Laporan Praktikum Godot 4

Muhammad Fauzan Lubis

191524026

D4-2A

Daftar Isi

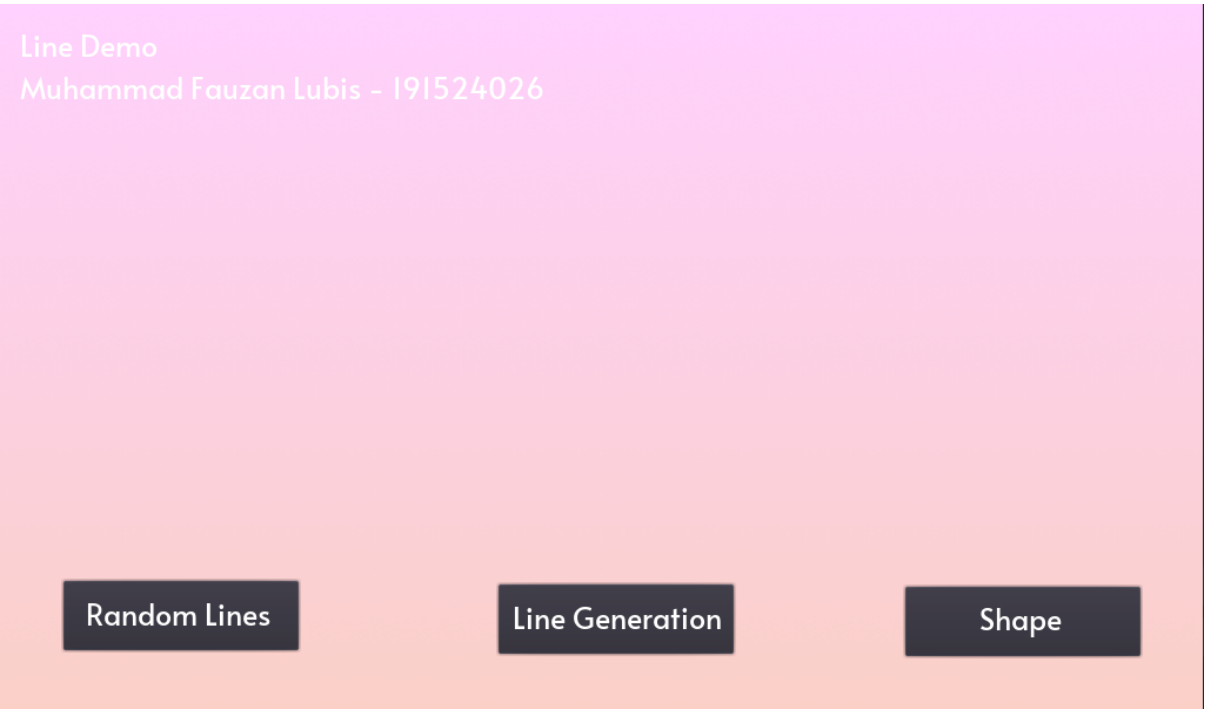
Rangkuman

Di bawah ini penjelasan dari beberapa langkah-langkah yang dilakukan pada praktikum ini,

Membuat UI

Seperti sudah dilakukan sebelumnya, dibuat suatu root Node2D di scene Menu. Scene ini akan menajdi scene utama dari project, jadi dimana user membuka

aplikasi akan langsung diberikan layar menu. Untuk layarnya sendiri akan terlihat seperti ini,



Dibuat dengan menambahkan RichTextLabel untuk teks label, nama, dan nim. Background merupakan suatu png, hanya dilakukan drag and drop ke scene. Terakhir, 3 button untuk mengontrol ke scene lainnya. Dengan ketiga tombol yang ada di kontrol pada skrip yang terhubung dengan Node2D root-nya.

```
extends Node2D


func _on_RandomLinesButton_pressed():
  get_tree().change_scene("res://random_lines/RandomLines.tscn")


func _on_LineGenerationButton2_pressed():
  get_tree().change_scene("res://line_generation/LineGen.tscn")


func _on_ShapeButton_pressed():
  get_tree().change_scene("res://shape/Shape.tscn")
```

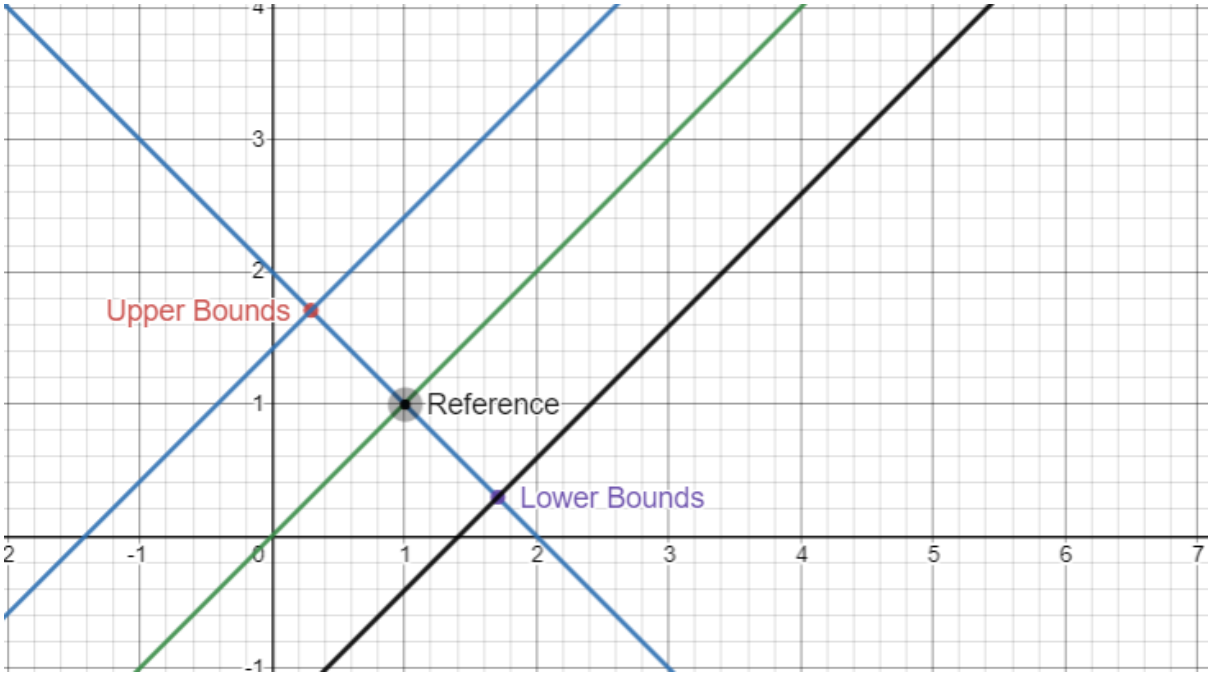
Sementara untuk font custom di lakukan setting langsung di panel RichTextLabel nya sendiri. Font yang dipakai adalah Alata Regular. Karena font yang dipakai bukan font bitmap bawaan Godot, ukuran dapat diatur.

