# Laporan Praktikum Godot

**Muhammad Fauzan Lubis**

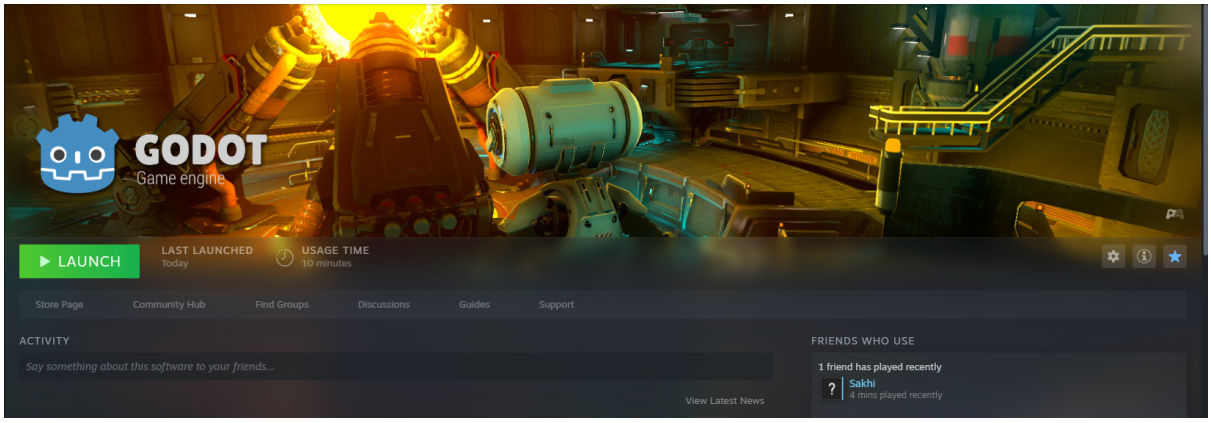
191524026

D4-2A

# Rangkuman Kegiatan

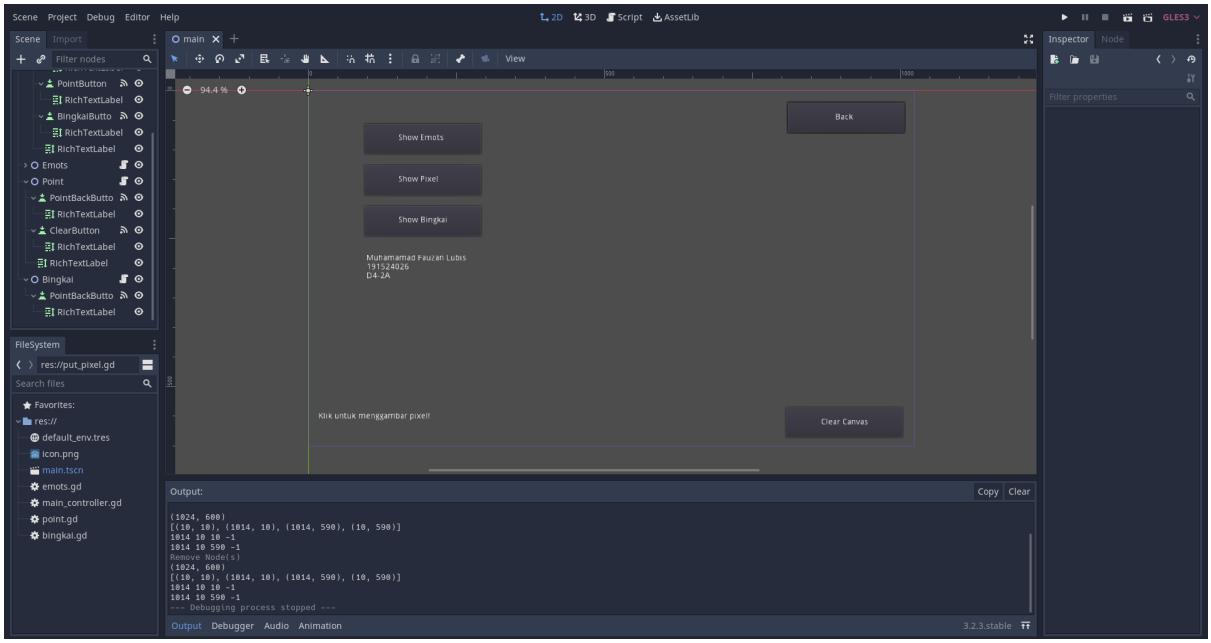
## Instalasi Godot

Instalasi yang saya pakai adalah instalasi melalui steam, dengan satu klik didpatkan Godot editor.




