LAPORAN PEMROGRAMAN KOMPUTER SEMESTER 117

SIMULASI INDUKSI ELEKTROMAGNETIK DENGAN PYGAME

Disusun Oleh:

Kelompok 2

Muhammad Akmalluddin	(1306621014)
Novi Ken Sydney	(1306621021)
Muhammad Rosyid Suseno	(1306621023)
Nova Nur Elisa Dewi	(1306621028)
Nadhifah Najwa Rasditya	(1306621071)

DOSEN PENGAMPU:

Dewi Muliyati, M.Si.

PROGRAM STUDI FISIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI JAKARTA

2022

A. PENDAHULUAN

IPA didefinisikan sebagai disiplin ilmu yang sangat luas serta berkaitan erat dengan kehidupan manusia. Menurut Trowbridge and Bybee (1990), sains atau IPA merupakan representasi dari hubungan dinamis yang meliputi tiga faktor utama, "*the extant body of scientific knowledge, the values of science and the method and procecces of science*". Artinya, sains adalah produk dan proses, serta mengandung nilai-nilai. Sehingga, dapat dikatakan jika IPA merupakan proses atau metode penyelidikan yang mencakup cara berpikir, sikap, langkah-langkah kegiatan scientis untuk memperoleh produk-produk IPA, seperti observasi, pengukuran, merumuskan, menguji hipotesis, mengumpulkan data, berkesperimen dan prediksi.

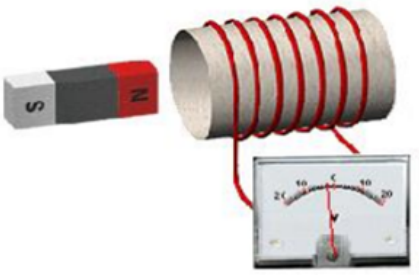
Dari definisi yang telah dijabarkan, menunjukkan bahwa IPA sangat perlu diterapkan dalam dunia pendidikan agar literasi tentang sains terus berkembang. Di Indonesia, IPA atau Ilmu Pengetahuan Alam merupakan pelajaran yang sudah diterapkan dari jenjang pendidikan dasar hingga pendidikan menengah atas. Penerapan yang cukup lama ini disebabkan IPA juga berperan dalam kemajuan teknologi dan berkaitan dengan fenomena-fenomena di kehidupan manusia.

Agar literasi sains terus berkembang, pembelajaran IPA dalam dunia pendidikan dari jenjang tingkat dasar hingga menengah atas harus menekankan pada pemahaman konsep. Pemahaman konsep (Conceptual Understanding) merupakan tujuan dalam mencapai hasil belajar. Menurut Tjandra (2005), konsep adalah kesimpulan dari suatu pengertian yang terdiri dari dua atau lebih fakta dengan memiliki ciri-ciri yang sama. Untuk pemahaman konsep yang maksimal, pendukung dunia pendidikan seperti guru harus mengembangkan simulasi yang berkaitan dengan materi yang dijelaskan, dan mengaitkannya terhadap lingkungan sekitar. Jika hal ini terimplementasikan dengan baik, maka kemampuan berpikir serta pemahaman konsep pada siswa dapat berkembang dengan baik. Berkembangnya pemahaman konsep pada siswa akan memberikan kesempatan kepadanya untuk memahami konsep lain yang lebih luas, sehingga semakin banyak alternatif yang dipilihnya dalam menghadapi masalah yang lebih kompleks.

Dalam mata pelajaran IPA, khususnya fisika kelas 9 terdapat bab yang membahas tentang induksi elektromagnetik. Induksi elektromagnetik merupakan materi yang meliputi medan magnetik, ggl induksi, fluks magnet, perubahan fluks magnet menimbulkan medan

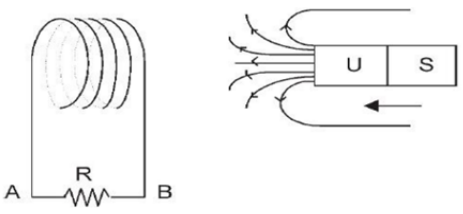
listrik. Percobaan tentang induksi elektromagnetik ini telah dilakukan oleh ilmuwan fisika yaitu Michael Faraday. Pada percobaan yang dilakukan yaitu menggerakkan magnet di sekitar kumparan.

Ilustrasi dari percobaan yang dilakukan oleh Faraday adalah sebagai berikut:



Saat magnet digerakkan mendekati atau menjauhi kumparan, akan timbul gerakan jarum galvanometer ke kanan atau ke kiri. Sedangkan, ketika magnet tidak digerakkan jarum pada galvanometer, maka jarum galvanometer tidak bergerak. Oleh karena itu, Faraday menyimpulkan bahwa medan magnet konstan tidak dapat menghasilkan arus, tetapi perubahan medan magnet dapat menghasilkan arus listrik. Arus listrik ini disebut sebagai arus induksi.

Berdasarkan percobaan yang dilakukan, menunjukkan bahwa gerakan magnet di dalam kumparan menyebabkan jarum galvanometer menyimpang. Penyimpangan jarum galvanometer ini membuktikan bahwa kedua ujung kumparan terdapat arus listrik. Peristiwa timbulnya arus listrik ini dikenal induksi elektromagnetik. Adapun beda potensial yang timbul pada ujung kumparan disebut gaya gerak listrik (GGL) induksi. Timbulnya GGL ini akibat adanya perubahan garis gaya magnet yang dilingkupi oleh kumparan.



GGL Induksi oleh Magnet yang Mendekati Kumparan

Dari penjelasan diatas menunjukkan jika bab tentang induksi elektromagnetik sangat membutuhkan animasi dan simulasi, sehingga materi yang abstrak dapat dideskripsikan secara nyata. Didukung dengan observasi yang dilakukan oleh Warjanto (2015) pada beberapa sekolah di Jakarta, ternyata masih banyak ditemui guru dalam mengajarkan materi tentang induksi elektromagnetik yang tidak menggunakan media pembelajaran interaktif dalam pembelajaran. Pengajaran lebih banyak menggunakan papan tulis dan spidol sebagai media pembelajaran. Untuk itu, peneliti memilih “Simulasi Induksi Elektromagnetik dengan Pygame”.

B. LANGKAH PROBLEM SOLVING

1. Problem Statement

   Simulasi ini dilakukan untuk mengetahui pengaruh dari medan magnet dan juga kumparan dalam induksi elektromagnetik

2. Mathematical Equations
   - Fluks Magnet

     $\Phi = BA$

     Keterangan :

     Φ = Fluks Magnet

     B = Induksi Magnet

     A = Luas Bidang

   - Hukum Faraday

     $\varepsilon = -N (\Delta\Phi/\Delta t)$

     Keterangan :

     ε = Ggl Induksi

     N = Jumlah lilitan

     $(\Delta\Phi/\Delta t)$ = Laju perubahan fluks magnet

   - GGL Induksi Diri (Hukum Henry)

     $\varepsilon = -L (dI/dt)$

     Keterangan :

     ε = Ggl Induksi diri

     L = Induktansi diri

     (dl/dt) = Besarnya perubahan arus tiap satuan waktu (A/s)

     Nilai L dapat dicari dengan rumus

     $L = N \Phi/L$

     Keterangan :

L = Induktansi diri

N = Jumlah lilitan kumparan

Φ = Fluks magnet

I = Kuat arus

3. Algoritma

Algoritma Classes

1. Mulai
2. Mengimpor pygame
3. Dari math mengimpor pi, exp
4. Jika *__name__* != '*__main__*':, maka
   4.1. Mengimpor modules.variables as v
   4.2. Jika salah mengimpor modules.variables as v
5. Mendefinisikan class Object:
   5.1. Mendefinisikan def *__init__*(self, name, x, y, sprite = None, size = None, drag = False):
      5.1.1. Menginisiasi self.name = name
      5.1.2. Jika sprite != None:
         5.1.2.1. Menginisiai self.sprite = sprite
         5.1.2.2. Menginisiasi sprite_size = sprite.get_rect().size
         5.1.2.3. Menginisiasi self.rect = pygame.Rect(x, y, sprite_size[0], sprite_size[1])
      5.1.3. Jika sprite == None and size != None:
         5.1.3.1. Menginisiasi self.rect = pygame.Rect(x, y, size[0], size[1])
      5.1.4. Jika salah
         5.1.4.1. Menginisiasi self.x = x
         5.1.4.2. Menginisiasi self.y = y
      5.1.5. Menginisiasi self.drag = drag
   5.2. Mendefinisikan def draw(self):
      5.2.1. Menginisiasi v.simWindow.blit(self.sprite, (self.rect.x, self.rect.y))
   5.3. Mendefinisikan def is_collided_with_mouse(self, mouse_pos):
      5.3.1. jika self.rect.collidepoint(mouse_pos):
         5.3.1.1. return True
      5.3.2. jika salah
         5.3.2.1. return False
   5.4. Mendefinisikan def move(self, pos):
      5.4.1. Menginisiasi self.rect.centerx = pos[0]
      5.4.2. Menginisiasi self.rect.centery = pos[1]

6. Mendefinisikan class Magnet (Object) :
    6.1. Mendefinisikan def __init__(self, name, x, y, sprite, B = 5, field_visible = False):
        6.1.1. Menginisiasi super().__init__(name, x, y, sprite = sprite)
        6.1.2. Menginisiasi self.B = B
        6.1.3. Menginisiasi self.field_visible = field_visible
        6.1.4. Menginisiasi self.inside_coil = False
        6.1.5. Menginisiasi self.magnetic_lines_list = []
        6.1.6. Menginisiasi self.save_magnetic_lines_rect()
        6.1.7. Menginisiasi self.rectunion = None
        6.1.8. Menginisiasi self.get_rectunion()
    6.2. Mendefinisikan def save_magnetic_lines_rect(self):
        6.2.1. Menginisiasi self.magnetic_lines_list = []
        6.2.2. Menginisiasi ellipse_width = v.field_width
        6.2.3. Menginisiasi ellipse_height = v.field_height
        6.2.4. Menginisiasi rect = pygame.Rect(0, 0, ellipse_width, ellipse_height)
        6.2.5. Menginisiasi rect.midbottom = self.rect.center
        6.2.6. Menginisiasi x = rect.x
        6.2.7. Menginisiasi y = rect.y
        6.2.8. Menginisiasi j = 0
        6.2.9. Melakukan pengulangan for _ in range(0, self.B): untuk
            6.2.9.1. Menginisiasi rect = pygame.Rect(x, y-j, ellipse_width, ellipse_height+j)
            6.2.9.2. Menginisiasi self.magnetic_lines_list.append(rect)
            6.2.9.3. Menginisiasi j += 10
        6.2.10. Menginisiasi j=0
        6.2.11. Melakukan pengulangan for _ in range(0, self.B):
            6.2.11.1. Menginisiasi rect = pygame.Rect(x, y+ellipse_height, ellipse_width, ellipse_height+j)
            6.2.11.2. Menginisiasi self.magnetic_lines_list.append(rect)
            6.2.11.3. Menginisiasi j += 10
    6.3. Mendefinisikan def get_rectunion(self):
        6.3.1. Menginisiasi lenght = len(self.magnetic_lines_list)-1
        6.3.2. Menginisiasi mid = lenght // 2
        6.3.3. Menginisiasi x = self.magnetic_lines_list[mid].topleft[0]
        6.3.4. Menginisiasi y = self.magnetic_lines_list[mid].topleft[1]
        6.3.5. Menginisiasi rectunion_height = self.magnetic_lines_list [-1].bottomleft[1] - y
        6.3.6. Menginisiasi rectunion = pygame.Rect(x, y, v.field_width, rectunion_height)
        6.3.7. Menginisiasi self.rectunion = rectunion
    6.4. Mendefinisikan def update_magnet(self):
        6.4.1. Menginisiasi self.save_magnetic_lines_rect()

