Laporan Praktikum Godot 6

Muhammad Fauzan Lubis

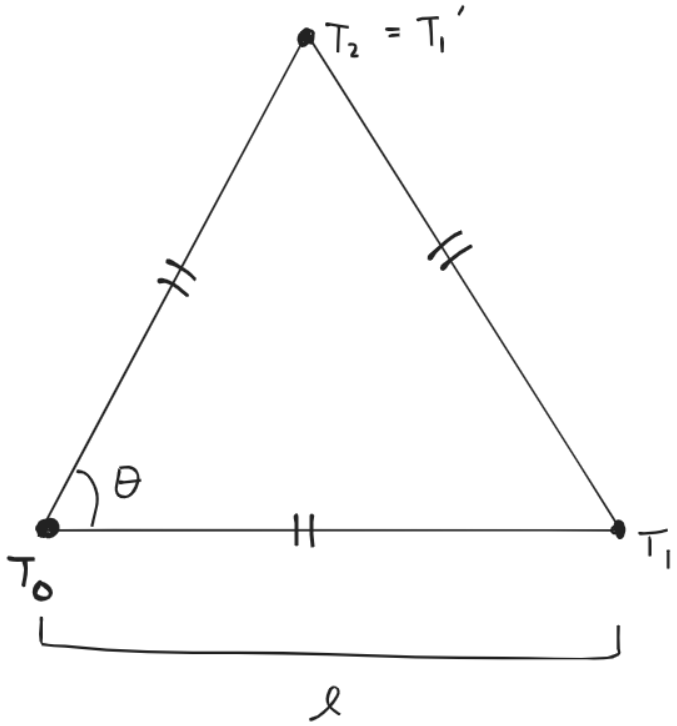
191524026

D4-2A

Rangkuman

Segitiga Sama Sisi

Dalam menggambar segitiga sama sisi, dilakukan transformasi rotasi dari titik utama (disini digunakan titik paling kiri). Visualisasi dari transformasi itu sendiri dapat dilihat seperti di bawah ini,



Titik awal $T_0(x_0, y_0)$ di translasikan dengan ℓ di arah horizontal yang akan menjadi titik $T_1(x_0 + \ell, y_0)$. Lalu titik paling atas merupakan suatu rotasi dari titik T_1 dengan anchor point T_0 dengan $\theta = 60^\circ$ (karena $\frac{1}{3}180^\circ = 60^\circ$). Untuk

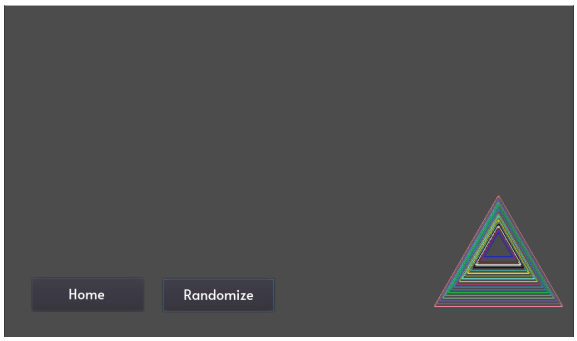
mempermudah melakukan transformasi, dibuat kelas baru, dengan 2 fungsi statis, source code dari kode tersebut adalah,

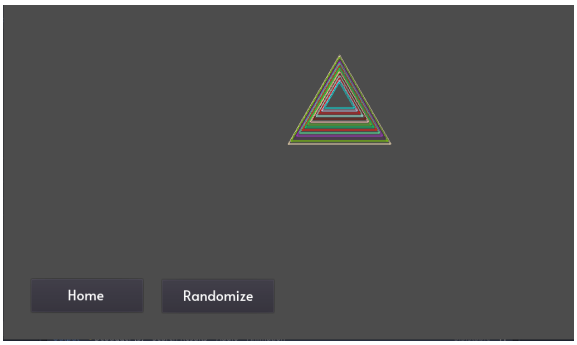
```
static func translate(start: Vector2, amount: Vector2) -> Vector2:
  return Vector2(start.x + amount.x, start.y + amount.y)

static func rotate(point: Vector2, anchor: Vector2, degree: float) -> Vector2:
  var rot_cos = cos(deg2rad(degree))
  var rot_sin = sin(deg2rad(degree))

  return Vector2(
    anchor.x + ((point.x - anchor.x) * rot_cos) - ((point.y - anchor.y) * rot_sin),
    anchor.y + ((point.x - anchor.x) * rot_sin) + ((point.y - anchor.y) * rot_cos)
  )
```