Random Line Generator

Sebelum dapat melakukan suatu scene line random, harus dibuat algortima nya terlebih dahulu. Karena melihat yang lain sudah ada kebutuhan garis putus-putus, dan juga garis tebal, saya coba langsung coba implementasikannya di sini. Basis dari implentasi adalah algoritma DDA, walaupun bukan yang paling cepat, namun sudah dapat melakukan semua range gradien yang ada.

Garis Tebal

Untuk membuat garis tebal maka apa yang saya pikirkan di sini adalah bukan untuk menggambar garis secara lurus seperti biasanya. Namun, dengan menggambar ketebalannya itu sendiri secara perpendikular, kurang lebih seperti ini,



Jadi yang akan digambar adalah garis dari titik upper bound (x_{ub}, y_{ub}), ke titik lower bound (x_{lb}, y_{lb}), yang dimana jarak d dari kedua garis tersebut adalah ketebalan garis itu sendiri. Hal itu akan dilakukan dari titik awal garis, hingga akhir. Sehingga terbentuk suatu garis dari kumpulan garis perpendikular yang akan terlihat tebal.

Permasalahannya sekarang tidak diketahui adalah titik untuk lower bounds, atapun upper bounds. Namun dengan melihat representasi di gambar itu sendiri bisa terlihat bahwa titik upper bounds hanyalah suatu mirror dari lower bounds, dengan garis yang awal sebagai yang melakukan mirroring-nya.

Jika dilihat bahwa titik upper bounds, dan titik lower bounds berada di dalam satu garis yang merupakan garis perpendikular dari garis yang awalnya di gambar. Diketahui bahwa suatu garis linear akan perpendikular dengan garis linear lainnya jika,

$$m_p = -\frac{1}{m} \dots (1)$$

Lalu untuk mendapatkan garis itu kita bisa dapatkan dengan,

$$(y_{lb} - y_p) = m_p(x_{lb} - x_p) \dots (2)$$

Dimana titik reference adalah (x_p, y_p), dan sekarang untuk mendapatkan titik dengan jarak d dari titik reference itu sendiri,

$$d^2 = (x_{ub} - x_p)^2 + (y_{ub} - y_p)^2 \dots (3)$$

Lalu kita bisa gabungkan (2) dan (3),

$$d^2 = (x_{lb} - x_p)^2 + (m_p(x_{lb} - x_p))^2$$

$$d^2 = (x_{lb} - x_p)^2 + m_p^2(x_{lb} - x_p)^2$$

$$d^2 = (1 + m_p^2)(x_{lb} - x_p)^2$$

$$\frac{d^2}{(1 + m_p^2)} = (x_{lb} - x_p)^2$$

$$\frac{d}{\sqrt{(1 + m_p^2)}} + x_p = x_{lb}$$

Sekarang sudah bisa dihitung x_{lb}, dan juga x_{ub} dengan mengganti jarak d menjadi $-d$. Untuk menghitung dengan menggunakan memasukan x masing-masing ke (2).

$$y_{lb} = m_p(x_{lb} - x_p) + y_p$$

Dan,

$$y_{ub} = m_p(x_{ub} - x_p) + y_p$$

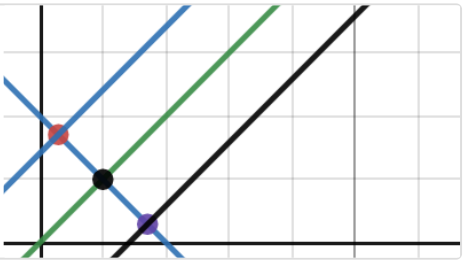
Lalu untuk melakukan update kedua titik upper bounds, dan lower bounds, bisa dilakukan dengan menambahkan x dengan $x_{inc} = dx/steps$ dan y dengan $y_{inc} = dy/steps$. Hal ini sudah ada di dalam algoritma DDA nya itu sendiri. Variabel ini bisa dipakai kembali karena garis utama yang di gambarkan memiliki m yang sama dengan pergerakan kedua titik tersebut.

Hasil interaktif dari penurunan rumus ini dapat dilihat di,

Line thicc

Line thicc

https://www.desmos.com/calculator/iqkxvqrpsr



Garis Putus-putus

Untuk garis putus-putus sendiri cukup mudah untuk diimplenatasikan, karena apa yang dilakukan hanyalah melakukan perhitungan dengan kapan garis harus muncul dan kapan harus ada. Hal ini dapat dilakukan dengan menghitung kapan akan garis mulai tidak muncul. Lalu, melakukan step over ke titik dimana garis akan muncul kembali.