6.4.2. Menginisiasi self.get_rectunion()

6.5. Mendefinisikan def draw_magnetic_field(self):

6.5.1. Jika self.field_visible: maka,

6.5.1.1. Menginisiasi last_index = len(self.magnetic_lines_list)-1

6.5.1.2. Menginisiasi mid = last_index // 2

6.5.1.3. Menginisiasi i=0

6.5.1.4. Melakukan pengulangan while i <= mid: maka

6.5.1.4.1. Menginisiasi pygame.draw.ellipse(

6.5.1.4.1.1. Menginisiasi v.simWindow,

6.5.1.4.1.2. Menginisiasi v.magnetic_field_color,

6.5.1.4.1.3. Menginisiasi self.magnetic_lines_list[i], v.field_lines_thickness)

6.5.1.4.2. Menginisiasi i+=1

6.5.1.5. Melakukan pengulangan while i <= last_index:

6.5.1.5.1. Menginisiasi pygame.draw.ellipse(

6.5.1.5.1.1. Menginisiasi v.simWindow,

6.5.1.5.1.2. Menginisiasi v.magnetic_field_color,

6.5.1.5.1.3. Menginisiasi self.magnetic_lines_list[i], v.field_lines_thickness)

6.5.1.5.2. Menginisiasi i+=1

6.6. Mendefinisikan def show_magnetic_field(self, event):

6.6.1. Jika event.type == pygame.KEYDOWN and event.key == pygame.K_f: maka

6.6.1.1. Jika self.field_visible: maka

6.6.1.1.1. Menginisiasi self.field_visible = False

6.6.1.2. Jika tidak self.field_visible:

6.6.1.2.1. Menginisiasi self.field_visible = True

6.7. Mendefinisikan def check_inside_coil(self, coil_rect):

6.7.1. Menginisiasi case_1 = coil_rect.collidepoint(self.rect.topleft) and coil_rect.collidepoint(self.rect.bottomleft)

6.7.2. Menginisiasi case_2 = coil_rect.collidepoint(self.rect.topright) and coil_rect.collidepoint(self.rect.bottomright)

6.7.3. jika case_1 or case_2: maka

6.7.3.1. self.inside_coil = True

6.7.4. Jika tidak coil_rect.colliderect(self.rect): maka

6.7.4.1. self.inside_coil = False

6.8. Mendefinisikan def relative_move(self, coil_rect, pos):

6.8.1. Menginisiasi self.check_inside_coil(coil_rect)

6.8.2. Jika tidak self.inside_coil: maka

6.8.2.1. Menginisiasi case_1 = (pos[1] - self.rect.height/2) >= coil_rect.bottom

6.8.2.2. Menginisiasi case_2 = (pos[1] + self.rect.height/2) <= coil_rect.top

6.8.2.3. Menginisiasi case_3 = self.rect.left > coil_rect.right
6.8.2.4. Menginisiasi case_4 = self.rect.right < coil_rect.left
6.8.2.5. Jika (case_1 or case_2) or (case_3 or case_4): maka
6.8.2.5.1. self.move(pos)
6.8.2.6. Jika tidak maka
6.8.2.6.1. Menginisiasi pom_x = self.rect.centerx
6.8.2.6.2. Menginisiasi pom_y = self.rect.centery
6.8.2.6.3. Menginisiasi self.move(pos)
6.8.2.6.4. Jika coil_rect.colliderect(self.rect): maka
6.8.2.6.4.1. Menginisiasi self.move((pom_x, pom_y))
6.8.2.6.4.2. Menginisiasi self.drag = False
6.8.3. Jika salah maka
6.8.3.1. Menginisiasi case_1 = (pos[1] + self.rect.height/2) <= coil_rect.bottom - 3
6.8.3.2. Menginisiasi case_2 = (pos[1] - self.rect.height/2) >= coil_rect.top + 3
6.8.3.3. Jika case_1 and case_2:
6.8.3.3.1. Menginisiasi self.move(pos)
6.8.3.4. Jika salah maka
6.8.3.4.1. self.drag = False
6.8.3.5. Menginisiasi self.update_magnet()
6.9. Mendefinisikan def change_magnet_features(self, event, handler):
6.9.1. jika event.type == pygame.KEYDOWN: maka
6.9.1.1. jika event.key == pygame.K_s:
6.9.1.1.1. jika self.B > v.induction_min:
6.9.1.1.1.1. Menginisiasi self.B -= 1
6.9.1.1.1.2. Menginisiasi self.update_magnet()
6.9.1.1.1.3. Menginisiasi handler.update_parameters('B', 1, '-')
6.9.1.2. jika event.key == pygame.K_w:
6.9.1.2.1. jika self.B < v.induction_max:
6.9.1.2.1.1. Menginisiasi self.B += 1
6.9.1.2.1.2. Menginisiasi self.update_magnet()
6.9.1.2.1.3. Menginisiasi handler.update_parameters('B', 1, '+')

7. Mendefinisikan class Coil(Object):
7.1. Mendefinisikan def __init__(self, name, x, y, num_coils = 10):
7.1.1. Menginisiasi super().__init__(name, x, y, size = [v.coil_width, v.coil_height])
7.1.2. Menginisiasi self.coil_color = v.coil_color
7.1.3. Menginisiasi self.num_coils = num_coils
7.1.4. Menginisiasi self.coils_list = []
7.1.5. Menginisiasi self.save_coils_rect()

7.1.6. Menginisiasi self.rectunion = None
7.1.7. Menginisiasi self.get_rectunion()
7.2. Mendefinisikan def save_coils_rect(self):
7.2.1. Menginisiasi self.coils_list = []
7.2.2. Melakukan pengulangan for _ in range(self.num_coils): untuk
7.2.2.1. Menginisiasi coil_rect = pygame.Rect(self.rect.x, self.rect.y, self.rect.width, self.rect.height)
7.2.2.2. Menginisiasi self.coils_list.append(coil_rect)
7.2.2.3. Menginisiasi self.rect.x += v.coil_spacing
7.2.3. Menginisiasi self.rect.x -= self.num_coils * v.coil_spacing
7.3. Mendefinisikan def get_rectunion(self):
7.3.1. Menginisiasi rectunion_width = self.coils_list[-1].topright[0] - self.rect.x
7.3.2. Menginisiasi rectunion = pygame.Rect(self.rect.x, self.rect.y, rectunion_width, self.rect.height)
7.3.3. Menginisiasi self.rectunion = rectunion
7.4. Mendefinisikan def draw_first_half(self):
7.4.1. Melakukan pengulangan for i in range(self.num_coils): untuk
7.4.1.1. Menginisiasi pygame.draw.arc(v.simWindow, v.coil_color,
7.4.1.1.1. Menginsiasi self.coils_list[i],
7.4.1.1.2. Menginisiasi pi/2, 3/2*pi,
7.4.1.1.3. Menginisiasi v.coil_thickness)
7.5. Mendefinisikan def draw_second_half(self):
7.5.1. Melakukan pengulangan for i in range(self.num_coils): untuk
7.5.1.1. Menginisiasi pygame.draw.arc(v.simWindow, v.coil_color,
7.5.1.1.1. Menginisiasi self.coils_list[i],
7.5.1.1.2. Menginisiasi 3/2*pi, pi/2,
7.5.1.1.3. Menginisiasi v.coil_thickness)
7.6. Mendefinisikan def draw_lightbulb(self):
7.6.1. Menginisiasi start_pos = self.rectunion.midleft
7.6.2. Menginisiasi end_pos = [self.rectunion.left, self.rectunion.y - v.coil_line_lenght]
7.6.3. Menginisiasi pygame.draw.line(v.simWindow, v.coil_color, start_pos, end_pos, v.coil_thickness-2)
7.6.4. Menginisiasi start_pos = self.rectunion.midright
7.6.5. Menginisiasi end_pos[0] = self.rectunion.right
7.6.6. Menginisiasi pygame.draw.line(v.simWindow, v.coil_color, start_pos, end_pos, v.coil_thickness-2)
7.6.7. Menginisiasi start_pos = [self.rectunion.left, self.rectunion.y - v.coil_line_lenght]
7.6.8. Menginisiasi pygame.draw.line(v.simWindow, v.coil_color, start_pos, end_pos, v.coil_thickness-2)

7.6.9. Menginisiasi circle_center = [self.rectunion.centerx, self.rectunion.y - v.coil_line_lenght]
7.6.10. Menginisiasi pygame.draw.circle(v.simWindow, v.lightbulb_color, circle_center, v.lightbulb_radius)
7.6.11. Menginisiasi pygame.draw.circle(v.simWindow, (93, 103, 105), circle_center, v.lightbulb_radius + 9, 4)
7.7. Mendefinisikan def update_coil(self):
7.7.1. Menginisiasi self.save_coils_rect()
7.7.2. Menginisiasi self.get_rectunion()
7.8. Mendefinisikan def change_coil_features(self, magnet, event, handler):
7.8.1. Jika event.type == pygame.KEYDOWN: maka
7.8.1.1. jika event.key == pygame.K_UP: maka
7.8.1.1.1. jika self.rect.height >= v.coil_min_height: maka
7.8.1.1.1.1. Menginisiasi self.rect.height -= 5
7.8.1.1.1.2. Menginisiasi handler.update_parameters('d', 5, '-')
7.8.1.1.1.3. Menginisiasi self.update_coil()
7.8.1.1.1.4. Jika self.rectunion.colliderect(magnet.rect): maka
7.8.1.1.1.4.1. jika self.rectunion.bottom < magnet.rect.bottom: maka
7.8.1.1.1.4.1.1. Menginisiasi self.rect.height += 5
7.8.1.1.1.4.1.2. Menginisiasi handler.update_parameters('d', 5, '+')
7.8.1.2. Jika event.key == pygame.K_DOWN: maka
7.8.1.2.1. jika self.rect.height <= v.coil_max_height:
7.8.1.2.1.1. Menginisiasi self.rect.height += 5
7.8.1.2.1.2. Menginisiasi handler.update_parameters('d', 5, '+')
7.8.1.2.1.3. Menginisiasi self.update_coil()
7.8.1.2.1.4. Jika self.rectunion.colliderect(magnet.rect): maka
7.8.1.2.1.4.1. Menginisiasi self.rect.height -= 5
7.8.1.2.1.4.2. Menginisiasi handler.update_parameters('d', 5, '-')
7.8.1.3. Jika event.key == pygame.K_LEFT: maka
7.8.1.3.1. jika self.num_coils > v.coil_min_num: maka
7.8.1.3.1.1. Menginisiasi self.num_coils -= 1

