Sonnenblumen und der goldene Schnitt

Description

Es ist mal wieder Zeit für einen grafischen Effekt. In diesem Tutorial geht es um eine Art Spiraleffekt, der wie das Innere einer Sonnenblume aussieht. Im Kern fußt es auf den Goldenen Schnitt und bietet viele Variationen. Dieser Effekt ist keine neue Erfindung, aber es ist interessant, die Umsetzung im GameMaker anzuschauen. Dabei greifen wir auf die Zeichenfunktionen zurück, statt auf Shader. So ist es bspw. möglich, den Effekt für einen Gegner im Spiel zu nutzen, der statt aus geometrischen Formen, aus kleinen Objekten/Sprites besteht.

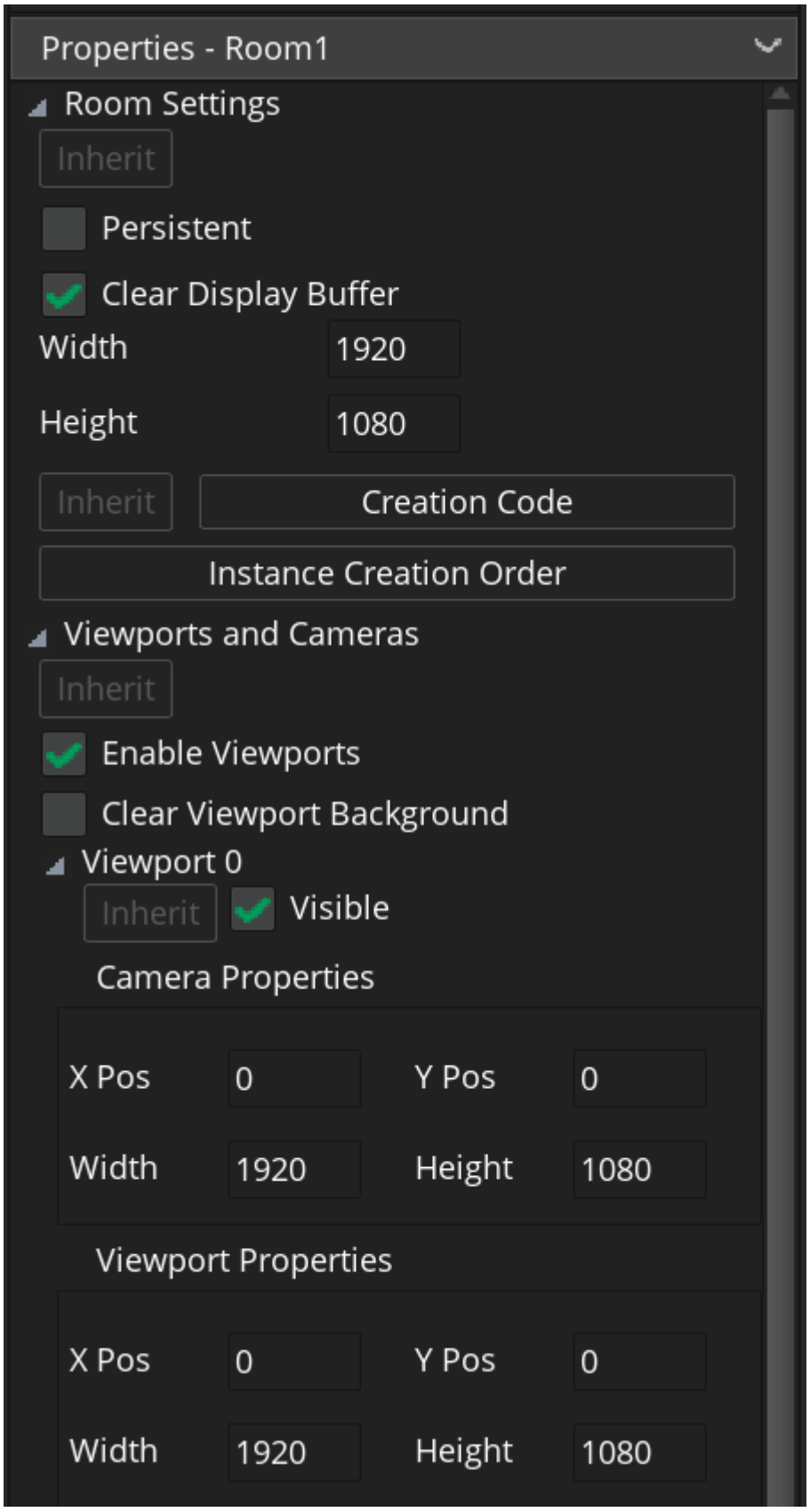
Unser Ziel

Das Resultat, oder besser, eine Variation davon, könnt ihr euch bereits hier ansehen. Schaut ihn ruhig etwas länger an, da in dieser Version die Formen alle 10 Sekunden durchgeschaltet werden.

