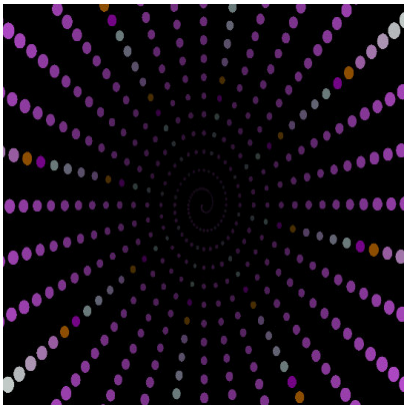
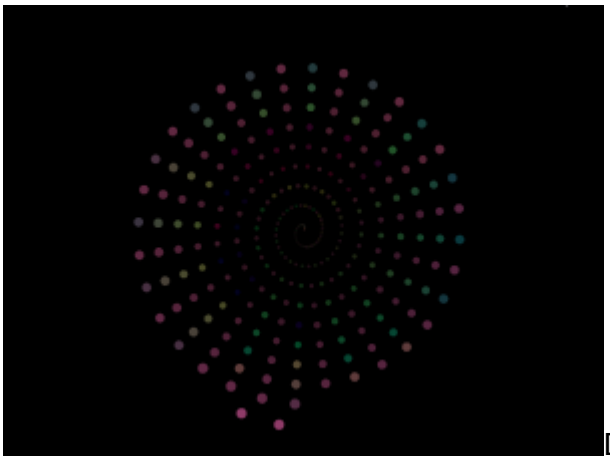


Spiraleffekt in GMS 2

Description

Im Nicole Marie T Intro gibt es einen Spiraleffekt. Die Spirale baut sich aus einzelnen Kreisen auf, wechselt zwischendurch die Farben und neigt sich anschließend um 45°. In diesem Tutorial zeige ich, wie das geht.



Dieses Tutorial baut auf das erlernte Wissen der Effekte Falling numbers und Schicker Hintergrundeffekt auf. Deshalb werde ich darauf verzichten, die dort gezeigten Grundlagen erneut zu erklären.

Im Kern funktioniert der Spiraleffekt ähnlich. Wir erstellen uns im Create-Event ein Array, welches wir mit den benötigten Werten pro Kreis füllen. Durch Alarm 0 bauen wir die Spirale langsam auf, etwa wie im Memory im GameMaker Tutorial. Dort wurden die Karten bei Spielstart einzeln auf den Tisch gelegt. Sobald alle Kreise sichtbar sind, wechseln wir die Farbe und neigen, ebenfalls in Alarm 0, langsam die Spirale um 45°.

Der Effekt bietet zahlreiche Möglichkeiten um ihn zu ändern und zu erweitern. Beispielsweise könnte man die „Ecken“ des Kreises ändern, die Form der Spirale, die Form durch Rechtecke oder Dreiecke austauschen und vieles mehr. Deiner Kreativität sind keine Grenzen gesetzt!

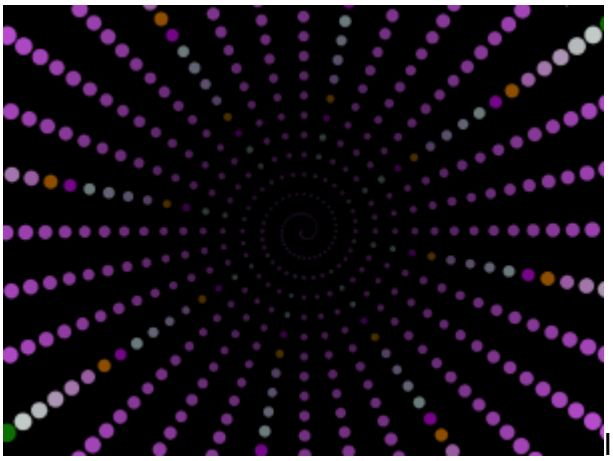
obj_spiraleffekt

Da GMS 2 bekanntlich den ersten Raum selbst erzeugt, brauchen wir lediglich das Objekt. Im Objekt legen wir die drei Events Create, Alarm 0 und Draw an.