Lalu untuk menggambar segitiganya sendiri,

```
#...
func draw_triangle(left_point: Vector2, length: float, color: Color = Color.white):
  var points = [left_point]

  points.append(Utils.translate(left_point, Vector2(length, 0)))
  points.append(Utils.rotate(left_point, points[1], 60))

  draw_points(points, color)
#...
```

Dan dalam scene-nya itu sendiri, segitiga tersebut dilakukan randomisasi, dan juga translasi ke dalamnya dengan merubah titik awal, dan besarnya. Hasilnya terlihat seperti di bawah ini,





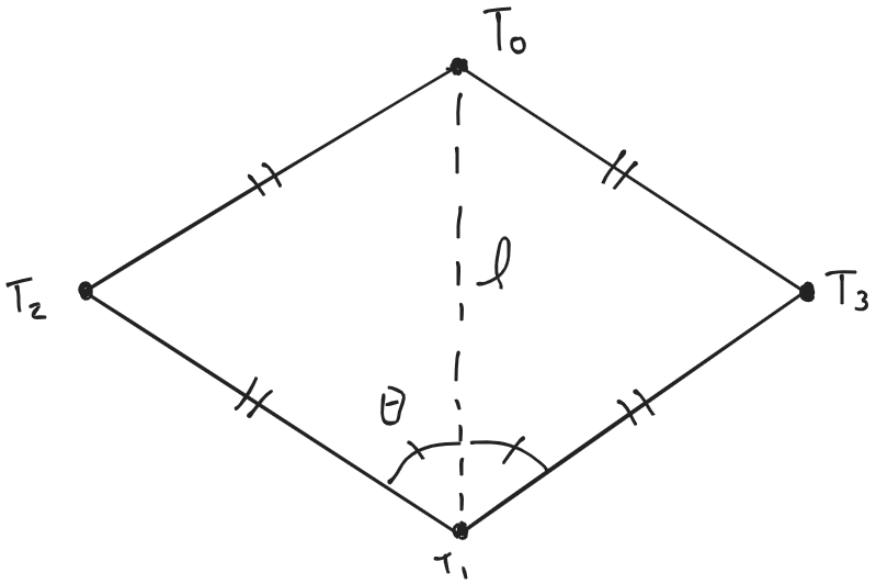
Dan dalam melakukan pengambarannya digunakan kode di bawah ini,

```
extends Shapes
```

```
func _draw():
  var viewport_size = get_viewport().size

  randomize()
  var size = rand_range(60, 250)
  var point = Vector2(rand_range(0, viewport_size.x - size), rand_range(size, viewport
_size.y))

  while size - 50 > 0:
    randomize()
    draw_triangle(point, size, Color(randf(), randf(), randf()))

    point = Utils.translate(point, Vector2(5, -5))
    size -= 10


func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")


func _on_Button2_pressed():
  update()
```

Belah Ketupat

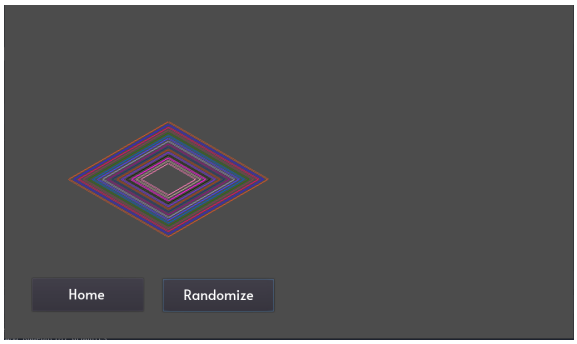
Dalam membuat belah ketupat, sistem pengambarannya mirip dengan menggambarkan segitiga. Hanya saja di sini dilakukan 2 kali rotasi, yang membuatnya kurang lebih menjadi 2 segitiga sama sisi dengan yang satu di rotasikan 180° dari yang lainnya.

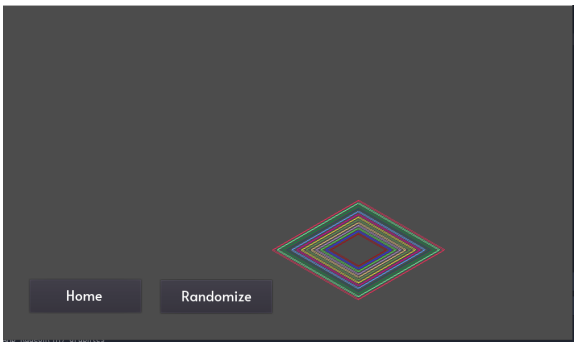


Terlihat dari gambar, cara untuk mendapatkan titik T_1, T_2 sama dengan mencarinya dengan yang ada di segitiga sama sisi. Sementara untuk mendapatkan titik T_3 dilakukan rotasi dengan $\theta = -60^\circ$. Lalu, kode untuk menggamarkannya sendiri,

```
#...
func draw_rhombus(top_point: Vector2, length: float, color: Color = Color.white):
  var points = [top_point]
  var bottom_points = Utils.translate(top_point, Vector2(0, length))

  points.append(Utils.rotate(top_point, bottom_points, 60))
  points.append(bottom_points)
  points.append(Utils.rotate(top_point, bottom_points, -60))

  draw_points(points, color)
#...
```

