Tanggal Percobaan : 07 Desember 2022

Tanggal Pengumpulan : 24 Desember 2022

MAKALAH PROYEK PEMOGRAMAN KOMPUTER

"SIMULASI GERAK PARABOLA PERMAINAN KETAPEL DENGAN KONSEP GAME ANGRY BIRDS"

Dibuat Untuk Memenuhi Salah Satu Tugas Mata Kuliah **Pemograman Komputer**

Dosen Pengampu : Drs. Andreas Handjoko Permana, M.Si

Mencerdaskan &
Memartabatkan Bangsa

DiSusun Oleh :

Kelompok 10

Achmad Nurnaafi (1306621057)

Haryanto (1306621059)

Yohanes Radito Putra (1306621048)

Program Studi Fisika

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Negeri Jakarta

2022

BAB I
PENDAHULUAN

1.1. Latar Belakang

Fisika dapat diajarkan tentang gerak parabola dalam beberapa cara, salah satunya adalah dengan menggunakan game sebagai media pembelajaran. Ketika siswa bermain, mereka dengan mudah memahami cara kerja permainan dan cara memainkannya. Siswa menyukai game tersebut karena game 2 memiliki tampilan yang menarik dibandingkan dengan buku pelajaran fisika. Oleh karena itu, penelitian ini menggunakan game Angry Birds untuk pembelajaran fisika.

Dalam game Angry Birds dimainkan dengan cara melempar burung ke udara. Semua gerakan burung yang dilempar ke udara membentuk gerakan parabola. Ini dapat digunakan sebagai alat untuk mempelajari topik gerak parabola. Sebuah penelitian tentang penggunaan game Angry Bird dalam pembelajaran pernah dilakukan oleh seorang guru IPA bernama John Burk di Westminster Schools di Atlanta, USA. Namun, penelitian ini belum pernah dilakukan di Indonesia.

Gerak parabola merupakan perpaduan antara gerak lurus beraturan (GLB) di arah sumbu-x dan gerak lurus berubah beraturan (GLBB) di arah sumbu-y. Artinya, kecepatan benda pada sumbu-x akan selalu tetap, baik besar maupun arahnya. Sementara itu, kecepatan benda pada sumbu-y akan mengalami GLBB diperlambat akibat pengaruh percepatan gravitasi. Nah, pengaruh gravitasi inilah yang menyebabkan gerak bendanya melengkung sehingga disebut gerak parabola.

1.2. Rumusan Masalah

1. Apa yang dimaksud dengan gerak parabola?
2. Gaya apa saja yang bekerja pada gerak parabola?
3. Bagaimana konsep dari pygame?
4. Jelaskan konsep simulasi projek pygame yang akan dibuat?
5. Bagaimana langkah-langkah pembuatan proyek pygame tersebut?
6. Bagaimana screen code, tampilan simulasi dari projek pygame tersebut?

1.3. Tujuan

1. Memahami konsep dari gerak parabola
2. Mempelajari salah satu modul python dalam mengembangkan game
3. Mengetahui konsep atau rencana dari simulasi game tersebut.
4. Mengetahui langkah-langkah pembuatan game.

BAB II
KAJIAN PUSTAKA

2.1. Gerak Parabola

Gerak Parabola merupakan gerak dua dimensi dari partikel yang dilemparkan miring ke udara dengan menganggap bahwa pengaruh gesekan udara terhadap gerak ini dapat di abaikan. Gerak parabola adalah gabungan antara GLB dengan GLBB. Gerak benda pada sumbu X adalah GLB dan Pada Sumbu Y adalah GLBB.

Sumbu X : GLB yakni gerak benda pada arah mendataar yang tidak dipengaruhi oleh gaya gravitasi, sehingga tidak ada percepatan atau perlambatan pada daerah ini.

Sumbu Y : GLBB pada arah vertical, yaitu gerak benda pada arah vertical yang dipengaruhi oleh gravitasi, sehingga ada percepatan arah ini. **(Rismalasari, 2013)**

2.2. Hukum Hooke

Hukum Hooke adalah hukum atau ketentuan mengenai gaya dalam ilmu fisika yang terjadi karena sifat elastisitas dari sebuah pegas.Ukuran elastisitas sebuah pegas berbeda-beda sesuai dengan ukuran kekuatan pegas tersebut.Ukuran kekuatan sebuah pegas disebut modulus elastis yang dikenal sebagai konstanta pegas (k). **(Elisa & Claudya, 2016)**

Hooke menemukan bahwa pertambahan panjang pegas yang timbul berbanding lurus dengan gaya yang diberikan. Lebih jauh lagi, Hooke juga menemukan bahwa pertambahan panjang pegas sangat bergantung pada karakteristik dari pegas tersebut. Pegas yang mudah teregang seperti karet gelang akan mengalami pertambahan panjang yang besar meskipun gaya yang diberikan kecil. Sebaliknya pegas yang sangat sulit teregang seperti pegas baja akan mengalami pertambahan panjang yang sedikit atau kecil meskipun diberi gaya yang besar. Karakteristik yang dimiliki masing-masing pegas ini dinyatakan sebagai tetapan gaya dari pegas tersebut. Pegas yang mudah teregang seperti karet gelang memiliki tetapan gaya yang kecil. Sebaliknya pegas yang sulit teregang seperti pegas baja memiliki tetapan gaya yang besar. Secara umum apa yang ditemukan Hooke bisa dinyatakan sebagai berikut:

$$F = k.\ x$$

2.3. Pygame

PyGame merupakan salah satu modul python. Pygame berfungsi untuk membangun sebuah game dari python. Didalam pygame terdapat beberapa fungsi yang bisa digunakan dalam pembuatan sebuah game, sepert pemutar musik dan lain sebagainya. Pygame bisa juga dikatakan library yang open source untuk membuat aplikasi yang berbasis multimedia dengan menggunakan Bahasa pemrograman python. Pygame adalah seperangkat modul Python yang dirancang untuk membuat permainan. Pygame menambahkan fungsi di atas dengan sangat baik di SDL perpustakaan. Hal ini memungkinkan Anda untuk membuat sebuah game dengan fitur yang lengkap dan sebuah program multimedia dalam bahasa python. Pygame sangat portable dan dapat berjalan pada hampir semua platform dan sistem operasi. **(Satria, 2018)**

2.4. Rencana Simulasi

Project Pygame Simulasi Fisika yang kelompok kami usung adalah permainan ketapel dengan konsep game angry birds, dimana pada game ini menggunakan hukum fisika gerak parabola dan gerak elastisitas hukum hooke. Tujuan dari game ini dimana ada sebuah ketapel yang pelurunya karakter angry bird harus mengenai target seekor pigs yang setiap tingkatan levelnya rintangan untuk mengenai target semakin sulit.

BAB III
LANGKAH-LANGKAH PEMOGRAMAN

1. Problem Statement

Membuat Program Pygame Simulasi Gerak Parabola Permainan Ketapel dengan Konsep Game Angry Birds

2. Mathematical Equation

- Import pygame
- Import random
- From math import
- Kecepatan arah vertical

$$V_{yt} = V_0 \ \sin a - gt$$

- Waktu mencapai titik tertinggi

$$t_H = \frac{v_0 \sin a}{g}$$

- Waktu untuk mencapai titik terjauh

$$t_R = \frac{2v_0 \sin a}{g}$$

- Jarak yang ditempuh arah mendatar

$$X = V_0 \cos a \, t$$

- Jarak benda menyentuk tanah

$$X_R = \frac{v_{0^2} \sin 2\theta}{g}$$

3. Algoritma

Interface

1. Import pygame
2. Import sys
3. Import Pygame.init()
4. Memproses display = None
5. Mendefinisikan init(Screen):

   5.1.Global display

```
    5.2.Menginisiasi display = screen
6.  Menghimpun data class Button :
    6.1.Mendefinisikan _init_
        6.1.1.  Menginisiasi self.x = x
        6.1.2.  Menginisiasi self.y = y
        6.1.3.  Menginisiasi self.w = w
        6.1.4.  Menginisiasi self.h = h
        6.1.5.  Menginisiasi self.colorActive = colorActive
        6.1.6.  Menginisiasi self.colorNotActive = colorNotActive
        6.1.7.  Menginisiasi self.action = action
        6.1.8.  Menginisiasi self.font = None
        6.1.9.  Menginisiasi self.text = None
        6.1.10. Menginisiasi.text_pos = None
    6.2.Mendefinisikan add_text
        6.2.1.  Menginisiasi self.font = pygame.font.Font(font, size)
        6.2.2.  Menginisiasi.self.text = self.font.render(text, True, text_color)
        6.2.3.  Menginisiasi.self.text_pos = self.text.get_rect()
        6.2.4.  Menginisiasi self.text_pos.center = (self.x + self.w/2,self.y + self.h/2)
    6.3.Mendefinisikan draw (self):
        6.3.1.  Jika, self.isActive():
            6.3.1.1.Jika not self.colorActive == None:
                6.3.1.1.1.  Pygame.draw.rect(display,self.colorActive,(self.x,self.y,self.w,self.h))
        6.3.2.  Else:
            6.3.2.1.Pygame.draw.rect(display,self.colorActive,(self.x,self.y,self.w,self.h))
        6.3.3.  Jika, self.text:
            6.3.3.1.Display.blit(self.text, self.text_pos)
    6.4.Mendefinisikan isActive(self):
        6.4.1.  Menginisiasi pos = pygame.mouse.get_pos()
        6.4.2.  Jika, (self.x < pos[0] < self.x + self.w) and (self.y < pos[1] < self.y + self.h):
            6.4.2.1.Return True
        6.4.3.   Else :
            6.4.3.1.Return False
7.   Menghimpun data class label(Button):
    7.1.Mendefinisikan draw(self):
        7.1.1.   Jika, self.text:
            7.1.1.1.Display.blit(self.text, self.text_pos)
```

Objects

```
1.   Import pygame
2.   Import sys
3.   From math import
4.   Import physics_engine
5.   Pygame.init()
6.   Mengatur display = None
7.   Mengatur height = None
8.   Mengatur clock = pygame.time.clock()
9.   Mengatur ground = 50
10. Mendefinisikan init(screen):
    10.1.Global width, height, display
    10.2.Memproses display = screen
    10.3.Menginisiasi (width, height) = display.get_rect().size
    10.4.Memproses height -= ground
11. Menghimpun data class Slab:
    11.1.Mendefinisikan _init_
        11.1.1. Menginisiasi self.x = x
        11.1.2. Menginisiasi self.y = y
        11.1.3. Menginisiasi.self.w = w
        11.1.4. Menginisiasi self.h = h
        11.1.5. Jika, self.w > self.h:
            11.1.5.1.         self.image = pygame.image.load("Images/wall_horizontal.png")
        11.1.6. Else:
```

```
      11.1.6.1.    self.image = pygame.image.load("Images/wall_vertical.png")
    11.1.7. Self.image = pygame.transform.scale(self.image,(self.w,self.h))
    11.1.8. Self.color = color
11.2.Mendefinisikan draw(self):
    11.2.1. display.blit(self.image, (self.x, self.y))
11.3.Mendefinisikan collision_manager
    11.3.1. Jika, type == "BALL":
        11.3.1.1.    Jika, (ball.y + ball.r > self.y) and (ball.y < self.y + self.h):
            11.3.1.1.1. Jika, (ball.x < self.x + self.w) and (ball.x + ball.r > self.x + self.w):
                11.3.1.1.1.1.   Memproses ball.x = 2*(self.x + self.w) - ball.x
                11.3.1.1.1.2.   Memproses ball.velocity.angle = - ball.velocity.angle
                11.3.1.1.1.3.  ball.velocity.magnitude *=physics_engine.elasticity
            11.3.1.1.2. elif ball.x + ball.r > self.x and (ball.x < self.x):
                11.3.1.1.2.1.   memproses ball.x = 2*(self.x - ball.r) - ball.x
                11.3.1.1.2.2.   memproses ball.velocity.angle = - ball.velocity.angle
                11.3.1.1.2.3.   ball.velocity.magnitude *= physics_engine.elasticity
        11.3.1.2.    Jika, (ball.x + ball.r > self.x) and (ball.x < self.x + self.w):
            11.3.1.2.1. Jika, ball.y + ball.r > self.y and ball.y < self.y:
                11.3.1.2.1.1.   Memproses ball.y = 2*(self.y - ball.r) - ball.y
                11.3.1.2.1.2.   Memproses ball.velocity.angle = pi - ball.velocity.angle
                11.3.1.2.1.3.   ball.velocity.magnitude *= physics_engine.elasticity
            11.3.1.2.2. elif (ball.y < self.y + self.h) and (ball.y + ball.r > self.y + self.h):
                11.3.1.2.2.1.   Memproses ball.y = 2*(self.y + self.h) - ball.y
                11.3.1.2.2.2.   Memproses ball.velocity.angle = pi - ball.velocity.angle
                11.3.1.2.2.3.   ball.velocity.magnitude *= physics_engine.elasticity
        11.3.1.3.    return ball
    11.3.2. else:
        11.3.2.1.    menginisiasi block = ball
        11.3.2.2.    jika, (block.y + block.h > self.y) and (block.y < self.y + self.h):
            11.3.2.2.1. jika, (block.x < self.x + self.w) and (block.x + block.w > self.x +
                        self.w):
```

```
            11.3.2.2.1.1.   memproses block.x = 2*(self.x + self.w) - block.x
            11.3.2.2.1.2.   memproses block.velocity.angle = - block.velocity.angle
            11.3.2.2.1.3.   memproses block.rotateAngle =  - block.velocity.angle
            11.3.2.2.1.4.   block.velocity.magnitude *= physics_engine.elasticity
        11.3.2.2.2. elif block.x + block.w > self.x and (block.x < self.x):
            11.3.2.2.2.1.   memproses block.x = 2*(self.x - block.w) - block.x
            11.3.2.2.2.2.   memproses block.velocity.angle = - block.velocity.angle
            11.3.2.2.2.3.   memproses block.rotateAngle =  - block.velocity.angle
            11.3.2.2.2.4.   block.velocity.magnitude *= physics_engine.elasticity
    11.3.2.3.       jika (block.x + block.w > self.x) and (block.x < self.x + self.w):
        11.3.2.3.1. jika, block.y + block.h > self.y and block.y < self.y:
            11.3.2.3.1.1.   memproses block.y = 2*(self.y - block.h) - block.y
            11.3.2.3.1.2.   memproses block.velocity.angle = pi - block.velocity.angle
            11.3.2.3.1.3.   memproses block.rotateAngle =  pi - block.velocity.angle
            11.3.2.3.1.4.   block.velocity.magnitude *= physics_engine.elasticity
        11.3.2.3.2. elif (block.y < self.y + self.h) and (block.y + block.h > self.y +
                    self.h):
            11.3.2.3.2.1.   memproses block.y = 2*(self.y + self.h) - block.y
            11.3.2.3.2.2.   memproses block.velocity.angle = pi - block.velocity.angle
            11.3.2.3.2.3.   memproses block.rotateAngle =  pi - block.velocity.angle
            11.3.2.3.2.4.   block.velocity.magnitude *= physics_engine.elasticity
    11.3.2.4.       return block
```

