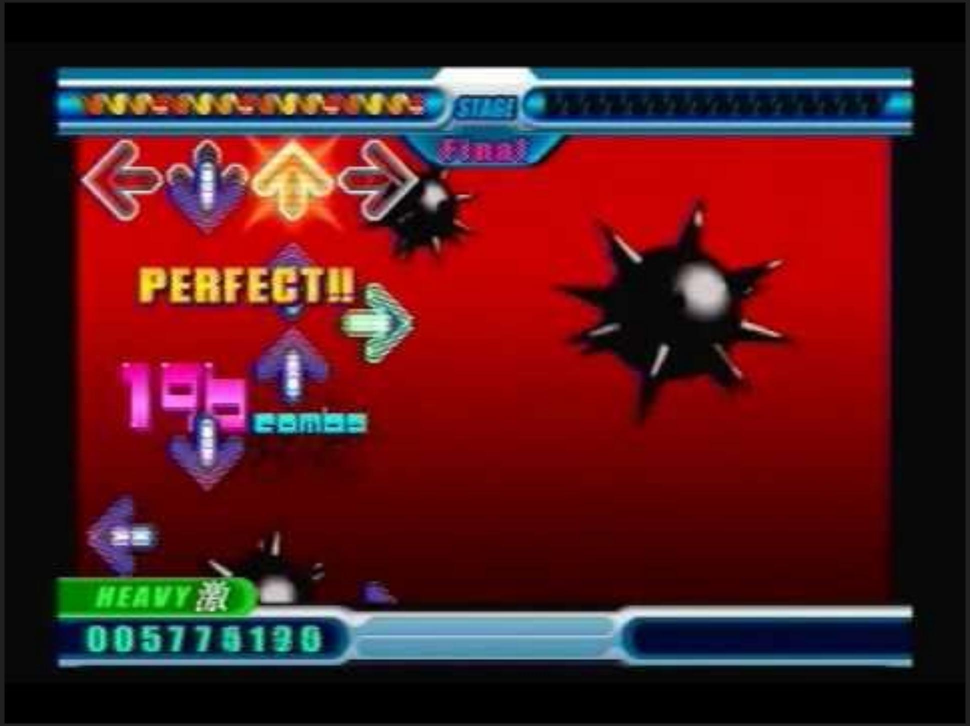
Not Into These Graphics

Making a NotITG Custom Mod Chart

Team 15

Dance Dance Revolution (DDR)

- Arrows scroll from edge of screen to receptors
- Player must step on directional pad when arrow meets receptor
- Steps are timed to music



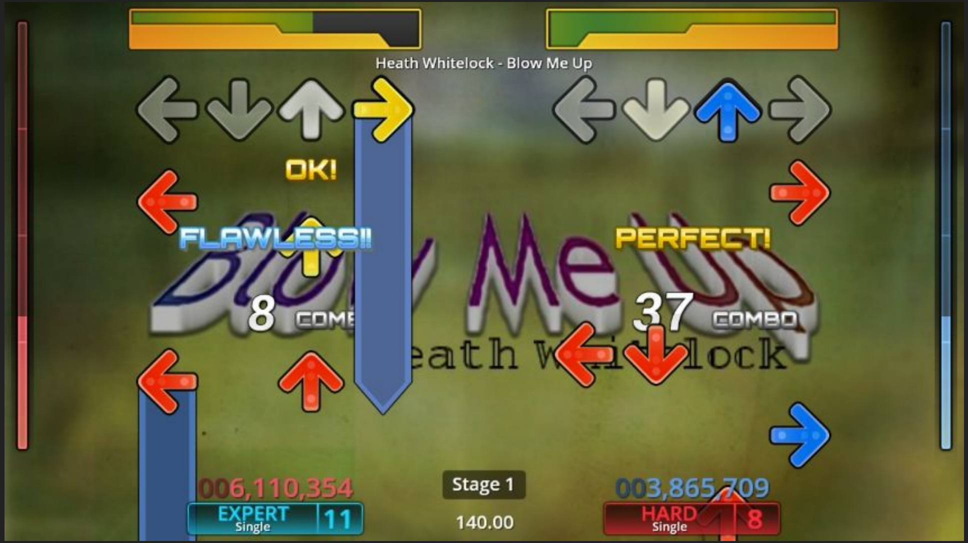
In The Groove (ITG)

- Arrows scroll from edge of screen to receptors
- Player must step on directional pad when arrow meets receptor
- Steps are timed to music

- But this time it's american

Stepmania

- Arrows scroll from edge of screen to receptors
- Player must step on directional pad when arrow meets receptor
- Steps are timed to music
- But this time it's open source



OpenITG

- Arrows scroll from edge of screen to receptors
- Player must step on directional pad when arrow meets receptor
- Steps are timed to music

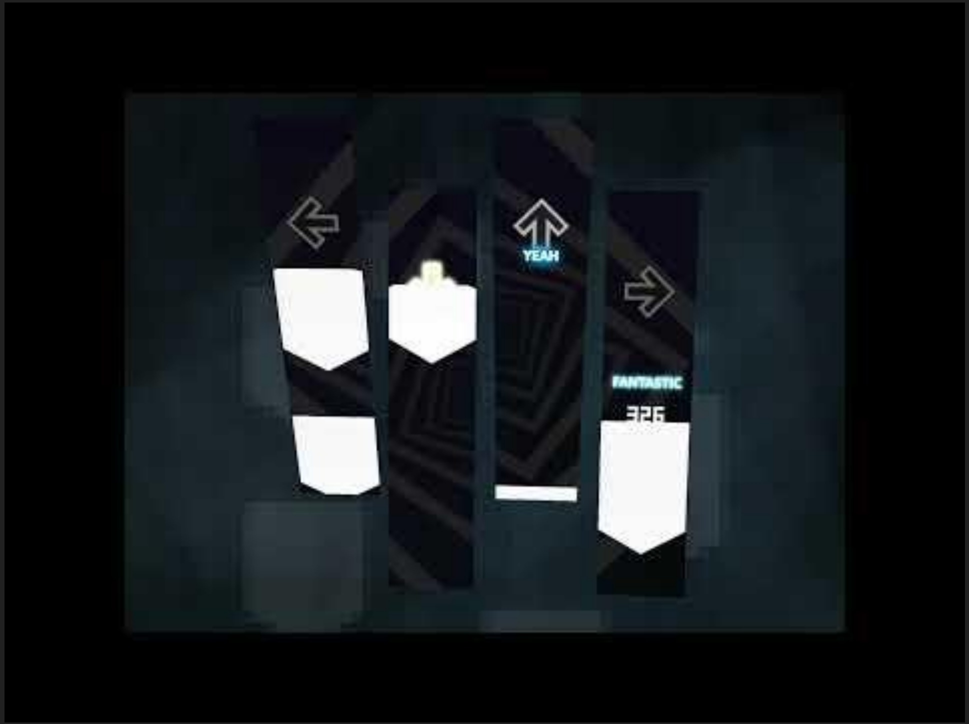
- But this time it’s made to be like that american version from before

Uh oh! You friccin moron. You just got
BEANED!!!
Tag your friends to totally BEAN! them!
facebook.com/greendelabean

NotITG

- Arrows scroll from edge of screen to receptors
- Player must step on directional pad when arrow meets receptor
- Steps are timed to music

- But this time there are mods



What does NotITG Support?