7.8.1.3.1.2. Menginisiasi handler.update_parameters('n', 1, '-')

7.8.1.4. Jika event.key == pygame.K_RIGHT: maka

7.8.1.4.1. Jika self.num_coils < v.coil_max_num:

7.8.1.4.1.1. Menginisiasi self.num_coils += 1

7.8.1.4.1.2. Menginisiasi handler.update_parameters('n', 1, '+')

7.8.1.4.1.3. Menginisiasi self.update_coil()

7.8.1.4.1.4. jika self.rectunion.colliderect(magnet.rect): maka

7.8.1.4.1.4.1. Menginisiasi self.num_coils -= 1

7.8.1.4.1.4.2. Menginisiasi handler.update_parameters('n', 1, '-')

7.8.2. Menginisiasi self.update_coil()

8. Mendefinisikan class PhysicsHandler:

8.1. Mendefinisikan def __init__(self, magnet, coil):

8.1.1. Menginisiasi self.dic = {}

8.1.2. Menginisiasi self.B = magnet.B * pow(10, -3) * 2

8.1.3. Menginisiasi self.n = coil.num_coils

8.1.4. Menginisiasi self.d = self.pixels_to_meters(coil.rect.height)

8.1.5. Menginisiasi self.l = coil.num_coils * self.pixels_to_meters(v.coil_spacing)

8.1.6. Menginisiasi self.E = 0.0

8.1.7. Menginisiasi self.E = 0.0

8.1.8. Menginisiasi self.fluxes = []

8.1.9. Menginisiasi self.time = []

8.1.10. Menginisiasi self.start = False

8.1.11. Menginisiasi self.update_dict_attributes()

8.2. Mendefinisikan def reduce_electromotive_force(self):

8.2.1. Menginisiasi delta = 0.35 * pow(10, -3)

8.2.2. Jika self.E >= delta:

8.2.2.1. Menginisiasi self.E -= delta

8.2.3. Jika self.E <= -delta:

8.2.3.1. Menginisiasi self.E += delta

8.2.4. Jia salah maka

8.2.4.1. Menginisiasi self.E = 0.0

8.2.5. Menginisiasi self.update_dict_attributes()

8.3. Mendefinisikan def write_parameters(self):

8.3.1. Menginisiasi font = pygame.font.Font('freesansbold.ttf', 16)

8.3.2. Menginisiasi x = 960

8.3.3. Menginisiasi y = 30

8.3.4. Melakukan pengulangan for name, param in self.dic.items(): untuk
8.3.4.1. Menginisiasi text = '{} = {} '.format(name, param)
8.3.4.2. Jika name == 'Induksi magnet (F)': maka
8.3.4.2.1. Menginisiasi text += 'mT'
8.3.4.3. Jika name == 'Tegangan':
8.3.4.3.1. Menginisiasi text += 'mV'
8.3.4.4. Jika name == 'Jumlah kumparan (L/R)':
8.3.4.4.1. Menjalankan pass
8.3.4.5. Jika salah maka
8.3.4.5.1. text += 'cm'
8.3.4.6. Menginisiasi obj = font.render(text, True, (255, 255, 255))
8.3.4.7. Menginisiasi obj_rect = obj.get_rect()
8.3.4.8. Menginisiasi obj_rect.topleft = (x, y)
8.3.4.9. Menginisiasi y += 25
8.3.4.10. Menginisiasi v.simWindow.blit(obj, obj_rect)

8.4. Mendefinisikan def update_dict_attributes(self):
8.4.1. Mendefinisikan d = self.d * pow(10, 2)
8.4.2. Mendefinisikan l = self.l * pow(10, 2)
8.4.3. Mendefinisikan B = self.B * pow(10, 3)
8.4.4. Mendefinisikan E = self.E * pow(10, 3)
8.4.5. Mendefinisikan self.dic['Jumlah kumparan (L/R)'] = self.n
8.4.6. Mendefinisikan self.dic['Diameter (Up/Down)'] = round(d, 2)
8.4.7. Mendefinisikan self.dic['Panjang kumparan'] = round(l, 2)
8.4.8. Mendefinisikan self.dic['Induksi magnet (F)'] = round(B, 2)
8.4.9. Mendefinisikan self.dic['Tegangan'] = round(E, 2)
8.5. Mendefinisikan def pixels_to_meters(self, value):
8.5.1. Melakukan return (value / 12) * pow(10, -2)
8.6. Mendefinisikan def update_parameters(self, key, value, operation):
8.6.1. jika key == 'B': maka
8.6.1.1. jika operation == '+': maka
8.6.1.1.1. Menginisiasi self.B += value * pow(10, -3) * 2
8.6.1.2. jika tidak maka self.B -= value * pow(10, -3) * 2 maka
8.6.2. jika key == 'n': maka
8.6.2.1. jika operation == '+':
8.6.2.1.1. Menginisiasi self.n += value
8.6.2.1.2. Menginisiasi self.l += value * self.pixels_to_meters(v.coil_spacing)
8.6.2.2. jika tidak maka
8.6.2.2.1. Menginisiasi self.n -= value
8.6.2.2.2. Menginisiasi self.l -= value * self.pixels_to_meters(v.coil_spacing)

8.6.3. jika key == 'd': maka
8.6.3.1. jika operation == '+': maka
8.6.3.1.1. Menginisiasi self.d += self.pixels_to_meters(value)
8.6.3.2. jika tidak maka
8.6.3.2.1. Menginisiasi self.d -= self.pixels_to_meters(value)
8.6.4. jika key == 'E': maka
8.6.4.1. Menginisiasi self.E = value
8.6.5. jika tidak maka
8.6.5.1. Menjalankan pass
8.6.6. Menginisiasi self.update_dict_attributes()
8.7. Mendefinisikan def calculate_flux(self, magnet, coil):
8.7.1. jika magnet.rectunion.colliderect(coil.rectunion): maka
8.7.1.1. Menginisiasi collisions = 0
8.7.1.2. Melakukan pengulangan for untuk line in magnet.magnetic_lines_list:
8.7.1.2.1. Melakukan pengulangan for untuk area in coil.coils_list:
8.7.1.2.1.1. jika area.colliderect(line): maka
8.7.1.2.1.1.1. Menginisiasi collisions += 1
8.7.1.3. Menginisiasi B = pow(10, -3)
8.7.1.4. Menginisiasi r = self.d/2
8.7.1.5. Menginisiasi S = r*r * pi
8.7.1.6. Menginisiasi F = B * S
8.7.1.7. Menginisiasi F = collisions * F
8.7.1.8. Mengembalikan nilai F dengan return F
8.7.2. Jika salah maka
8.7.2.1. Mengembalikan nilai 0 dengan return 0
8.8. Mendefinisikan def calculate_electromotive_force(self, start_time, end_time):
8.8.1. Menginisiasi delta_t = end_time - start_time
8.8.2. Menginisiasi delta_F = self.fluxes[1] - self.fluxes[0]
8.8.3. Menginisiasi E = - delta_F / delta_t
8.8.4. Menginisiasi self.E += E
8.8.5. Menginisiasi self.update_dict_attributes()
8.9. Mendefinisikan def relative_move(self, event):
8.9.1. Jika tidak event.type == pygame.MOUSEBUTTONUP: maka
8.9.1.1. Menginisiasi rel = event.rel
8.9.1.2. Menginisiasi is_moved = (rel[0] <= -1 or rel[0] >= 1) and (rel[1] <= -1 or rel[1] >= 1)
8.9.1.3. Mengembalikan nilai is_moved dengan return is_moved
8.9.2. jika salah maka
8.9.2.1. Mengembalikan nilai False dengan return False

8.10. Mendefinisikan def monitoring(self, event, magnet, coil, time):
8.10.1. jika self.relative_move(event): maka
8.10.1.1. jika tidak self.start:
8.10.1.1.1. Menginisiasi self.start = True
8.10.1.1.2. Menginisiasi self.time.append(time)
8.10.1.1.3. Menginisiasi self.fluxes.append(self.calculate_flux(magnet, coil))
8.10.1.2. jika tidak self.relative_move(event) and self.start: maka
8.10.1.2.1. Menginisiasi self.start = False
8.10.1.2.2. Menginisiasi self.time.append(time)
8.10.1.2.3. Menginisiasi self.fluxes.append(self.calculate_flux(magnet, coil))
8.10.1.2.4. Menginisiasi self.calculate_electromotive_force(self.time[0], self.time[1])
8.10.1.2.5. Menginisiasi self.time.clear()
8.10.1.2.6. Menginisiasi self.fluxes.clear()
8.11. Mendefinisikan def sigmoid(self, x):
8.11.1. Mengembalikan nilai 220 / (1 + exp(-x+8)) + 35 dengan return 220 / (1 + exp(-x+8)) + 35
8.12. Mendefinisikan def change_light_strength(self):
8.12.1. Menginisiasi E = abs(self.E * pow(10, 3))
8.12.2. Menginisiasi val = round(self.sigmoid(E))
8.12.3. Menginisiasi v.lightbulb_color = [val, val, 0]

Algoritma Utama

1. Mulai
2. Mengimpor modules.variables sebagai v
3. Mengimpor modules.classes sebagai c
4. Mengimpor pygame
5. Mengimpor waktu sebagai t
6. Menginisiasi magnet = c.Magnet('magnet', 450, 500, v.sp_magnet)
7. Menginisiasi coil = c.Coil('coil', 300, 280, 25)
8. Menginisiasi handler = c.PhysicsHandler(magnet, coil)
9. Melakukan loop dengan pengulangan while True
9.1. Menginisiasi v.simWindow.fill(v.background)
9.2. Menginisiasi coil.draw_lightbulb()
9.3. Menginisiasicoil.draw_first_half()
9.4. Menginisiasi magnet.draw_magnetic_field()
9.5. Menginisiasi magnet.draw()
9.6. Menginisiasi coil.draw_second_half()