Maps

```
1. import pygame
2. import sys
3. import physics_engine
4. import objects
5. import interface
6. pygame.init()
7. mengatur width = None
```

```
8.  mengatur height = None
9.  mengatur display = None
10. menginisiasi clock = pygame.time.clock()
11. mengatur ground = 50
12. mengatur d_velocity = 2.0
13. mendefinisikan init(screen):
    13.1.global width, height, display
    13.2.menginisiasi display = screen
    13.3.menginisiasi (width, height) = display.get_rect().size
    13.4.memproses height -= ground
    13.5.interface.init(display)
14. mendefinisikan all_rest(pigs, birds, blocks):
    14.1.memproses threshold = 0.15
    14.2.for pig in pigs:
        14.2.1. if pig.velocity.magnitude >= threshold:
            14.2.1.1.    return False
    14.3.for bird in birds:
        14.3.1. if bird.velocity.magnitude >= threshold:
            14.3.1.1.    return False
    14.4.for block in blocks:
        14.4.1. if block.velocity.magnitude >= threshold:
            14.4.1.1.    return False
    14.5.return True
15. mendefinisikan close():
    15.1.pygame.quit()
    15.2.sys.exit()
16. menghimpun data class Maps:
    16.1.mendefinisikan_init_(self):
        16.1.1. memproses self.level = 1
        16.1.2. memproses self.max_level = 3
        16.1.3. memproses self.color = {'background': (51, 51, 51)}
        16.1.4. memproses self.score = 0
    16.2.mendefinisikan wait_level(self):
        16.2.1. memproses time = 0
        16.2.2. while time < 3:
            16.2.2.1.     for event in pygame.event.get():
                16.2.2.1.1. jika, event.type == pygame.QUIT:
                    16.2.2.1.1.1.   close()
                16.2.2.1.2. jika, event.type == pygame.KEYDOWN:
                    16.2.2.1.2.1.   if event.key == pygame.K_q:
                        16.2.2.1.2.1.1. close()
            16.2.2.2.     time += 1
            16.2.2.3.     clock.tick(1)
    16.3.mendefinisikan check_win(self, pigs, birds):
        16.3.1. jika, pigs == []:
            16.3.1.1.     mencetak ("WON!")
            16.3.1.2.     return True
        16.3.2. jika, (not pigs == []) and birds == []:
            16.3.2.1.     mencetak ("LOST!")
            16.3.2.2.     return False
    16.4.mendefinisikan draw_map(self):
        16.4.1. membuat list birds = []
        16.4.2. membuat list pigs = []
        16.4.3. membuat list blocks = []
        16.4.4. membuat list walls = []
        16.4.5. memproses self.score = 0
        16.4.6. jika, self.level == 1:
            16.4.6.1.     for i in range(4):
                16.4.6.1.1. new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None,
                            "BIRD")
                16.4.6.1.2. birds.append(new_bird)
            16.4.6.2.     pigs.append(physics_engine.Pig(950, height - 60, 25))
        16.4.6.3.     pigs.append(physics_engine.Pig(1055, 350 - 60, 25))
        16.4.6.4.     pigs.append(physics_engine.Pig(1100, height - 60, 25))
        16.4.6.5.     blocks.append(physics_engine.Block(900, height - 100, 100))
        16.4.6.6.     blocks.append(physics_engine.Block(900, 350 - 2*60, 100))
        16.4.6.7.     walls.append(objects.Slab(850, 350, 400, 20))
        16.4.6.8.     walls.append(objects.Slab(1030, 450, 20, height - 450))
      16.4.7. elif self.level == 2:
        16.4.7.1.     for i in range(4):
          16.4.7.1.1. new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
          16.4.7.1.2. birds.append(new_bird)
        16.4.7.2.     pigs.append(physics_engine.Pig(1000, height - 60, 25))
        16.4.7.3.     pigs.append(physics_engine.Pig(1000, 400 - 60, 25))
        16.4.7.4.     pigs.append(physics_engine.Pig(1150, height - 60, 25))
        16.4.7.5.     blocks.append(physics_engine.Block(900, height - 100, 100))
        16.4.7.6.     blocks.append(physics_engine.Block(1100, 400 - 100, 100))
        16.4.7.7.     walls.append(objects.Slab(810, 400, 450, 20))
        16.4.7.8.     walls.append(objects.Slab(800, 100, 20, 320))
      16.4.8. elif self.level == 3:
        16.4.8.1.     for i in range(5):
          16.4.8.1.1. new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
          16.4.8.1.2. birds.append(new_bird)
        16.4.8.2.     pigs.append(physics_engine.Pig(900, height - 60, 25))
        16.4.8.3.     pigs.append(physics_engine.Pig(width - 400, 400 - 60, 25))
        16.4.8.4.     pigs.append(physics_engine.Pig(1150, height - 60, 25))
        16.4.8.5.     walls.append(objects.Slab(800, 400, 20, height - 400))
        16.4.8.6.     walls.append(objects.Slab(1050, 550, 30, height - 550))
        16.4.8.7.     walls.append(objects.Slab(width - 500, 400, 400, 30))
        16.4.8.8.     walls.append(objects.Slab(width - 500, 150, 30, 400 - 150))
    16.5.mendefinisikan replay_level(self):
      16.5.1. memproses self.level -= 1
      16.5.2. self.draw_map()
    16.6.mendefinisikan start_again(self):
      16.6.1. memproses self.level = 1
      16.6.2. self.draw_map()
    16.7.mendefinisikan level_cleared(self):
      16.7.1. self.level += 1
      16.7.2. level_cleared_text = interface.Label(480, 100, 400, 200, None, self.color['background'])
      16.7.3. jika, self.level <= self.max_level:
        16.7.3.1. level_cleared_text.add_text("LEVEL " + str(self.level - 1) + " CLEARED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
      16.7.4. else:
        16.7.4.1. level_cleared_text.add_text("ALL LEVEL CLEARED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
      16.7.5. score_text=interface.Label(500, 300, 300, 100, None, self.color['background'])
      16.7.6. score_text.add_text("SCORE: " + str(self.score), 55, "Fonts/Comic_Kings.ttf", (236, 240, 241))
      16.7.7. replay = interface.Button(100, 500, 300, 100, self.replay_level, (244, 208, 63), (247, 220, 111))
      16.7.8. replay.add_text("PLAY AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
      16.7.9. jika, self.level <= self.max_level:
        16.7.9.1. next = interface.Button(500, 500, 300, 100, self.draw_map, (88, 214, 141), (171, 235, 198))
        16.7.9.2. next.add_text("CONTINUE", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
      16.7.10. else:
        16.7.10.1. next = interface.Button(500, 500, 300, 100, self.start_again, (88, 214, 141), (171, 235, 198))
        16.7.10.2.   next.add_text("START AGAIN",60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
    16.7.11.   exit = interface.Button(900, 500, 300, 100, close, (241, 148, 138), (245, 183, 177))
    16.7.12.   exit.add_text("QUIT",60,"Fonts/arfmoochikncheez.ttf",self.color['background'])
    16.7.13.   while True:
        16.7.13.1.   for event in pygame.event.get():
            16.7.13.1.1.   jika, event.type == pygame.QUIT:
                16.7.13.1.1.1. close()
            16.7.13.1.2.   jika, event.type == pygame.KEYDOWN:
                16.7.13.1.2.1. if event.key == pygame.K_q:
                    16.7.13.1.2.1.1.   close()
            16.7.13.1.3.   jika, event.type == pygame.MOUSEBUTTONDOWN:
                16.7.13.1.3.1. jika, replay.isActive():
                    16.7.13.1.3.1.1.   replay.action()
                16.7.13.1.3.2. jika, next.isActive():
                    16.7.13.1.3.2.1.   next.action()
                16.7.13.1.3.3. jika, exit.isActive():
                    16.7.13.1.3.3.1.   exit.action()
        16.7.13.2.   replay.draw()
        16.7.13.3.   next.draw()
        16.7.13.4.   exit.draw()
        16.7.13.5.   level_cleared_text.draw()
        16.7.13.6.   score_text.draw()
        16.7.13.7.   pygame.display.update()
        16.7.13.8.   clock.tick(60)
16.8.mendefinisikan level_failed(self):
    16.8.1. level_failed_text = interface.Label(450, 100, 400, 200, None, self.color['background'])
```

```
16.8.2. level_failed_text.add_text("LEVEL FAILED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
16.8.3. score_text=interface.Label(500, 300, 300, 100, None, self.color['background'])
16.8.4. score_text.add_text("SCORE: " + str(self.score), 55, "Fonts/Comic_Kings.ttf", (236, 240, 241))
16.8.5. replay = interface.Button(200, 500, 300, 100, self.draw_map, (244, 208, 63), (247, 220, 111))
16.8.6. replay.add_text("TRY AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
16.8.7. exit = interface.Button(800, 500, 300, 100, close, (241, 148, 138), (245, 183, 177))
16.8.8. exit.add_text("QUIT",60,"Fonts/arfmoochikncheez.ttf",self.color['background'])
16.8.9. while True:
    16.8.9.1.    for event in pygame.event.get():
        16.8.9.1.1. jika, event.type == pygame.QUIT:
            16.8.9.1.1.1.   close()
        16.8.9.1.2. jika, event.type == pygame.KEYDOWN:
            16.8.9.1.2.1.   if event.key == pygame.K_q:
                16.8.9.1.2.1.1. close()
        16.8.9.1.3. jika, event.type == pygame.MOUSEBUTTONDOWN:
            16.8.9.1.3.1.   jika, replay.isActive():
                16.8.9.1.3.1.1. replay.action()
            16.8.9.1.3.2.   jika, next.isActive():
                16.8.9.1.3.2.1. next.action()
            16.8.9.1.3.3.   jika, exit.isActive():
                16.8.9.1.3.3.1. exit.action()
16.8.10.   replay.draw()
16.8.11.   exit.draw()
16.8.12.   level_failed_text.draw()
16.8.13.   score_text.draw()
```

```
    16.8.14. pygame.display.update()
    16.8.15. clock.tick(60)
16.9.mendefinisikan start_level(self, birds, pigs, blocks, walls):
    16.9.1. loop = True
    16.9.2. slingshot = physics_engine.Slingshot(200, height - 200, 30, 200)
    16.9.3. birds[0].load(slingshot)
    16.9.4. mouse_click = False
    16.9.5. flag = 1
    16.9.6. pigs_to_remove = []
    16.9.7. blocks_to_remove = []
    16.9.8. score_text = interface.Label(50, 10, 100, 50, None, self.color['background'])
    16.9.9. score_text.add_text("SCORE: " + str(self.score), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
    16.9.10. birds_remaining = interface.Label(120, 50, 100, 50, None, self.color['background'])
    16.9.11. birds_remaining.add_text("BIRDS REMAINING: " + str(len(birds)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
    16.9.12. pigs_remaining = interface.Label(110, 90, 100, 50, None, self.color['background'])
    16.9.13. pigs_remaining.add_text("PIGS REMAINING: " + str(len(pigs)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
    16.9.14. while loop:
        16.9.14.1. for event in pygame.event.get():
            16.9.14.1.1. jika, event.type == pygame.QUIT:
                16.9.14.1.1.1. close()
            16.9.14.1.2. jika, event.type == pygame.KEYDOWN:
                16.9.14.1.2.1. jika, event.key == pygame.K_q:
                    16.9.14.1.2.1.1. close()
                16.9.14.1.2.2. jika, event.key == pygame.K_r:
                    16.9.14.1.2.2.1. self.draw_map()
                16.9.14.1.2.3. jika, event.key == pygame.K_p:
```

```
            16.9.14.1.2.3.1.    self.pause()
        16.9.14.1.2.4. jika, event.key == pygame.K_ESCAPE:
            16.9.14.1.2.4.1.    self.pause()
    16.9.14.1.3.        jika, event.type == pygame.MOUSEBUTTONDOWN:
        16.9.14.1.3.1. jika, birds[0].mouse_selected():
            16.9.14.1.3.1.1.    mouse_click = True
    16.9.14.1.4.        jika, event.type == pygame.MOUSEBUTTONUP:
        16.9.14.1.4.1. mouse_click = False
        16.9.14.1.4.2. jika, birds[0].mouse_selected():
        16.9.14.1.4.3. flag = 0
16.9.14.2.      jika, (not birds[0].loaded) and all_rest(pigs, birds, blocks):
    16.9.14.2.1.        mencetak ("LOADED!")
    16.9.14.2.2.        birds.pop(0)
    16.9.14.2.3.        jika, self.check_win(pigs, birds) == 1:
        16.9.14.2.3.1. self.score += len(birds)*100
        16.9.14.2.3.2. self.level_cleared()
    16.9.14.2.4.        elif self.check_win(pigs,birds) == 0:
        16.9.14.2.4.1. self.level_failed()
    16.9.14.2.5.        jika, not birds == []:
        16.9.14.2.5.1. birds[0].load(slingshot)
    16.9.14.2.6.        flag = 1
16.9.14.3.      jika, mouse_click:
    16.9.14.3.1.        birds[0].reposition(slingshot, mouse_click)
16.9.14.4.      jika, not flag:
    16.9.14.4.1.        birds[0].unload()
16.9.14.5.      color = self.color['background']
16.9.14.6.      for i in range(3):
    16.9.14.6.1.        color = (color[0] + 5, color[1] + 5, color[2] + 5)
    16.9.14.6.2.        pygame.draw.rect(display, color, (0, i*300, width, 300))
16.9.14.7.      pygame.draw.rect(display, (77, 86, 86), (0, height, width, 50))
16.9.14.8.      slingshot.draw(birds[0])
```

```
16.9.14.9. for i in range(len(pigs)):
    16.9.14.9.1. for j in range(len(blocks)):
        16.9.14.9.1.1. pig_v,block_v=pigs[i].velocity.magnitude,blocks[j].velocity.magnitude
        16.9.14.9.1.2. pigs[i],blocks[j],result_block_pig=physics_engine.collision_handler(pigs[i], blocks[j], "BALL_N_BLOCK")
        16.9.14.9.1.3. pig_v1,block_v1=pigs[i].velocity.magnitude,blocks[j].velocity.magnitude
        16.9.14.9.1.4. jika, result_block_pig:
            16.9.14.9.1.4.1. jika, abs(pig_v - pig_v1) > d_velocity:
                16.9.14.9.1.4.1.1. blocks_to_remove.append(blocks[j])
                16.9.14.9.1.4.1.2. blocks[j].destroy()
            16.9.14.9.1.4.2. jika, abs(block_v - block_v1) > d_velocity:
                16.9.14.9.1.4.2.1. pigs_to_remove.append(pigs[i])
                16.9.14.9.1.4.2.2. pigs[i].dead()
16.9.14.10. for i in range(len(birds)):
    16.9.14.10.1. jika, not (birds[i].loaded or birds[i].velocity.magnitude == 0):
        16.9.14.10.1.1. for j in range(len(blocks)):
            16.9.14.10.1.1.1. birds_v, block_v = birds[i].velocity.magnitude, blocks[j].velocity.magnitude
            16.9.14.10.1.1.2. birds[i], blocks[j], result_bird_block = physics_engine.collision_handler(birds[i], blocks[j], "BALL_N_BLOCK")
            16.9.14.10.1.1.3. birds_v1, block_v1 = birds[i].velocity.magnitude, blocks[j].velocity.magnitude
            16.9.14.10.1.1.4. jika, result_bird_block:
                16.9.14.10.1.1.4.1. jika, abs(birds_v - birds_v1) > d_velocity:
                    16.9.14.10.1.1.4.1.1. jika, not blocks[j]in blocks_to_remove:
```

```
                    16.9.14.10.1.1.4.1.1.1.  blocks_to_remove.append(blocks[j])
                    16.9.14.10.1.1.4.1.1.2.  blocks[j].destroy()
16.9.14.11.    for i in range(len(pigs)):
    16.9.14.11.1.       pigs[i].move()
    16.9.14.11.2.       for j in range(i+1, len(pigs)):
        16.9.14.11.2.1.         pig1_v,pig2_v=pigs[i].velocity.magnitude,pigs[j].velocity.magnitude
        16.9.14.11.2.2.         pigs[i],pigs[j],result=physics_engine.collision_handler(pigs[i], pigs[j], "BALL")
        16.9.14.11.2.3.         pig1_v1,pig2_v1=pigs[i].velocity.magnitude,pigs[j].velocity.magnitude
        16.9.14.11.2.4.         result = True
        16.9.14.11.2.5.         jika, result:
            16.9.14.11.2.5.1.  jika, abs(pig1_v - pig1_v1) > d_velocity:
                16.9.14.11.2.5.1.1.    jika, not pigs[j] in pigs_to_remove:
                    16.9.14.11.2.5.1.1.1.       pigs_to_remove.append(pigs[j])
                    16.9.14.11.2.5.1.1.2.       pigs[j].dead()
            16.9.14.11.2.5.2.  jika, abs(pig2_v - pig2_v1) > d_velocity:
                16.9.14.11.2.5.2.1.    jika, not pigs[i] in pigs_to_remove:
                    16.9.14.11.2.5.2.1.1.       pigs_to_remove.append(pigs[i])
                    16.9.14.11.2.5.2.1.2.       pigs[i].dead()
    16.9.14.11.3.       for wall in walls:
        16.9.14.11.3.1.         pigs[i] = wall.collision_manager(pigs[i])
    16.9.14.11.4.       pigs[i].draw()
16.9.14.12.    for i in range(len(birds)):
    16.9.14.12.1.       jika, (not birds[i].loaded) and birds[i].velocity.magnitude:
        16.9.14.12.1.1.         birds[0].move()
        16.9.14.12.1.2.         for j in range(len(pigs)):
            16.9.14.12.1.2.1.  bird_v,pig_v=birds[i].velocity.magnitude,pigs[j].velocity.magnitude
            16.9.14.12.1.2.2. birds[i],pigs[j],result_bird_pig=physics_engine.collision_handler(birds[i], pigs[j], "BALL")
            16.9.14.12.1.2.3. bird_v1,pig_v1=birds[i].velocity.magnitude,pigs[j].velocity.magnitude
            16.9.14.12.1.2.4. result = True
            16.9.14.12.1.2.5. jika, result_bird_pig:
                16.9.14.12.1.2.5.1. jika, abs(bird_v - bird_v1) > d_velocity:
                    16.9.14.12.1.2.5.1.1. jika, not pigs[j] in pigs_to_remove:
                        16.9.14.12.1.2.5.1.1.1. pigs_to_remove.append(pigs[j])
                        16.9.14.12.1.2.5.1.1.2. pigs[j].dead()
    16.9.14.12.2. jika, birds[i].loaded:
        16.9.14.12.2.1. birds[i].project_path()
    16.9.14.12.3. for wall in walls:
        16.9.14.12.3.1. birds[i] = wall.collision_manager(birds[i])
    16.9.14.12.4. birds[i].draw()
16.9.14.13. for i in range(len(blocks)):
    16.9.14.13.1. for j in range(i + 1, len(blocks)):
        16.9.14.13.1.1. block1_v, block2_v = blocks[i].velocity.magnitude, blocks[j].velocity.magnitude
        16.9.14.13.1.2. blocks[i],blocks[j],result_block=physics_engine.block_collision_handler(blocks[i], blocks[j])
        16.9.14.13.1.3. block1_v1,block2_v1=blocks[i].velocity.magnitude, blocks[j].velocity.magnitude
        16.9.14.13.1.4. jika, result_block:
            16.9.14.13.1.4.1. jika, abs(block1_v - block1_v1) > d_velocity:
                16.9.14.13.1.4.1.1. jika, not blocks[j] in blocks_to_remove:
                    16.9.14.13.1.4.1.1.1. blocks_to_remove.append(blocks[j])
                    16.9.14.13.1.4.1.1.2. blocks[j].destroy()
            16.9.14.13.1.4.2. jika, abs(block2_v - block2_v1) > d_velocity:
                16.9.14.13.1.4.2.1. jika, not blocks[i] in blocks_to_remove:
                    16.9.14.13.1.4.2.1.1. blocks_to_remove.append(blocks[i])
```

```
          16.9.14.13.1.4.2.1.2. blocks[i].destroy()
  16.9.14.13.2. blocks[i].move()
  16.9.14.13.3. for wall in walls:
    16.9.14.13.3.1. blocks[i] = wall.collision_manager(blocks[i], "BLOCK")
  16.9.14.13.4. blocks[i].draw()
16.9.14.14. for wall in walls:
  16.9.14.14.1. wall.draw()
16.9.14.15. score_text.add_text("SCORE:"+str(self.score),25,"Fonts/Comic_Kings.ttf", (236, 240, 241))
16.9.14.16. score_text.draw()
16.9.14.17. birds_remaining.add_text("BIRDS REMAINING:"+str(len(birds)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
16.9.14.18. birds_remaining.draw()
16.9.14.19. pigs_remaining.add_text("PIGS REMAINING: " + str(len(pigs)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
16.9.14.20. pigs_remaining.draw()
16.9.14.21. pygame.display.update()
16.9.14.22. jika, all_rest(pigs, birds, blocks):
  16.9.14.22.1. for pig in pigs_to_remove:
    16.9.14.22.1.1. jika, pig in pigs:
      16.9.14.22.1.1.1. pigs.remove(pig)
      16.9.14.22.1.1.2. self.score += 100
  16.9.14.22.2. for block in blocks_to_remove:
    16.9.14.22.2.1. jika, block in blocks:
      16.9.14.22.2.1.1. blocks.remove(block)
      16.9.14.22.2.1.2. self.score += 50
  16.9.14.22.3. pigs_to_remove = []
  16.9.14.22.4. blocks_to_remove = []
16.9.14.23. clock.tick(60)
```