- Movement of elements in 3D space (transformation, scaling, rotation, etc)
- Predefined effects (scewing, spinning, transformations upon the arrow paths, etc)
- Ability to define your own effects
- Ability to set effects on specific beats
- Ability to smoothly transition between different intensity of effects over time (ease functions)
- Shader support (GLSL Fragmentation and Vertex Shaders)
- Much more

Demo

(Because trying to explain some of this without context is difficult)

Backup Demo

SELECT MUSIC EVENT

Xero0l
xeypooma
XMOD
Xyore
Yasmyn64
yukiko1212
-YOSEFU-
[ipies Wip Files]
Blow My Mind
Humanity
NCS
Jurassic
Kitsu Template
Umi's Secret
Mirin Template

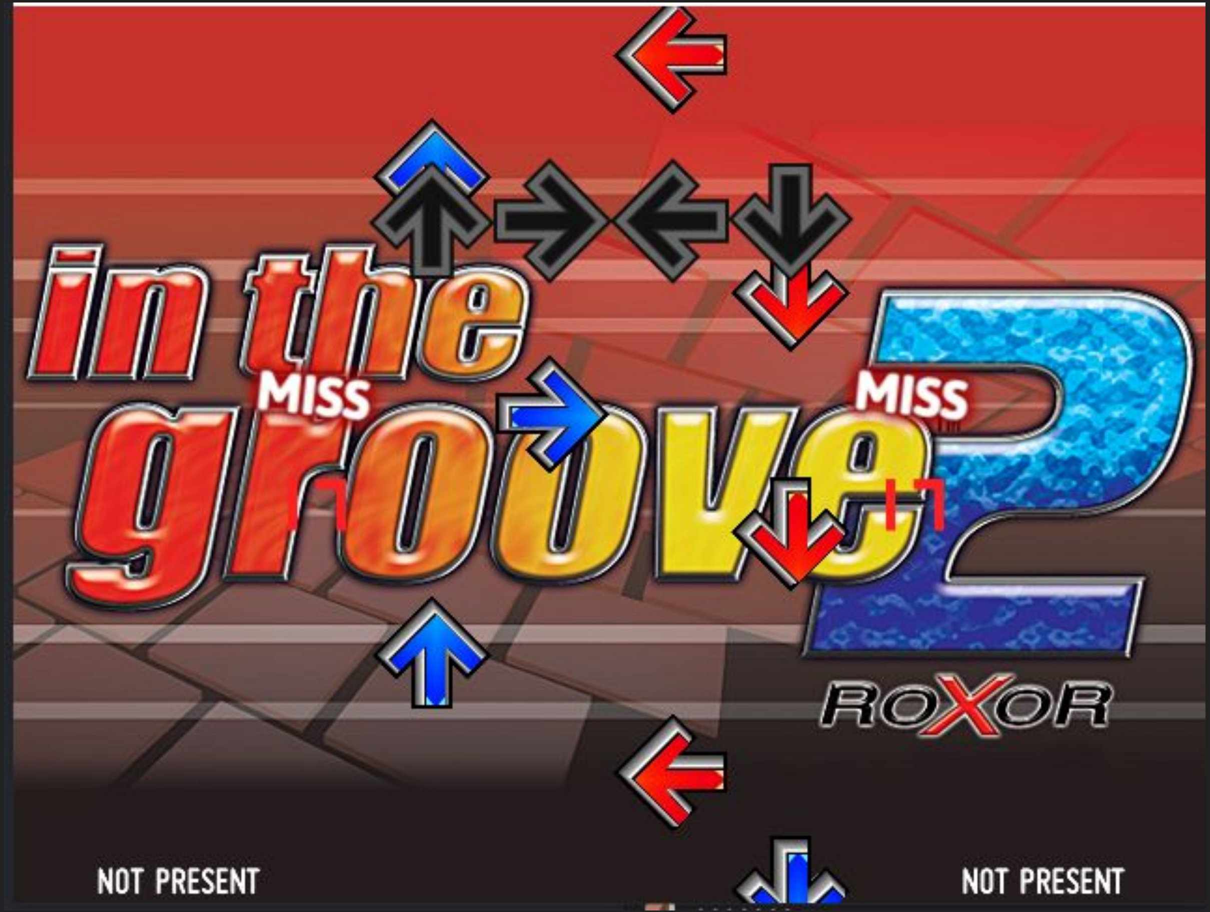
ARTIST
BPM LENGTH

? STEPS ? MINES ? ?
? JUMPS ? HANDS ? CARD
? HOLDS ? ROLLS

? STEPS ? MINES ? ?
? JUMPS ? HANDS ? CARD
? HOLDS ? ROLLS

The Beginning Section

- Center the Playfields
- Receptors Fade Out
- Column Swaps
- Danger/Warning Graphics
- The Spin

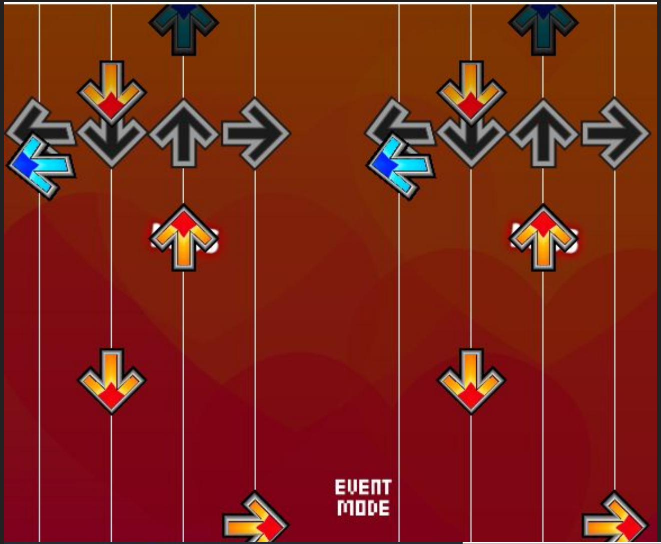


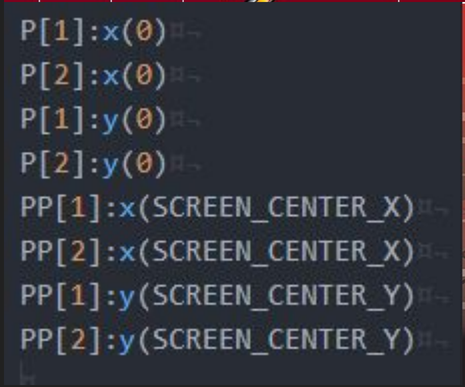
Centering the Playfields

By default, there are two playfields (One for player 1, and another for player 2). These playfields are normally side by side.

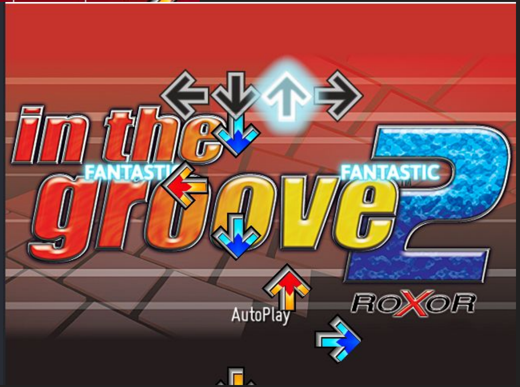
Centering them is easy thanks to NotITG.