9.7. Untuk event dalam pygame.event.get():
9.7.1. Membuat kondisi jika event.type == pygame.MOUSEBUTTONDOWN dan magnet.is_collided_with_mouse(event.pos):
9.7.1.1. Menginisiasi magnet.drag = True
9.7.2. Membuat kondisi jika event.type == pygame.MOUSEMOTION dan magnet.drag:
9.7.2.1. Menginisiasi handler.monitoring(event, magnet, coil, t.time())
9.7.2.2. Menginisiasi magnet.relative_move(coil.rectunion, event.pos)
9.7.3. Membuat kondisi jika event.type == pygame.MOUSEBUTTONUP:
9.7.3.1. Menginisiasi handler.monitoring(event, magnet, coil, t.time())
9.7.3.2. Menginisiasi magnet.drag = False
9.7.4. Membuat kondisi jika event.type == pygame.QUIT:
9.7.4.1. exit()
9.8. Mendeklarasikan magnet.show_magnetic_field(event)
9.9. Mendeklarasikan magnet.change_magnet_features(event, handler)
9.10. Mendeklarasikan coil.change_coil_features(magnet, event, handler)
10. Mendeklarasikan handler.write_parameters()
11. Mendeklarasikan handler.change_light_strength()
12. Mendeklarasikan handler.reduce_electromotive_force()
13. Mendeklarasikan v.write_author_name(v.simWindow)
14. Mendeklarasikan pygame.display.update()
15. Mendeklarasikan v.clock.tick(60)
16. Selesai

Algoritma Variabel

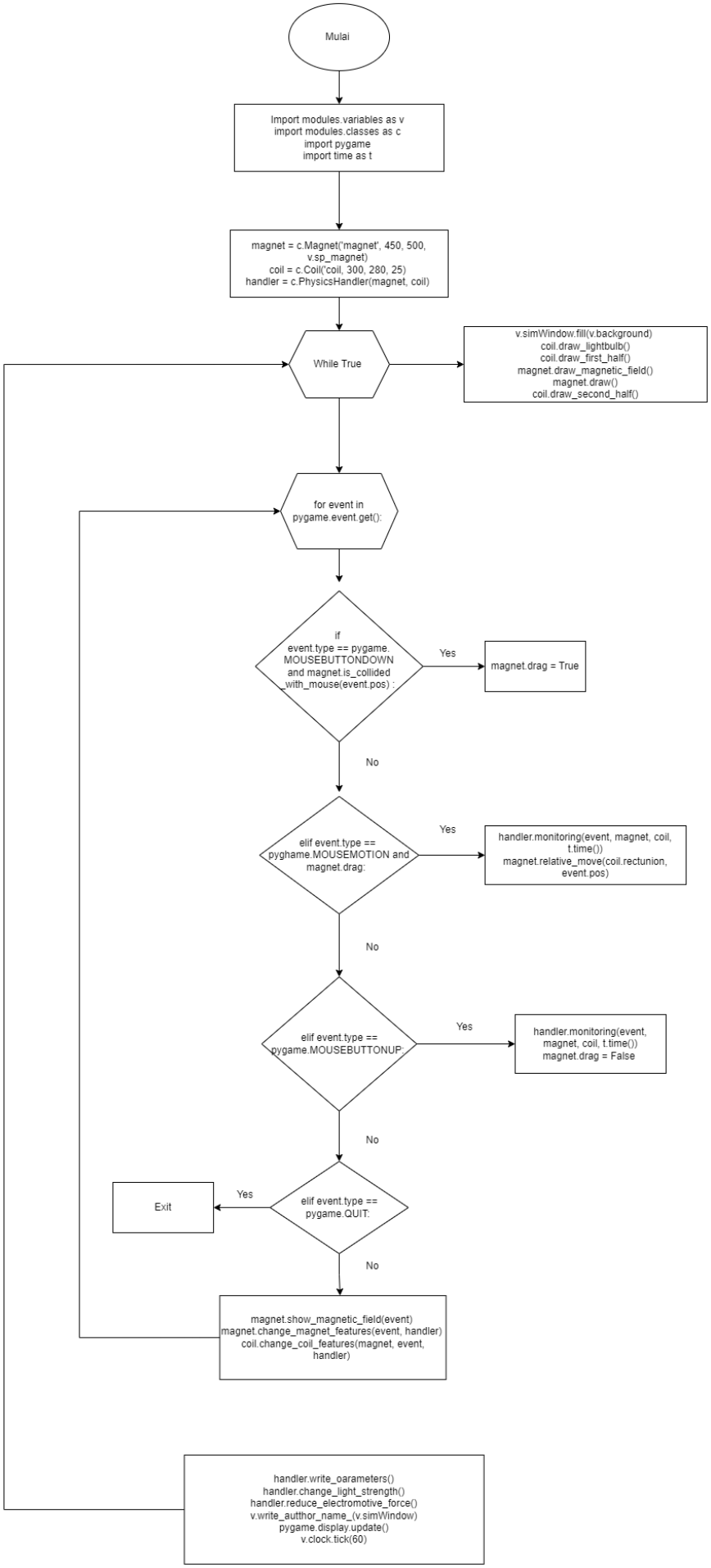
1. Mengimpor pygame
2. Mengimpor os
3. Memanggil pygame.init()
4. Menginisiasi (dirname, prom) = os.path.split(os.path.dirname(_file_))
5. Menginisiasi magnet_address = dirname + '\\resources\\magnet.png'
6. Menginisiasi icon_address = dirname + '\\resources\\icon.png'
7. Menginisiasi sp_magnet = pygame.image.load(magnet_address)
8. Menginisiasi icon = pygame.image.load(icon_address)
9. Menginisiasi wind_width = 1200
10. Menginisiasi wind_height = 800
11. Menginisiasi simWindow = pygame.display.set_mode((wind_width, wind_height))

12. Menginisiasi pygame.display.set_caption('KELOMPOK 2: SIMULASI ELEKTROMAGNETIK')
13. Menginisiasi pygame.display.set_icon(icon)
14. Menginisiasi background = (0, 0, 0)
15. Menginisiasi magnetic_field_color = (97, 189, 194)
16. Menginisiasi coil_color = (184, 115, 51)
17. Menginisiasi lightbulb_color = [35, 35, 0]
18. Menginisiasi field_width = 330
19. Menginisiasi field_height = 60
20. Menginisiasi field_lines_thickness = 2
21. Menginisiasi induction_min = 1
22. Menginisiasi induction_max = 10
23. Menginisiasi coil_width = 50
24. Menginisiasi = 145
25. Menginisiasi coil_min_height = 95
26. Menginisiasi coil_max_height = 200
27. Menginisiasi coil_spacing = 16
28. Menginisiasi coil_thickness = 5
29. Menginisiasi coil_min_num = 3
30. Menginisiasi coil_max_num = 35
31. Menginisiasi coil_line_lenght = 110
32. Menginisiasi lightbulb_radius = 25
33. Menginisiasi clock = pygame.time.Clock()
34. Membuat fungsi def write_author_name(simWindow):
    34.1. Menginisiasi font = pygame.font.Font('freesansbold.ttf', 16)
    34.2. Menginisiasi author_text = font.render('KASIH TUTORIAL', True, (255, 255, 255))
    34.3. Menginisiasi author_text_rect = author_text.get_rect()
    34.4. Menginisiasi author_text_rect.center = (wind_width * 0.9, wind_height * 0.95)
    34.5. Menginisiasi simWindow.blit(author_text, author_text_rect)