Di scene nya juga sama dengan segitiga, dimana dilakukan randomisasi. Lalu, dilakukan translasi titik awal dengan mengurangi komponen x dan y nya, juga mengurangi besar dari ℓ. Hasilnya terlihat seperti di bawah ini,





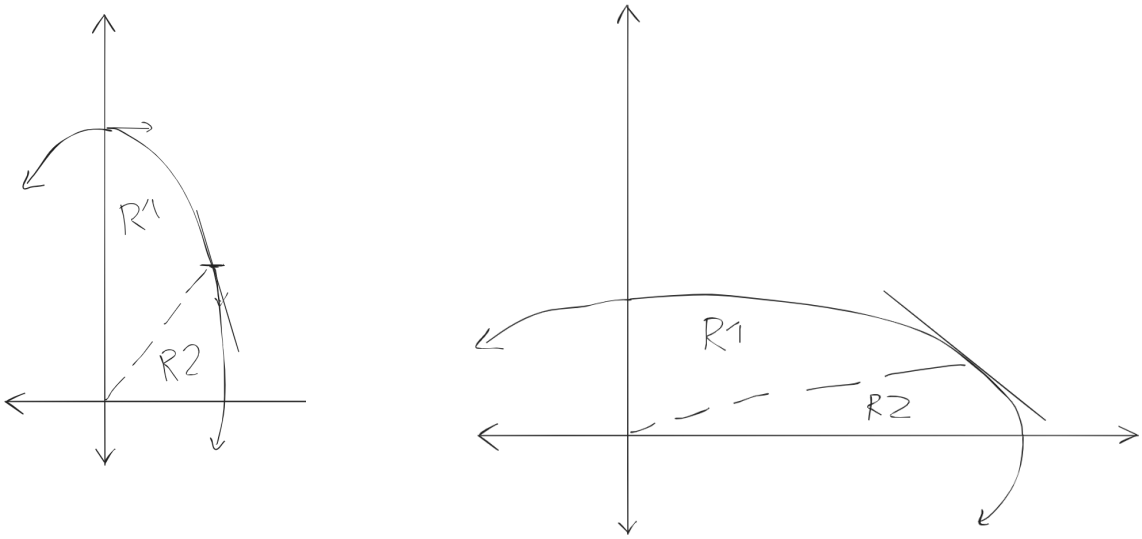
Dan kode yang mengaturnya,

```
extends Shapes


func _draw():
  var viewport_size = get_viewport().size

  var size = rand_range(60, 250)
  var point = Vector2(rand_range(0, viewport_size.x - size), rand_range(size, viewport
_size.y))

  while size - 50 > 0:
    randomize()
    draw_rhombus(point, size, Color(randf(), randf(), randf()))
```

```
    point = Utils.translate(point, Vector2(0, 5))
    size -= 10



func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")


func _on_Button2_pressed():
  update()
```

Elips

Dalam menggambar elips di sini digunakan algoritma midpoint elips,



Dimana sistem kerjanya kurang lebih sama dengan algoritma lingkaran midpoint, hanya saja di sini apa yang digambarkan memiliki simetri 4. Juga dalam menggambarkan salah satu simetrinya terdapat 2 bagian, dimana bagian pertama $R1$ memiliki $m < -1$, dan $R2$ yang memiliki $m > -1$, kedua nya akan digambarkan dengan cara yang berbeda.

Di $R1$ digambarkan dengan $x + 1$, jadi dilakukan penggambaran "ke depan" dengan keputusan dimana apakah y akan menjadi $y - 1$ atau tetap y. Sementara untuk $R2$, digambarkan dengan $y + 1$, dimana akan digambarkan "ke bawah". Keputusan yang ada di sini apakah x tetap, atau menjadi $x + 1$.

