Laporan Praktikum Godot 2

Muhammad Fauzan Lubis

D4-2A

191524026

Rangkuman

Implementasi LineDDA dan Bressenham

Line DDA

```
func draw_line_dda(va: Vector2, vb: Vector2, _line_color: Color):
  var steps: int

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

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

  var xIncrement: float
  var yIncrement: float

  var x: float = va.x
  var y: float = va.y

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

  xIncrement = dx / steps
  yIncrement = dy / steps

  put_pixel(round(x), round(y), _line_color)
  for _k in range(0, steps):
    x += xIncrement
    y += yIncrement
    put_pixel(round(x), round(y), _line_color)
```

Bresenham

```
func draw_line_bresenham(va: Vector2, vb: Vector2, _line_color: Color):
  var dx: int = abs(va.x - vb.x)
  var dy: int = abs(va.y - vb.y)

  var p: int = 2 * dy - dx
  var twoDy: int = 2 * dy
  var twoDyDx: int = 2* (dy - dx)

  var x: int
  var y: int
  var xEnd: int

  if va.x > vb.x:
    x = vb.x
    y = vb.y
    xEnd = va.x
  else:
    x = va.x
    y = va.y
    xEnd = vb.x
  put_pixel(x, y, _line_color)

  while x < xEnd:
    x += 1

    if p < 0:
      p += twoDy
    else:
      y += 1
      p += twoDyDx

    put_pixel(x, y, _line_color)
```

Saya sendiri tidak melihat adanya permasalahan dalam kedua algoritma ini, kecuali pada Line DDA dimana ketika melakukan penggambaran garis dengan dx dan dy nol maka akan keluar exception divide by zero. Sehingga saya melakukan pengecekan jika keduanya nol, langsung melakukan return void karena memang tidak akan ada yang di gambar juga.

Membuat Bingkai dan Garis Kartesian

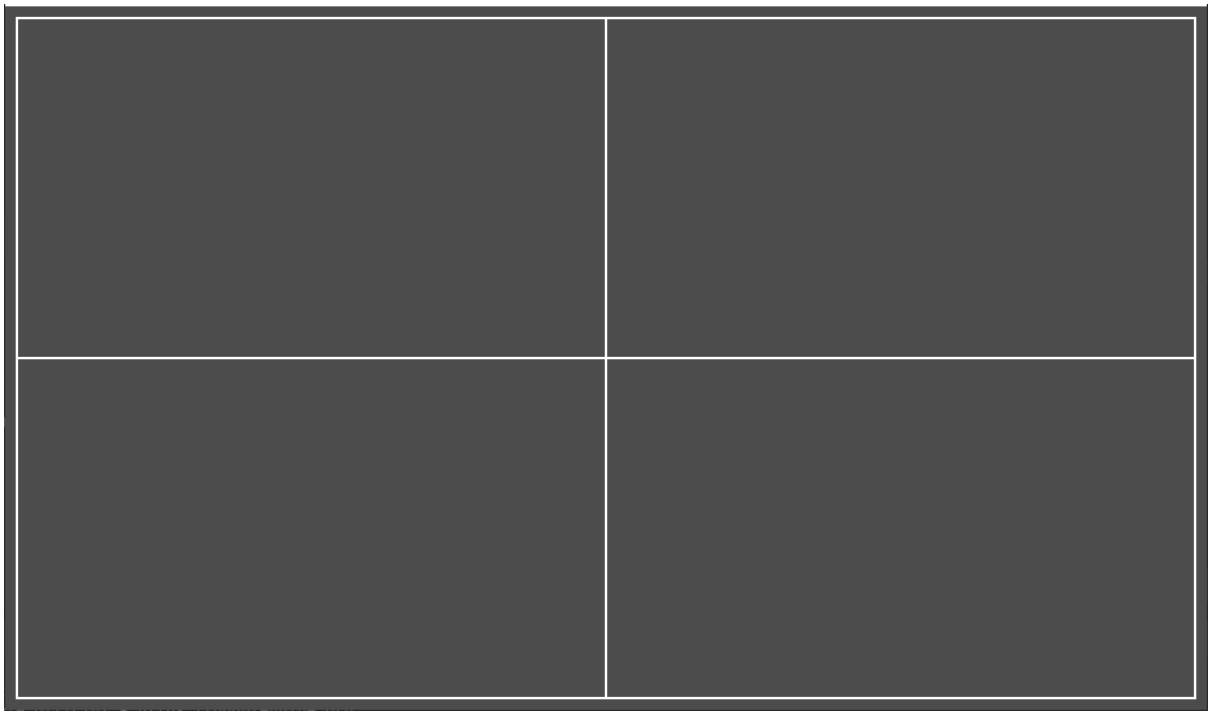
```
func draw_frame(_margin: int):
  var viewport_size = get_viewport_rect().size

  var top_left = Vector2(_margin, _margin)
  var top_right = Vector2(viewport_size.x - _margin, _margin)
  var bottom_left = Vector2(_margin, viewport_size.y - _margin)
  var bottom_right = Vector2(viewport_size.x - _margin, viewport_size.y - _margin)

  # Vertical Lines
```

```
    draw_line_dda(top_left, bottom_left, line_color)
    draw_line_dda(top_right, bottom_right, line_color)

    # Horizontal Lines
    draw_line_bresenham(top_left, top_right, line_color)
    draw_line_bresenham(bottom_left, bottom_right, line_color)

    # Cartesian Line
    var center_x = viewport_size.x / 2
    var center_y = viewport_size.y / 2

    draw_line_dda(
      Vector2(center_x, _margin),
      Vector2(center_x, viewport_size.y - _margin),
      line_color)

    draw_line_bresenham(
      Vector2(_margin, center_y),
      Vector2(viewport_size.x - _margin, center_y),
      line_color)
```