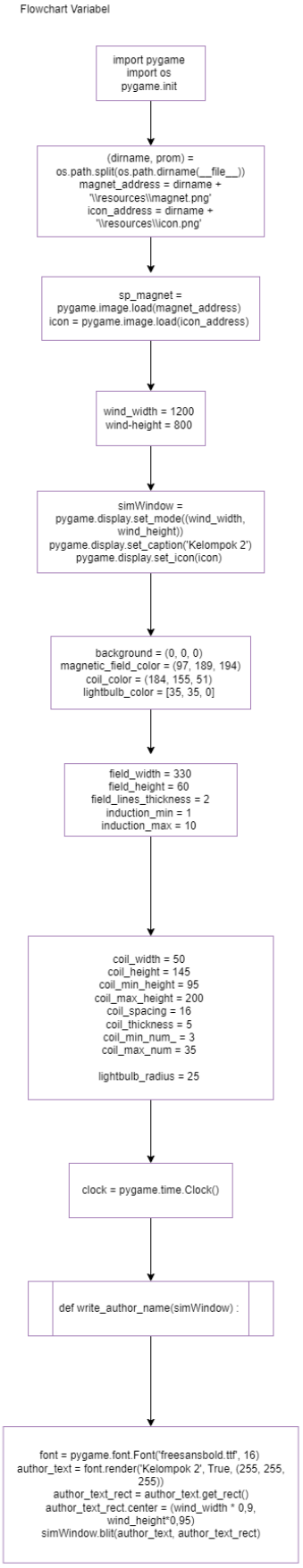
Flowchart

- Main Flowchart

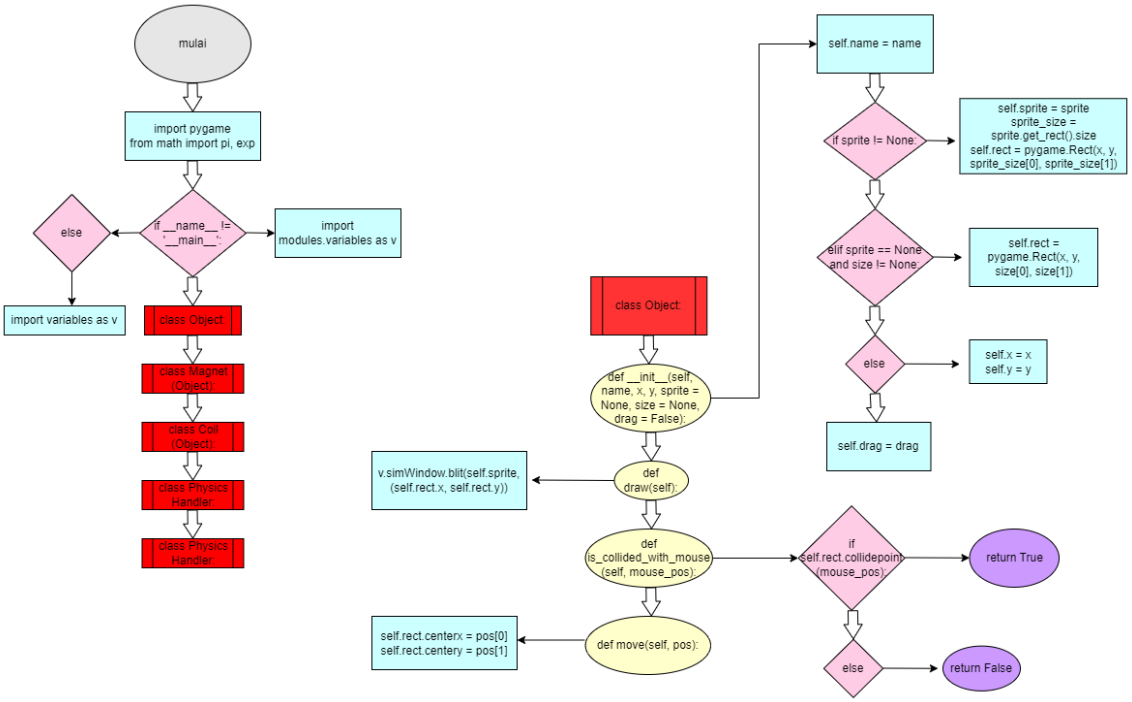
FLOWCHART UTAMA



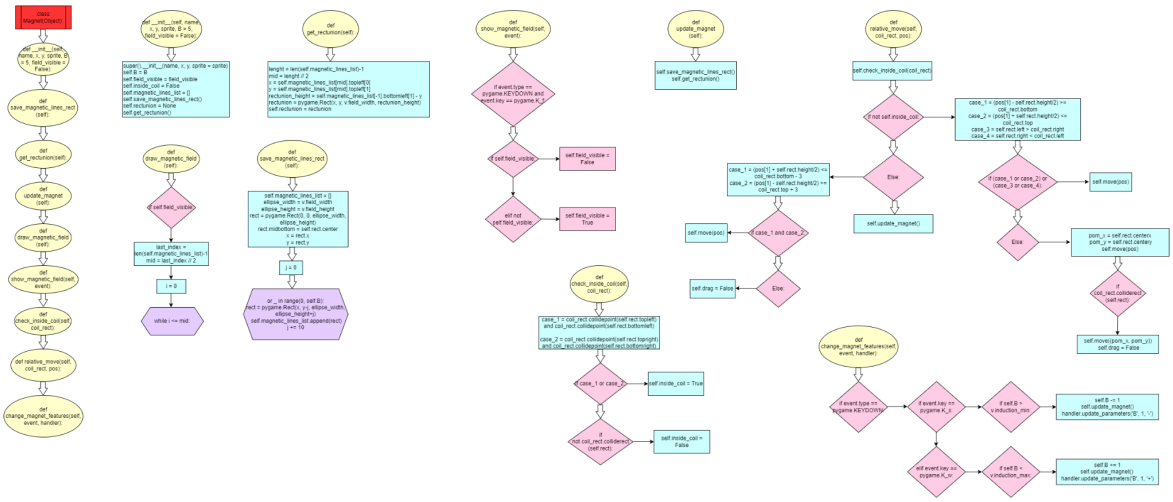
- Flowchart Variable



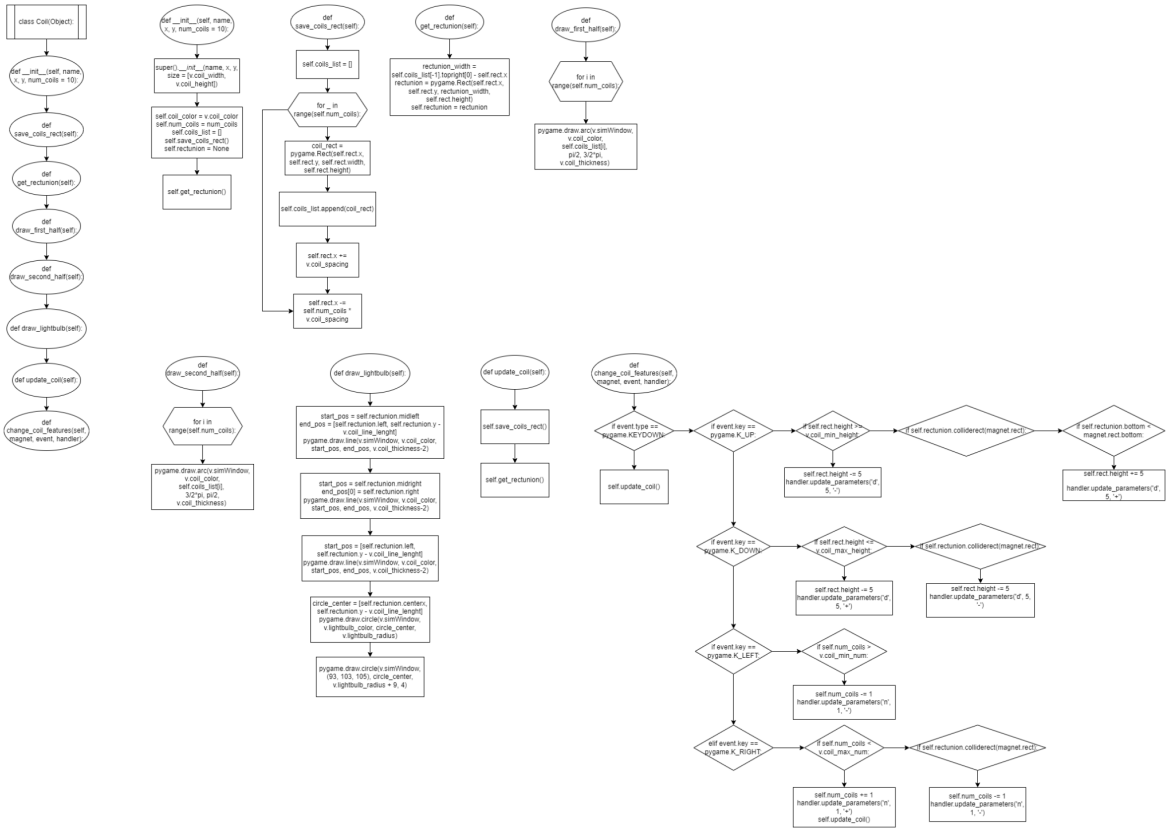
- Flowchart Classes
  - Class Object



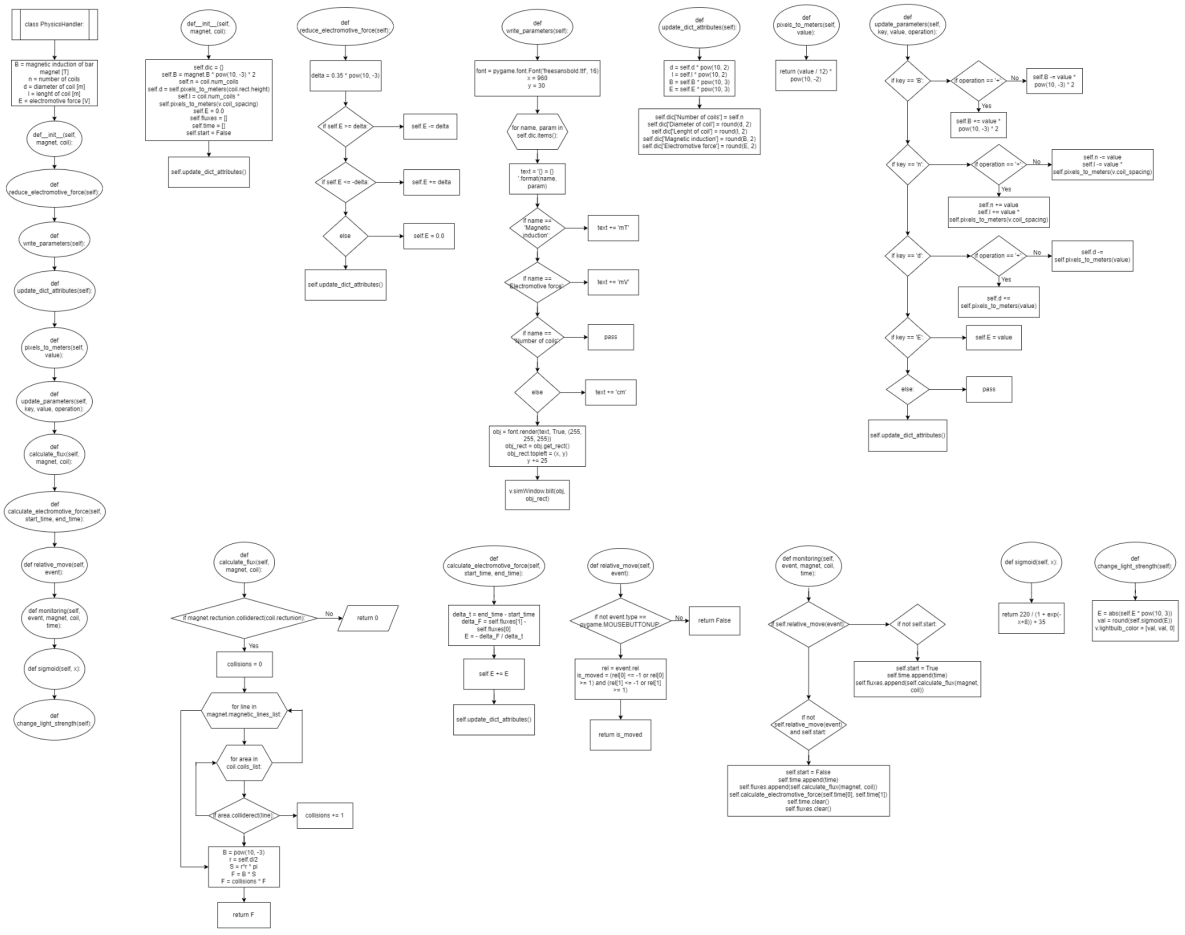
  - Class Magnet



- Class Coil (object)



- Class PhysicsHandler



Source Code

- Main Code

```
import modules.variables as v

import modules.classes as c

import pygame

import time as t



magnet = c.Magnet('magnet', 450, 500, v.sp_magnet)

coil = c.Coil('coil', 300, 280, 25)
```

```
handler = c.PhysicsHandler(magnet, coil)


while True:
    v.simWindow.fill(v.background)
    coil.draw_lightbulb()
    coil.draw_first_half()
    magnet.draw_magnetic_field()
    magnet.draw()
    coil.draw_second_half()


    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN and
magnet.is_collided_with_mouse(event.pos):
            magnet.drag = True
        elif event.type == pygame.MOUSEMOTION and magnet.drag:
            handler.monitoring(event, magnet, coil, t.time())
            magnet.relative_move(coil.rectunion, event.pos)
        elif event.type == pygame.MOUSEBUTTONUP:
            handler.monitoring(event, magnet, coil, t.time())
            magnet.drag = False
        elif event.type == pygame.QUIT:
            exit()
```

```
            magnet.show_magnetic_field(event)
            magnet.change_magnet_features(event, handler)
            coil.change_coil_features(magnet, event, handler)

        handler.write_parameters()
        handler.change_light_strength()
        handler.reduce_electromotive_force()
        v.write_author_name(v.simWindow)
        pygame.display.update()
        v.clock.tick(60)
```