Implementasi dari midpoint elipsnya itu sendiri,

```
#...
func draw_ellipse_midpoint(rx: float, ry: float, center_point: Vector2, params: Ellips
eParams):
  var x = 0
  var y = ry

  var ry_square = ry * ry
  var rx_square = rx * rx

  var d1 = ((ry * ry) - (rx * rx * ry) + (0.25 * rx * rx))
  var dx = 2 * ry * ry * x
  var dy = 2 * rx * rx * y

  # While m < -1, Region 1
  while(dx < dy):
    put_pixel_every_quadrant(x, y, center_point, params)

    if d1 < 0:
      x += 1
      dx = dx + (2 * ry * ry)
      d1 = d1 + dx + (ry * ry)
    else:
      x += 1
      y -= 1
      dx = dx + (2 * ry * ry)
      dy = dy - (2 * rx * rx)
      d1 = d1 + dx - dy + (ry * ry)


  var d2 = (((ry * ry) * ((x + 0.5) * (x + 0.5))) +
      ((rx * rx) * ((y - 1) * (y - 1))) -
       (rx * rx * ry * ry))

  while y >= 0:
    put_pixel_every_quadrant(x, y, center_point, params)

    if d2 > 0:
      y -= 1
      dy = dy - (2 * rx * rx)
      d2 = d2 + (rx * rx) - dy
    else:
      y -= 1
      x += 1
      dx = dx + (2 * ry * ry)
      dy = dy - (2 * rx * rx)
      d2 = d2 + dx - dy + (rx * rx)

func put_pixel_every_quadrant(x: float, y: float, center_point: Vector2, params: Ellip
seParams):
  var to_draw = params.quadrant_to_draw
  var colors = params.quadrant_color

  var points = PoolVector2Array([
    Vector2(x + center_point.x, y + center_point.y),
    Vector2(x + center_point.x, -y + center_point.y),
    Vector2(-x + center_point.x, -y + center_point.y),
```

```
    Vector2(-x + center_point.x, y + center_point.y)
  ])

  for i in range(4):
    if !to_draw[i]:
      continue
    else:
      points[i] = Utils.rotate(points[i], center_point, params.degree)
      put_pixel(points[i].x, points[i].y, colors[i])
#...
```

Di implementasi juga ada kelas konfigurasi dalam penggambaran elipsnya sendiri. Terdapat opsi untuk tidak menggambarkan kuandran tertentu, mapping kuadrannya seperti di bawah ini,

$$\begin{aligned}\text{Quadrant } 1 &\rightarrow (x > 0, y > 0)\\ \text{Quadrant } 2 &\rightarrow (x > 0, y < 0)\\ \text{Quadrant } 3 &\rightarrow (x < 0, y < 0)\\ \text{Quadrant } 4 &\rightarrow (x < 0, y > 0)\end{aligned}$$

Terdapat juga konfigurasi warna untuk setiap kuadran, dan juga rotasi dari elips dengan anchornya merupakan titik tengahnya. Kode kelas konfigurasinya sendiri,

```
class_name EllipseParams

var quadrant_to_draw = []
var quadrant_color = PoolColorArray()
var degree: float

func _init(_quadrant_to_draw: Array = [], _quadrant_color: PoolColorArray = [], _degre
e: float = 0):
  if _quadrant_color.size() < 4:
    initialize_quadrant_color()
  else:
    quadrant_color = _quadrant_color

  if _quadrant_to_draw.size() < 4:
    initialize_quadrant_to_draw()
  else:
    quadrant_to_draw = _quadrant_to_draw

  degree = _degree


func initialize_quadrant_color():
  for _i in range(4):
    quadrant_color.append(Color.white)
```

```
func initialize_quadrant_to_draw():
  for _i in range(4):
    quadrant_to_draw.append(true)


func randomize_quadrant_color():
  for i in range(4):
    randomize()
    quadrant_color[i] = Color(randf(), randf(), randf())


func set_degree(_degree: float):
  degree = _degree


func randomize_degree():
  randomize()
  degree = rand_range(0, 360)
```

Lalu dalam scenenya dilakukan randomisasi posisi, panjang (rx), tinggi (ry), warna setiap kuadran, dan juga rotasi dari elipsnya sendiri. Tampilannya seperti di bawah ini,