Source Code

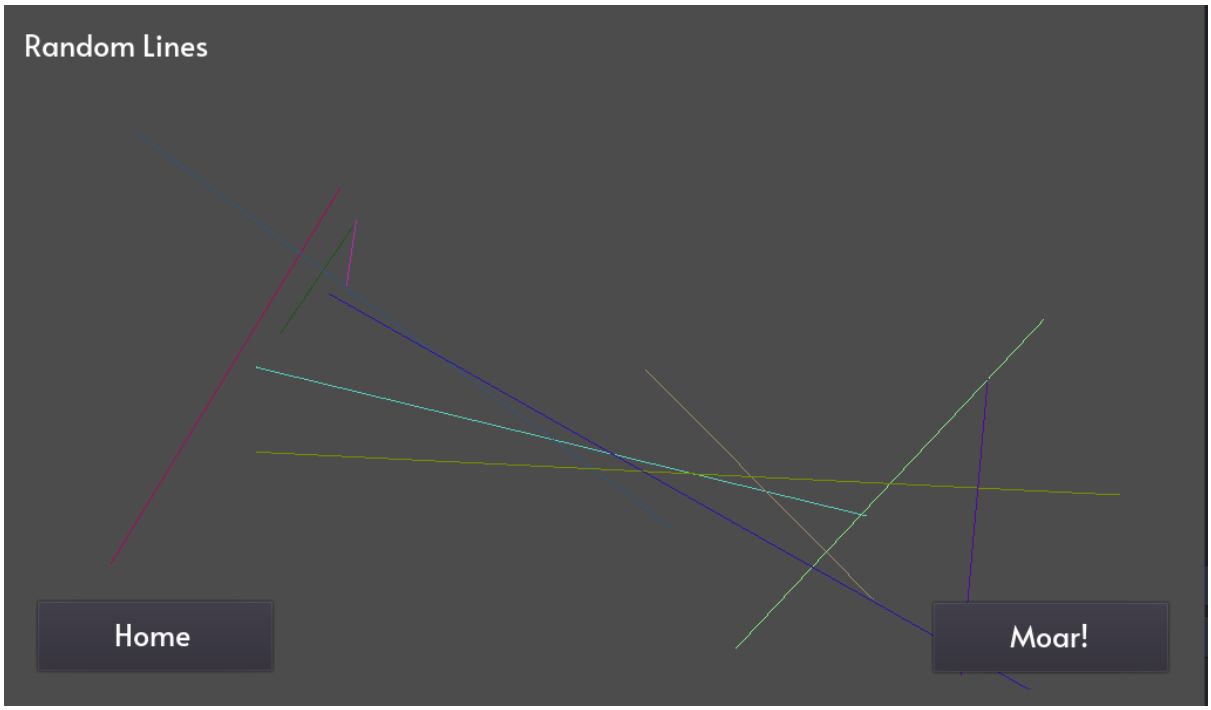
Hasil dari implementasi dari persamaan dan sistem putus-putus yang sudah dijelaskan di atas, kurang lebih seperti ini,

```
func draw_line_custom(
  va: Vector2,
  vb: Vector2,
  _line_color: Color,
  thickness: float = 1,
  dotted: bool = false,
  dash: int = 1,
  gap: int = 0
):
  var steps

  var dx = vb.x - va.x
  var dy = vb.y - va.y

  if dx == 0 and dy == 0:
    return

  var x = va.x
  var y = va.y
```

```
  var upper_bounds: Vector2
  var lower_bounds: Vector2

  thickness = thickness / 2

  if dx == 0:
    upper_bounds = Vector2(x - thickness, y)
    lower_bounds = Vector2(x + thickness, y)
  elif dy == 0:
    upper_bounds = Vector2(x, y + thickness)
    lower_bounds = Vector2(x, y - thickness)
  else:
    var m: float = dy / dx
    var m_p: float = (-1) * (1 / m)

    var tmp = thickness / sqrt(1 + pow(m_p, 2))

    var x_lb = tmp + x
    var x_ub = -tmp + x

    var y_lb = (m_p * (x_lb - x)) + y
    var y_ub = (m_p * (x_ub - x)) + y

    upper_bounds = Vector2(x_ub, y_ub)
    lower_bounds = Vector2(x_lb, y_lb)

  if abs(dx) > abs(dy):
    steps = abs(dx)
  else:
    steps = abs(dy)

  var xIncrement = dx / steps
  var yIncrement = dy / steps

  var i = 1
  var skip = dash

  put_pixel(round(x), round(y), _line_color)
  while i <= steps:
    if i == skip and dotted:
      var x_add = xIncrement * gap
      var y_add = yIncrement * gap

      upper_bounds.x += x_add
      upper_bounds.y += y_add

      lower_bounds.x += x_add
      lower_bounds.y += y_add

      skip += dash + gap
      i += gap
    else:
      var x_add = xIncrement
      var y_add = yIncrement

      upper_bounds.x += x_add
      upper_bounds.y += y_add
```

```
        lower_bounds.x += x_add
        lower_bounds.y += y_add

        i += 1

        draw_thickness(upper_bounds, lower_bounds, _line_color)


func draw_thickness(start: Vector2, end: Vector2, color: Color):
  var dy = end.y - start.y
  var dx = end.x - start.x

  var steps = abs(dx) if (abs(dx) > abs(dy)) else abs(dy)

  var x = start.x
  var y = start.y

  put_pixel(x, y, color)

  if steps > 0:
    var x_inc = dx / steps
    var y_inc = dy / steps

    for i in steps:
      x += x_inc
      y += y_inc

      put_pixel(round(x), round(y), color)
```