Moved both playfields to (0,0), recentered using proxies.





```
P[1]:x(0)
P[2]:x(0)
P[1]:y(0)
P[2]:y(0)
PP[1]:x(SCREEN_CENTER_X)
PP[2]:x(SCREEN_CENTER_X)
PP[1]:y(SCREEN_CENTER_Y)
PP[2]:y(SCREEN_CENTER_Y)
```



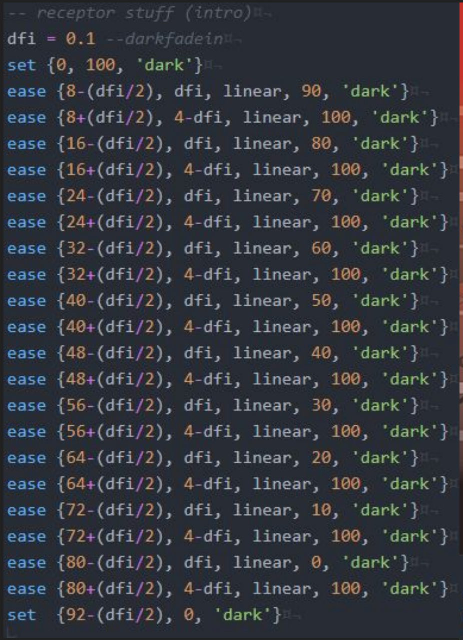
Receptor Fade Out and Column Swaps

NotITG already supports a way to change the receptor's alpha (dark) and move columns around (flip, invert)

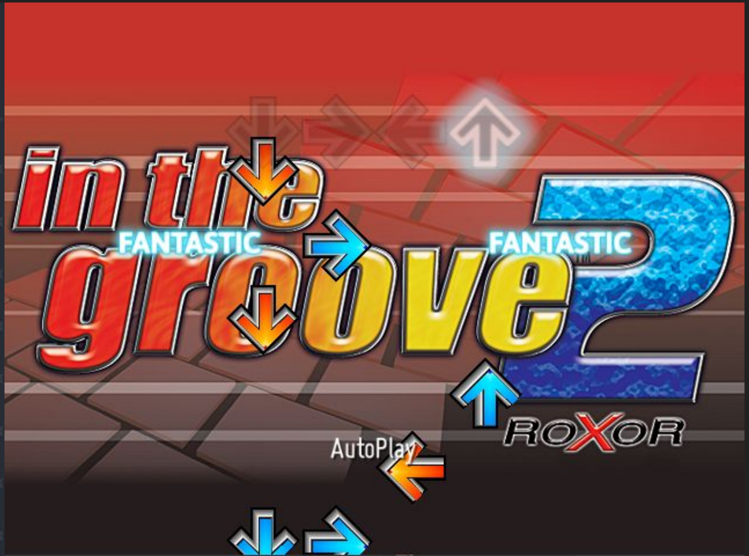
Flips reverses the arrow positions (left swaps with right, up swaps with down).

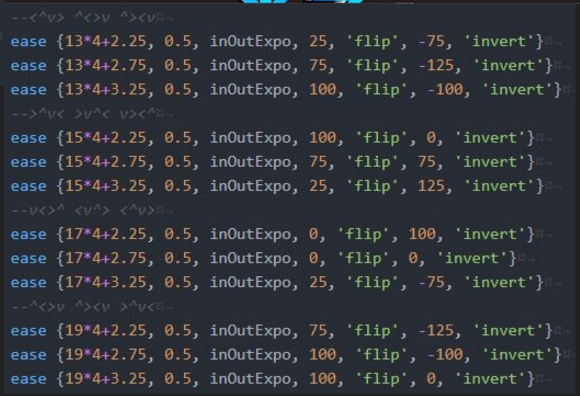
Invert swaps adjacent arrow positions (left swaps with down, up swaps with right).

Combining different percentages of flip and invert allows for different configurations of arrow positions.



```
-- receptor stuff (intro)
dfi = 0.1 --darkfadein
set {0, 100, 'dark'}
ease {8-(dfi/2), dfi, linear, 90, 'dark'}
ease {8+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {16-(dfi/2), dfi, linear, 80, 'dark'}
ease {16+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {24-(dfi/2), dfi, linear, 70, 'dark'}
ease {24+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {32-(dfi/2), dfi, linear, 60, 'dark'}
ease {32+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {40-(dfi/2), dfi, linear, 50, 'dark'}
ease {40+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {48-(dfi/2), dfi, linear, 40, 'dark'}
ease {48+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {56-(dfi/2), dfi, linear, 30, 'dark'}
ease {56+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {64-(dfi/2), dfi, linear, 20, 'dark'}
ease {64+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {72-(dfi/2), dfi, linear, 10, 'dark'}
ease {72+(dfi/2), 4-dfi, linear, 100, 'dark'}
ease {80-(dfi/2), dfi, linear, 0, 'dark'}
ease {80+(dfi/2), 4-dfi, linear, 100, 'dark'}
set  {92-(dfi/2), 0, 'dark'}
```

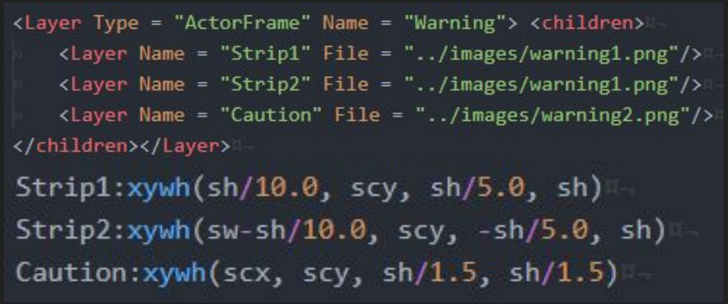




```
--<^v> ^<>v ^><v
ease {13*4+2.25, 0.5, inOutExpo, 25, 'flip', -75, 'invert'}
ease {13*4+2.75, 0.5, inOutExpo, 75, 'flip', -125, 'invert'}
ease {13*4+3.25, 0.5, inOutExpo, 100, 'flip', -100, 'invert'}
-->^v< >v^< v><^
ease {15*4+2.25, 0.5, inOutExpo, 100, 'flip', 0, 'invert'}
ease {15*4+2.75, 0.5, inOutExpo, 75, 'flip', 75, 'invert'}
ease {15*4+3.25, 0.5, inOutExpo, 25, 'flip', 125, 'invert'}
--v<>^ <v^> <^v>
ease {17*4+2.25, 0.5, inOutExpo, 0, 'flip', 100, 'invert'}
ease {17*4+2.75, 0.5, inOutExpo, 0, 'flip', 0, 'invert'}
ease {17*4+3.25, 0.5, inOutExpo, 25, 'flip', -75, 'invert'}
--^<>v ^><v >^v<
ease {19*4+2.25, 0.5, inOutExpo, 75, 'flip', -125, 'invert'}
ease {19*4+2.75, 0.5, inOutExpo, 100, 'flip', -100, 'invert'}
ease {19*4+3.25, 0.5, inOutExpo, 100, 'flip', 0, 'invert'}
```

Danger/Warning Graphics

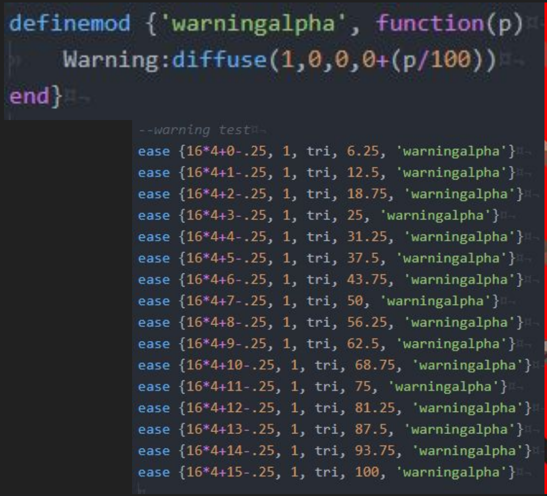
Added the warning triangle and strip to the background. Scaled images based on the screen height and positioned based on this height with some *math*.



```
<Layer Type = "ActorFrame" Name = "Warning"> <children>
    <Layer Name = "Strip1" File = "../images/warning1.png"/>
    <Layer Name = "Strip2" File = "../images/warning1.png"/>
    <Layer Name = "Caution" File = "../images/warning2.png"/>
</children></Layer>
```