Fungsi ini akan menggambar garis bingkai dan juga garis y dan x dalam bidang kartesian dengan (0, 0) berada di tengah viewport. Sistemnya simpel, hanya dengan pertama mencari titik-titik ujung dari bingkai, lalu memanggil algoritma Line DDA dalam garis vertikal, dan Bresenham untuk garis horizontal. Hal yang sama dilakukan dengan garis x dan y dalam bidang kartesian, hanya saja yang di cari adalah titik tengah dalam x dan y.

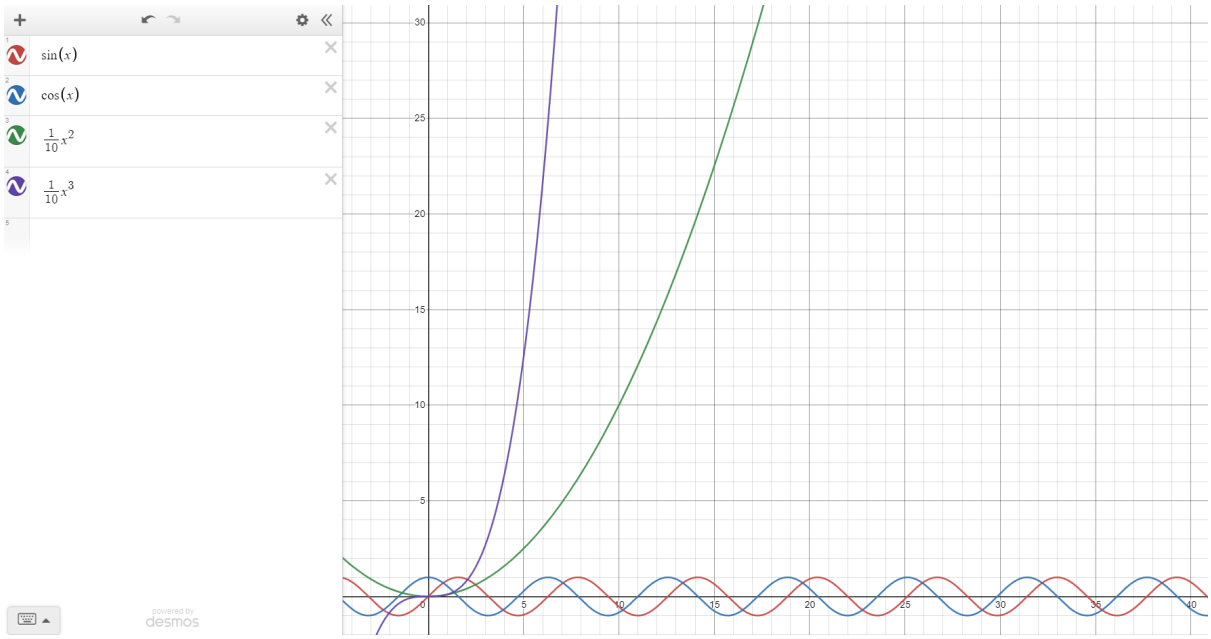
Screenshot

Hasil dari fungsi itu jika dipanggil akan terlihat seperti ini,



Graphing Persamaan

Persamaan yang saya pilih sangat standar sekali, yaitu $\frac{1}{10}x^2$, $\frac{1}{10}x^3$. $\sin x$, dan $\cos x$. Jika dilihat dalam graphing calculator maka akant terlihat seperti ini,



Fungsi kudratic, dan kubik di kalikan dengan $\frac{1}{10}$ agar y dari fungsi tidak akan terlalu jauh ke atas.