Semantara scene dari random line generator ini sendiri terlihat seperti ini,



Dengan tombol di kanan bawah akan menambah garis random yang akan keluar di layar. Implementasi dari scene ini sendiri terlihat seperti ini,

```
extends Lines

const INITIAL_LINES = 10
var start_coordinates = PoolVector2Array()
var end_coordinates = PoolVector2Array()
var colors = PoolColorArray()


func get_random_coordinate() -> Vector2:
  var viewport_size: Vector2 = get_viewport().size
  randomize()

  var x = rand_range(0, viewport_size.x)
  var y = rand_range(0, viewport_size.y)

  return Vector2(x, y)


func get_random_color() -> Color:
  return Color(randf(), randf(), randf())


func _ready():
  for _i in range(INITIAL_LINES):
    start_coordinates.append(get_random_coordinate())
    end_coordinates.append(get_random_coordinate())
    colors.append(get_random_color())


func _draw():
  for i in range(start_coordinates.size()):
    draw_line_custom(start_coordinates[i], end_coordinates[i], colors[i])


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


func _on_Moar_pressed():
  start_coordinates.append(get_random_coordinate())
  end_coordinates.append(get_random_coordinate())
  colors.append(get_random_color())

  update()
```

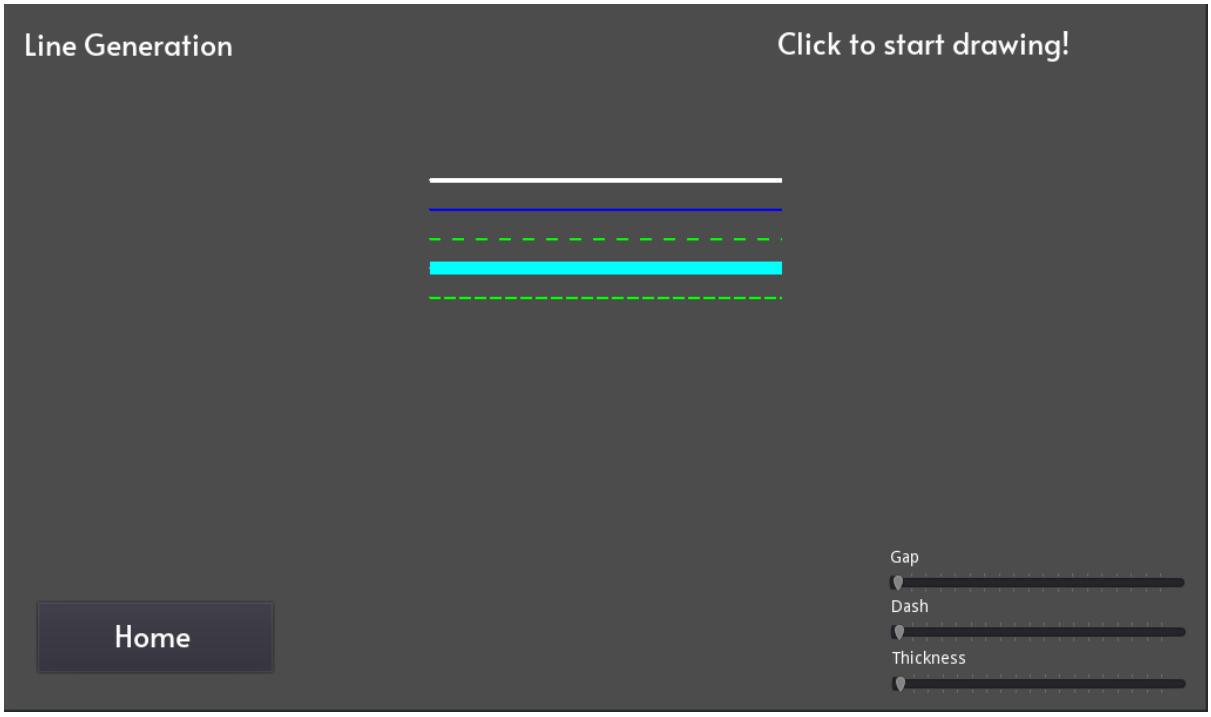
Semua garis random di generate vektornya pada saat scene ini di load, dan ditambahkan ke dalam suatu array yang menampungnya, semantara untuk garis yang diminta tambah maka akan juga ditambahkan ke array tersebut.

Penggambaran garis itu sendiri dilakukan pada saat draw dengan melakukan looping pada array.

Line Generation

Karena penjelasan dari garis itu sendiri sudah dijelaskan di atas, di sini hanya akan dijelaskan yang ada di dalam scene ini sendiri. Dalam scene ini ada garis yang sudah di set sebagai suatu demontrasi bahwa sistem yang sudah di implementasikan berjalan dengan baik. Lalu, ada sistem yang dapat melakukan click and drag untuk menggambar bebas, dengan paramter garis juga seperti dash, gap, dan thickness.