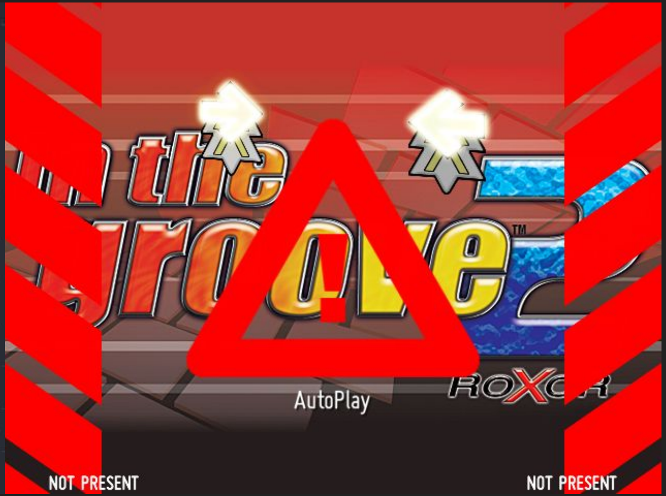
```
Strip1:xywh(sh/10.0, scy, sh/5.0, sh)
Strip2:xywh(sw-sh/10.0, scy, -sh/5.0, sh)
Caution:xywh(scx, scy, sh/1.5, sh/1.5)
```

Created a mod to change the alpha on the images, and used this to make it pulse to the beat.



```
definemod {'warningalpha', function(p)
    Warning:diffuse(1,0,0,0+(p/100))
end}
```

```
--warning test
ease {16*4+0-.25, 1, tri, 6.25, 'warningalpha'}
ease {16*4+1-.25, 1, tri, 12.5, 'warningalpha'}
ease {16*4+2-.25, 1, tri, 18.75, 'warningalpha'}
ease {16*4+3-.25, 1, tri, 25, 'warningalpha'}
ease {16*4+4-.25, 1, tri, 31.25, 'warningalpha'}
ease {16*4+5-.25, 1, tri, 37.5, 'warningalpha'}
ease {16*4+6-.25, 1, tri, 43.75, 'warningalpha'}
ease {16*4+7-.25, 1, tri, 50, 'warningalpha'}
ease {16*4+8-.25, 1, tri, 56.25, 'warningalpha'}
ease {16*4+9-.25, 1, tri, 62.5, 'warningalpha'}
ease {16*4+10-.25, 1, tri, 68.75, 'warningalpha'}
ease {16*4+11-.25, 1, tri, 75, 'warningalpha'}
ease {16*4+12-.25, 1, tri, 81.25, 'warningalpha'}
ease {16*4+13-.25, 1, tri, 87.5, 'warningalpha'}
ease {16*4+14-.25, 1, tri, 93.75, 'warningalpha'}
ease {16*4+15-.25, 1, tri, 100, 'warningalpha'}
```

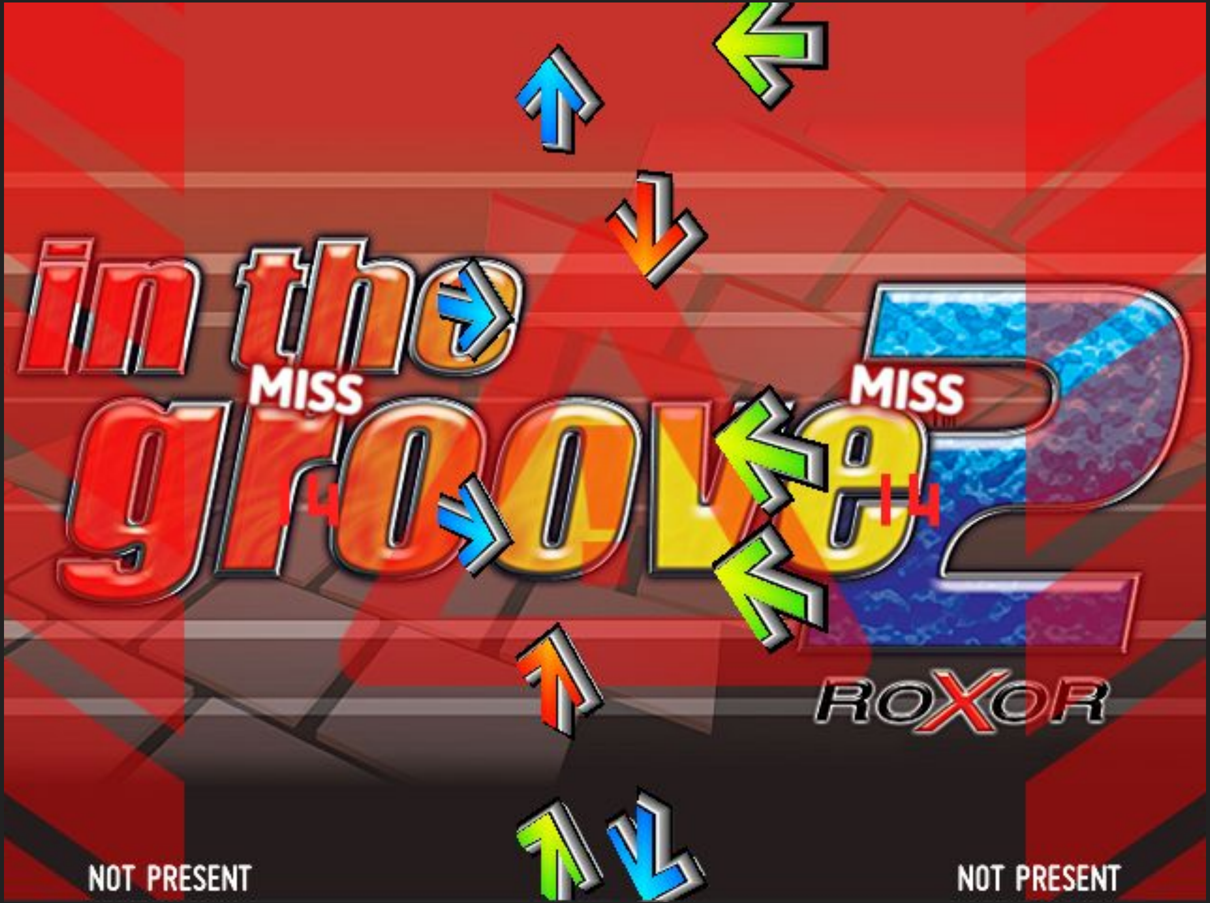


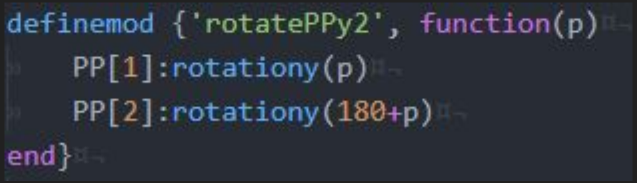
The Spin

NotITG already implements 3D rotation, but the y-axis it rotates upon is at x=0, z=0 (This is why the playfields were positioned at (0,0), so the center would be aligned to this axis, allowing the desired rotation).