Jadi di bawah ini adalah implementasi untuk menggambar garisnya,

```
const SCALING = 10

func convert_to_normal_coordinate(vec: Vector2) -> Vector2:
  var viewport_size = get_viewport_rect().size

  return Vector2(
    ((viewport_size.x / 2 ) + vec.x * SCALING) ,
    ((viewport_size.y / 2) - vec.y * SCALING)
  )

func draw_expression(_expression: String, _margin: float,
  steps: float = 1, _line_color: Color = Color('#ffffff')):
  var expression = Expression.new()
  expression.parse(_expression, ['x'])

  var y: float = expression.execute([0])

  var viewport_size = get_viewport_rect().size
  var viewport_length = ((viewport_size.x / 2) - _margin) / SCALING
  var viewport_height = ((viewport_size.y / 2) - _margin) / SCALING

  var x = steps
  while abs(x) < viewport_length and abs(y) < viewport_height:
    y = expression.execute([x])

    var current_position = Vector2(float(x), float(y))
    current_position = convert_to_normal_coordinate(current_position)

    put_pixel(current_position.x, current_position.y, _line_color)

    x += steps
```

Terlihat bahwa ada 2 fungsi di dalam kode tersebut, fungsi pertama di gunakan untuk megkonversi koordinat dalam kartesian ke sistem koordinat yang di dalam Godot. Terlihat juga bahwa dalam konversi menggunakan konstanta SCALING. Hal ini dilakukan agar zoom dari graph dapat diatur dalam scriptnya. Hal ini digunakan karena fungsi trigonometri hanya akan berada di $-1 \leq y \leq 1$ maka tinggi nya hanya akan 2 pixel. Jika SCALING 10 digunakan maka tinggi 1 akan sama dengan 10 pixel.

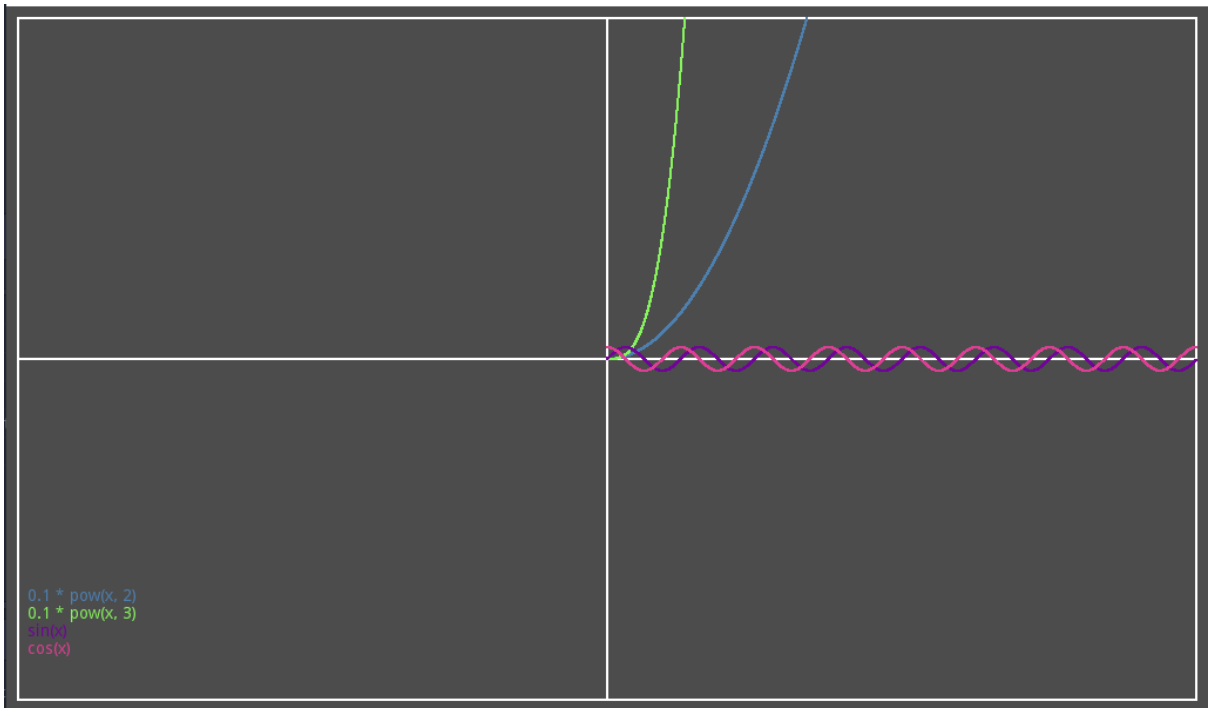
Fungsi kedua merupakan fungsi yang menggambar persamaannya itu sendiri. Fungsi menggunakan Expression yang ada di dalam Godot yang dapat mengambil expresi matematika seperti `pow(x, 2)` (x^2) dan membuatnya dapat di eksekusi. Hal ini dilakukan agar satu fungsi dapat melakukan graphing ke fungsi apapun.