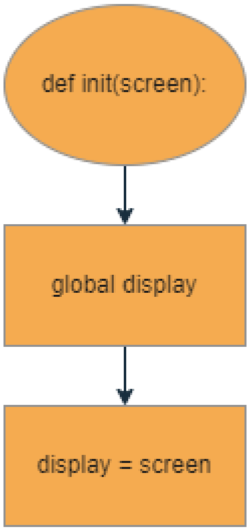
Program Utama

```
1. import pygame, import sys, import random, from math import, import physics_engine, import objects, import maps, import interface
2. pygame.init()
3. mengatur width = 1300
4. mengatur height = 700
5. menginisiasi display = pygame.display.set_mode((width, height))
6. menginisiasi clock = pygame.time.Clock()
7. physics_engine.init(display)
8. objects.init(display)
9. maps.init(display)
10. interface.init(display)
11. pygame.display.set_caption("Angry Birds --- Oleh Kelompok 10")
12. mengatur background = (51, 51, 51)
13. mendefinisikan close():
    13.1.pygame.quit()
    13.2.sys.exit()
14. mendefinisikan start_game(map):
    14.1.map.draw_map()
15. mendefinisikan GAME():
    15.1.map = maps.Maps()
    15.2.welcome = interface.Label(450, 100, 400, 200, None, background)
    15.3.welcome.add_text("ANGRY BIRDS", 80, "Fonts/arfmoochikncheez.ttf", (236, 240, 241))
    15.4.start = interface.Button(200, 400, 300, 100, start_game, (244, 208, 63), (247, 220, 111))
    15.5.start.add_text("START GAME", 60, "Fonts/arfmoochikncheez.ttf", background)
    15.6.exit = interface.Button(800, 400, 300, 100, close, (241, 148, 138), (245, 183, 177))
    15.7.exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", background)
    15.8.while True:
        15.8.1. for event in pygame.event.get():
            15.8.1.1.     jika, event.type == pygame.QUIT:
                15.8.1.1.1. close()
            15.8.1.2.     jika, event.type == pygame.KEYDOWN:
                15.8.1.2.1. jika, event.key == pygame.K_q:
                    15.8.1.2.1.1.   close()
            15.8.1.3.     jika, event.type == pygame.MOUSEBUTTONDOWN:
                15.8.1.3.1. jika, exit.isActive():
                    15.8.1.3.1.1.   exit.action()
                15.8.1.3.2. jika, start.isActive():
                    15.8.1.3.2.1.   start_game(map)
        15.8.2. display.fill(background)
        15.8.3. start.draw()
        15.8.4. exit.draw()
        15.8.5. welcome.draw()
        15.8.6. pygame.display.update()
        15.8.7. clock.tick(60)
16. GAME ()
```

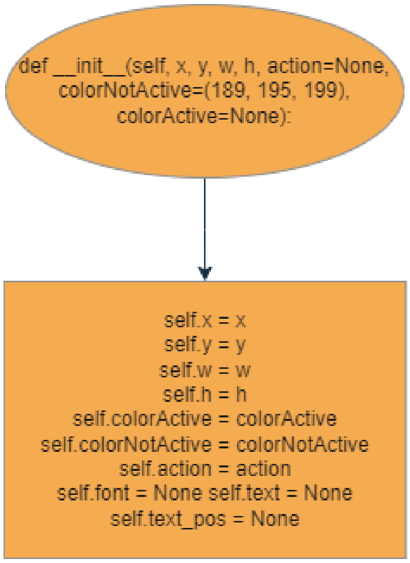
4. Flowchart

Interface

1. Pendefinisian init (Screen)



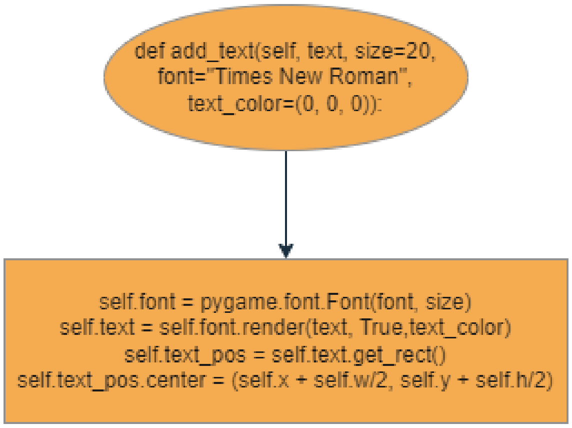
2. Pendefinisian _init_



```
def __init__(self, x, y, w, h, action=None,
colorNotActive=(189, 195, 199),
colorActive=None):
```

```
self.x = x
self.y = y
self.w = w
self.h = h
self.colorActive = colorActive
self.colorNotActive = colorNotActive
self.action = action
self.font = None self.text = None
self.text_pos = None
```