Scene itu terlihatt seperti ini,



Lalu setelah dilakukan beberapa gambar bebas dengan sistem click and drag, dengan parameter garis di naikan semua,

Source Code

Untuk source dari scene tersebut,

```
extends Lines

var line_to_draw = Array()

var start_coor: Vector2
var end_coor: Vector2


func _ready():
  var viewport_size = get_viewport().size
  var center_x = viewport_size.x / 2
  var center_y = viewport_size.y / 2

  line_to_draw.append(
    LineParams.new(
      Vector2(center_x - 150, center_y - 150),
      Vector2(center_x + 150, center_y - 150),
      Color.white,
      3
    )
  )
  line_to_draw.append(
    LineParams.new(
      Vector2(center_x - 150, center_y - 125),
      Vector2(center_x + 150, center_y - 125),
      Color.blue
    )
  )
  line_to_draw.append(
```

```
    LineParams.new(
      Vector2(center_x - 150, center_y - 100),
      Vector2(center_x + 150, center_y - 100),
      Color.green,
      1,
      true,
      10,
      10
    )
  )
  line_to_draw.append(
    LineParams.new(
      Vector2(center_x - 150, center_y - 75),
      Vector2(center_x + 150, center_y - 75),
      Color.aqua,
      10
    )
  )
  line_to_draw.append(
    LineParams.new(
      Vector2(center_x - 150, center_y - 50),
      Vector2(center_x + 150, center_y - 50),
      Color.green,
      1,
      true,
      10,
      3
    )
  )

  $DrawLabel.text = "Click to start drawing!"


func _draw():
  for params in line_to_draw:
    draw_line_from_params(params)


func add_from_input():
  line_to_draw.append(
    LineParams.new(
      start_coor,
      end_coor,
      Color(randf(), randf(), randf()),
      $Thickness.value,
      $Gap.value > 1 and $Dash.value > 1,
      $Dash.value,
      $Gap.value
    )
  )

  update()


func _input(event):
  if event is InputEventMouseButton:
    if event.button_index == BUTTON_LEFT and event.pressed:
      start_coor = event.position
```

```
      $DrawLabel.text = "Release to draw!"
    elif event.button_index == BUTTON_LEFT and not event.pressed:
      end_coor = event.position
      $DrawLabel.text = "Click to start drawing!"

      add_from_input()


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

Terlihat bahwa terdapat array yang akan berisi garis untuk demontrasi dan juga garis yang dibentuk dengan menyimpan parameternya ke dalam LineParams, yang merupakan suatu kelas sendiri.

```
var start: Vector2
var end: Vector2
var color: Color
var thickness: float
var dotted: bool
var dash: int
var gap: int


func _init(
  va: Vector2,
  vb: Vector2,
  _line_color: Color,
  _thickness: float = 1,
  _dotted: bool = false,
  _dash: int = 1,
  _gap: int = 0
):
  start = va
  end = vb
  color = _line_color
  thickness = _thickness
  dotted = _dotted
  dash = _dash
  gap = _gap
```

Dalam menggambar objek parameter ini dilakukan dengan menggunakan fungsi yang berbeda di file Lines,

```
func draw_line_from_params(params: LineParams):
  draw_line_custom(
    params.start,
    params.end,
    params.color,
    params.thickness,
    params.dotted,
```

```
    params.dash,
    params.gap
  )
```

Hal ini dilakukan agar kode terlihat lebih enak di lihat, dan juga agar dapat dengan mudah disimpan ke dalam satu array. Dengan menyimpan semua parameter dalam satu objek di array, kita dapat mudah untuk menambahkan garis baru, dengan penggambaran hanya dengan melakukan looping terhadap array yang menyimpan array tersebut, dan memanggil fungsi yang tepat.

Lalu, untuk sistem click and drag itu sendiri digunakan fungsi `_input(event)` yang akan menerima semua input yang masuk ke dalam Node2D yang terhubung dengan skrip itu. Salah satunya adalah event jika mouse di klik, dan kita dapat mengikuti mouse itu sampai mousenya melepaskan klik nya. Titik awal dari klik dianggap titik awal, dan jika klik dilepaskan, maka itu dianggap sebagai titik akhir.

Shape

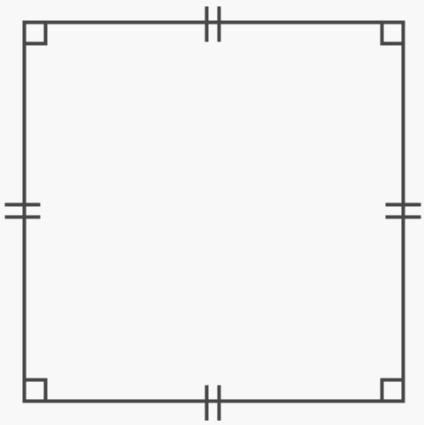
Sebelum membuat algoritma untuk bentuk yang dibuat, harus dibuat fungsi untuk menggambar titik dari array,

```
func draw_points(
  points: PoolVector2Array,
  color: Color,
  thickness: float = 1,
  dotted: bool = false,
  dash: int = 1,
  gap: int = 0
):
  for i in range(points.size()):
    var start = points[i]
    var end = points[(i + 1) % points.size()]