Statt dieses Sonnenblumenmuster lassen sich mit leichten Änderungen auch andere Formen wie normale Spiralen erschaffen.

Kamera

Die Drehung des Effekts basiert auf einer Kameradrehung. Die muss man nicht haben, sieht aber wesentlich besser aus, wenn man nur den Effekt zeigt. Der Code dafür ist weiter unten eingebettet, aber damit er funktioniert, müsst ihr den View einschalten. So sehen meine Einstellungen aus:



Raumeinstellungen

Lass uns programmieren!

Der Effekt ist nicht schwer zu programmieren und hat erstaunlich wenig Zeilen. Die Theorie dahinter ist aber etwas umfangreicher. Wer sie verstehen will, sollte sich zunächst den oben stehenden Link zum Goldenen Schnitt anschauen.

Event Create

```
dotSize = 42;
radius = room_width+dotSize;
phi = (1 + sqrt(5)) / 2;

step = 0;
sspeed = 2;
frame = 1000;

// Camera
angle = 0;
```

Die Variable *dotSize* gibt den Radius der Punkte an. Genauer gesagt wird er später aufgrund dieser Zahl errechnet. *radius* gibt die Reichweite des Effekts an. Im Beispiel geht er über die Raumgrenzen hinaus. Da sich der Effekt zeitweise zusammen zieht, sieht das recht beeindruckend aus. Eine schöne Variante:

```
radius = sqrt(room_width/2*room_height/2);
```

oder:

```
radius = room_height/2-dotSize;
```

Auf die Variablen *step*, *sspeed* und *frame* gehe ich im Draw-Event ein. Wichtig ist nur, dass man später *sspeed* für weitere Variationen gut nutzen kann.

Zu guter Letzt setzen wir noch den Winkel für die Kamera.

Event Step

```
camera_set_view_angle(view_camera[0], angle);
angle += .65;
```

Über *camera_set_view_angle* steuern wir unsere Kamera. Diese wird über den Winkel in 0,65 Schritten pro Step gedreht. Das sind 39°/Sekunde. Nach rund 9,231 Sekunden erreichen wir eine volle Drehung. Lässt sich natürlich je nach Geschmack verändern.

Event-Draw

```
draw_set_alpha(.7);
```

```
var t = frac(step/frame);
var count = 360*(1-cos(t*2*pi)+1);

for (var i=0; i<count; i++)
{
 var f = i / count;
 var a = i * phi;
 var dist = f * radius;
 var xx = .5 + cos(a * 2*pi) * dist;
 var yy = .5 + sin(a * 2*pi) * dist;
 var sig = power(1-cos((f-t*4)*2*pi)*.5+.5, 2);
 var r = sig * f * dotSize;

 var hue = f/2*i;
 var sat = 210;
 var val = 180*sig+r;

 var col = make_color_hsv(hue, sat, val);
 draw_set_color(col);

 draw_circle(room_width/2+xx, room_height/2+yy, r, 0);
}

step += sspeed;
```

Ja, das war es schon. Im Prinzip besteht der Effekt aus einer for-Schleife und zahlreichen Variablen. Doch schauen wir sie uns zeilenweise an.

Der Alpha-Wert von 0,7 wirkt vor allem bei Überlappungen. Das sieht m. M. n. deutlich besser aus, als 1.

Wie man sieht, kommen hier unsere drei Variablen, *step*, *sspeed* und *frame* zum Einsatz. Diese steuern die Geschwindigkeit des Ablaufs. In der Variable *t* kommen *step* und *frame* vor. Dabei wird *step* ganz unten von *sspeed* erhöht. D. h. diese kümmert sich um den Fortschritt der Animation. Ohne *step* würde sich im Beispiel nur die Kamera drehen bzw. wenn *sspeed* null wäre. Zu *frame* komme ich gleich.