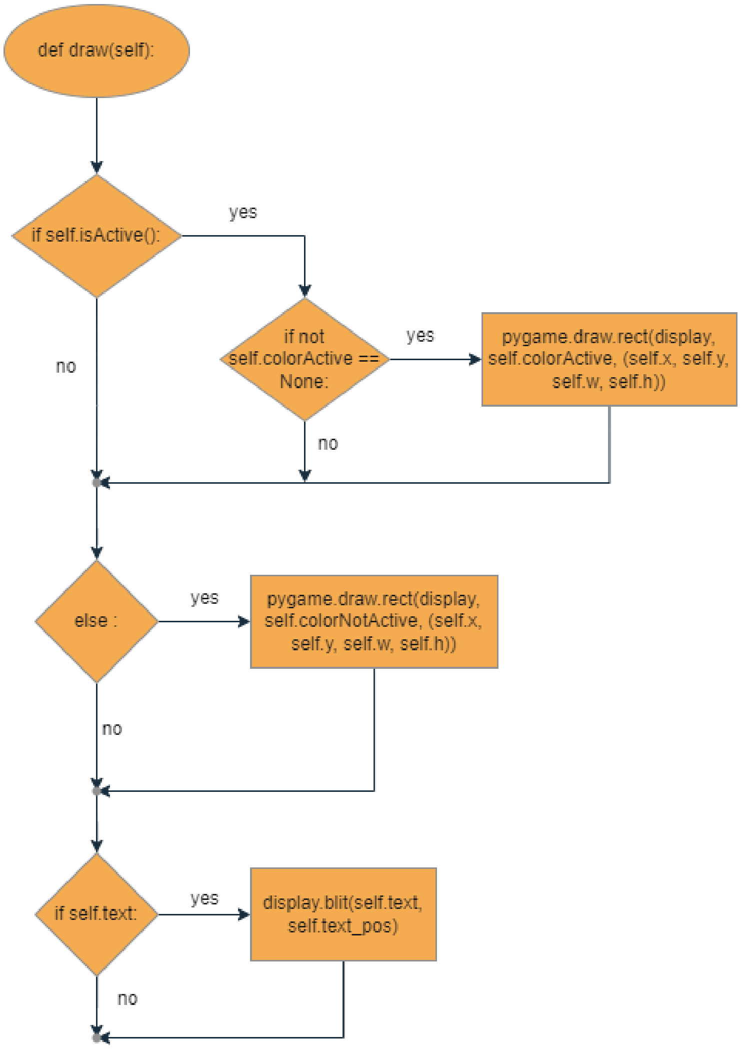
3. Pendefinisian add_text



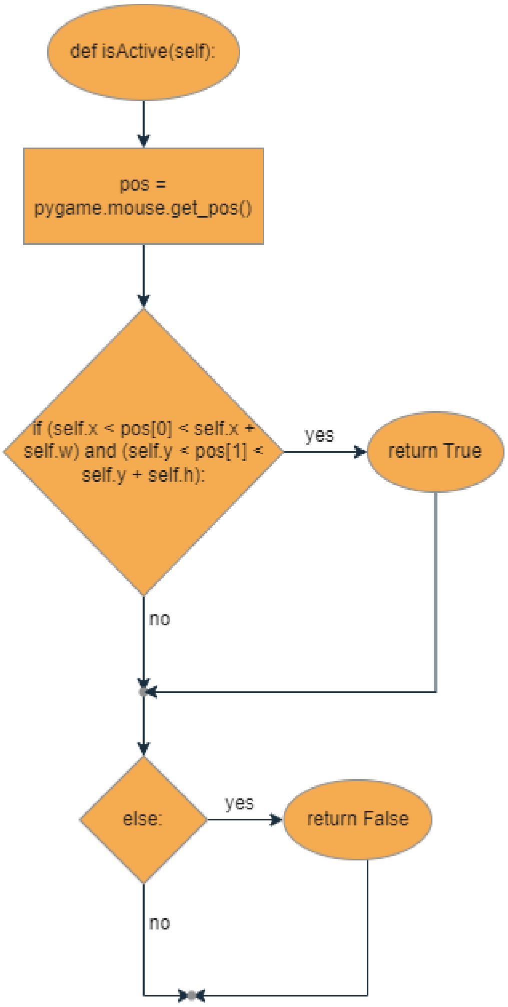
```
def add_text(self, text, size=20,
font="Times New Roman",
text_color=(0, 0, 0)):
```

```
self.font = pygame.font.Font(font, size)
self.text = self.font.render(text, True,text_color)
self.text_pos = self.text.get_rect()
self.text_pos.center = (self.x + self.w/2, self.y + self.h/2)
```

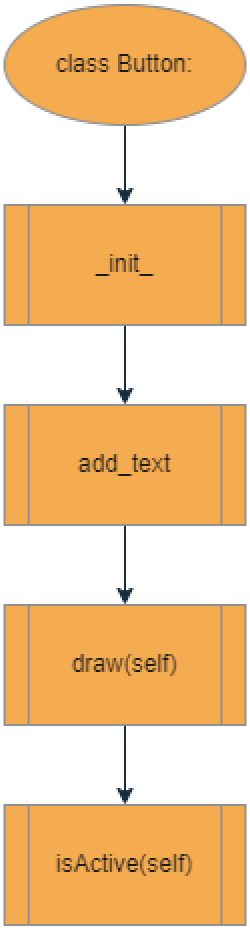
4. Pendefinisian draw


def draw(self):
if self.isActive():
yes
no
if not self.colorActive == None:
yes
pygame.draw.rect(display, self.colorActive, (self.x, self.y, self.w, self.h))
no
else :
yes
pygame.draw.rect(display, self.colorNotActive, (self.x, self.y, self.w, self.h))
no
if self.text:
yes
display.blit(self.text, self.text_pos)
no

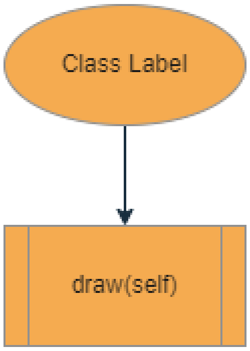
5. Pendefinisian isActive


def isActive(self):
pos = pygame.mouse.get_pos()
if (self.x < pos[0] < self.x + self.w) and (self.y < pos[1] < self.y + self.h):
yes
return True
no
else:
yes
return False
no

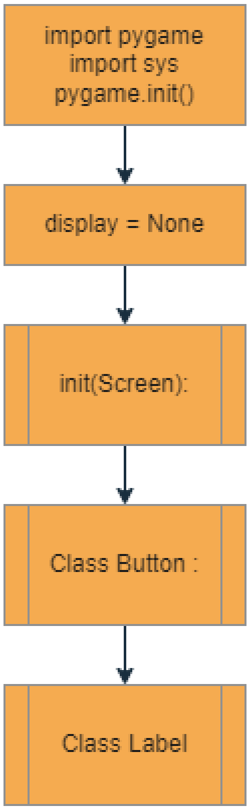
6. Class Button



7. Class Label

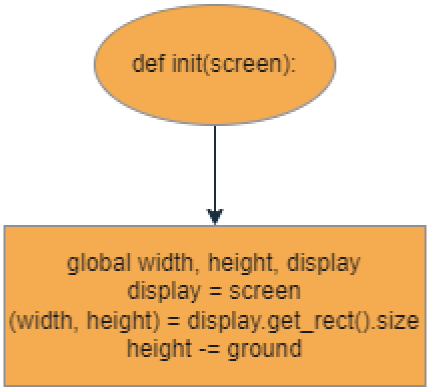


8. Program Utama Interface

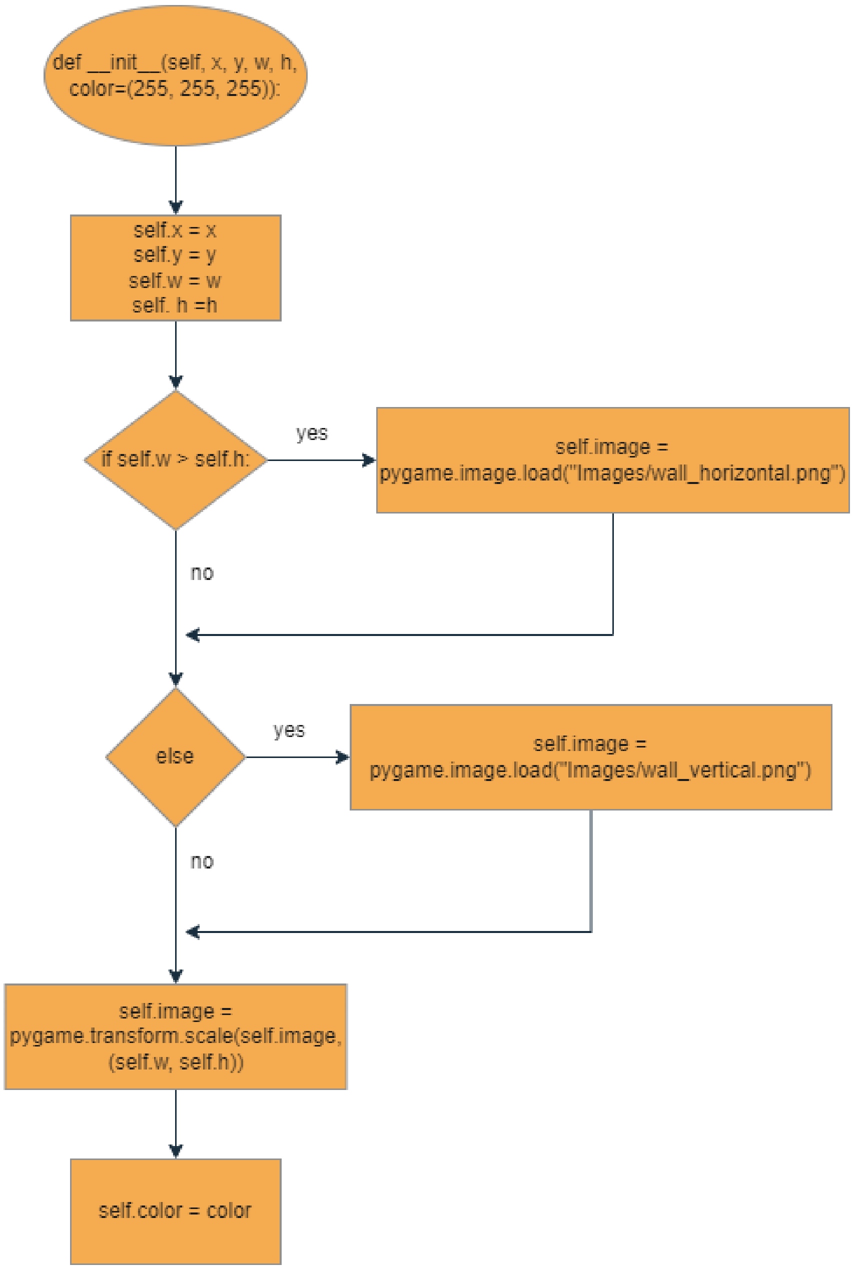


Objects

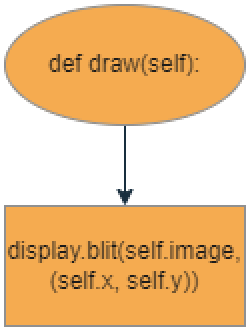
1. Pendefinisian init (Screen) :


def init(screen):
global width, height, display
display = screen
(width, height) = display.get_rect().size
height -= ground

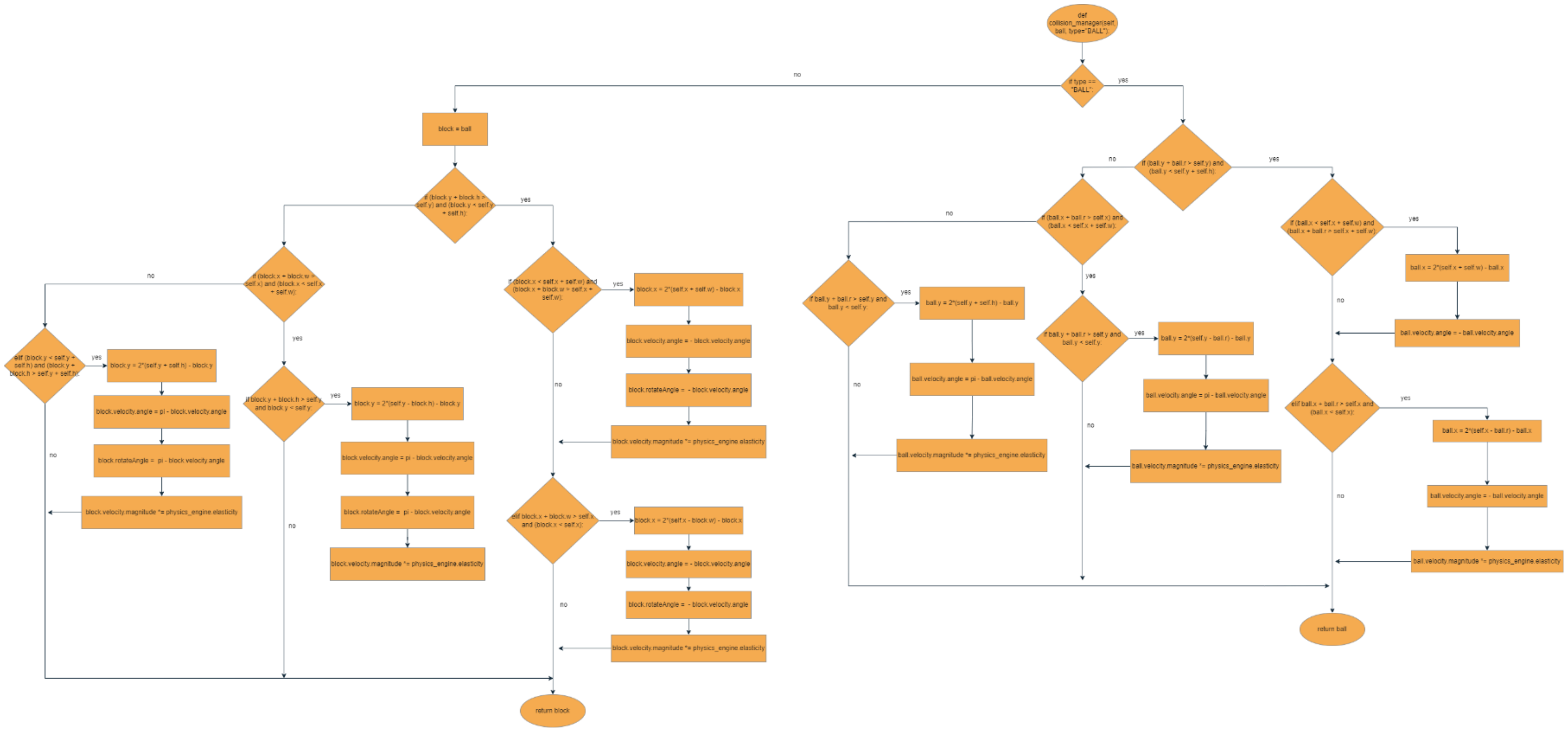
2. Pendefinisian _init_


def __init__(self, x, y, w, h, color=(255, 255, 255)):
self.x = x
self.y = y
self.w = w
self. h =h
if self.w > self.h:
yes
self.image = pygame.image.load("Images/wall_horizontal.png")
no
else
yes
self.image = pygame.image.load("Images/wall_vertical.png")
no
self.image = pygame.transform.scale(self.image, (self.w, self.h))
self.color = color

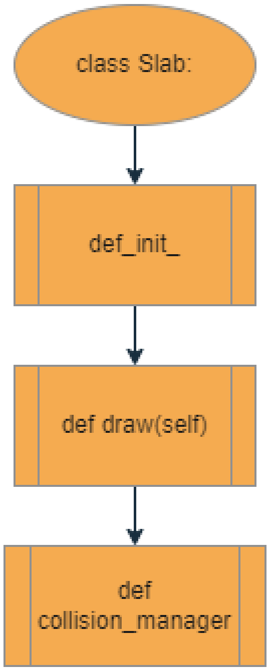
3. Pendefinisian draw(self)