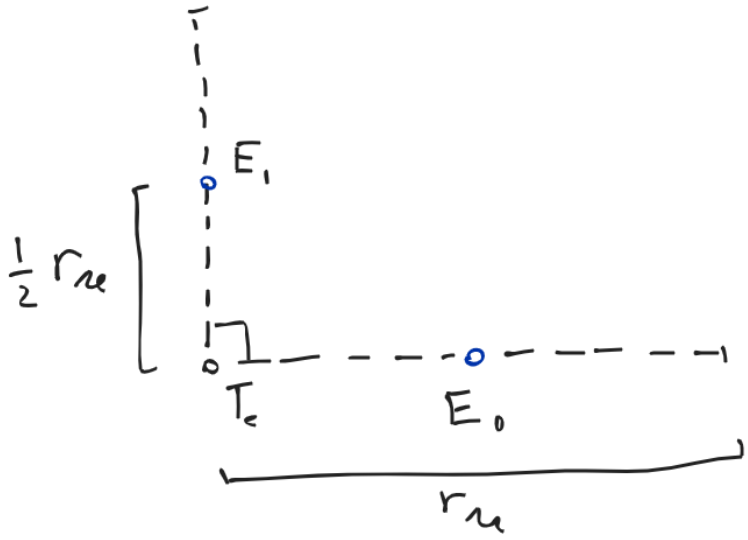
Untuk kode dari scenenya sendiri,

```
extends Shapes

const N_ELLIPSE = 10

func _draw():
  var viewport_size = get_viewport().size

  for _i in range(N_ELLIPSE):
    var params = EllipseParams.new()
    params.randomize_quadrant_color()
    params.randomize_degree()

    randomize()
    var rx = rand_range(20, 200)
    var ry = rand_range(20, 200)
```

```
    randomize()
    var point = Vector2(rand_range(0, viewport_size.x - rx), rand_range(ry, viewport_s
ize.y))

    draw_ellipse_midpoint(rx, ry, point, params)


func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")

func _on_Button2_pressed():
  update()
```

Bunga 4 Kelopak

Dalam menggambarkan bungan dengan 4 kelopak, dilakukan penggambaran 4 elips, dengan rotasi yang berbeda beda, dan titik tengah yang dilakukan rotasi sesuai dengan posisinya.



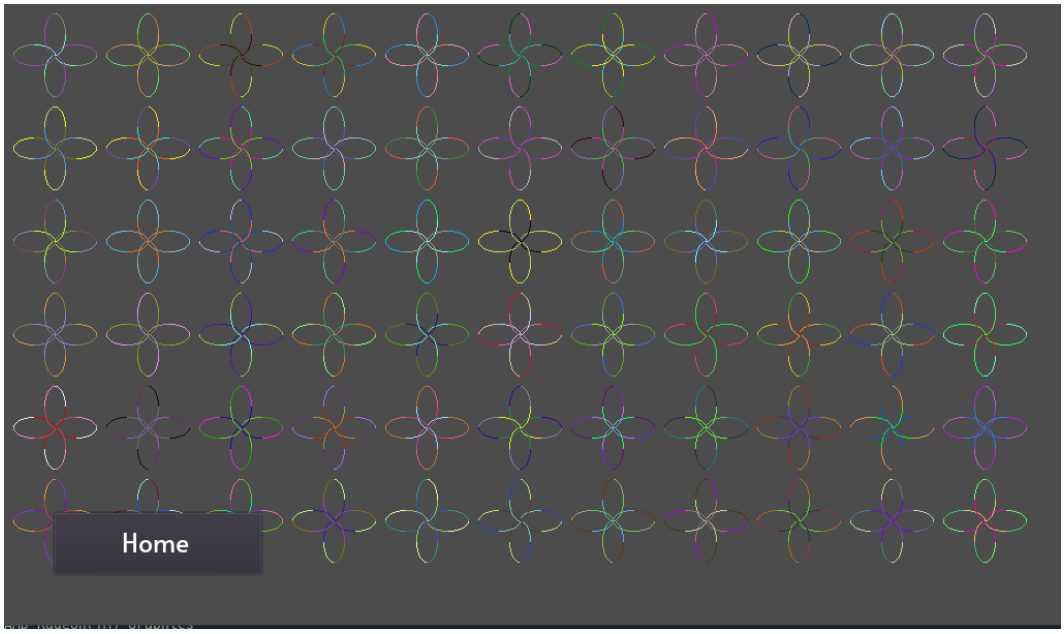
Seperti gambar di atas, panjang dari elips tidak berubah, hanya saja rotasi yang berubah. Titik tengah dari bunga itu sendiri T_c dilakukan translasi dengan $x + \frac{1}{2}r_x$, dari situ bisa didapatkan E_0. Dari situ titik lainnya bisa di dapatkan dengan melakukan rotasi dengan anchor T_c, dan titik yang dirotasikan E_0, besarannya sendiri $n_E * 90^\circ$, sehingga untuk mendapatkan besaran rotasi dari $\theta_{E_n} = n * 90^\circ$, dengan $n = \{0, 1, 2, 3\}$.