    draw_line_custom(start, end, color, thickness, dotted, dash, gap)
```

Fungsi ini akan melakukan looping terhadap array dari titik dan menggambarkan garis dari titik pertama hingga titik terakhir, dimana akan di hubungkan kembali dengan titik awal. Lalu, berikut ini implementasi dari setiap bentuk yang diharuskan untuk implementasi,

Persegi

Persegi di sini simpel saja, fungsi akan mengambil parameter besar nya, dan juga titik kiri atas dari persegi tersebut.

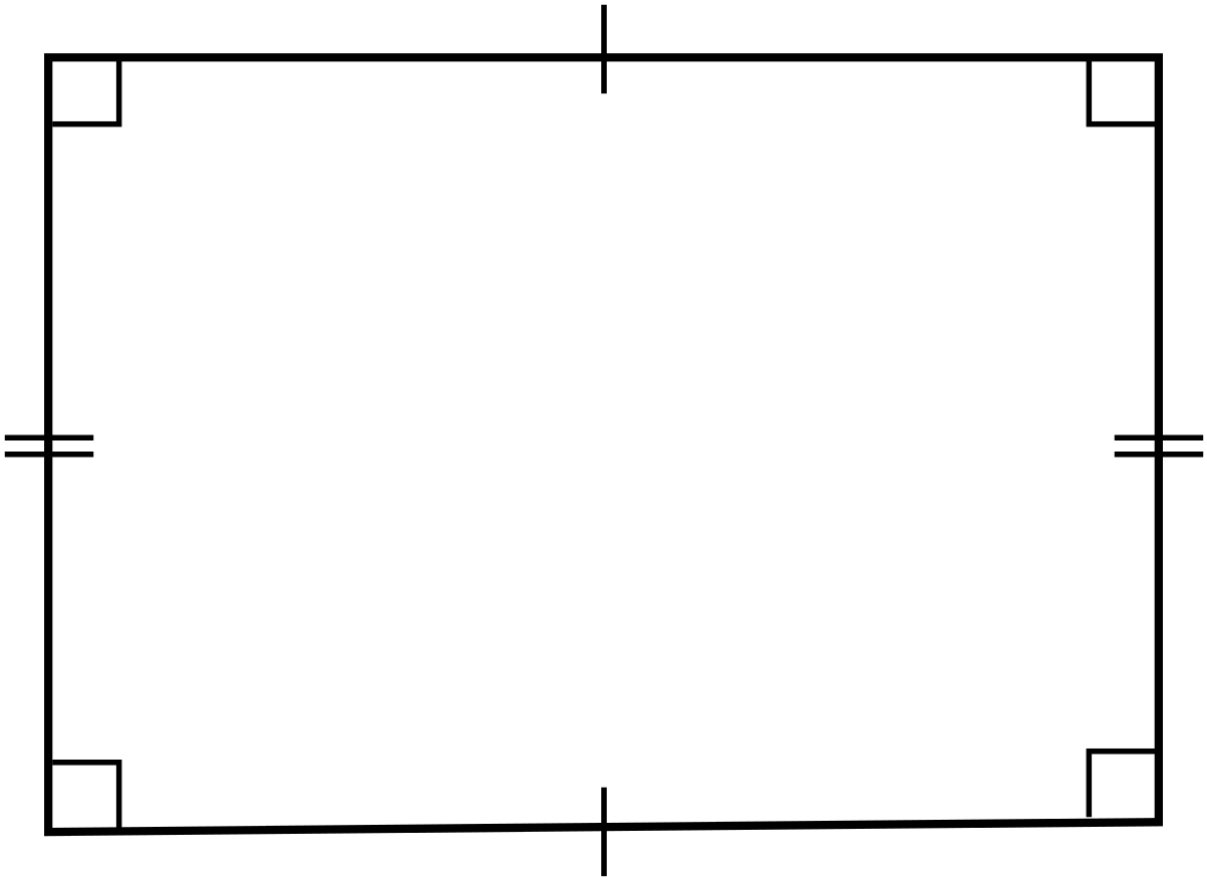


Baru dari situ akan dibuat titik sebelah kanan atas, kanan bawah, dan kiri bawah. Implementasinya sendiri terlihat seperti ini,

```
func draw_square(
  length: float, point: Vector2, color: Color, thickness: float = 1, dash: int = 1, ga
p: int = 0
):
  draw_points(
    [
      point,
      Vector2(point.x + length, point.y),
      Vector2(point.x + length, point.y + length),
      Vector2(point.x, point.y + length)
    ],
    color,
    thickness,
    dash > 1 and gap > 1,
    dash,
    gap
  )
```

Persegi Panjang

Persegi panjang kurang lebih sama dengan sebelumnya, namun berbedanya ada di dalam parameter yang akan mengambil panjang dan lebar, tidak hanya besaran.

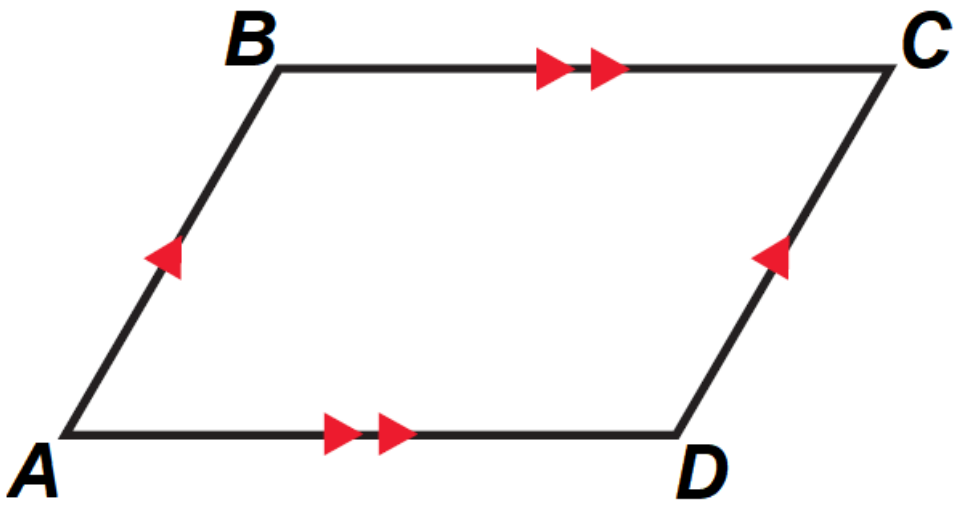


Lalu implementasinya,