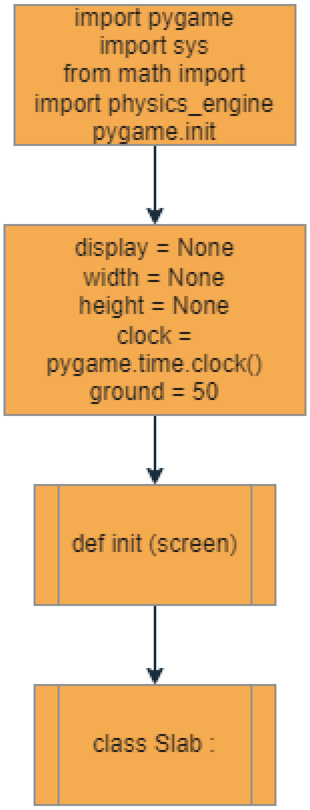
4. Pendefinisian collision_manager


def collision_manager(self, ball, type="BALL"):
if type == "BALL":
no
yes
block = ball
if (block.y + block.h > self.y) and (block.y < self.y + self.h):
yes
if (block.x + block.w > self.x) and (block.x < self.x + self.w):
no
yes
elif (block.y < self.y + self.h) and (block.y + block.h > self.y + self.h):
yes
block.y = 2*(self.y + self.h) - block.y
block.velocity.angle = pi - block.velocity.angle
block.rotateAngle = pi - block.velocity.angle
block.velocity.magnitude *= physics_engine.elasticity
no
if block.y + block.h > self.y and block.y < self.y:
yes
block.y = 2*(self.y - block.h) - block.y
block.velocity.angle = pi - block.velocity.angle
block.rotateAngle = pi - block.velocity.angle
block.velocity.magnitude *= physics_engine.elasticity
no
if (block.x < self.x + self.w) and (block.x + block.w > self.x + self.w):
yes
block.x = 2*(self.x + self.w) - block.x
block.velocity.angle = - block.velocity.angle
block.rotateAngle = - block.velocity.angle
block.velocity.magnitude *= physics_engine.elasticity
no
elif block.x + block.w > self.x and (block.x < self.x):
yes
block.x = 2*(self.x - block.w) - block.x
block.velocity.angle = - block.velocity.angle
block.rotateAngle = - block.velocity.angle
block.velocity.magnitude *= physics_engine.elasticity
no
return block
if (ball.y + ball.r > self.y) and (ball.y < self.y + self.h):
no
yes
if (ball.x + ball.r > self.x) and (ball.x < self.x + self.w):
no
yes
if ball.y + ball.r > self.y and ball.y < self.y:
yes
ball.y = 2*(self.y + self.h) - ball.y
ball.velocity.angle = pi - ball.velocity.angle
ball.velocity.magnitude *= physics_engine.elasticity
no
if ball.y + ball.r > self.y and ball.y < self.y:
yes
ball.y = 2*(self.y - ball.r) - ball.y
ball.velocity.angle = pi - ball.velocity.angle
ball.velocity.magnitude *= physics_engine.elasticity
no
if (ball.x < self.x + self.w) and (ball.x + ball.r > self.x + self.w):
yes
ball.x = 2*(self.x + self.w) - ball.x
ball.velocity.angle = - ball.velocity.angle
no
elif ball.x + ball.r > self.x and (ball.x < self.x):
yes
ball.x = 2*(self.x - ball.r) - ball.x
ball.velocity.angle = - ball.velocity.angle
ball.velocity.magnitude *= physics_engine.elasticity
no
return ball

5. Class Slab

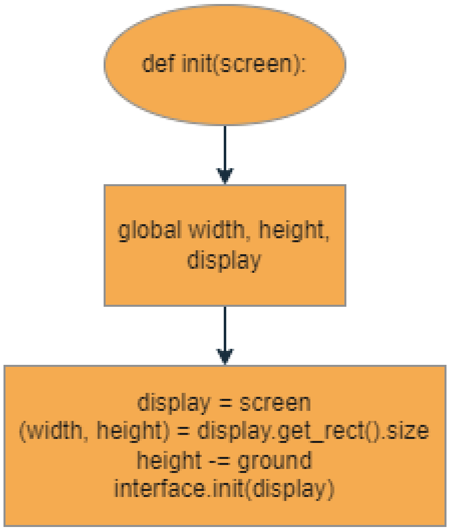


6. Program Utama Objects

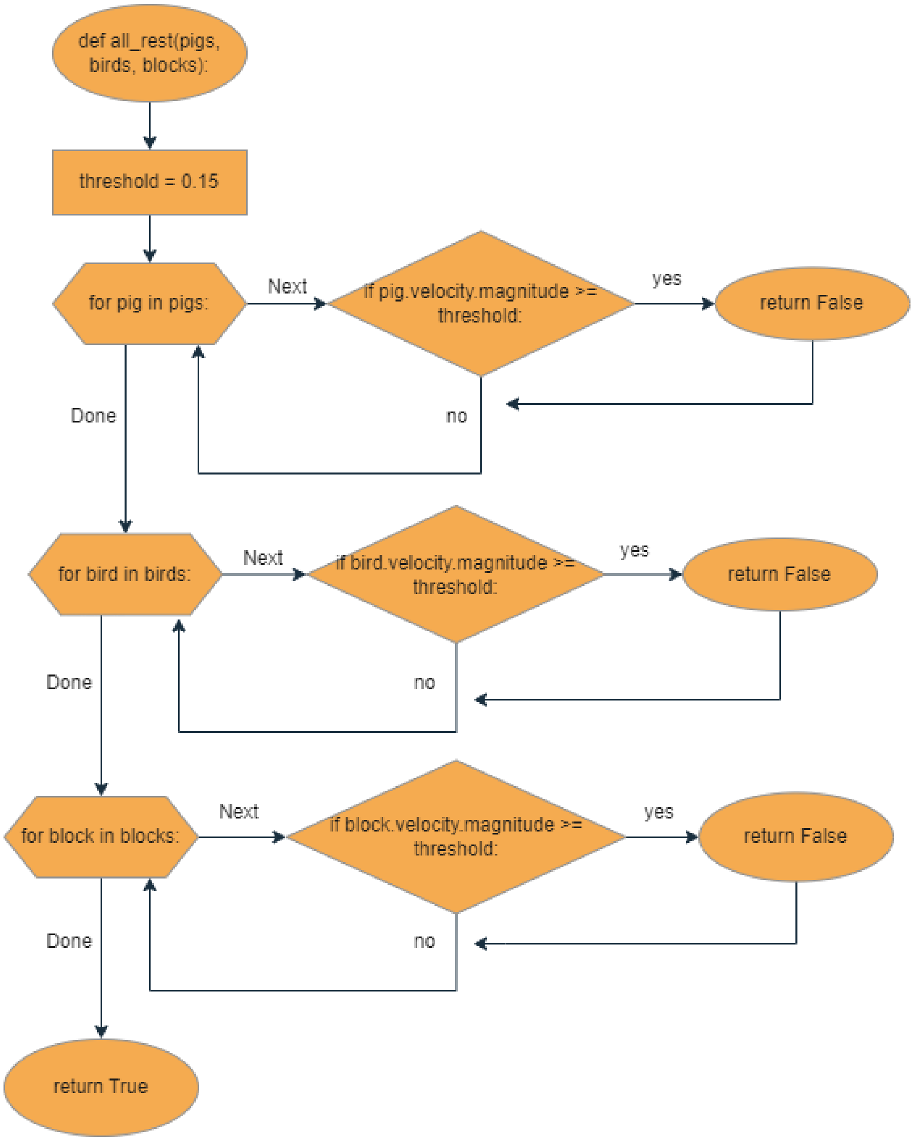


Maps

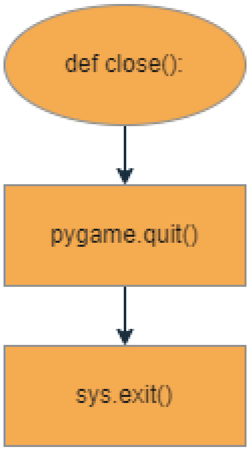
1. Pendefinisian init (screen)



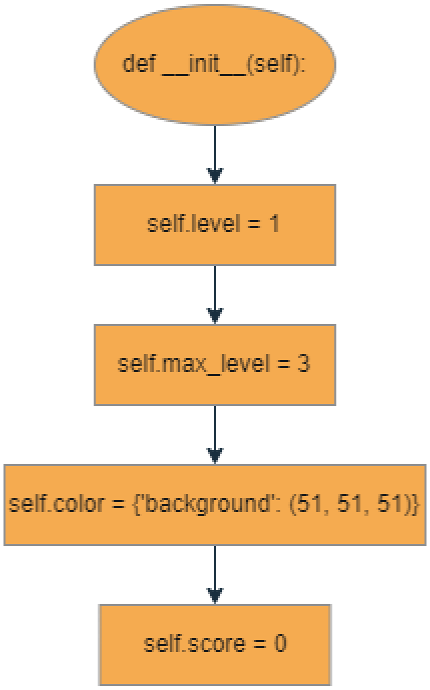
2. Pendefinisian all_rest



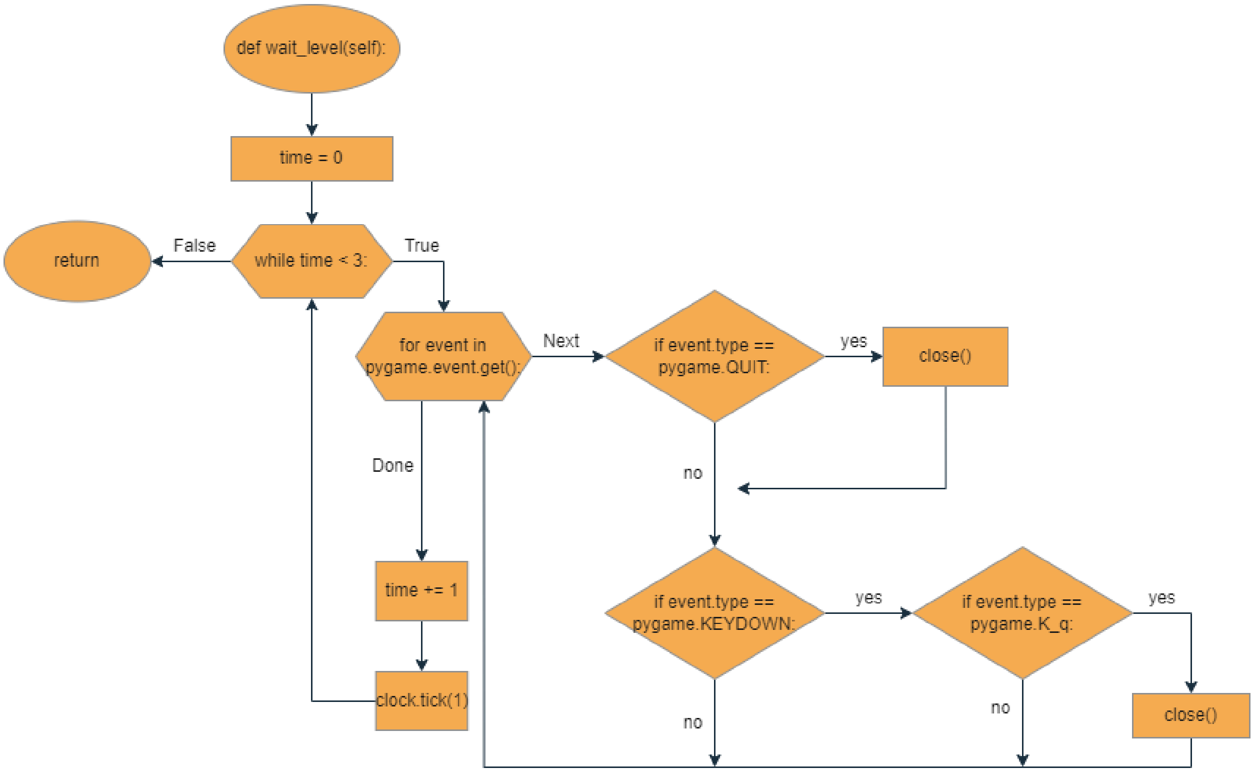
3. Pendefinisian close



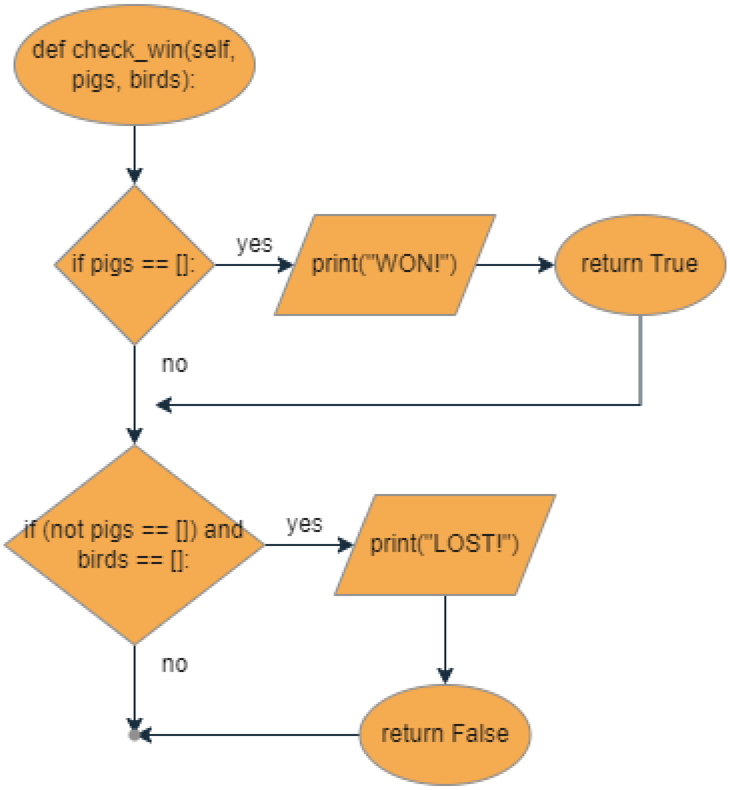
4. Pendefinisian _init_(self)