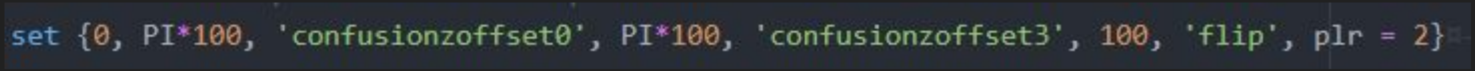
There is no backside to the arrows.

Use player 2's playfield as the backside. (This also requires flipping the arrows positions and rotating the left and right arrows 180 deg to align properly)

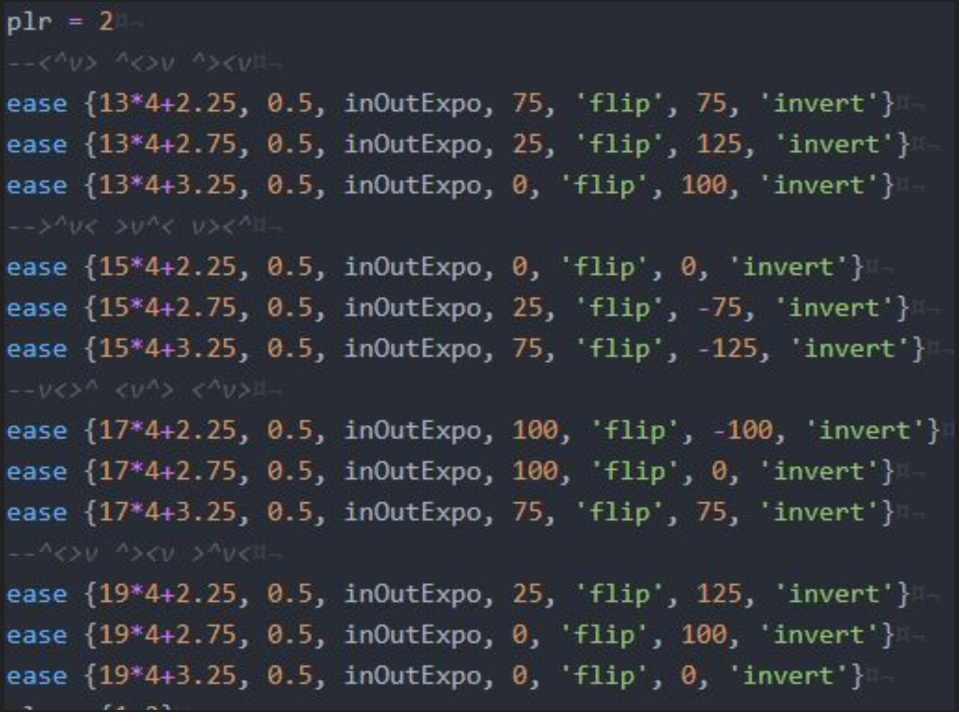




```
definemod {'rotatePPy2', function(p)
    PP[1]:rotationy(p)
    PP[2]:rotationy(180+p)
end}
```



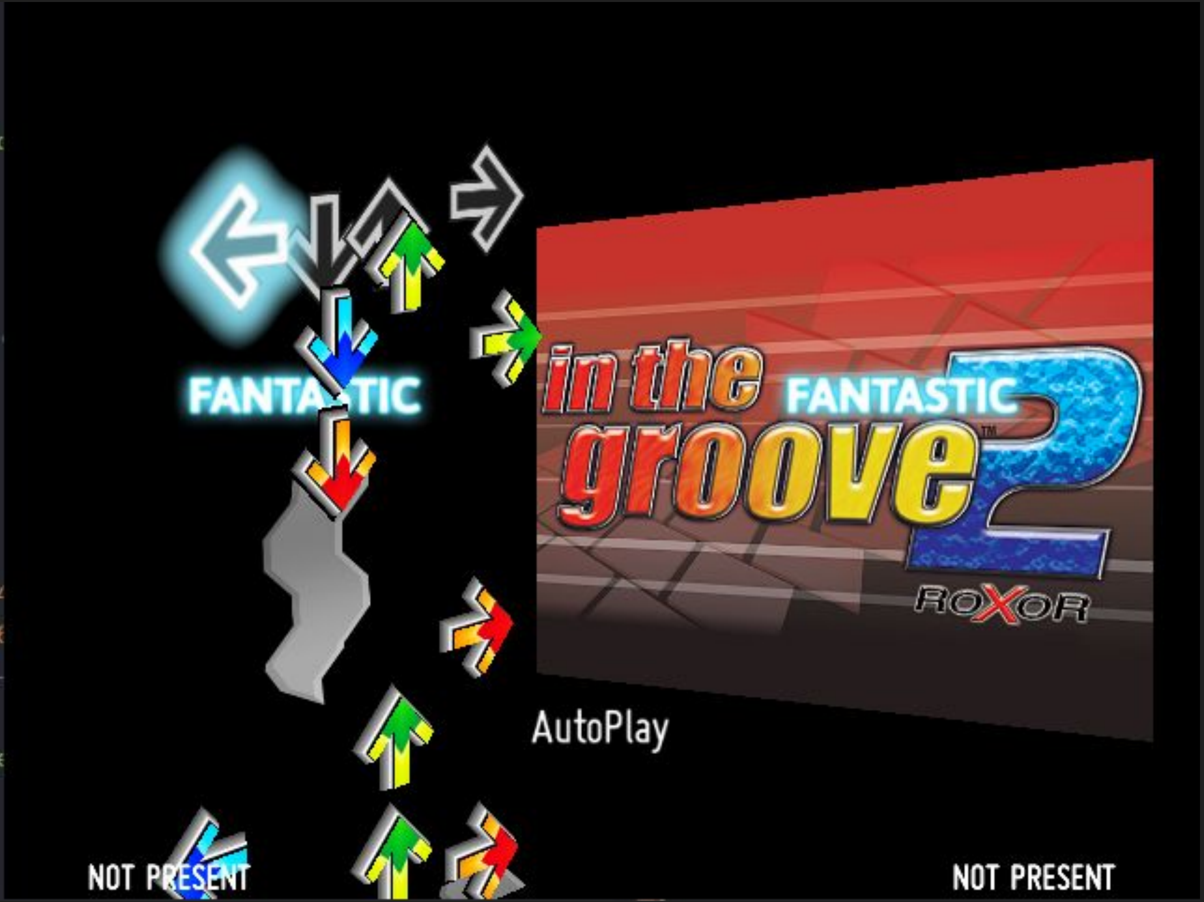
```
set {0, PI*100, 'confusionzoffset0', PI*100, 'confusionzoffset3', 100, 'flip', plr = 2}
```



```
plr = 2
--<^v> ^<>v ^><v
ease {13*4+2.25, 0.5, inOutExpo, 75, 'flip', 75, 'invert'}
ease {13*4+2.75, 0.5, inOutExpo, 25, 'flip', 125, 'invert'}
ease {13*4+3.25, 0.5, inOutExpo, 0, 'flip', 100, 'invert'}
-->^v< >v^< v><^
ease {15*4+2.25, 0.5, inOutExpo, 0, 'flip', 0, 'invert'}
ease {15*4+2.75, 0.5, inOutExpo, 25, 'flip', -75, 'invert'}
ease {15*4+3.25, 0.5, inOutExpo, 75, 'flip', -125, 'invert'}
--v<>^ <v^> <^v>
ease {17*4+2.25, 0.5, inOutExpo, 100, 'flip', -100, 'invert'}
ease {17*4+2.75, 0.5, inOutExpo, 100, 'flip', 0, 'invert'}
ease {17*4+3.25, 0.5, inOutExpo, 75, 'flip', 75, 'invert'}
--^<>v ^><v >^v<
ease {19*4+2.25, 0.5, inOutExpo, 25, 'flip', 125, 'invert'}
ease {19*4+2.75, 0.5, inOutExpo, 0, 'flip', 100, 'invert'}
ease {19*4+3.25, 0.5, inOutExpo, 0, 'flip', 0, 'invert'}
```