- Classes Code

```
import pygame
from math import pi, exp
if __name__ != '__main__':
    import modules.variables as v
else:
    import variables as v

class Object:
    def __init__(self, name, x, y, sprite = None, size = None, drag = False):
```

```
        self.name = name
        if sprite != None:
            self.sprite = sprite
            sprite_size = sprite.get_rect().size
            self.rect = pygame.Rect(x, y, sprite_size[0], sprite_size[1])
        elif sprite == None and size != None:
            self.rect = pygame.Rect(x, y, size[0], size[1])
        else:
            self.x = x
            self.y = y

        self.drag = drag

    def draw(self):
        v.simWindow.blit(self.sprite, (self.rect.x, self.rect.y))

    def is_collided_with_mouse(self, mouse_pos):
        if self.rect.collidepoint(mouse_pos):
            return True
        else:
            return False
```

```
    def move(self, pos):
        self.rect.centerx = pos[0]
        self.rect.centery = pos[1]


class Magnet(Object):
    def __init__(self, name, x, y, sprite, B = 5, field_visible = False):
        super().__init__(name, x, y, sprite = sprite)
        self.B = B
        self.field_visible = field_visible
        self.inside_coil = False
        self.magnetic_lines_list = []
        self.save_magnetic_lines_rect()
        self.rectunion = None
        self.get_rectunion()

    def save_magnetic_lines_rect(self):
            self.magnetic_lines_list = []
            ellipse_width = v.field_width
            ellipse_height = v.field_height
```

```
        rect = pygame.Rect(0, 0, ellipse_width, ellipse_height)
        rect.midbottom = self.rect.center
        x = rect.x
        y = rect.y

        j = 0
        for _ in range(0, self.B):
            rect = pygame.Rect(x, y-j, ellipse_width, ellipse_height+j)
            self.magnetic_lines_list.append(rect)
            j += 10

        j = 0
        for _ in range(0, self.B):
            rect = pygame.Rect(x, y+ellipse_height, ellipse_width, ellipse_height+j)
            self.magnetic_lines_list.append(rect)
            j += 10

def get_rectunion(self):
    lenght = len(self.magnetic_lines_list)-1
    mid = lenght // 2
    x = self.magnetic_lines_list[mid].topleft[0]
```

```
    y = self.magnetic_lines_list[mid].topleft[1]
    rectunion_height = self.magnetic_lines_list[-1].bottomleft[1] - y
    rectunion = pygame.Rect(x, y, v.field_width, rectunion_height)
    self.rectunion = rectunion


def update_magnet(self):
    self.save_magnetic_lines_rect()
    self.get_rectunion()


def draw_magnetic_field(self):
    if self.field_visible:
        last_index = len(self.magnetic_lines_list)-1
        mid = last_index // 2


        i = 0
        while i <= mid:
            pygame.draw.ellipse(
                v.simWindow,
                v.magnetic_field_color,
                self.magnetic_lines_list[i], v.field_lines_thickness)
            i += 1
```

```
            while i <= last_index:
                pygame.draw.ellipse(
                    v.simWindow,
                    v.magnetic_field_color,
                    self.magnetic_lines_list[i], v.field_lines_thickness)
                i += 1


    def show_magnetic_field(self, event):
        if event.type == pygame.KEYDOWN and event.key == pygame.K_f:
            if self.field_visible:
                self.field_visible = False
            elif not self.field_visible:
                self.field_visible = True


    def check_inside_coil(self, coil_rect):
        case_1 = coil_rect.collidepoint(self.rect.topleft) and
coil_rect.collidepoint(self.rect.bottomleft)
        case_2 = coil_rect.collidepoint(self.rect.topright) and
coil_rect.collidepoint(self.rect.bottomright)
        if case_1 or case_2:
            self.inside_coil = True
```

```
    if not coil_rect.colliderect(self.rect):
        self.inside_coil = False


def relative_move(self, coil_rect, pos):
    self.check_inside_coil(coil_rect)


    if not self.inside_coil:
        case_1 = (pos[1] - self.rect.height/2) >= coil_rect.bottom
        case_2 = (pos[1] + self.rect.height/2) <= coil_rect.top
        case_3 = self.rect.left > coil_rect.right
        case_4 = self.rect.right < coil_rect.left
        if (case_1 or case_2) or (case_3 or case_4):
            self.move(pos)
        else:
            pom_x = self.rect.centerx
            pom_y = self.rect.centery
            self.move(pos)
            if coil_rect.colliderect(self.rect):
                self.move((pom_x, pom_y))
                self.drag = False
```

```
        else:
            case_1 = (pos[1] + self.rect.height/2) <= coil_rect.bottom - 3
            case_2 = (pos[1] - self.rect.height/2) >= coil_rect.top + 3
            if case_1 and case_2:
                self.move(pos)
            else:
                self.drag = False

        self.update_magnet()

    def change_magnet_features(self, event, handler):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_s:
                if self.B > v.induction_min:
                    self.B -= 1
                    self.update_magnet()
                    handler.update_parameters('B', 1, '-')
            elif event.key == pygame.K_w:
                if self.B < v.induction_max:
                    self.B += 1
                    self.update_magnet()
```

```
                handler.update_parameters('B', 1, '+')



class Coil(Object):
    def __init__(self, name, x, y, num_coils = 10):
        super().__init__(name, x, y, size = [v.coil_width, v.coil_height])
        self.coil_color = v.coil_color
        self.num_coils = num_coils
        self.coils_list = []
        self.save_coils_rect()
        self.rectunion = None
        self.get_rectunion()


    def save_coils_rect(self):
        self.coils_list = []
        for _ in range(self.num_coils):
            coil_rect = pygame.Rect(self.rect.x, self.rect.y, self.rect.width, self.rect.height)
            self.coils_list.append(coil_rect)
            self.rect.x += v.coil_spacing


        self.rect.x -= self.num_coils * v.coil_spacing
```

```
def get_rectunion(self):
    rectunion_width = self.coils_list[-1].topright[0] - self.rect.x
    rectunion = pygame.Rect(self.rect.x, self.rect.y, rectunion_width, self.rect.height)
    self.rectunion = rectunion


def draw_first_half(self):
    for i in range(self.num_coils):
        pygame.draw.arc(v.simWindow, v.coil_color,
            self.coils_list[i],
            pi/2, 3/2*pi,
            v.coil_thickness)


def draw_second_half(self):
    for i in range(self.num_coils):
        pygame.draw.arc(v.simWindow, v.coil_color,
            self.coils_list[i],
            3/2*pi, pi/2,
            v.coil_thickness)


def draw_lightbulb(self):
```

```
        start_pos = self.rectunion.midleft

        end_pos = [self.rectunion.left, self.rectunion.y - v.coil_line_lenght]

        pygame.draw.line(v.simWindow, v.coil_color, start_pos, end_pos,
v.coil_thickness-2)


        start_pos = self.rectunion.midright

        end_pos[0] = self.rectunion.right

        pygame.draw.line(v.simWindow, v.coil_color, start_pos, end_pos,
v.coil_thickness-2)


        start_pos = [self.rectunion.left, self.rectunion.y - v.coil_line_lenght]

        pygame.draw.line(v.simWindow, v.coil_color, start_pos, end_pos,
v.coil_thickness-2)


        circle_center = [self.rectunion.centerx, self.rectunion.y - v.coil_line_lenght]

        pygame.draw.circle(v.simWindow, v.lightbulb_color, circle_center,
v.lightbulb_radius)


        pygame.draw.circle(v.simWindow, (93, 103, 105), circle_center,
v.lightbulb_radius + 9, 4)


    def update_coil(self):

        self.save_coils_rect()
```

```
    self.get_rectunion()


def change_coil_features(self, magnet, event, handler):
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_UP:
            if self.rect.height >= v.coil_min_height:
                self.rect.height -= 5
                handler.update_parameters('d', 5, '-')
                self.update_coil()
                if self.rectunion.colliderect(magnet.rect):
                    if self.rectunion.bottom < magnet.rect.bottom:
                        self.rect.height += 5
                        handler.update_parameters('d', 5, '+')
        elif event.key == pygame.K_DOWN:
            if self.rect.height <= v.coil_max_height:
                self.rect.height += 5
                handler.update_parameters('d', 5, '+')
                self.update_coil()
                if self.rectunion.colliderect(magnet.rect):
                    self.rect.height -= 5
                    handler.update_parameters('d', 5, '-')
```

```
            elif event.key == pygame.K_LEFT:
                if self.num_coils > v.coil_min_num:
                    self.num_coils -= 1
                    handler.update_parameters('n', 1, '-')

            elif event.key == pygame.K_RIGHT:
                if self.num_coils < v.coil_max_num:
                    self.num_coils += 1
                    handler.update_parameters('n', 1, '+')
                    self.update_coil()
                    if self.rectunion.colliderect(magnet.rect):
                        self.num_coils -= 1
                        handler.update_parameters('n', 1, '-')

        self.update_coil()



class PhysicsHandler:
    ''' Parameters meaning:
            B = Induksi magnet (F) [T]
            n = Jumlah kumparan (L/R)
```

```
        d = Diameter (Up/Down) [m]

        l = Panjang kumparan [m]

        E = Tegangan [V]

'''

def __init__(self, magnet, coil):

    self.dic = {}

    self.B = magnet.B * pow(10, -3) * 2

    self.n = coil.num_coils

    self.d = self.pixels_to_meters(coil.rect.height)

    self.l = coil.num_coils * self.pixels_to_meters(v.coil_spacing)

    self.E = 0.0

    self.fluxes = []

    self.time = []

    self.start = False


    self.update_dict_attributes()


def reduce_electromotive_force(self):

    delta = 0.35 * pow(10, -3)

    if self.E >= delta:

        self.E -= delta
```

```
    elif self.E <= -delta:
        self.E += delta
    else:
        self.E = 0.0

    self.update_dict_attributes()

def write_parameters(self):
    font = pygame.font.Font('freesansbold.ttf', 16)
    x = 960
    y = 30
    for name, param in self.dic.items():
        text = '{} = {} '.format(name, param)
        if name == 'Induksi magnet (F)':
            text += 'mT'
        elif name == 'Tegangan':
            text += 'mV'
        elif name == 'Jumlah kumparan (L/R)':
            pass
        else:
            text += 'cm'
```

```
        obj = font.render(text, True, (255, 255, 255))
        obj_rect = obj.get_rect()
        obj_rect.topleft = (x, y)
        y += 25

        v.simWindow.blit(obj, obj_rect)

def update_dict_attributes(self):
    # transfers meters to centimeters and
    # teslas to militeslas when writing on screen
    d = self.d * pow(10, 2)
    l = self.l * pow(10, 2)
    B = self.B * pow(10, 3)
    E = self.E * pow(10, 3)
    self.dic['Jumlah kumparan (L/R)'] = self.n
    self.dic['Diameter (Up/Down)'] = round(d, 2)
    self.dic['Panjang kumparan'] = round(l, 2)
    self.dic['Induksi magnet (F)'] = round(B, 2)
    self.dic['Tegangan'] = round(E, 2)
```

```
def pixels_to_meters(self, value):
    return (value / 12) * pow(10, -2)


def update_parameters(self, key, value, operation):
    if key == 'B':
        if operation == '+':
            self.B += value * pow(10, -3) * 2
        else:
            self.B -= value * pow(10, -3) * 2
    elif key == 'n':
        if operation == '+':
            self.n += value
            self.l += value * self.pixels_to_meters(v.coil_spacing)
        else:
            self.n -= value
            self.l -= value * self.pixels_to_meters(v.coil_spacing)
    elif key == 'd':
        if operation == '+':
            self.d += self.pixels_to_meters(value)
        else:
            self.d -= self.pixels_to_meters(value)
```

```
    elif key == 'E':
        self.E = value
    else:
        pass

    self.update_dict_attributes()

def calculate_flux(self, magnet, coil):
    if magnet.rectunion.colliderect(coil.rectunion):
        collisions = 0
        for line in magnet.magnetic_lines_list:
            for area in coil.coils_list:
                if area.colliderect(line):
                    collisions += 1

        # one magnetic line has value of 1 mT
        B = pow(10, -3)
        r = self.d/2
        S = r*r * pi
        F = B * S
        F = collisions * F
```

```
            return F


        else:
            return 0


    def calculate_electromotive_force(self, start_time, end_time):
        delta_t = end_time - start_time
        delta_F = self.fluxes[1] - self.fluxes[0]
        E = - delta_F / delta_t
        self.E += E


        self.update_dict_attributes()


    def relative_move(self, event):
        if not event.type == pygame.MOUSEBUTTONUP:
            rel = event.rel
            is_moved = (rel[0] <= -1 or rel[0] >= 1) and (rel[1] <= -1 or rel[1] >= 1)
            return is_moved
        else:
            return False
```

```
def monitoring(self, event, magnet, coil, time):
    if self.relative_move(event):
        if not self.start:
            self.start = True
            self.time.append(time)
            self.fluxes.append(self.calculate_flux(magnet, coil))
    elif not self.relative_move(event) and self.start:
        self.start = False
        self.time.append(time)
        self.fluxes.append(self.calculate_flux(magnet, coil))
        self.calculate_electromotive_force(self.time[0], self.time[1])
        self.time.clear()
        self.fluxes.clear()


def sigmoid(self, x):
    # Modified sigmoid function
    return 220 / (1 + exp(-x+8)) + 35


def change_light_strength(self):
    E = abs(self.E * pow(10, 3))
    val = round(self.sigmoid(E))
```

```
        v.lightbulb_color = [val, val, 0]
```