## Interface Godot

Interface yang diberikan sangatlah intuitf dengan sebentar saja saya dapat menggunakan Godot editor dengan mudah. Beberapa hal yang ada di sana adalah adanya view yang berbeda untuk setiap tipe dari kegiatan yang ingin dilakukan. Jadi di sana terdapat 2D untuk melakukan pemodelan pada 2 dimensi, 3D untuk pemodelan 3 dimesi, Script dimana dilakukan coding, dan AssetLib untuk mencari asset di website Godot-nya, maupun lokal.




## GDScript

Godot Script yang di support oleh Godot dalam sisi scripting game engine Godot itu sendiri. Bahasa pemrograman-nya sendiri mudah dipahami, apalgi bagi programmer yang sudah pernah menggunakan Python. Pemrogramannya juga berdasarkan pada sistem linking, sehingga penggunaan dengan komponen yang ada di sisi "modelling" lebih mudah.

## Style Guide GDScript

Style yang digunakan kurang lebih sama dengan style PEP standar yang digunakan oleh python. Sehingga membaca dan membuat kode sangatlah mudah.

## API GDScript

Dalam melakukan coding di GDScript sangatlah "berkondisi" di sini maksudnya setiap kode yang ada pasti di link antara ke suatu Node, Canvas, maupun Komponen yang ada. Membuatnya mudah untuk membagi kode-kode sesuai kebutuhannya. Namun, yang masih kurang jelas untuk saya adalah dengan cara membangun suatu sistem kode yang reusable.

Selain dari itu juga banyak API yang di desain mirip dengan Python, seperti pada String yang bisa dijadikan format dengan menggunakan "%s" atau "%d" dll. Hal ini sangat membantu developer untuk berfokus ke bagian penting codingannya.

## Custom Drawing 2D

Dalam Node yang bertipe Node2D, kita dapat menggunakan kode untuk "menggambar" suatu objek, garis, atau apapun. Di situ diberikan beberapa API level tinggi sepeti fungsi-fungsi `draw_string`, `draw_rect`, dan lainnya. Namun terdapat juga fungsi seperti `draw_primtives` yang memang menggambar pada level yang lebih dasar. Fungsi itu hanya akan membangun suatu polygon dari titik-titik yang diberikan.

## Membuat Smiley

Untuk membuat suatu smiley, saya menggunakan beberap fungsi yang ada dalam class `CanvasItem` untuk mempermudah dalam membuat wajah, dan muka secara general. Kodenya sendiri ada di bawah ini,

```
extends Node2D

var colors: Dictionary = {
  "yellow": Color("#ffeb3b"),
  "black": Color("#000000")
}

func _draw():
  # Face base
  draw_circle(Vector2(200, 200), 100, colors["yellow"])

  # Eyes
  draw_circle(Vector2(160, 175), 7, colors["black"])
  draw_circle(Vector2(240, 175), 7, colors["black"])

  # Smile!
  draw_arc(Vector2(200, 215), 55, deg2rad(0),
    deg2rad(180), 32, colors["black"], 3)
```

Kode tidak terlalu banyak, hanya membuat bentuk wajah secara general, lalu kedua bulatan mata, dan arc untuk membuat senyumnya dari setengah lingkaran. Hasil dari kode berikut adalah,


## put_pixel

Fungsi `put_pixel` saya implmentasikan sesuai dengan ketentuan, menggunakan `draw_primitive`. Fungsi, dan node-nya sendiri kodenya adalah sebgaian berikut,

```
extends Node2D

var point_color = Color("#ffffff")
var x_y_array = []

func put_pixel(x: float, y: float, color: Color, pixel_size: float = 5):
  var radius = pixel_size / 2
  var points_to_draw = PoolVector2Array()

  points_to_draw.append(Vector2(x - radius, y + radius))
  points_to_draw.append(Vector2(x + radius, y + radius))
  points_to_draw.append(Vector2(x + radius, y - radius))
  points_to_draw.append(Vector2(x - radius, y - radius))

  var color_for_pixel = [color, color, color, color]

  # parameter uvs f uv mapping (?)
  draw_primitive(points_to_draw, color_for_pixel, points_to_draw)

func _draw():
  for coordinates in x_y_array:
    put_pixel(coordinates[0], coordinates[1], point_color)

func _input(event):
  # Check if mouse is clicked and the node is active
  if event is InputEventMouseButton and self.is_visible_in_tree():
    # Preventing double click
```