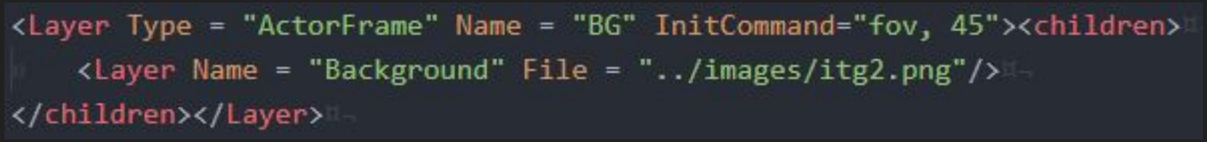
The Drop

- 3D Playfield and Background Movement
- Playfield Proxy Field Shader

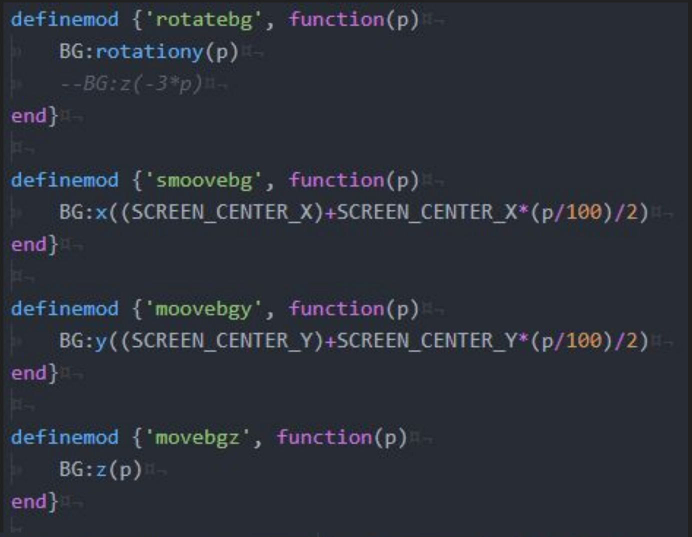


Background Movement

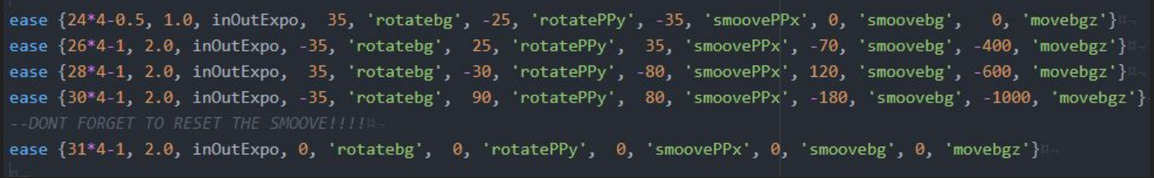
- By default, NotITG uses parallel projection (rotating the background will just make it look squished)
- Changing to perspective projection is as easy as setting an fov.
- NotITG also supports moving and rotating objects in 3D space, but not through eases
- However, a mod can be defined to do this, allowing easing of pre-set 3D movements



```
<Layer Type = "ActorFrame" Name = "BG" InitCommand="fov, 45"><children>
    <Layer Name = "Background" File = "../images/itg2.png"/>
</children></Layer>
```



```
definemod {'rotatebg', function(p)
    BG:rotationy(p)
    --BG:z(-3*p)
end}

definemod {'smoovebg', function(p)
    BG:x((SCREEN_CENTER_X)+SCREEN_CENTER_X*(p/100)/2)
end}

definemod {'moovebgy', function(p)
    BG:y((SCREEN_CENTER_Y)+SCREEN_CENTER_Y*(p/100)/2)
end}

definemod {'movebgz', function(p)
    BG:z(p)
end}
```

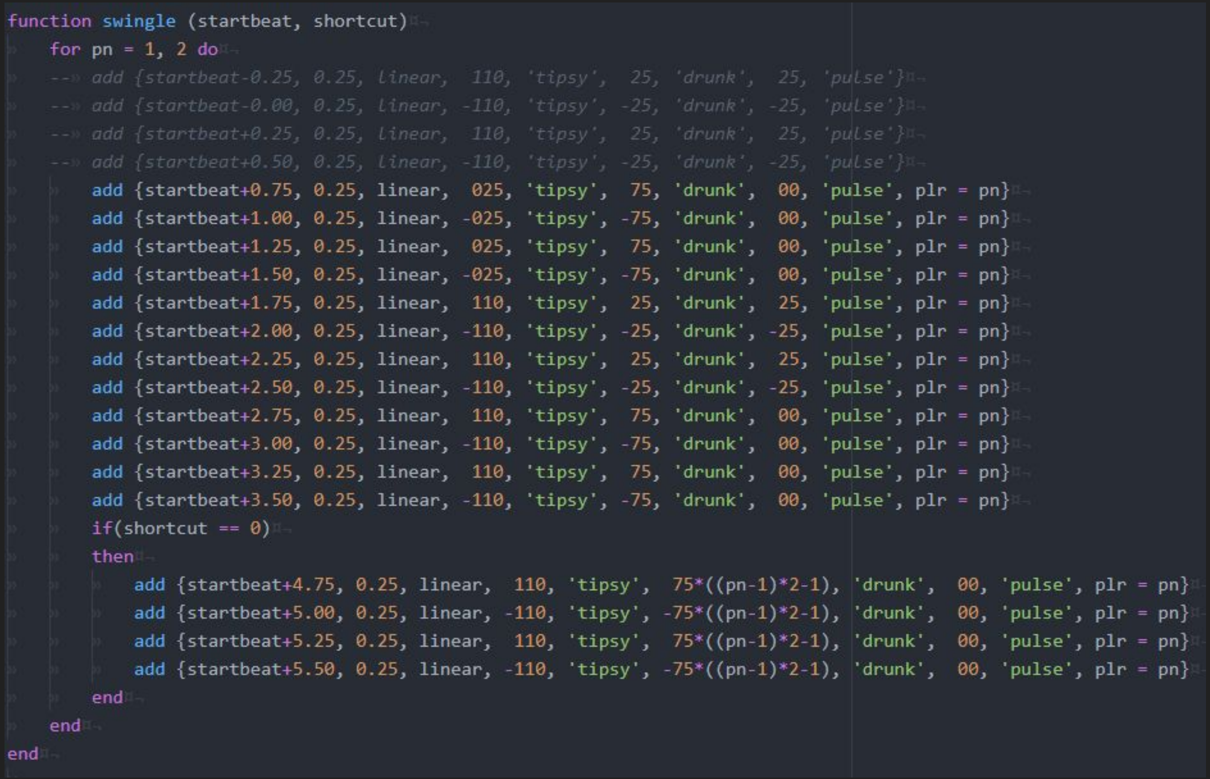


```
ease {24*4-0.5, 1.0, inOutExpo,  35, 'rotatebg', -25, 'rotatePPy', -35, 'smoovePPx', 0, 'smoovebg',   0, 'movebgz'}
ease {26*4-1, 2.0, inOutExpo, -35, 'rotatebg',  25, 'rotatePPy',  35, 'smoovePPx', -70, 'smoovebg', -400, 'movebgz'}
ease {28*4-1, 2.0, inOutExpo,  35, 'rotatebg', -30, 'rotatePPy', -80, 'smoovePPx', 120, 'smoovebg', -600, 'movebgz'}
ease {30*4-1, 2.0, inOutExpo, -35, 'rotatebg',  90, 'rotatePPy',  80, 'smoovePPx', -180, 'smoovebg', -1000, 'movebgz'}
--DONT FORGET TO RESET THE SMOOVE!!!!
ease {31*4-1, 2.0, inOutExpo, 0, 'rotatebg',  0, 'rotatePPy',  0, 'smoovePPx', 0, 'smoovebg', 0, 'movebgz'}
```