Zunächst ist es wichtig zu wissen, dass *count* in Abhängigkeit zu *t* steht. Beide Werte zählen bis zu einem Maximum hoch und gehen dann zum Ursprungswert zurück. *t* geht von 0 bis 1 und steht für time. *count* geht von 360 bis 1080 (also 3×360). *t* springt von 1 auf 0 zurück, count wird rückwärts gezählt. Da *count* von *t* und *t* von *frame* abhängig ist, versteht man diese Variable besser. *frame* ist der Teiler und sagt letztlich nichts anderes aus, als die einzelnen „Bruchstücke” der Animation bzw. wie viele Stücke ein Durchlauf hat.

Durch die Formel *360*(1-cos(t*2*pi)+1)* bekommen wir eine gewisse Dynamik in die Animation, weil die Amplitude normalisiert wird. D. h. statt einem geraden Verlauf erhalten wir, je nach Phase, unterschiedliche Geschwindigkeiten.

Die for-Schleife

count ist also eine Variable, die zwischen 360 und 1080 liegen kann. Diese Variable ist das Maximum

der for-Schleife. D. h. pro Step (sind 60 pro Sekunde) wird die Schleife zwischen 360 und 1080 mal durchgeführt. Würde man in der Schleife statt *i<count* die *t*-Variable als *i<count*t* einsetzen, würde man sehen, wie sich der Effekt langsam aufbaut. Nun zu den nächsten Variablen:

- *f* steht für fraction, also den Bruchteil. Es ist eine Zahl zwischen 0 und 1.
- *a* für angle, den Winkel. Hier kommt der goldene Schnitt zum Einsatz. Tipp: Versucht mal *a = i / 18*. Interessant sind auch *i * pi* und *i / pi*.
- *dist* ist die Distanz. Damit erreichen wir in *xx* und *yy* die Spirale. Hätten wir hier bspw. nur *dist = radius*, wäre es lediglich ein Kreis.
- *xx* ist die x-Koordinate des jeweiligen Kreises
- *yy* ist die y-Koordinate des jeweiligen Kreises
- *sig* ist ein weiterer Parameter für Dynamik. Es steht für Signal und sorgt dafür, dass die Kreise nicht konstant größer werden, sondern manche Bereiche wieder kleiner. Auch damit lässt sich herumspielen.
- *r* ist der Radius des Kreises. Indem wir es mit *sig* und *f* multiplizieren, werden die Kreise von Innen nach Außen immer größer.

Farben

Jetzt geht es endlich an die Farbe. Hierbei ist es besser, statt RGB den HSV-Farbraum zu verwenden, was aber nichts mit dem Fußballverein zu tun hat. Wir arbeiten also mit

- Farbwert
- Sättigung
- Leuchtkraft

Alle Werte im GameMaker gehen von 0 bis 255. Vorsicht: Die Sättigung sieht in HTML5 nicht so intensiv aus wie unter Windows.

Im vorliegenden Beispiel ist die Sättigung konstant, wir variieren lediglich Farben und Helligkeit. Dadurch bekommen wir weitere Dynamik in die Animation. Auch hier kann man sehr viel experimentieren.