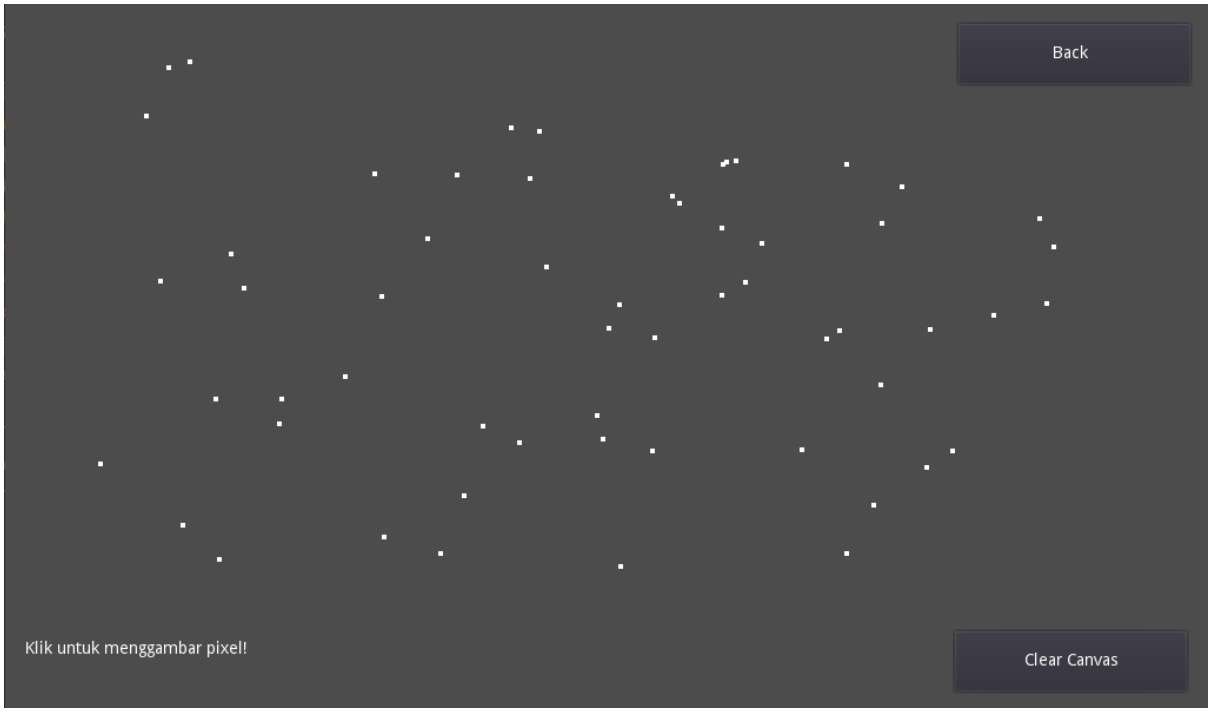
```
    if event.is_pressed():
      x_y_array.append([event.position.x, event.position.y])
      update()


func _on_ClearButton_pressed():
  x_y_array.clear()
  update()
```

Fungsi put_pixel akan mengambil x, dan y untuk posisi tengahnya, lalu dari situ akan menabahkan "padding" agar terlihat dengan jelas. "Padding" itu sendiri diatur dengan parameter tambahan yang opsional untuk mengatur seberapa tebal satu pixel yang akan diperlihatkan.

Untuk kode yang lainnya, saya menambahkan fungsionalitas tambahan dimana pixel dapat di spawn dengan mengklik ke viewport programnya. Juga ada tombol untuk menghapus semua pixel yang sudah ada di viewport.