Arrow Movement

To make the arrows bounce around to the music, a combination of different predefined mods were used:

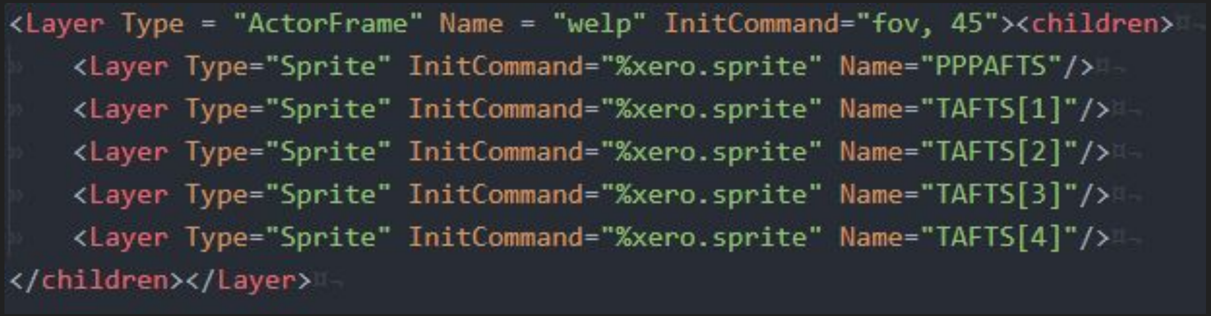
- Tipsy: Changes arrows relative y position according to a wave function
- Drunk: Changes arrows relative x position according to a wave function
- Pulse: Changes arrows relative size according to a wave function



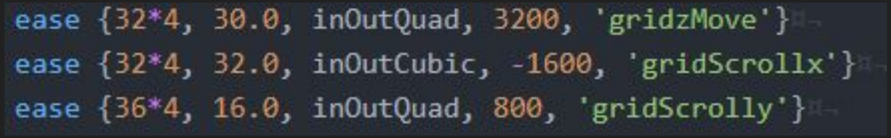
```lua
function swingle (startbeat, shortcut)
    for pn = 1, 2 do
    --  add {startbeat-0.25, 0.25, linear,  110, 'tipsy',  25, 'drunk',  25, 'pulse'}
    --  add {startbeat-0.00, 0.25, linear, -110, 'tipsy', -25, 'drunk', -25, 'pulse'}
    --  add {startbeat+0.25, 0.25, linear,  110, 'tipsy',  25, 'drunk',  25, 'pulse'}
    --  add {startbeat+0.50, 0.25, linear, -110, 'tipsy', -25, 'drunk', -25, 'pulse'}
        add {startbeat+0.75, 0.25, linear,  025, 'tipsy',  75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+1.00, 0.25, linear, -025, 'tipsy', -75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+1.25, 0.25, linear,  025, 'tipsy',  75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+1.50, 0.25, linear, -025, 'tipsy', -75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+1.75, 0.25, linear,  110, 'tipsy',  25, 'drunk',  25, 'pulse', plr = pn}
        add {startbeat+2.00, 0.25, linear, -110, 'tipsy', -25, 'drunk', -25, 'pulse', plr = pn}
        add {startbeat+2.25, 0.25, linear,  110, 'tipsy',  25, 'drunk',  25, 'pulse', plr = pn}
        add {startbeat+2.50, 0.25, linear, -110, 'tipsy', -25, 'drunk', -25, 'pulse', plr = pn}
        add {startbeat+2.75, 0.25, linear,  110, 'tipsy',  75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+3.00, 0.25, linear, -110, 'tipsy', -75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+3.25, 0.25, linear,  110, 'tipsy',  75, 'drunk',  00, 'pulse', plr = pn}
        add {startbeat+3.50, 0.25, linear, -110, 'tipsy', -75, 'drunk',  00, 'pulse', plr = pn}
        if(shortcut == 0)
        then
            add {startbeat+4.75, 0.25, linear,  110, 'tipsy',  75*((pn-1)*2-1), 'drunk',  00, 'pulse', plr = pn}
            add {startbeat+5.00, 0.25, linear, -110, 'tipsy', -75*((pn-1)*2-1), 'drunk',  00, 'pulse', plr = pn}
            add {startbeat+5.25, 0.25, linear,  110, 'tipsy',  75*((pn-1)*2-1), 'drunk',  00, 'pulse', plr = pn}
            add {startbeat+5.50, 0.25, linear, -110, 'tipsy', -75*((pn-1)*2-1), 'drunk',  00, 'pulse', plr = pn}
        end
    end
end
```

How to fake a 3D matrix of playfields

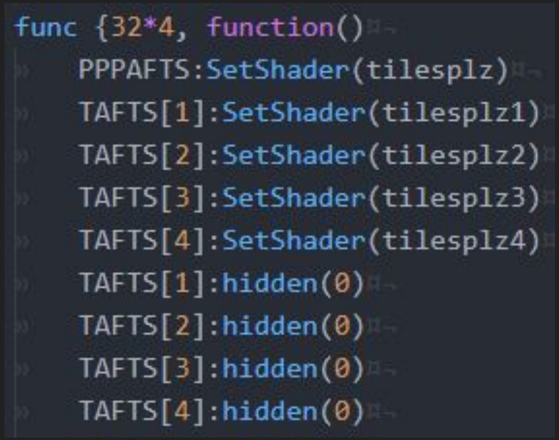
- Set up 5 proxies of playfields at different z positions. Start with only the furthest back one being within the camera, and move them back gradually to reveal more proxies.
- Apply a shader to each proxy to make it look like there is a grid of playfields
- Use the shader to make it looks like the playfields are moving in the x and y directions.
- Use a chromakey in the shader to remove the background (make the background transparent)