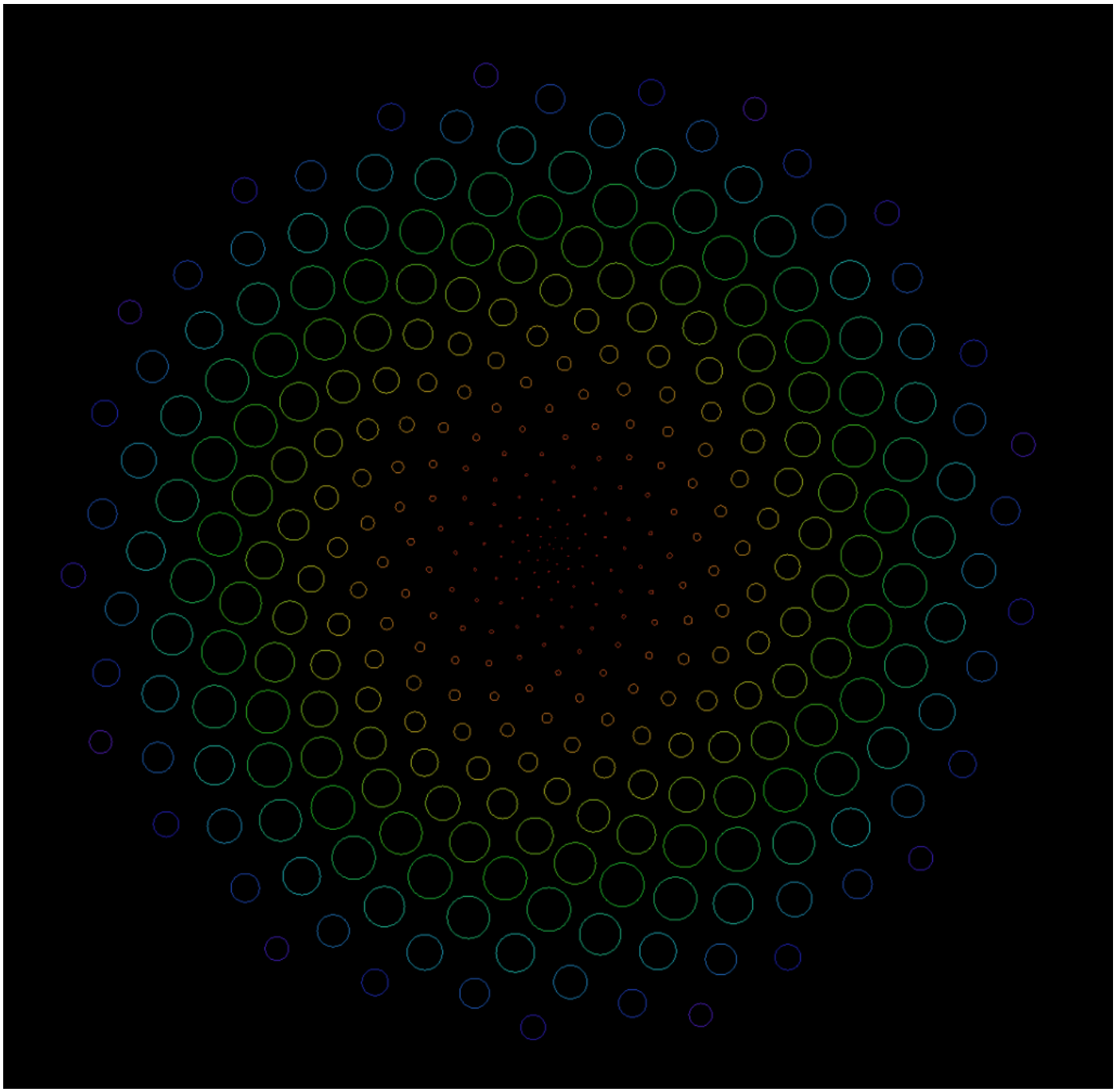
Am Ende wird die Farbe mit *col = make_color_hsv(hue, sat, val)* generiert und mit *draw_set_color(col)* gesetzt.

Geometrie

Die Kreise werden dann mit folgender Formel gezeichnet:

```
draw_circle(room_width/2+xx, room_height/2+yy, r, 0);
```

Setzt man hinten statt 0 eine 1, bekommt man es als Outline.



Beispiel für eine Variation

Im verlinkten Beispiel wechsle ich per Timer alle zehn Sekunden zwischen verschiedenen Geometrien, mal voll, mal outline, durch. Der Abschnitt sieht dann so aus:

```
if (timer == 1)
{
 draw_triangle(room_width/2+xx, room_height/2+yy, room_width/2+xx+r*2, room_he
} else if (timer == 2) {
 draw_rectangle(room_width/2+xx, room_height/2+yy, room_width/2+xx+r*2, room_h
} else if (timer == 3) {
 draw_rectangle(room_width/2+xx, room_height/2+yy, room_width/2+xx+r*2, room_h
} else if (timer == 4) {
```

```
 draw_circle(room_width/2+xx, room_height/2+yy, r, 0);
} else {
 draw_circle(room_width/2+xx, room_height/2+yy, r, 1);
}
```

Hierfür definiert man im Create-Event den Timer und ruft alarm[0] nach 10 Sekunden auf:

```
timer = 1;
alarm[0] = room_speed*10;
```

Dann fehlt nur noch alarm[0]:

```
if (timer < 5)
{
 timer++;
} else {
 timer = 1;
}

alarm[0] = room_speed*10;
```

Fertig. Viel Spaß beim experimentieren!

Date Created
10. Juni 2022
Author
sven