Hasilnya sendiri terlihat seperti di bawah ini,




## Box dan Garis Diagonal

Implementasi dari garis cukup straightforward, namun terdapat permasalah dengan implementasi saya dengan garis diagonal. Kodenya sendiri adalah seperti di bawah ini,

```
extends Node2D

var window_height
var window_width

var line_color = Color("#ffffff")

# Margin for the windows
const MARGIN = 10

# Point array constants
const TOP_LEFT = 0
const TOP_RIGHT = 1
const BOTTOM_RIGHT = 2
const BOTTOM_LEFT = 3

func put_pixel(x: float, y: float, color: Color, pixel_size: float = 5):
  var radius = pixel_size / 2
  var points_to_draw = PoolVector2Array()

  points_to_draw.append(Vector2(x - radius, y + radius))
  points_to_draw.append(Vector2(x + radius, y + radius))
  points_to_draw.append(Vector2(x + radius, y - radius))
  points_to_draw.append(Vector2(x - radius, y - radius))

  var color_for_pixel = [color, color, color, color]

  # parameter uvs untuk uv mapping (?)
  draw_primitive(points_to_draw, color_for_pixel, points_to_draw)

func get_edge_points() -> Array:
  return [Vector2(MARGIN, MARGIN),
    Vector2(window_width - MARGIN, MARGIN),
    Vector2(window_width - MARGIN, window_height - MARGIN),
    Vector2(MARGIN, window_height - MARGIN)]

func draw_line_pixel(start_point: Vector2, end_point: Vector2):
  var x_change = max(-1, min(1, (end_point.x - start_point.x)))
  var y_change = max(-1, min(1, (end_point.y - start_point.y)))

  var x_point = start_point.x
  var y_point = start_point.y

  while (x_point != end_point.x) or (y_point != end_point.y):
    put_pixel(x_point, y_point, line_color)

    x_point += x_change
    y_point += y_change

func draw_diagonal_pixel(start_point: Vector2, end_point: Vector2):
  var x_change = (end_point.x - start_point.x)
  var y_change = (end_point.y - start_point.y)

  var gradient = y_change / x_change

  var y_value
```

```
  var start_x
  var end_x
  var increment

  start_x = end_point.x
  end_x = start_point.x
  y_value = start_point.y
  increment = -1

  print (start_x, " ", end_x, " ", y_value, " ", increment)

  for x in range(start_x, end_x, increment):
    put_pixel(x, y_value, line_color)

    y_value += gradient


func _draw():
  print(get_viewport_rect().size)
  window_height = get_viewport_rect().size.y
  window_width = get_viewport_rect().size.x

  var edge_points = get_edge_points()
  print(edge_points)

  # Horizontal line
  draw_line_pixel(edge_points[TOP_LEFT], edge_points[TOP_RIGHT])
  draw_line_pixel(edge_points[BOTTOM_LEFT], edge_points[BOTTOM_RIGHT])

  # Vertical line
  draw_line_pixel(edge_points[BOTTOM_LEFT], edge_points[TOP_LEFT])
  draw_line_pixel(edge_points[BOTTOM_RIGHT], edge_points[TOP_RIGHT])

  # Diagonal line
  draw_diagonal_pixel(edge_points[TOP_LEFT], edge_points[BOTTOM_RIGHT])
  draw_diagonal_pixel(edge_points[BOTTOM_LEFT], edge_points[TOP_RIGHT])
```

Kode dibuat untuk selalu melakukan pengecekan ke besar viewport setiap kali melakukan draw. Selain itu implementasi-nya kurang lebih simpel, untuk fungsi garis yang tegak (vertikal atau horizontal) lalu perubahan itu di clip ke -1 atau +1. Namun, pada yang diagonal melihat perubahannya dilakukan dengan gradien, sayangnya sekarang fungsi yang menggunakan gradien belum bisa digunakan untuk yang tegak.

# Lesson Learned

## What went well?

Kurang lebih semua proses yang ada dapat dilakukan dengan lancar. Dari mulai melakukan familiaritas ke Godot Engine cukup mudah apalagi saya pernah

menggunakan Unity sebelumnya, sehingga beberapa fungsionalitas sudah familiar. Dalam scripting juga, karena Python merupakan salah satu bahasa pertama yang saya pelajari, sehingga GDScript mudah dipahami.

## What didn't go well?

Implementasi dalam garis gradien. Sebenarnya gampang, namun apa yang saya lakukan adalah overthingking dengan implementasinya yang harus mengikuti suatu sistem grid (karena pixel), namun akhirnya tidak melakukannya karena waktu yang sudah cukup malam, dan juga saya yang belum begitu mengerti algoritma dalam clipping pixel.

## What might have been better handled if done differently?

KISS, keep it simple system. Dalam melakukan implementasi suatu hal, lebih baik untuk mencoba melakukan implementasinya secara simpel, selama tidak dihalangi dengan kewajiban, atau limitas.