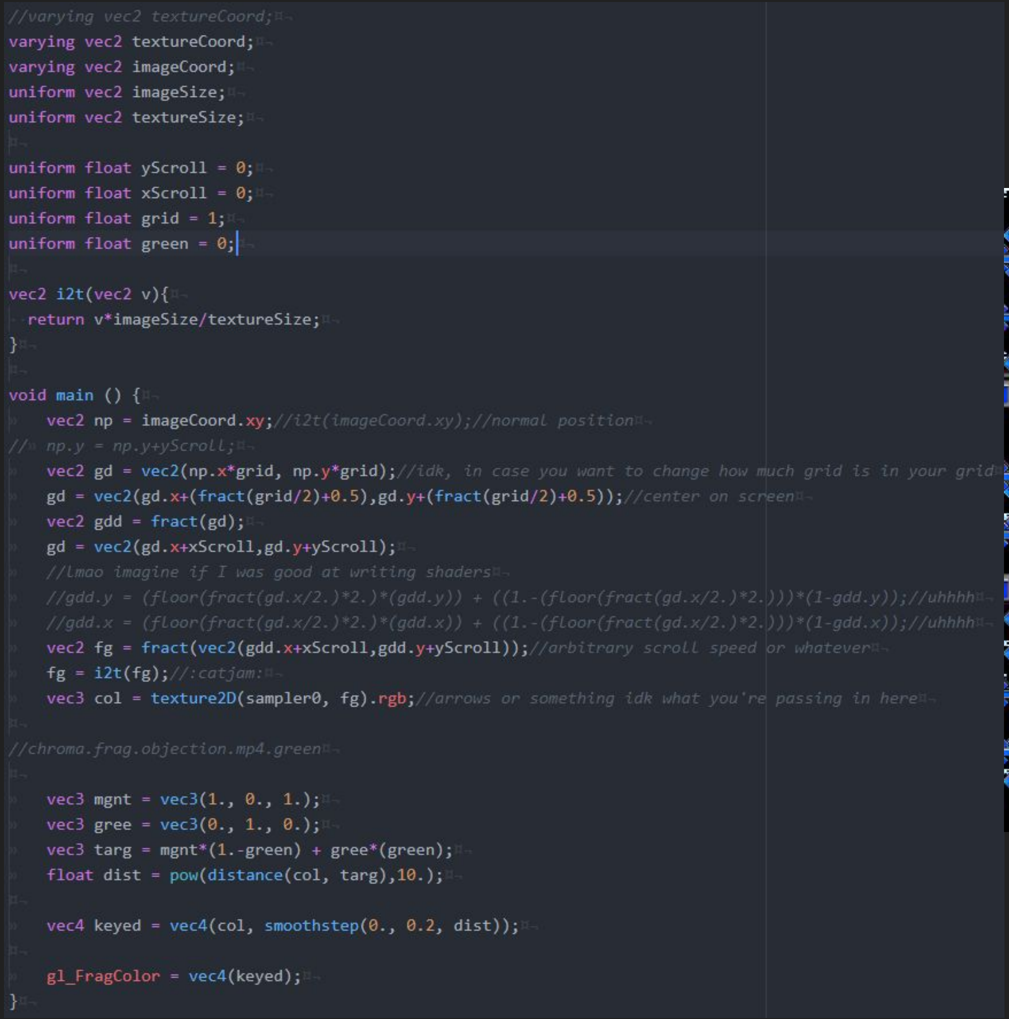
```
<Layer Type = "ActorFrame" Name = "welp" InitCommand="fov, 45"><children>
    <Layer Type="Sprite" InitCommand="%xero.sprite" Name="PPPAFTS"/>
    <Layer Type="Sprite" InitCommand="%xero.sprite" Name="TAFTS[1]"/>
    <Layer Type="Sprite" InitCommand="%xero.sprite" Name="TAFTS[2]"/>
    <Layer Type="Sprite" InitCommand="%xero.sprite" Name="TAFTS[3]"/>
    <Layer Type="Sprite" InitCommand="%xero.sprite" Name="TAFTS[4]"/>
</children></Layer>
```



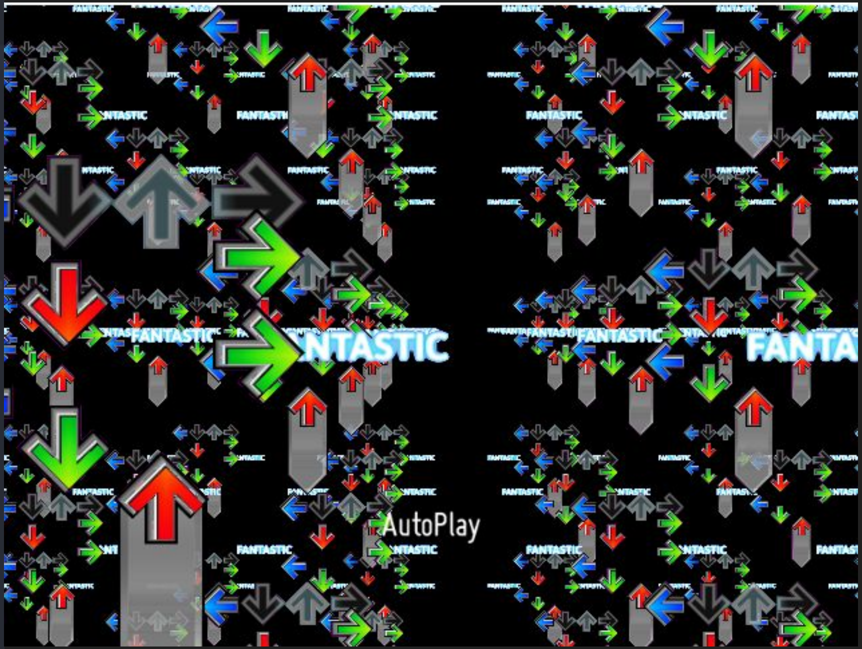
```
ease {32*4, 30.0, inOutQuad, 3200, 'gridzMove'}
ease {32*4, 32.0, inOutCubic, -1600, 'gridScrollx'}
ease {36*4, 16.0, inOutQuad, 800, 'gridScrolly'}
```



```
func {32*4, function()
    PPPAFTS:SetShader(tilesplz)
    TAFTS[1]:SetShader(tilesplz1)
    TAFTS[2]:SetShader(tilesplz2)
    TAFTS[3]:SetShader(tilesplz3)
    TAFTS[4]:SetShader(tilesplz4)
    TAFTS[1]:hidden(0)
    TAFTS[2]:hidden(0)
    TAFTS[3]:hidden(0)
    TAFTS[4]:hidden(0)
```



```glsl
//varying vec2 textureCoord;
varying vec2 textureCoord;
varying vec2 imageCoord;
uniform vec2 imageSize;
uniform vec2 textureSize;

uniform float yScroll = 0;
uniform float xScroll = 0;
uniform float grid = 1;
uniform float green = 0;

vec2 i2t(vec2 v){
  return v*imageSize/textureSize;
}

void main () {
    vec2 np = imageCoord.xy;//i2t(imageCoord.xy);//normal position
//  np.y = np.y+yScroll;
    vec2 gd = vec2(np.x*grid, np.y*grid);//idk, in case you want to change how much grid is in your grid
    gd = vec2(gd.x+(fract(grid/2)+0.5),gd.y+(fract(grid/2)+0.5));//center on screen
    vec2 gdd = fract(gd);
    gd = vec2(gd.x+xScroll,gd.y+yScroll);
    //lmao imagine if I was good at writing shaders
    //gdd.y = (floor(fract(gd.x/2.)*2.)*(gdd.y)) + ((1.-(floor(fract(gd.x/2.)*2.)))*(1-gdd.y));//uhhhh
    //gdd.x = (floor(fract(gd.x/2.)*2.)*(gdd.x)) + ((1.-(floor(fract(gd.x/2.)*2.)))*(1-gdd.x));//uhhhh
    vec2 fg = fract(vec2(gdd.x+xScroll,gdd.y+yScroll));//arbitrary scroll speed or whatever
    fg = i2t(fg);//:catjam:
    vec3 col = texture2D(sampler0, fg).rgb;//arrows or something idk what you're passing in here

//chroma.frag.objection.mp4.green

    vec3 mgnt = vec3(1., 0., 1.);
    vec3 gree = vec3(0., 1., 0.);
    vec3 targ = mgnt*(1.-green) + gree*(green);
    float dist = pow(distance(col, targ),10.);

    vec4 keyed = vec4(col, smoothstep(0., 0.2, dist));

    gl_FragColor = vec4(keyed);
}
```



Questions?