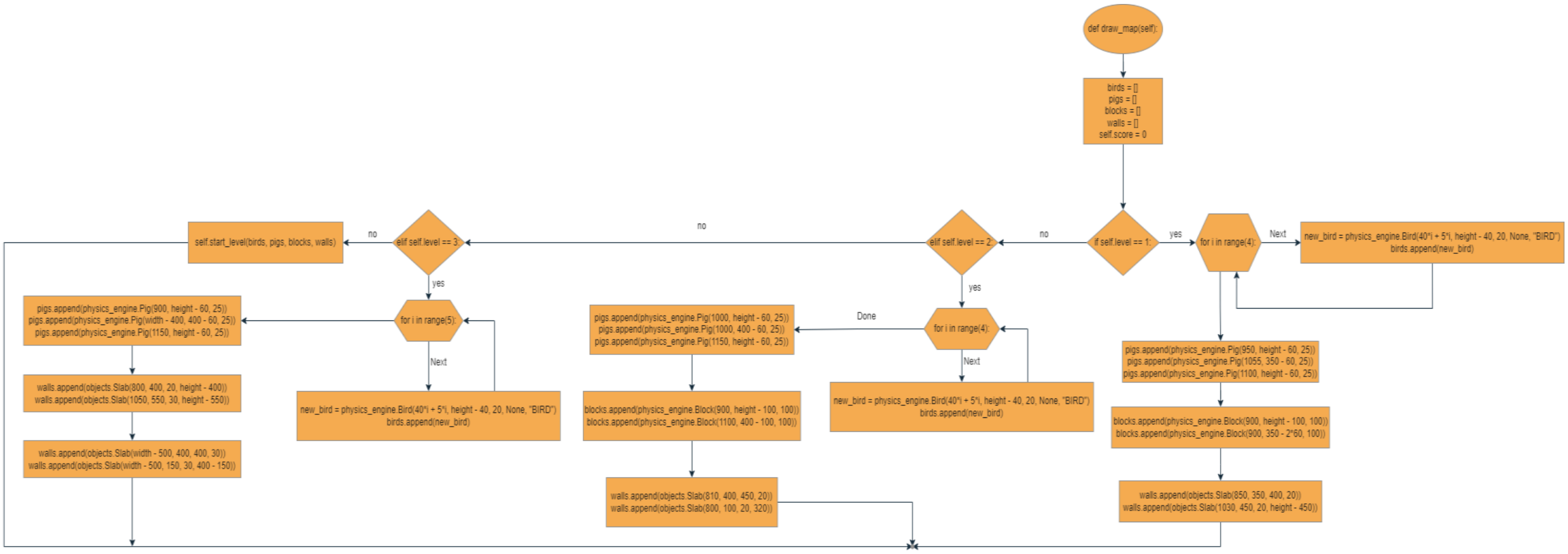
5. Pendefinisian wait_level(self)


def wait_level(self):
time = 0
return
False
while time < 3:
True
for event in pygame.event.get():
Next
if event.type == pygame.QUIT:
yes
close()
no
Done
time += 1
clock.tick(1)
if event.type == pygame.KEYDOWN:
yes
if event.type == pygame.K_q:
yes
close()
no
no

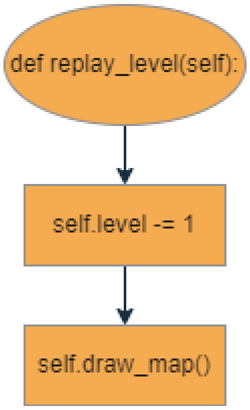
6. Pendefinisian check_win



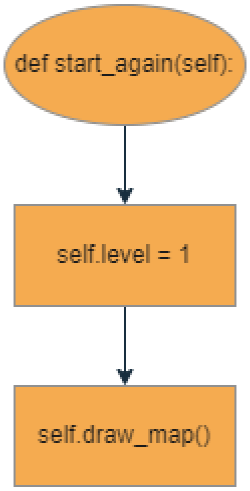
7. Pendefinisian draw_map(self)


def draw_map(self):
birds = []
pigs = []
blocks = []
walls = []
self.score = 0
if self.level == 1:
yes
for i in range(4):
Next
new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
birds.append(new_bird)
pigs.append(physics_engine.Pig(950, height - 60, 25))
pigs.append(physics_engine.Pig(1055, 350 - 60, 25))
pigs.append(physics_engine.Pig(1100, height - 60, 25))
blocks.append(physics_engine.Block(900, height - 100, 100))
blocks.append(physics_engine.Block(900, 350 - 2*60, 100))
walls.append(objects.Slab(850, 350, 400, 20))
walls.append(objects.Slab(1030, 450, 20, height - 450))
no
elif self.level == 2:
yes
for i in range(4):
Next
new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
birds.append(new_bird)
Done
pigs.append(physics_engine.Pig(1000, height - 60, 25))
pigs.append(physics_engine.Pig(1000, 400 - 60, 25))
pigs.append(physics_engine.Pig(1150, height - 60, 25))
blocks.append(physics_engine.Block(900, height - 100, 100))
blocks.append(physics_engine.Block(1100, 400 - 100, 100))
walls.append(objects.Slab(810, 400, 450, 20))
walls.append(objects.Slab(800, 100, 20, 320))
no
elif self.level == 3:
yes
for i in range(5):
Next
new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
birds.append(new_bird)
pigs.append(physics_engine.Pig(900, height - 60, 25))
pigs.append(physics_engine.Pig(width - 400, 400 - 60, 25))
pigs.append(physics_engine.Pig(1150, height - 60, 25))
walls.append(objects.Slab(800, 400, 20, height - 400))
walls.append(objects.Slab(1050, 550, 30, height - 550))
walls.append(objects.Slab(width - 500, 400, 400, 30))
walls.append(objects.Slab(width - 500, 150, 30, 400 - 150))
no
self.start_level(birds, pigs, blocks, walls)

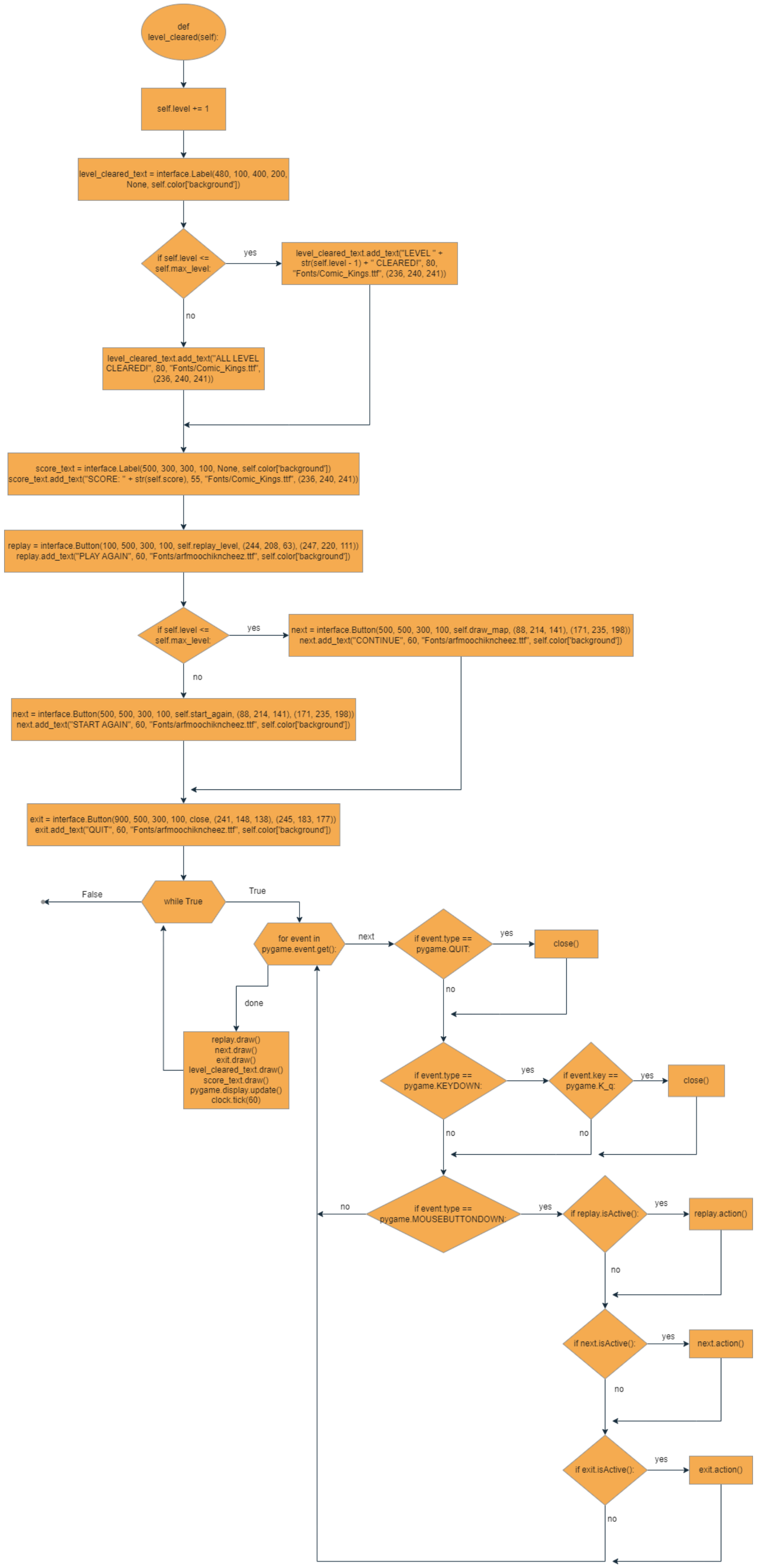
8. Pendefinisian replay_level(self)



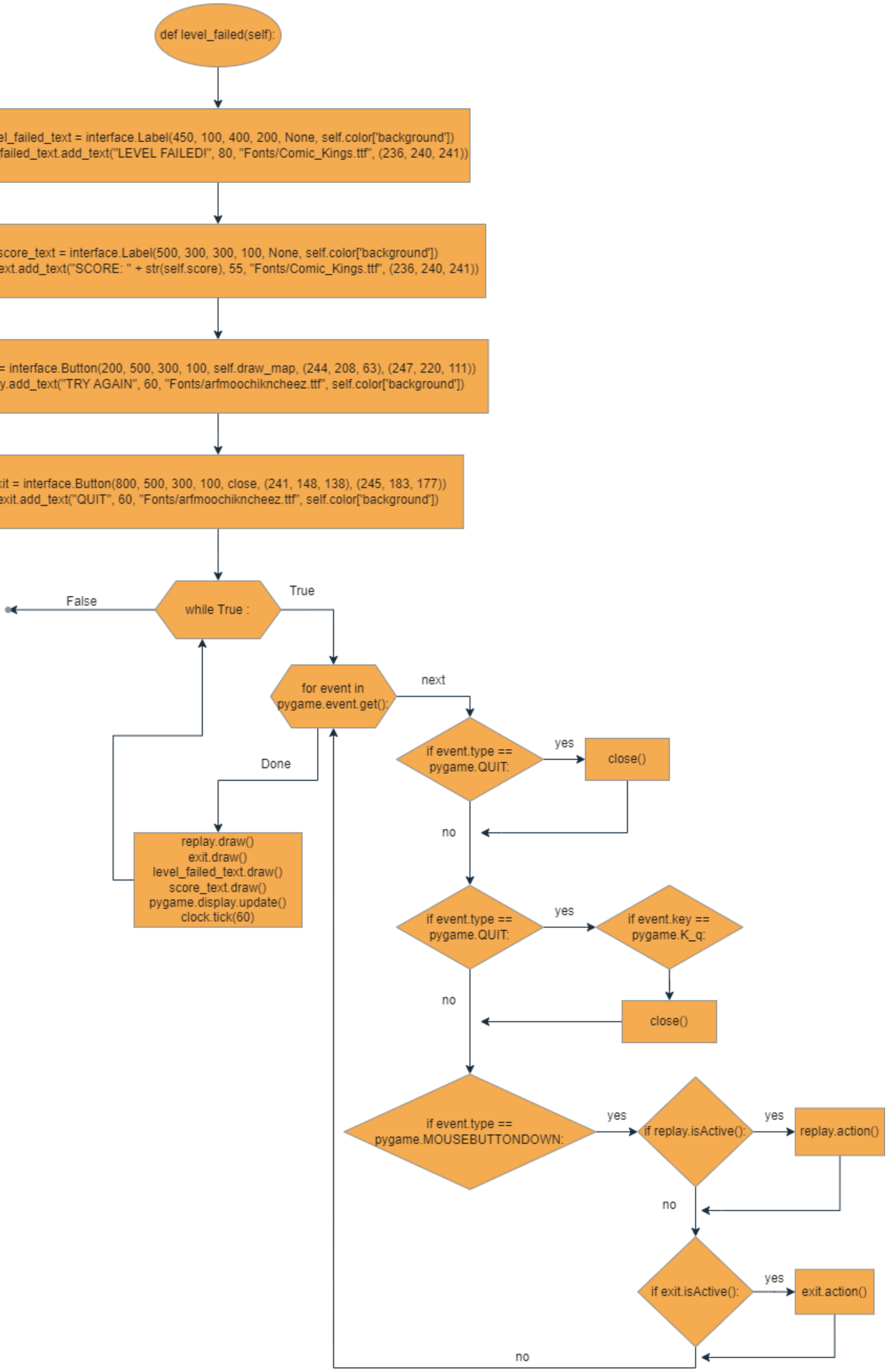
9. Pendefinisian start_again(self)