- Variables Code

```
import pygame

import os


pygame.init()


# Sprite addresses

(dirname, prom) = os.path.split(os.path.dirname(__file__))

magnet_address = dirname + '\\resources\\magnet.png'

icon_address = dirname + '\\resources\\icon.png'


# Sprites

sp_magnet = pygame.image.load(magnet_address)

icon = pygame.image.load(icon_address)


# Window variables

wind_width = 1200

wind_height = 800


simWindow = pygame.display.set_mode((wind_width, wind_height))
```

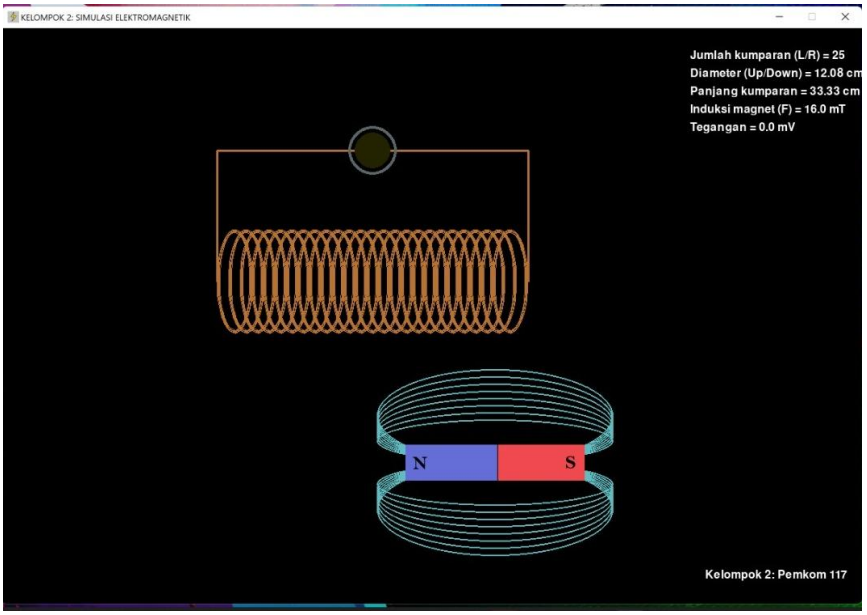
```
pygame.display.set_caption('Electromagnetism')

pygame.display.set_icon(icon)


# Color variables

background = (0, 0, 0)

magnetic_field_color = (97, 189, 194)

coil_color = (184, 115, 51)

# Lightbulb color will change during simulation

lightbulb_color = [35, 35, 0]


# Other variables

field_width = 330

field_height = 60

field_lines_thickness = 2

induction_min = 1

induction_max = 10


coil_width = 50

coil_height = 145

coil_min_height = 95

coil_max_height = 200
```

```
coil_spacing = 16

coil_thickness = 5

coil_min_num = 3

coil_max_num = 35

coil_line_lenght = 110


lightbulb_radius = 25


# Clock variable

clock = pygame.time.Clock()


# Text (author credits)

def write_author_name(simWindow):

    font = pygame.font.Font('freesansbold.ttf', 16)

    author_text = font.render('Made by: Kusla75', True, (255, 255, 255))

    author_text_rect = author_text.get_rect()

    author_text_rect.center = (wind_width * 0.9, wind_height * 0.95)


    simWindow.blit(author_text, author_text_rect)
```

C. ANALISIS HASIL

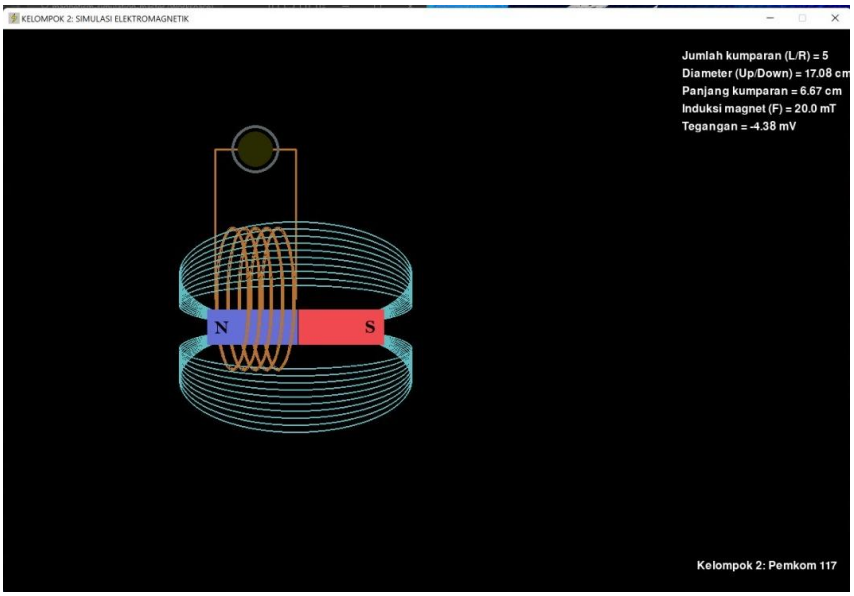
Berdasarkan program yang telah dibuat, sebelum menggunakan simulasi pengguna dapat mengatur jumlah kumparan yang akan digunakan. Selanjutnya, pengguna juga dapat menampilkan medan magnet pada magnet dan mengatur besarnya nilai yang digunakan. Untuk mengatur jumlah kumparan dapat diatur dengan menekan arah kanan dan kiri pada keyboard, untuk mengatur diameter dari kumparan dilakukan dengan menekan arah atas dan bawah, induksi magnet dapat dimunculkan dengan menekan tombol (F), dan untuk mengatur besar kecil medan magnet dapat diatur dengan menekan tombol (W/S). .Dari penjalanan program, hasilnya adalah sebagai berikut.

1. Awal Mula Program

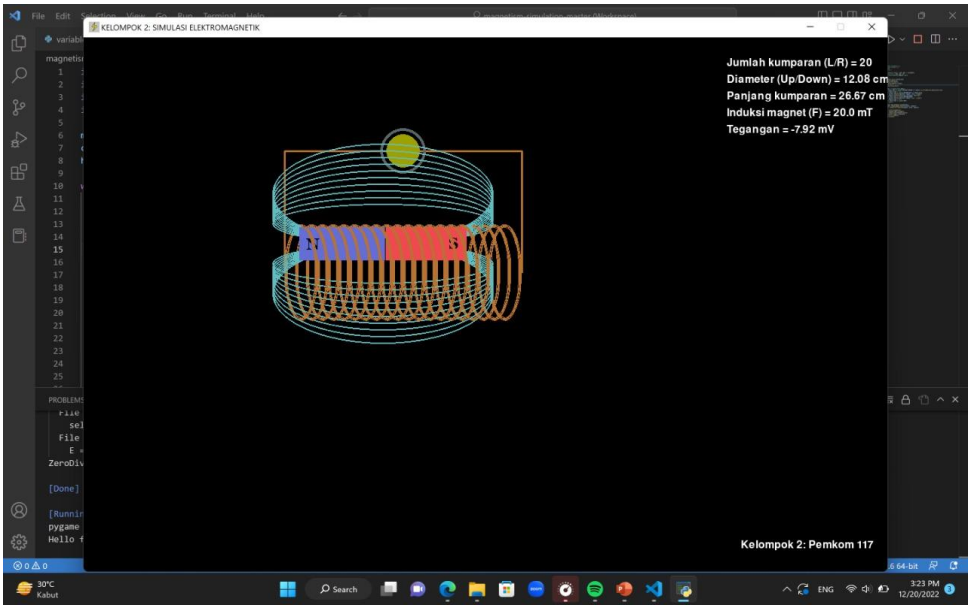


Gambar diatas adalah tampilan awal setelah pengguna mengatur jumlah kumparan dan menampilkan medan magnet . Terlihat bahwa pada ujung kanan atas terdapat keterangan seperti jumlah kumparan yang digunakan, diameter dari kumparan, panjang kumparan, induksi magnet dan tegangan yang terbaca jika magnet dimasukkan dan digerakkan atau didiamkan dalam kumparan.