```
func draw_rectange(
  length: float,
  width: float,
  point: Vector2,
  color: Color,
  thickness: float = 1,
  dash: int = 1,
  gap: int = 0
):
  draw_points(
    [
      point,
      Vector2(point.x + length, point.y),
      Vector2(point.x + length, point.y + width),
      Vector2(point.x, point.y + width)
    ],
    color,
    thickness,
    dash > 1 and gap > 1,
    dash,
    gap
  )
```

Jajaran Genjang

Karena sebenarnya jajaran genjang hanyalah suatu persegi yang dilakukan transformasi miring. Implenatasi yang saya lakukan akan menerima bearapa tinggi, dan panjang dari persegi itu sendiri. Lalu diambil juga seberapa jauh dilakukan perubahannya itu sendiri, atau di sini dipanggil delta.

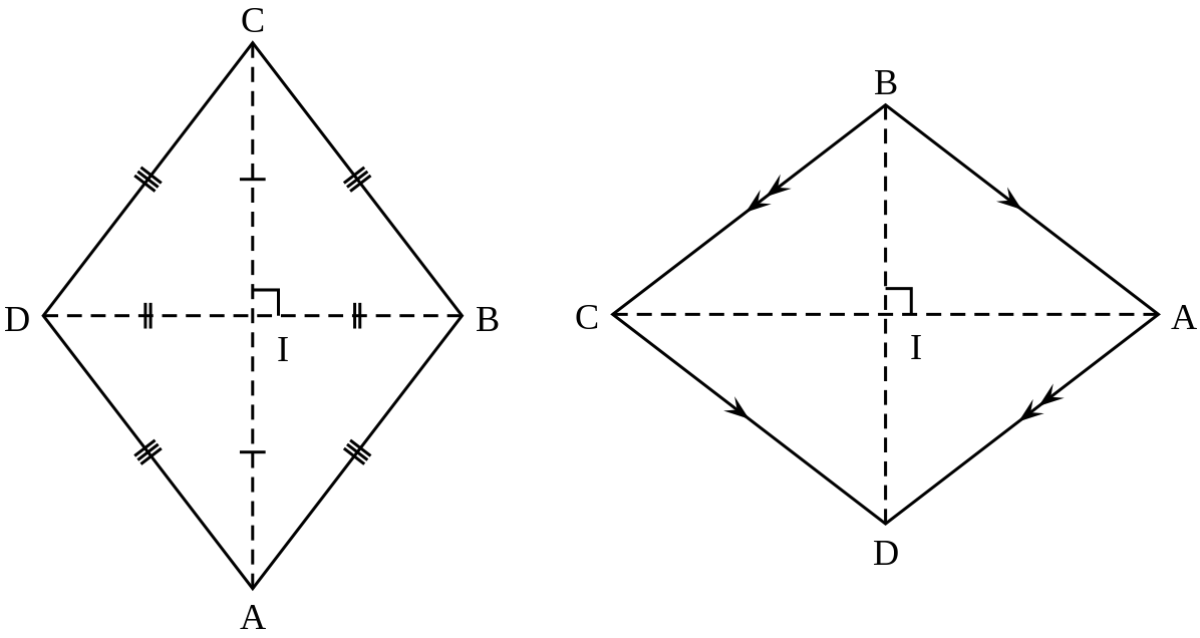


Implementasinya sendiri seperti di bawah ini,

```
func draw_parallelogram(
  base: float,
  delta: float,
  height: float,
  point: Vector2,
  color: Color,
  thickness: float = 1,
  dash: int = 1,
  gap: int = 0
):
  draw_points(
    [
      point,
      Vector2(point.x + base, point.y),
      Vector2(point.x + base - delta, point.y + height),
      Vector2(point.x - delta, point.y + height)
    ],
    color,
    thickness,
    dash > 1 and gap > 1,
    dash,
    gap
  )
```

Belah Ketupat

Dalam belah ketupat diambil titik tengah dari belah ketupat itu sendiri, dan diambil parameter panjang dan lebar dari belah ketupat itu.

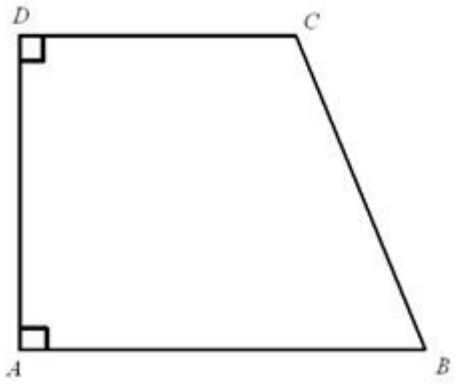


Lalu keempat titik yang ada dibuat dari, antara menambahkan atau mengurangi setengah dari tinggi atau panjang yang didapatkan dari paramter. Implementasi dari tekniknya seperti ini,

```
func draw_rhombus(
  length: float,
  height: float,
  center_point: Vector2,
  color: Color,
  thickness: float = 1,
  dash: int = 1,
  gap: int = 0
):
  var half_length = length / 2
  var half_height = height / 2