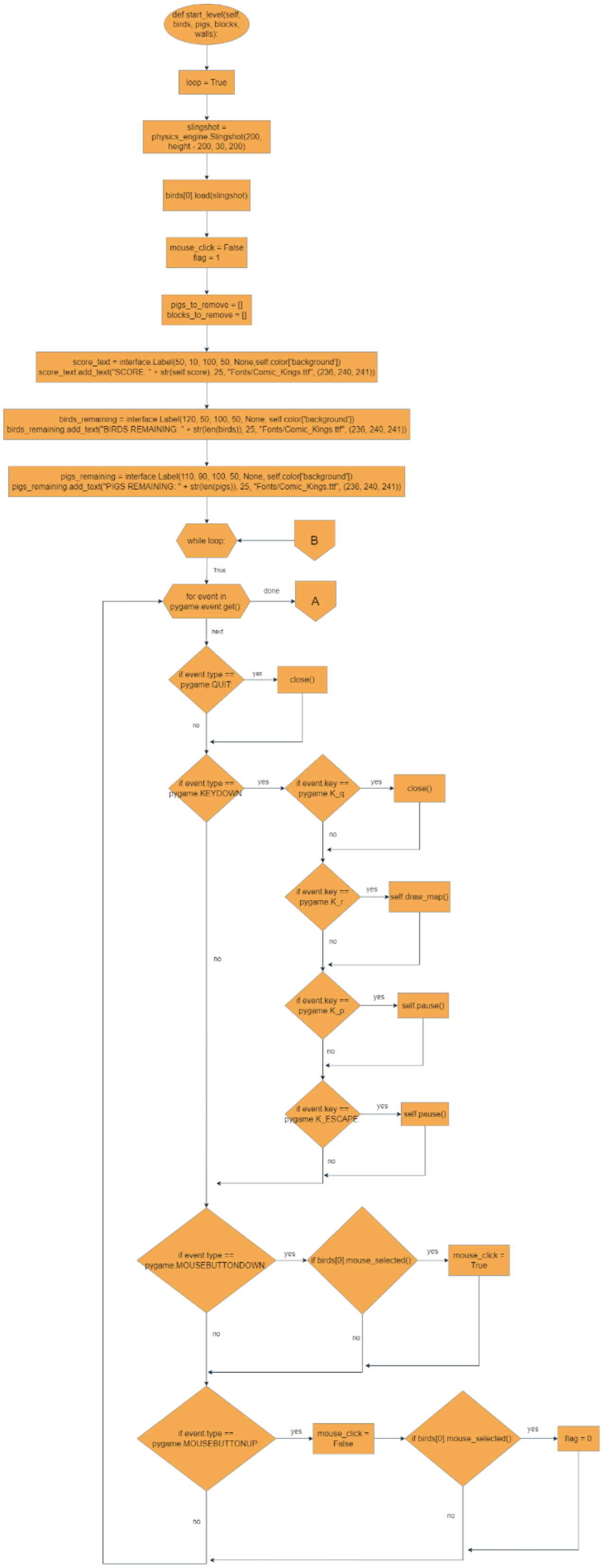
10. Pendefinisian level_cleared(self)

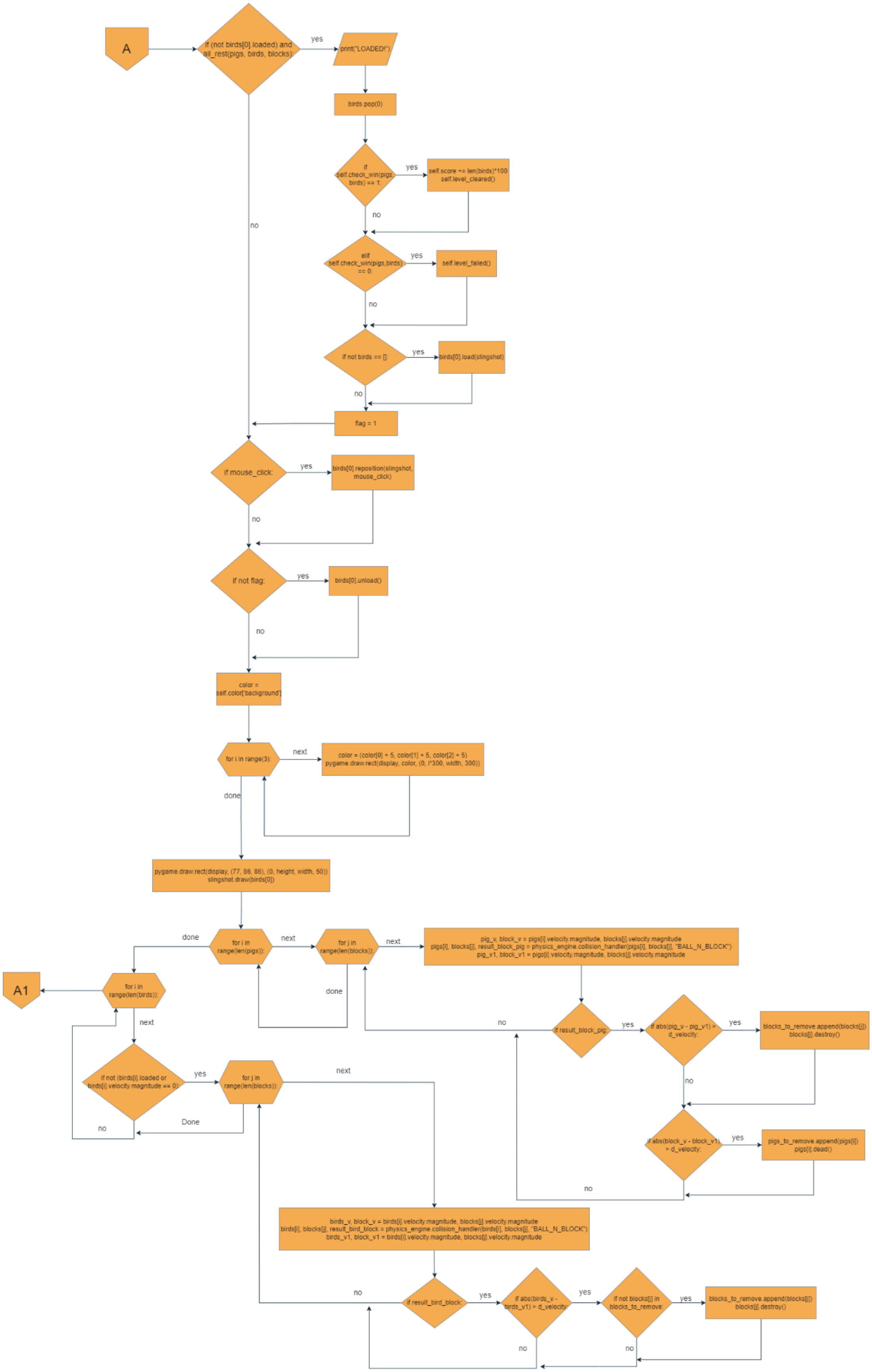

def level_cleared(self):
self.level += 1
level_cleared_text = interface.Label(480, 100, 400, 200, None, self.color['background'])
if self.level <= self.max_level:
yes
level_cleared_text.add_text("LEVEL " + str(self.level - 1) + " CLEARED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
no
level_cleared_text.add_text("ALL LEVEL CLEARED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
score_text = interface.Label(500, 300, 300, 100, None, self.color['background'])
score_text.add_text("SCORE: " + str(self.score), 55, "Fonts/Comic_Kings.ttf", (236, 240, 241))
replay = interface.Button(100, 500, 300, 100, self.replay_level, (244, 208, 63), (247, 220, 111))
replay.add_text("PLAY AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
if self.level <= self.max_level:
yes
next = interface.Button(500, 500, 300, 100, self.draw_map, (88, 214, 141), (171, 235, 198))
next.add_text("CONTINUE", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
no
next = interface.Button(500, 500, 300, 100, self.start_again, (88, 214, 141), (171, 235, 198))
next.add_text("START AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
exit = interface.Button(900, 500, 300, 100, close, (241, 148, 138), (245, 183, 177))
exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
False
while True
True
for event in pygame.event.get():
next
if event.type == pygame.QUIT:
yes
close()
no
done
replay.draw()
next.draw()
exit.draw()
level_cleared_text.draw()
score_text.draw()
pygame.display.update()
clock.tick(60)
if event.type == pygame.KEYDOWN:
yes
if event.key == pygame.K_q:
yes
close()
no
no
no
if event.type == pygame.MOUSEBUTTONDOWN:
yes
if replay.isActive():
yes
replay.action()
no
if next.isActive():
yes
next.action()
no
if exit.isActive():
yes
exit.action()
no

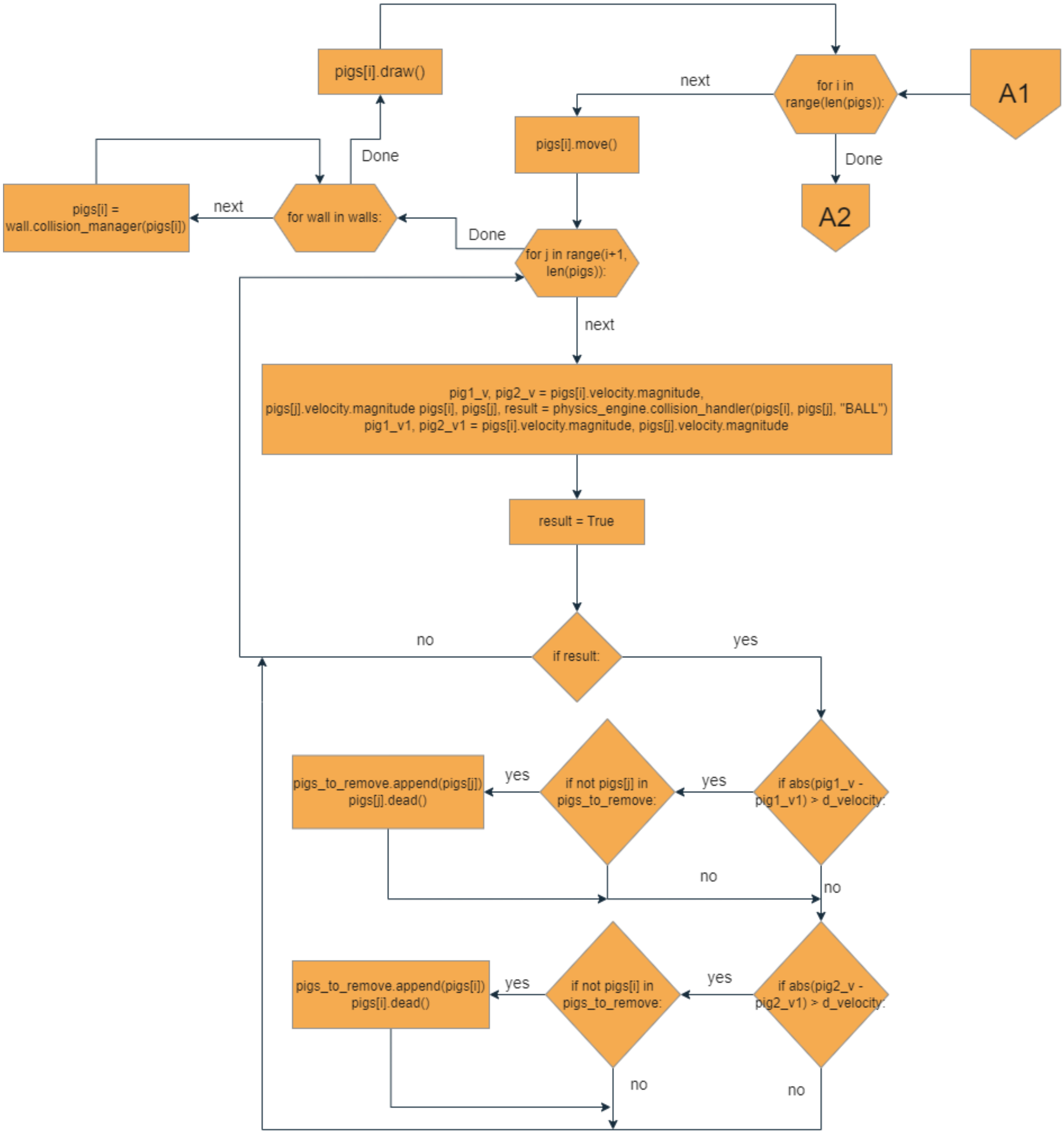
11. Pendefinisian level failed(self)

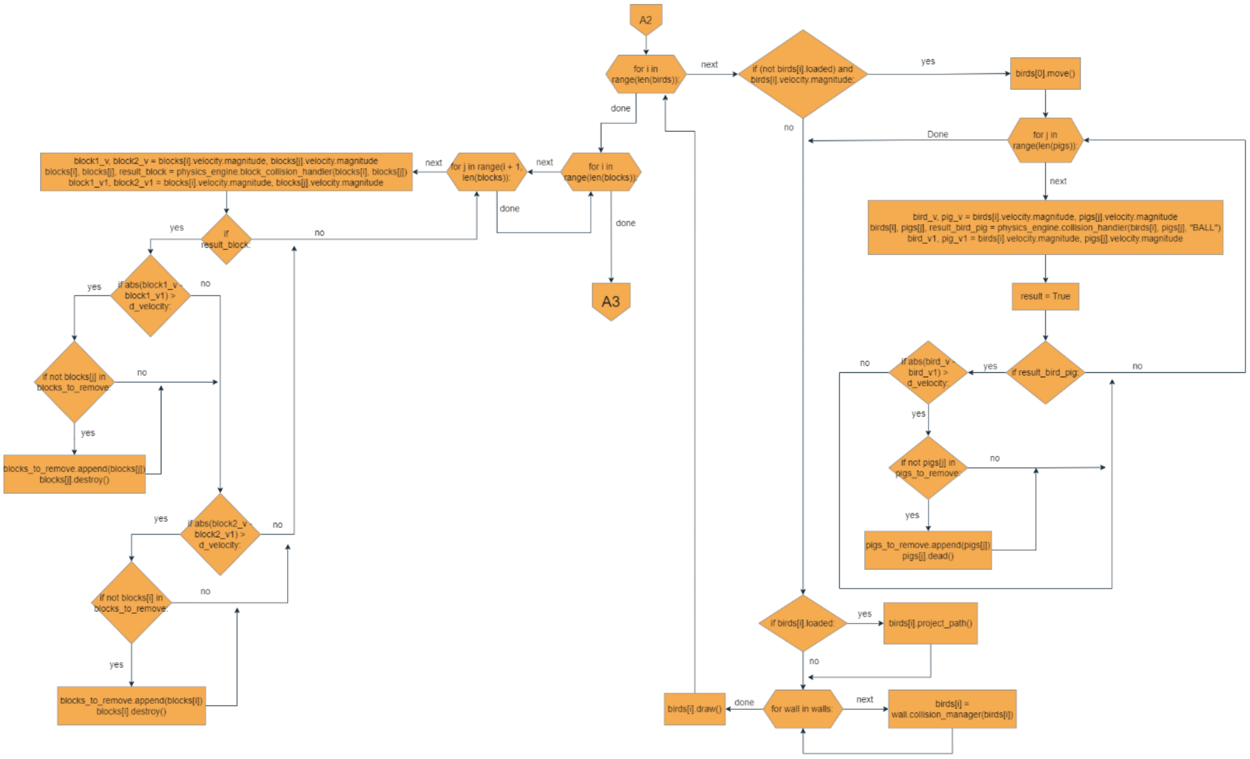

def level_failed(self):
level_failed_text = interface.Label(450, 100, 400, 200, None, self.color['background'])
level_failed_text.add_text("LEVEL FAILED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
score_text = interface.Label(500, 300, 300, 100, None, self.color['background'])
score_text.add_text("SCORE: " + str(self.score), 55, "Fonts/Comic_Kings.ttf", (236, 240, 241))
replay = interface.Button(200, 500, 300, 100, self.draw_map, (244, 208, 63), (247, 220, 111))
replay.add_text("TRY AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
exit = interface.Button(800, 500, 300, 100, close, (241, 148, 138), (245, 183, 177))
exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
False
while True :
True
for event in pygame.event.get():
next
if event.type == pygame.QUIT:
yes
close()
Done
no
replay.draw()
exit.draw()
level_failed_text.draw()
score_text.draw()
pygame.display.update()
clock.tick(60)
if event.type == pygame.QUIT:
yes
if event.key == pygame.K_q:
no
close()
if event.type == pygame.MOUSEBUTTONDOWN:
yes
if replay.isActive():
yes
replay.action()
no
if exit.isActive():
yes
exit.action()
no