Dalam menggambarnya sendiri, fungsi akan mengambil nilai y pertama di $x = 0$, baru akan melakukan looping dengan setiap step diatur dengan besaran

step di parameter. Looping dilakukan selama nilai x atau y masih di bawah bingkai frame yang sudah di bentuk sebelumnya.

Screenshot

Hasil dari pemanggilan fungsi tersebut dengan persamaan garis yang sudah dipilih, dan step 0.1 akan terlihat seperti ini,



Warna dari setiap persamaan yang dilakukan graphing dipilih secara random setiap kali dilakukan draw.

Code Challenge Animasi

Kode yang digunakan dalam code challenge ini sebenarnya tidak sebegitu berbeda dengan kode yang digunakan sebelumnya. Jadi saya tidak memisahkan Node untuk setiap graph, tetapi hanya array yang menyimpan setiap persamaan yang akan di gambarkan.

```
var line_to_draw: PoolStringArray = ['0.1 * pow(x, 2)', '0.1 * pow(x, 3)',
  'sin(x)', 'cos(x)']
```

Untuk menggati fungsi yang di gambarkan sendiri, digunakan button hanya saja yang dilakukannya hanya maju satu index, dan akan kembali ke 0 jika sudah di

ujung. Sementara dalam melakukan penggambaran ada beberapa hal yang diubah dalam `draw_expression`.

```
const SCALING = 10
var current_steps: int = 0

func draw_expression(_expression: String, _margin: float,
  steps: float = 1, _line_color: Color = Color('#ffffff')):
  var expression = Expression.new()
  expression.parse(_expression, ['x'])

  var steps_shift = current_steps * steps
  var y: float = expression.execute([steps_shift])

  var viewport_size = get_viewport_rect().size
  var viewport_length = ((viewport_size.x / 2) - _margin) / SCALING
  var viewport_height = ((viewport_size.y / 2) - _margin) / SCALING

  var x = steps
  while abs(x) < viewport_length and abs(y) < viewport_height:
    y = expression.execute([x + steps_shift])

    var current_position = Vector2(float(x), float(y))
    current_position = convert_to_normal_coordinate(current_position)

    put_pixel(current_position.x, current_position.y, _line_color)

    x += steps
```