  draw_points(
    [
      Vector2(center_point.x - half_length, center_point.y),
      Vector2(center_point.x, center_point.y + half_height),
      Vector2(center_point.x + half_length, center_point.y),
      Vector2(center_point.x, center_point.y - half_height)
    ],
    color,
    thickness,
    dash > 1 and gap > 1,
    dash,
    gap
  )
```

Trapesium

Di sini akan diambil seberapa panjang garis horizontal yang berada di atas, atau upper base. Juga berapa panjang garis horizontal yang berada di bawah, atau lower base. Terahir adalah tinggi dari trapesium itu sendiri. Titik awal adalah D yang merupakan parameter juga, lalu ke C, B, dan terakhir A.

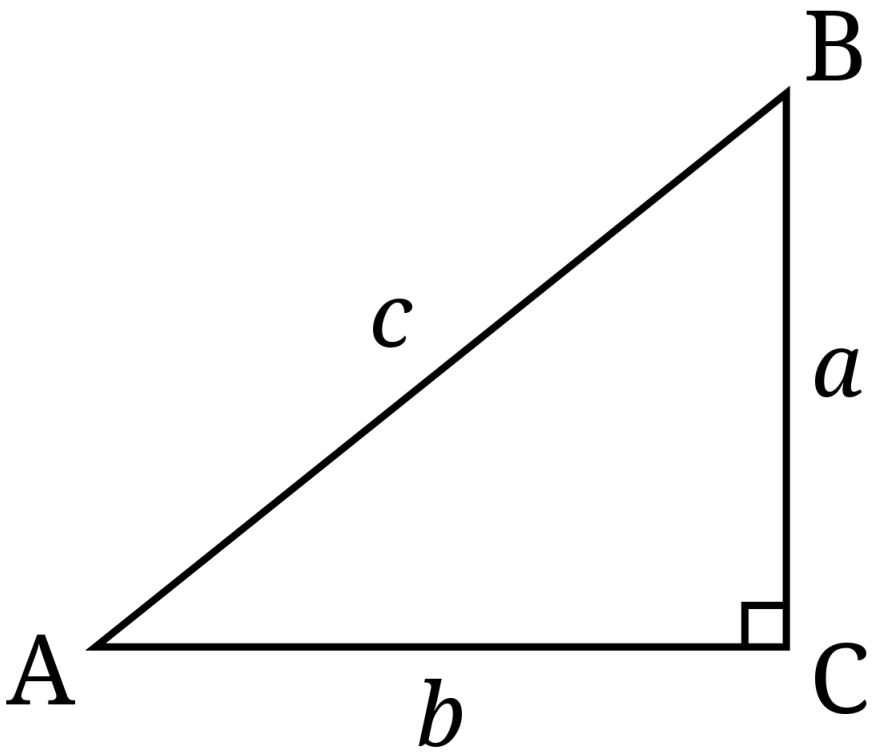


Sementara untuk implementasinya,

```
func draw_trapesium(
  top_base: float,
  bottom_base: float,
  height: float,
  point: Vector2,
  color: Color,
  thickness: float = 1,
  dash: int = 1,
  gap: int = 0
):
  draw_points(
    [
      point,
      Vector2(point.x + top_base, point.y),
      Vector2(point.x + bottom_base, point.y + height),
      Vector2(point.x, point.y + height)
    ],
    color,
    thickness,
    dash > 1 and gap > 1,
    dash,
    gap
  )
```

Segitiga Siku-Siku

Simple saja, karena segitiganya adalah segitiga siku-siku maka yang dibutuhkan hanyalah panjang dan tinggi yang dibutuhkan. Juga titik kanan atas dari segitiga siku-siku yang akan digambar.



Untuk implementasinya,

```
func draw_triangle(
  height: float,
  width: float,
  point: Vector2,
  color: Color,
  thickness: float = 1,
  dash: int = 1,
  gap: int = 0
):
  draw_points(
    [point, Vector2(point.x + width, point.y + height), Vector2(point.x, point.y + hei
ght)],
    color,
    thickness,
    dash > 1 and gap > 1,
    dash,
    gap
  )
```

Scene

Dari semua fungsi yang sudah dibuat dibuat scene yang menampilkan semuanya, dan juga versi dari bentuknya yang ada ketebalan dan putus-putus. Scenenya terlihat seperti ini,

Dengan kode yang dipakai,

```
extends Shapes

func _draw():
  draw_square(50, Vector2(100, 100), Color.blue)
  draw_square(50, Vector2(100, 175), Color.darkcyan, 5, 10, 10)

  draw_rectange(50, 25, Vector2(200, 100), Color.lightgray)
  draw_rectange(50, 25, Vector2(200, 175), Color.mediumaquamarine, 5, 10, 10)

  draw_parallelogram(50, 25, 50, Vector2(300, 100), Color.orange)
  draw_parallelogram(50, 25, 50, Vector2(300, 175), Color.mistyrose, 5, 10, 10)

  draw_rhombus(25, 50, Vector2(400, 100), Color.navajowhite)
  draw_rhombus(25, 50, Vector2(400, 175), Color.aliceblue, 5, 10, 10)

  draw_trapesium(25, 50, 25, Vector2(500, 100), Color.skyblue)
  draw_trapesium(25, 50, 25, Vector2(500, 175), Color.lightgray, 5, 10, 10)

  draw_triangle(50, 50, Vector2(600, 100), Color.moccasin)
  draw_triangle(50, 50, Vector2(600, 175), Color.lightgreen, 5, 10, 10)

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

Code Challenge

Jika dilihat dari kode yang dipakai di scene-scene sebelumnya extends dari semuanya bukanlah ke Node2D (kecuali Menu). Hal ini karena semua fungsi sudah dimasukan ke dalam classnya sendiri.

Class Stroke

Kelas stroke ini menyimpan kode yang berhubungan dengan penggambaran garis, seperti halnya yang sudah diperlihatkan di penjelasan garis tebal. Namun di project ini kelas ini bernama Lines.

Class Shape

Kelas ini merupakan suatu subclass dari kelas Stroke sebelumnya itu sendiri, hal ini dikarenakan fungsi yang digunakan di sini membutuhkan fungsi garis yang ada di kelas Stroke. Isi dari kode ini sudah terlihat pada penjelasan implementasi kode shape. Di dalam project ini kelas ini bernama Shapes.

Lesson Learned

Kebanyakan hal sudah berjalan dengan baik, namun ada beberapa poin yang harus saya keluarkan,

1. Jika suatu variabel integer di bagi dengan integer lainnya, walaupun akan dilakukan assignment ke variabel float hasilnya akan tetap integer. Hal ini disebabkan oleh hal yang disebut integer division.
2. Suatu kelas yang extends ke Node2D tidak dapat dibuat menjadi objek. Karena fungsi seperti _draw sepertinya memang akan dipanggil langsung oleh scene nya itu sendiri, sehingga harus langsung tekonek dengan scenenya.