Jumlah rotasi (θ_{E_n}) yang sama digunakan untuk rotasi elips yang digambarkan dengan titik tengahnya E_n. Implementasi dari penggambaran ini sendiri,

```
 #...
  func draw_flower_4(rx: float, ry: float, center: Vector2, degree: float = 0, draw: A
rray = []):
  var params = EllipseParams.new(draw)
  params.randomize_quadrant_color()
  params.set_degree(degree)

  var point = Utils.translate(center, Vector2(rx, 0))

  draw_ellipse_midpoint(rx, ry, Utils.rotate(point, center, degree), params)

  params.set_degree(degree + 90)
  draw_ellipse_midpoint(rx, ry, Utils.rotate(point, center, degree + 90), params)

  params.set_degree(degree + 180)
  draw_ellipse_midpoint(rx, ry, Utils.rotate(point, center, degree + 180), params)

  params.set_degree(degree + 270)
  draw_ellipse_midpoint(rx, ry, Utils.rotate(point, center, degree + 270), params)
#...
```

Di parameternya ada tambahan rotasi awal dari bunganya sendiri, rotasi awal ini (θ_0) ditambahkan ke fungsinya, dan menjadi $\theta_{E_n} = \theta_0 + n * 90^\circ$. Sementara itu di dalam scenenya sendiri dilakukan pengambarannya dengan memenuhi viewportnya. Caranya adalah dengan terus menggambar dengan titik awalnya terus di translasi x nya dengan panjangnya dan padding. Jika sudah menemukan limit, translasi y dengan panjang dan padding, dan x di mulai dari awal lagi.

Hasilnya sendiri terlihat seperti ini,



Dan kode dari scenenya sendiri,

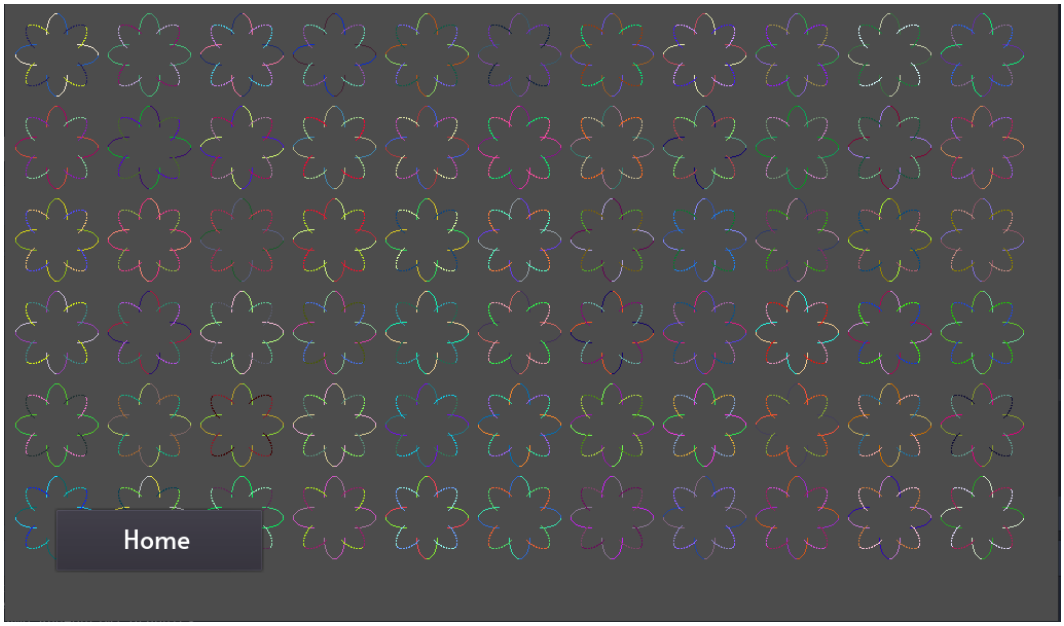
```
extends Shapes

const PADDING = 10

func _draw():
  var rx = 20
  var ry = 10

  var viewport_size = get_viewport().size
  var point = Vector2(rx * 2 + PADDING, rx * 2 + PADDING)

  while point.y < viewport_size.y - PADDING:
    while point.x < viewport_size.x - PADDING:
      draw_flower_4(rx, ry, point)

      point = Vector2(point.x + (rx * 4) + PADDING, point.y)

    point = Vector2(rx * 2 + PADDING, point.y + (rx * 4) + PADDING)

func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")
```