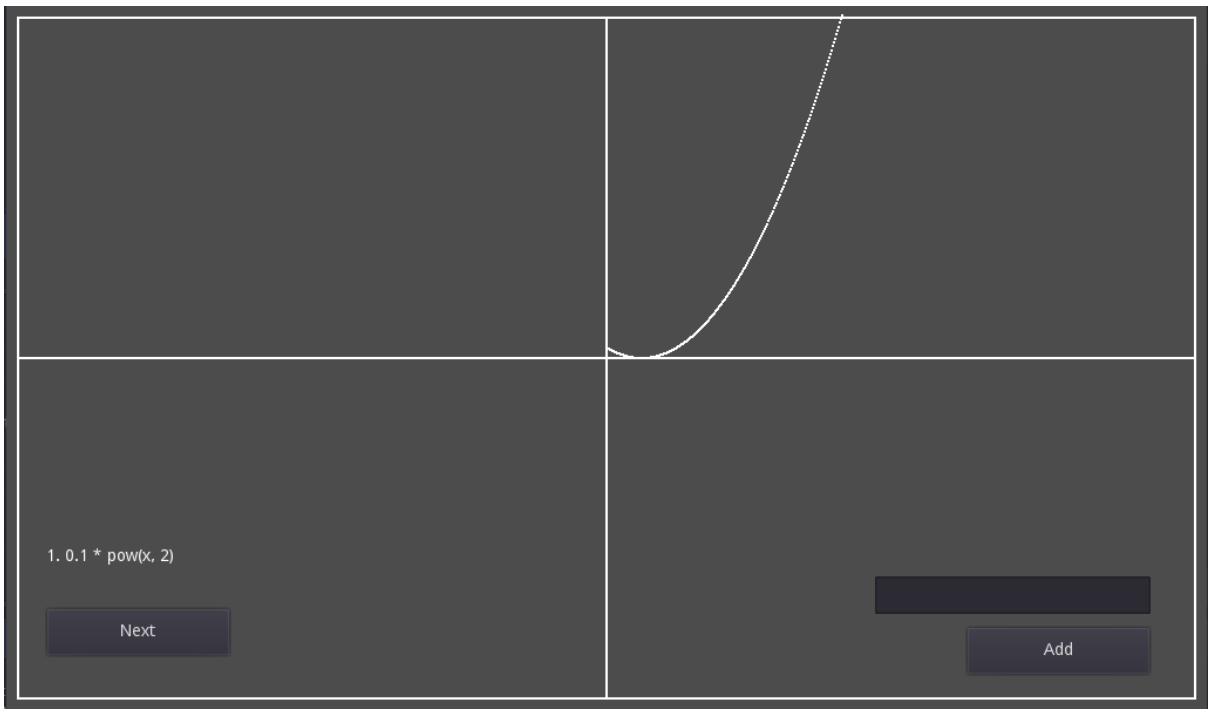
Sekarang dalam menggambar persamaan x di shift sesuai dengan sudah berapa frame yang sudah berjalan, dan karena setiap frame akan di gambarkan satu step, dikalikan dengan steps-nya. Lalu untuk menjalankan variabel `current_steps` harus di update setiap frame, maka digunakan fungsi `_process(delta)`, jika menggunakan `_update(delta)` tidak berjalan, dan saya cek kembali ke tutorial juga tidak ada fungsi sistem `_update(delta)`. Kodenya sendiri simpel,

```
func _process(delta):
  current_steps -= 1

  update()
```

Variabel di kurangi agar persamaan akan terlihat maju, yang sebenarnya hanya melakukan graph ke x negatif.

Screenshot

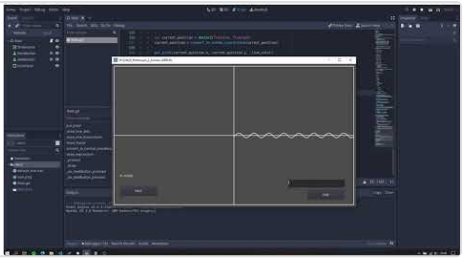
Jika script dijalankan maka akan terlihat seperti ini,



Karena bentuknya interaktif, dan juga berbentuk animasi, demo nya dapat dilihat di link di bawah ini,

Godot Graph Animation

https://youtu.be/z9pfxeWa-sQ



Lesson Learned

What went well?

Kebanyakan hal berjalan seperti harusnya, materi yang ada terlihat cukup jelas dan apa yang harus dilakukan juga cukup jelas (kecuali beberapa bagian).

What didn't go well?

Dalam mengkode bagian animasi di dalam instruksi di instruksikan untuk menggunakan fungsi `_update(delta)` namun tidak berjalan di saya, apakah karena beda versi? Atau perbedaan lainnya? saya tidak tahu. Namun, apa yang

saya temukan adalah `_process(delta)` yang berjalan seperti apa yang saya pikir harus dilakukan.

What might have been better handled if done differently?

Mungkin dengan melakukan graph dengan menggunakan line dan bukan pixel akan lebih bagus lagi. Juga dengan animasi mungkin penjelasannya lebih diperjelas lagi.

What recommendations would you give to others who might be involved in future projects of a similar type?

Mencoba untuk membuat sistem yang common, baru dengan inheritance di implementasikan ke hal yang lebih spesifik.

Temuan Lainnya

Bahwa Godot memiliki sistem yang dapat melakukan parse untuk fungsi matematis dengan class Expression. Hanya saja tidak ada fungsi `^` namun ada `pow(x, 2)`.