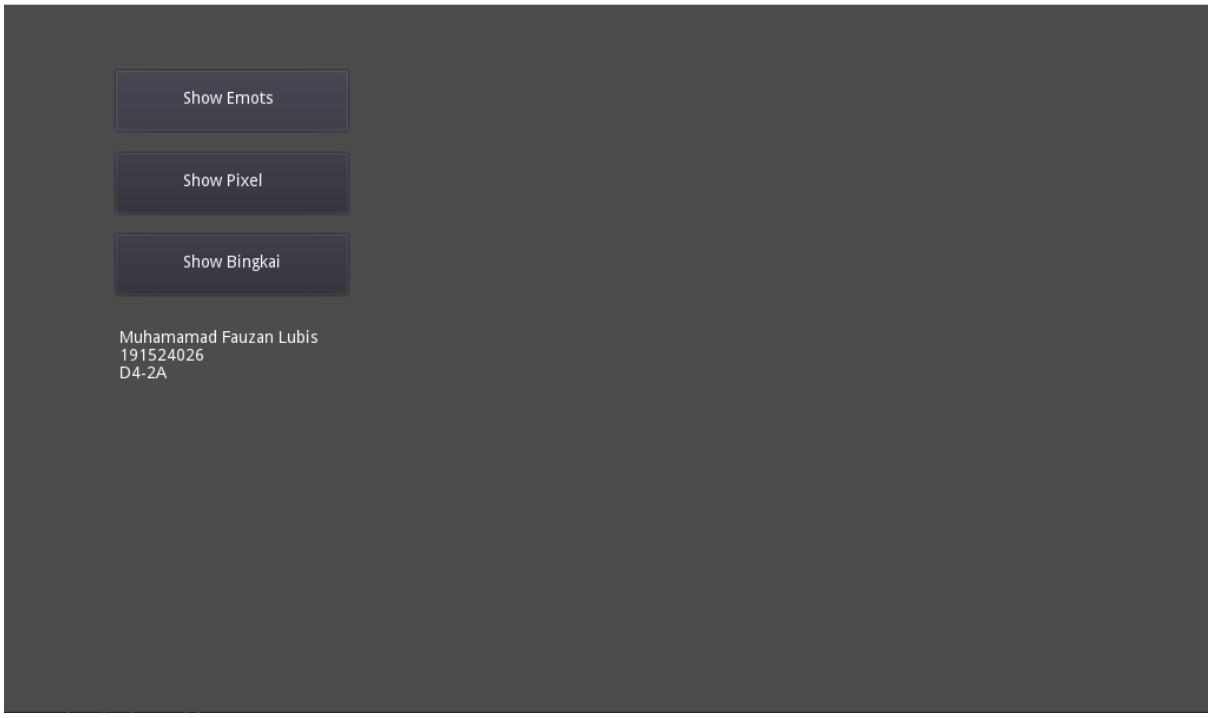
## What recommendations would you give to others who might be involved in future projects of a similar type?

Guanakan tooling dan juga hal yang sudah ada. Don't reinvent the wheel.

## Temuan Lainnya

### Sistem UI Godot

Dalam membuat tugas ini saya juga mencoba sistem UI yang ada di Godot. Ternyata dengan menggunakan Canvas saya dapat membangun sistem menu yang men-hide node lainnya hingga dibutuhkan.




Untuk kode yang mengaturnya sendiri ada di bawah ini,

```
extends Node2D

func _ready():
  $Point.hide()
  $Emots.hide()
  $Bingkai.hide()

func _on_EmotButton_pressed():
  $Menu.get_child(0).hide()
  $Emots.show()


func _on_EmotsBackButton_pressed():
  $Emots.hide()
  $Menu.get_child(0).show()


func _on_PointBackButton_pressed():
  $Point.hide()
  $Menu.get_child(0).show()


func _on_PointButton_pressed():
  $Point.show()
  $Menu.get_child(0).hide()


func _on_BingkaiButton_pressed():
  $Bingkai.show()
  $Menu.get_child(0).hide()
```

```
func _on_PointBackButton2_pressed():
  $Bingkai.hide()
  $Menu.get_child(0).show()
```

Terlihat bahwa setiap tombol hanya melakukan hide dan show. Button yang diatur di atas mengatur semua button Menu dan button Back pada setiap Node.

## Interaktivitas User

Pada tugas yang membangun `put_pixel` saya melihat ada potensi untuk menjadikannya "fun", sehingga apa yang saya lakukan adalah membuatnya dapat di klik untuk menambah pixel lainnya. Dan kode yang mengatur event klik nya juga mudah di implementasikan, karena handler low-levelnya sudah di implentasikan dari sananya. Kode yang mengaturnya seperti ini,

```
...

func _input(event):
  # Check if mouse is clicked and the node is active
  if event is InputEventMouseButton and self.is_visible_in_tree():
    # Preventing double click
    if event.is_pressed():
      x_y_array.append([event.position.x, event.position.y])
      update()

...
```

Hanya saja harus diperhatikan bahwa tidak bisa dilakukan draw ke canvas dalam fungsi lain selai draw, sehingga apa yang saya lakukan adalah menambahkan posisi klik-nya. Lalu dipanggil update yang akan melakukan draw lagi, dimana saat draw akan di loop untuk setiap posisi yang sudah di klik.

Untuk screenshotnya tidak bisa, sehingga silahkan dicoba langsung dalam program-nya 😊.