Event-Create

```
game_set_speed(60, gamespeed_fps);

// Wir verschieben das Objekt in die Mitte des Raumes.
x = room_width / 2;
y = room_height / 2;

rad = 0;       // Radius

pixel_pro_kreis = 32;
kreise = 24;
all_pixel = pixel_pro_kreis * kreise;

pixel_aktuell = 0;

turn = 0;
var c = 0;
col_count = 0;
neigung = 0;

for(k=0; k<kreise; k++)
{
    dd = 0;
    turn2 = 0;
    for(var i=0; i<pixel_pro_kreis; i++)
    {
        pixel[c, 0] = x + lengthdir_x(rad+42*k, dd) + turn2;    // x
        pixel[c, 1] = y + lengthdir_y(rad+42*k, dd);  // y
        pixel[c, 2] = 227;     // r
        pixel[c, 3] = 95;     // g
        pixel[c, 4] = 128+i+k*2;    // b
        pixel[c, 5] = make_colour_rgb(pixel[c, 2], pixel[c, 3], pixel[c, 4]);
        pixel[c, 6] = (k+1)/kreise+0.25;   // Alpha

        c++;
        turn2++;

        if (dd < 360)
        {
            dd += 360 / pixel_pro_kreis;    // Abstand in Grad. Anzahl der Pix
        } else {
```

```
            dd = 0;
        }
    }
}
```

 In GMS 1.x arbeitet man gerne mit **room_speed**, aber das funktioniert in GMS 2 nur bedingt gut. Stattdessen benutzen wir hier **game_get_speed** und legen die Geschwindigkeit mit **game_set_speed** fest.

Nun haben wir ein kleines Problem mit den Bezeichnungen der Variablen. Die Spirale besteht aus Kreisen (Ringen) und jeder Ring aus einer bestimmten Anzahl an Kreisen (**draw_circle**). Damit es weniger Missverständnisse gibt, habe ich die Kreise (**draw_circle**) bei den Variablen *Pixel* genannt. Das hat den Vorteil, dass wir später statt Kreise auch Dreiecke oder andere Formen nehmen können, ohne gleich alle Variablen ändern zu müssen. Wir tun also so, als wäre jede Form nur ein Bildpunkt.

Wir definieren Pixel pro Kreis und die Anzahl der Kreise, welche die Spirale haben wird. Letzteres hängt vor allem vom Raum bzw. der Bildschirmbreite ab. Nachdem wir weitere Variablen definiert haben, kommen unsere beiden Schleifen für die Kreise und die einzelnen Pixel pro Kreis. Jeder „Pixel“ besitzt sieben Werte. Das ließe sich beliebig erweitern oder gar kürzen, aber mit diesen Werten hat man eine sehr gute Ausgangsposition. Wir haben:

- x-Koordinate
- y-Koordinate
- Farbwert Rot
- Farbwert Grün
- Farbwert Blau
- Summe der Farbe
- Alpha

Am Ende rufen wir Alarm 0 auf.

Event-Draw

Bevor wir den Alarm anschauen, gehen wir in unser Draw-Event.

```
var c = 0;
```

```
var turn2 = 0;

for(k=0; k<kreise; k++)
{
    var dd = 0;

    for(var i=0; i<pixel_pro_kreis; i++)
    {
        if (pixel_aktuell > c)
        {
            var radius = round(2+(c/42));
            draw_set_alpha(pixel[c, 6]);
            draw_set_color(pixel[c, 5]);
            //draw_set_circle_precision(24);
            draw_circle(pixel[c, 0], pixel[c, 1], radius, 0);

            if (dd < 360)
            {
                dd += 360 / pixel_pro_kreis;    // Abstand in Grad. Anzahl der
            } else {
                dd = 0;
                turn = 0;
            }

            // Spirale
            pixel[c, 0] = x + lengthdir_x(neigung*4+turn2, dd+turn);    // x m
            pixel[c, 1] = y + lengthdir_y(4+turn2, dd+turn);  // y
            pixel[c, 6] = (k+1)/kreise+0.05;    // Alpha

            // Der Pixel wird immer heller
            if (col_count = 3)
            {
                if (pixel[c, 3] < 254)
                {
                    pixel[c, 2] += 227;
                    pixel[c, 3] += k;
                    pixel[c, 4] += 255;
                } else {
                    pixel[c, 3] = 110;
                    pixel[c, 4] = 0;
                    pixel[c, 5] = 100+i+k;
                }
                // Definiert die Farbe anhand der neuen RGB-Wert Daten
                pixel[c, 5] = make_colour_rgb(pixel[c, 2], pixel[c, 3], pixel[
                col_count = 0;
            } else {
                col_count++;
            }

            c++;

            if (turn < 360)
            {
                turn += 0.001; // Drehgeschwindigkeit
            } else {
```