Bunga Kelopak 8

Untuk menggambarkan bunga kelopak 8, yang saya lakukan adalah menggambar 2 bunga kelopak 4, yang satunya di putarkan 45°. Juga semua elips yang di gambar hanya digambarkan kuadran 1, dan 2. Kodenya kurang lebih seperti ini,

```
#...
func draw_flower_8(rx: float, ry: float, center: Vector2):
  draw_flower_4(rx, ry, center, 0, [true, true, false, false])
  draw_flower_4(rx, ry, center, 45, [true, true, false, false])
#...
```

Dan di scenenya kurang lebih sama dengan bunga kelopak 4, memenuhi viewport. Hasilnya terlihat seperti ini,

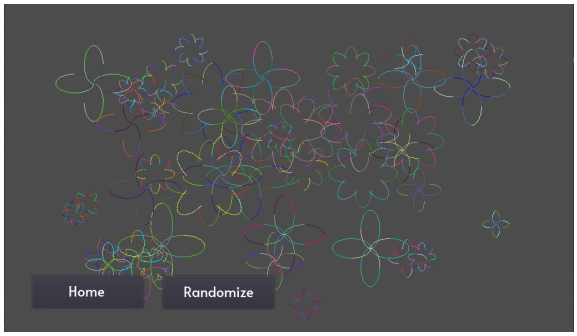


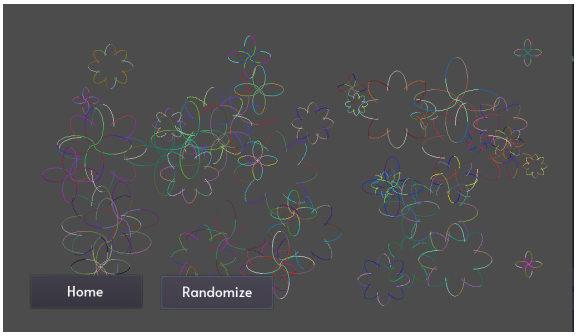
Kode dari scenenya itu sendiri,

```
extends Shapes

const PADDING = 10

func _draw():
  var rx = 20
  var ry = 10

  var viewport_size = get_viewport().size
  var point = Vector2(rx * 2 + PADDING, rx * 2 + PADDING)

  while point.y < viewport_size.y - PADDING:
    while point.x < viewport_size.x - PADDING:
      draw_flower_8(rx, ry, point)

      point = Vector2(point.x + (rx * 4) + PADDING, point.y)

    point = Vector2(rx * 2 + PADDING, point.y + (rx * 4) + PADDING)


func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")
```

Batik

Tugasnya di sini adalah dengan membuat scene dimana bunga kelopak 4, dan 8 di gambarkan secara acak. Jadi hasilnya adalah,





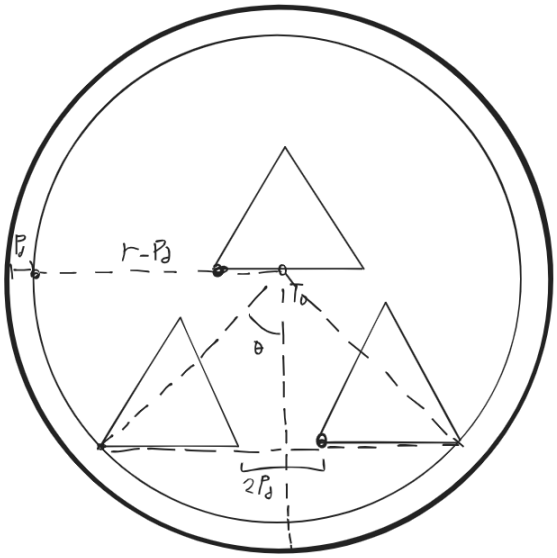
Lalu, kode dari scenenya sendiri,

```
extends Shapes

const NUM_EACH_CLOVE = 25

func _draw():
  var viewpor_size = get_viewport().size

  for i in range(NUM_EACH_CLOVE * 2):
    var rx = rand_range(10, 40)
    var ry = rx / 2

    var point = Vector2(rand_range(rx * 4, viewpor_size.x - rx * 4), rand_range(rx *
 4, viewpor_size.y - rx * 4))
    if i % 2 == 0:
      draw_flower_4(rx, ry, point)
    else:
      draw_flower_8(rx, ry, point)


func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")


func _on_Button2_pressed():
  update()
```

Family Crest

Dari yang saya cari terdapat banyak sekali family crest di jepang, dan apa yang saya pilih adalah,