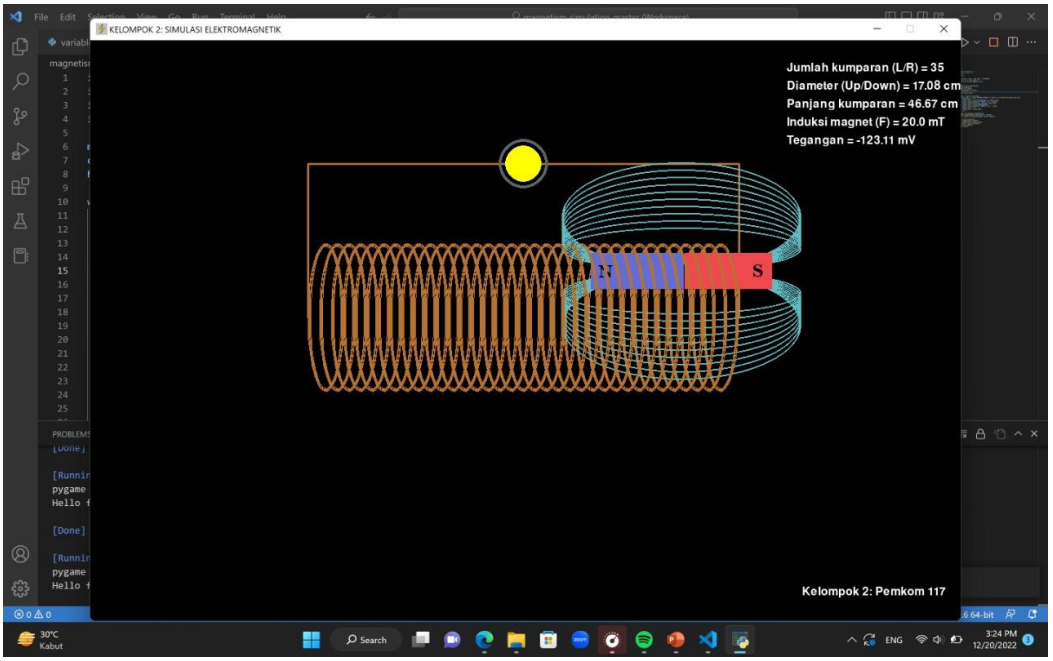
2. Percobaan Program



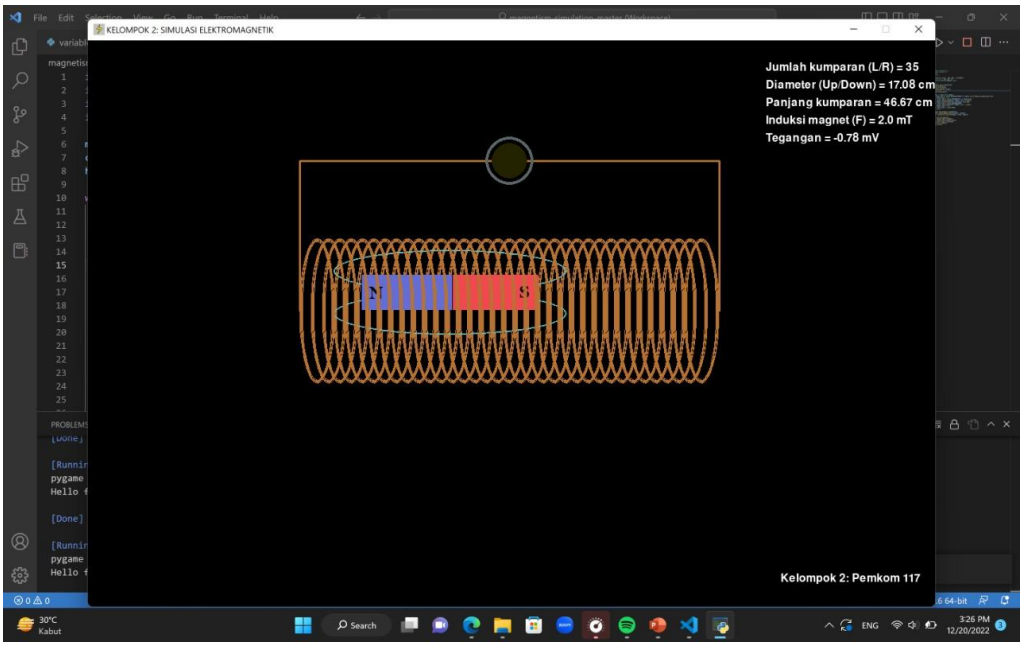
Dari gambar 2 terlihat jika jumlah kumparan bernilai 5 dan magnet digerakkan masuk kedalam kumparan, induksi magnet yang diatur = 20 mT, maka tegangan yang terbaca adalah -4,38 mV,dan lampu tidak menyala.



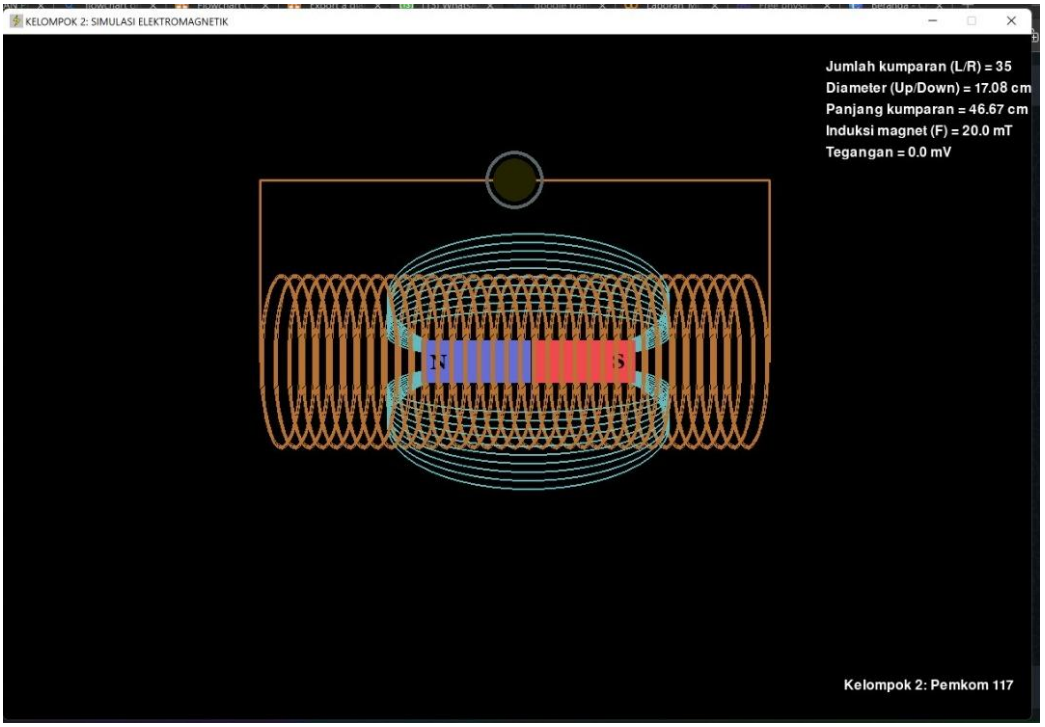
Dari gambar 3 terlihat jika jumlah kumparan bernilai 20 dan magnet digerakkan masuk kedalam kumparan, induksi magnet yang diatur = 20 mT, maka tegangan yang terbaca adalah -7,92 mV dan lampu menyala redup.



Dari gambar 4 terlihat jika jumlah kumparan bernilai 35 dan magnet digerakkan menjauhi kumparan, induksi magnet yang diatur = 20 mT, maka tegangan yang terbaca adalah -123,51 mV dan lampu menyala terang.



Dari gambar 5 terlihat jika jumlah kumparan bernilai 35 dan magnet digerakkan masuk ke dalam kumparan,, induksi magnet yang diatur = 2 mT, maka tegangan yang terbaca adalah -0,78 mV dan lampu tidak menyala.



Dari gambar 6 terlihat jika jumlah kumparan bernilai 35 dan magnet masuk ke dalam kumparan dan hanya diam, induksi magnet yang diatur = 20 mT, maka tegangan yang terbaca adalah -0mV dan lampu tidak menyala.

Dari simulasi yang telah dilakukan, dapat dianalisis bahwa banyak nya kumparan akan memengaruhi besar tegangan yang terbaca dan nyalanya lampu yang dirangkaian kumparan. Semakin banyak jumlah kumparan yang digunakan, maka tegangan akan semakin besar dan nyalanya lampu akan semakin terang. Selanjutnya medan magnet, dan pergerakan dari magnet memengaruhi tegangan. Apabila medan magnet yang digunakan semakin tinggi, dan magnet digerakkan masuk atau menjauhi kumpatran maka teganag yang terbaca akan semakin besar. Namun, jika kumparan yang digunakan banyak, dan medan magnet yang digunakan tinggi tetapi magnet hanya diam dikumparan, akan menghasilkan tegangan yang terbaca bernilai nol. Hal ini menyimpulkan jika pergerakan magnet juga memengaruhi tegangan yang terbaca

D. KESIMPULAN

1. Simulasi yang dilakukan kali ini adalah simulasi Induksi Elektromagnetik dengan Pygame
2. N (Kumparan) dan B (Induksi Magnetik) memengaruhi nilai ε
3. Nilai fluks (Φ) juga memengaruhi nilai ε
4. Berdasarkan rumus $\varepsilon = -N (\Delta\Phi/\Delta t)$, diketahui bahwa nilai semakin besar nilai N maka semakin kecil nilai ε, selain itu semakin besar nilai fluks (Φ) maka semakin besar pula nilai induksi elektromagnetik

E. DAFTAR PUSTAKA

Warjanto, S. (2015, October). Pengembangan Media Pembelajaran Induksi Elektromagnetik. In *Prosiding Seminar Nasional Fisika (E-Journal)* (Vol. 4, pp. SNF2015-II).

Trowbridge, L.W & Bybee. 1990.“Becoming A Secondary Schol Science Teache”.Merril Publishing Company, Ohio.