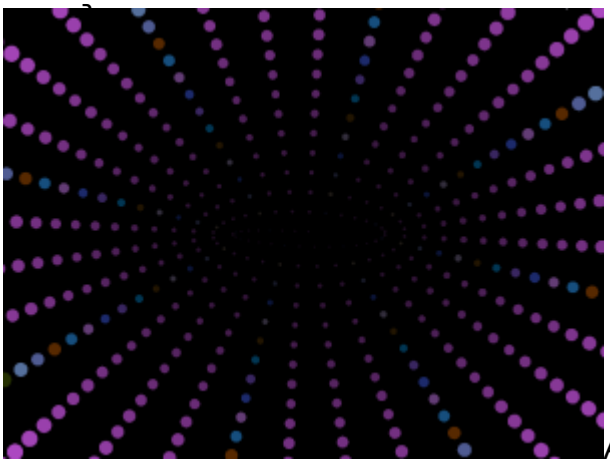
```
                turn = 0;
            }

   // Regelt den Abstand der Spiralen
            if (turn2 < (360*6))
            {
                turn2 += 1.05;
            } else {
                turn2 = 0;
            }
        }
    }
}
```

Im Prinzip haben wir wieder unsere beiden schleifen und zeichnen innerhalb der zweiten Schleife unsere Pixel. In Zeile 15 habe ich *draw_set_circle_precision(24);* auskommentiert. Damit wird die Genauigkeit der Kreise festgelegt. Der Wert 4 führt zu einem Viereck, der Wert 5 zu einem Fünfeck und so weiter. Dreiecke lassen sich so nicht zeichnen, dafür müsste man **draw_rectangle** statt **draw_circle** nehmen, aber immerhin haben wir hier schon einige Möglichkeiten zur Veränderung.

Der Rest vom Code regelt vorwiegend Abstand, Geschwindigkeit und Farbe der Pixel. Ab Zeile 32 verändern wir die Farbe jedes Pixel systematisch. Die Farbwerte befinden sich innerhalb einer Palette und lassen sich sehr leicht anpassen.

Alarm 0

```
// Erzeugt zunächst alle Kreise der Spirale
if (pixel_aktuell < all_pixel)
{
    pixel_aktuell+=2;
    alarm[0] = 0.01 * game_get_speed(gamespeed_fps);
} else {
 // Wenn alle Kreise da sind, neigen wir die Spirale
 // Bevor wir damit beginnen, wechseln wir aber noch die Farben
    if (neigung = 0)
    {
        for(var i=0; i<all_pixel; i++)
        {
            pixel[i, 2] = 227;
            pixel[i, 3] = 95;
            pixel[i, 4] = 255;
            pixel[i, 5] = make_colour_rgb(pixel[i, 2], pixel[i, 3], pixel[i, 4
            col_count=0;
        }
    }

 // Und hier entsteht die Neigung, bis wir bei 45° angekommen sind.
    if (neigung < 45)
    {
        neigung += 0.1;
        alarm[0] = 0.05 * game_get_speed(gamespeed_fps);
```

 Aufgrund der Kommentare ist der Code ziemlich

selbsterklärend. Die Geschwindigkeit des Aufbaus lässt sich über die Alarm-Parameter sehr einfach regeln. Ebenfalls das kippen der Spirale. Wer den Aufbau des Effekts überspringen möchte, kann im Create-Event auch *pixel_aktuell = all_pixel;* statt *pixel_aktuell = 0;* schreiben.

Der Farbwechsel in Alarm 0 kommt nach *if (neigung = 0)*. Hier geben wir jedem Pixel eine neue Farbe. Aufgrund der Regelung im Draw-Event mit *if (col_count = 3)* bekommt jedes vierte Pixel im Kreis eine andere Farbe.

Das war es auch schon. Wie gesagt, lassen sich damit viele tolle Dinge anstellen. Mit dem selben Prinzip lässt sich auch der alte Pixeltunnel-Effekt von den zahlreichen Objekten zu einem Objekt umstellen. Allerdings ist hier die Berechnung der exakten Bewegung etwas komplexer. In Tests musste ich leider feststellen, dass die Geschwindigkeit in diesem Fall kaum schneller ist, als wenn man

alles über ein Array regelt. 10-20% kann man aber trotzdem rausholen.

Date Created
15. Juni 2018
Author
sven