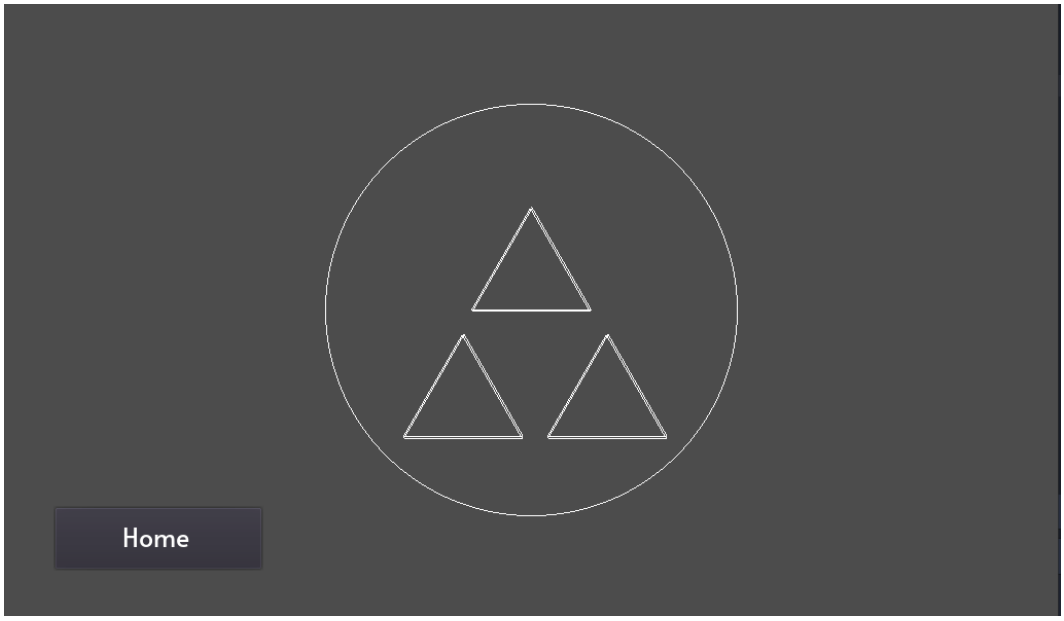
Untuk menggambarnya, saya merencanakan transformasi dari titiknya seperti ini,



Dari titik awal T_0, dilakukan translasi ke kiri, dengan mengurangi $x - (r + P_d)$, dimana r adalah radius dari lingkaran luar, dan P_d merupakan padding yang sudah di set. Untuk mencari titik segitiga kiri bawah, dilakukan rotasi dari titik yang sudah di translasi sebesar $\theta = 60^\circ$. Juga untuk panjang dari segitiga sama sisi digunakan fungsi $\ell = r \sin 60^\circ - P_d$.

Dari situ dilakukan lagi translasi sebesar $x + 2P_d$ untuk mendapatkan titik awal dari segitiga kanan bawah. Untuk segitiga paling atas hanya butuh dilakukan translasi dari titik awal sebesar $x - \frac{1}{2}\ell$.

Hasilnya akan terlihat seperti ini,



Dan kode implementasinya sendiri,

```
extends Shapes

const RADIUS = 200
const PAD = 25

func _draw():
  var viewport_size = get_viewport().size
  var center = Vector2(viewport_size.x / 2, viewport_size.y / 2)
  var edge = Vector2(center.x - RADIUS + PAD, center.y)

  var triagle_len = 0.7 * RADIUS - PAD
  var left_triange_point = Utils.rotate(edge, center, -45)
  var right_triangle_point = Utils.translate(left_triange_point, Vector2(triagle_len +
PAD, 0))


  draw_circle_midpoint(RADIUS, center, CircleParams.new())
  draw_triangle(left_triange_point, triagle_len)
  draw_triangle(right_triangle_point, triagle_len)
  draw_triangle(Utils.translate(center, Vector2(-(triagle_len / 2), 0)), triagle_len)


func _on_Button_pressed():
  get_tree().change_scene("res://menu/Menu.tscn")
```

Lesson Learned

1. Godot sudah mempunyai sistem komputasi vektornya sendiri, jadi bisa melakukan hal seperti perkalian vektor dengan skalar, atau pertambahan

vektor.

2. Menulis laporan godot ini memakan waktu yang lebih lama dibandingkan melakukan implementasinya di godotnya sendiri.
3. Godot berat dijalankan dengan beberapa kode di atas, walaupun dengan fungsi yang sudah ada di godot nya sendiri. Mungkin karena kode yang ada di GDScript harus di translasi dahulu ke level yang "rendah"