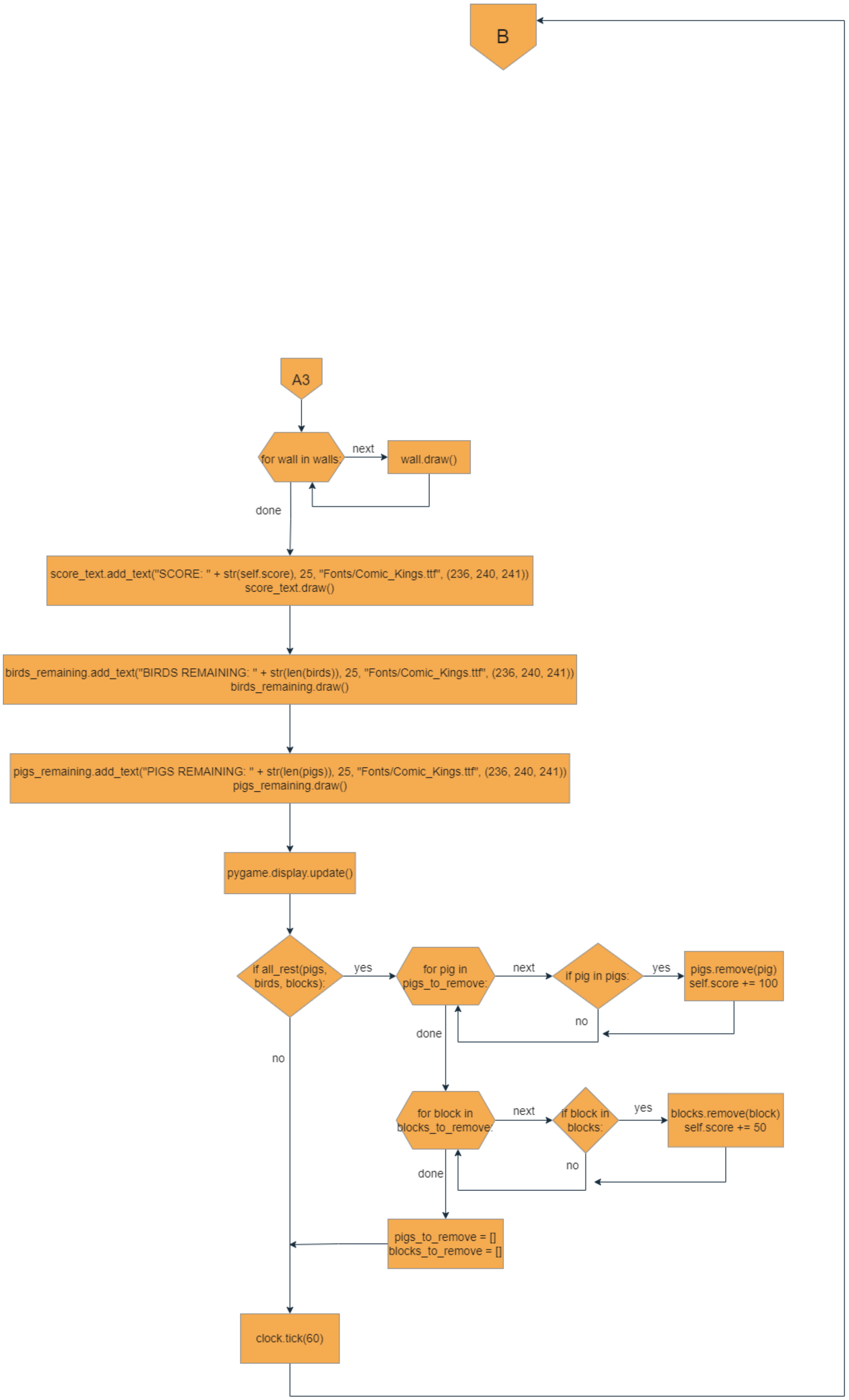
12. Pendefinisian start_level


def start_level(self, birds, pigs, blocks, walls):
loop = True
slingshot = physics_engine.Slingshot(200, height - 200, 30, 200)
birds[0].load(slingshot)
mouse_click = False
flag = 1
pigs_to_remove = []
blocks_to_remove = []
score_text = interface.Label(50, 10, 100, 50, None,self.color['background'])
score_text.add_text("SCORE: " + str(self.score), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
birds_remaining = interface.Label(120, 50, 100, 50, None, self.color['background'])
birds_remaining.add_text("BIRDS REMAINING: " + str(len(birds)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
pigs_remaining = interface.Label(110, 90, 100, 50, None, self.color['background'])
pigs_remaining.add_text("PIGS REMAINING: " + str(len(pigs)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
while loop:
B
True
for event in pygame.event.get():
done
A
next
if event.type == pygame.QUIT:
yes
close()
no
if event.type == pygame.KEYDOWN:
yes
if event.key == pygame.K_q:
yes
close()
no
if event.key == pygame.K_r:
yes
self.draw_map()
no
if event.key == pygame.K_p:
yes
self.pause()
no
if event.key == pygame.K_ESCAPE:
yes
self.pause()
no
no
if event.type == pygame.MOUSEBUTTONDOWN:
yes
if birds[0].mouse_selected():
yes
mouse_click = True
no
no
if event.type == pygame.MOUSEBUTTONUP:
yes
mouse_click = False
if birds[0].mouse_selected():
yes
flag = 0
no
no

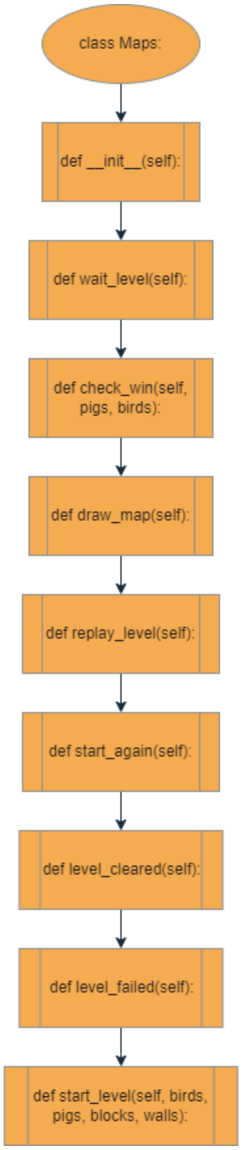

A
if (not birds[0].loaded) and all_rest(pigs, birds, blocks):
yes
print("LOADED!")
birds.pop(0)
if self.check_win(pigs, birds) == 1:
yes
self.score += len(birds)*100
self.level_cleared()
no
elif self.check_win(pigs,birds) == 0:
yes
self.level_failed()
no
if not birds == []:
yes
birds[0].load(slingshot)
no
flag = 1
no
if mouse_click:
yes
birds[0].reposition(slingshot, mouse_click)
no
if not flag:
yes
birds[0].unload()
no
color = self.color['background']
for i in range(3):
next
color = (color[0] + 5, color[1] + 5, color[2] + 5)
pygame.draw.rect(display, color, (0, i*300, width, 300))
done
pygame.draw.rect(display, (77, 86, 86), (0, height, width, 50))
slingshot.draw(birds[0])
done
for i in range(len(pigs)):
next
for j in range(len(blocks)):
next
pig_v, block_v = pigs[i].velocity.magnitude, blocks[j].velocity.magnitude
pigs[i], blocks[j], result_block_pig = physics_engine.collision_handler(pigs[i], blocks[j], "BALL_N_BLOCK")
pig_v1, block_v1 = pigs[i].velocity.magnitude, blocks[j].velocity.magnitude
done
A1
for i in range(len(birds)):
next
no
if result_block_pig:
yes
if abs(pig_v - pig_v1) > d_velocity:
yes
blocks_to_remove.append(blocks[j])
blocks[j].destroy()
no
if not (birds[i].loaded or birds[i].velocity.magnitude == 0):
yes
for j in range(len(blocks)):
next
Done
no
if abs(block_v - block_v1) > d_velocity:
yes
pigs_to_remove.append(pigs[i])
pigs[i].dead()
no
birds_v, block_v = birds[i].velocity.magnitude, blocks[j].velocity.magnitude
birds[i], blocks[j], result_bird_block = physics_engine.collision_handler(birds[i], blocks[j], "BALL_N_BLOCK")
birds_v1, block_v1 = birds[i].velocity.magnitude, blocks[j].velocity.magnitude
no
if result_bird_block:
yes
if abs(birds_v - birds_v1) > d_velocity:
yes
if not blocks[j] in blocks_to_remove:
yes
blocks_to_remove.append(blocks[j])
blocks[j].destroy()
no
no


pigs[i].draw()
next
for i in range(len(pigs)):
A1
Done
pigs[i].move()
A2
pigs[i] = wall.collision_manager(pigs[i])
next
for wall in walls:
Done
for j in range(i+1, len(pigs)):
next
pig1_v, pig2_v = pigs[i].velocity.magnitude,
pigs[j].velocity.magnitude pigs[i], pigs[j], result = physics_engine.collision_handler(pigs[i], pigs[j], "BALL")
pig1_v1, pig2_v1 = pigs[i].velocity.magnitude, pigs[j].velocity.magnitude
result = True
no
if result:
yes
pigs_to_remove.append(pigs[j])
pigs[j].dead()
yes
if not pigs[j] in pigs_to_remove:
yes
if abs(pig1_v - pig1_v1) > d_velocity:
no
no
pigs_to_remove.append(pigs[i])
pigs[i].dead()
yes
if not pigs[i] in pigs_to_remove:
yes
if abs(pig2_v - pig2_v1) > d_velocity:
no
no

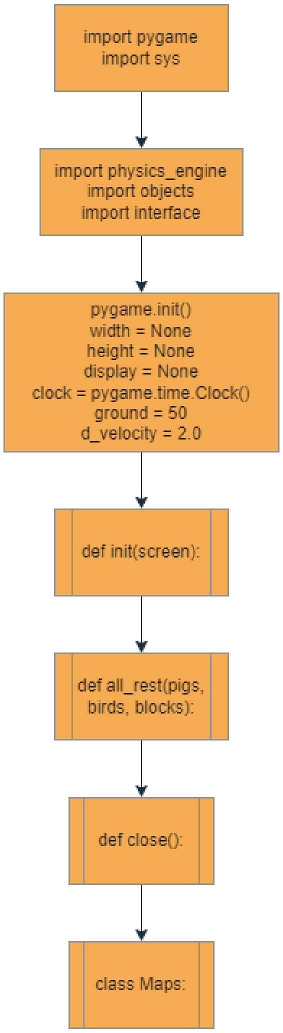

A2
for i in range(len(birds)):
next
if (not birds[i].loaded) and birds[i].velocity.magnitude:
yes
birds[0].move()
for j in range(len(pigs)):
Done
next
bird_v, pig_v = birds[i].velocity.magnitude, pigs[j].velocity.magnitude
birds[i], pigs[j], result_bird_pig = physics_engine.collision_handler(birds[i], pigs[j], "BALL")
bird_v1, pig_v1 = birds[i].velocity.magnitude, pigs[j].velocity.magnitude
result = True
if result_bird_pig:
yes
no
if abs(bird_v - bird_v1) > d_velocity:
no
yes
if not pigs[j] in pigs_to_remove:
no
yes
pigs_to_remove.append(pigs[j])
pigs[j].dead()
no
if birds[i].loaded:
yes
birds[i].project_path()
no
for wall in walls:
next
birds[i] = wall.collision_manager(birds[i])
done
birds[i].draw()
done
for i in range(len(blocks)):
next
for j in range(i + 1, len(blocks)):
next
block1_v, block2_v = blocks[i].velocity.magnitude, blocks[j].velocity.magnitude
blocks[i], blocks[j], result_block = physics_engine.block_collision_handler(blocks[i], blocks[j])
block1_v1, block2_v1 = blocks[i].velocity.magnitude, blocks[j].velocity.magnitude
done
done
A3
if result_block:
yes
no
if abs(block1_v - block1_v1) > d_velocity:
yes
no
if not blocks[j] in blocks_to_remove:
no
yes
blocks_to_remove.append(blocks[j])
blocks[j].destroy()
if abs(block2_v - block2_v1) > d_velocity:
yes
no
if not blocks[i] in blocks_to_remove:
no
yes
blocks_to_remove.append(blocks[i])
blocks[i].destroy()


B
A3
for wall in walls:
next
wall.draw()
done
score_text.add_text("SCORE: " + str(self.score), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
score_text.draw()
birds_remaining.add_text("BIRDS REMAINING: " + str(len(birds)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
birds_remaining.draw()
pigs_remaining.add_text("PIGS REMAINING: " + str(len(pigs)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
pigs_remaining.draw()
pygame.display.update()
if all_rest(pigs, birds, blocks):
yes
for pig in pigs_to_remove:
next
if pig in pigs:
yes
pigs.remove(pig)
self.score += 100
no
done
no
for block in blocks_to_remove:
next
if block in blocks:
yes
blocks.remove(block)
self.score += 50
no
done
pigs_to_remove = []
blocks_to_remove = []
clock.tick(60)

13. Class Maps

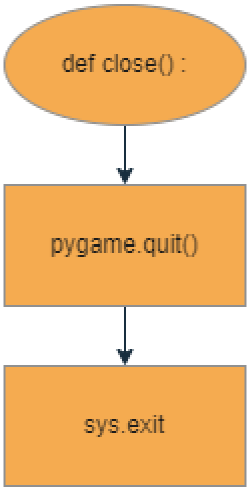


14. Program Utama Maps

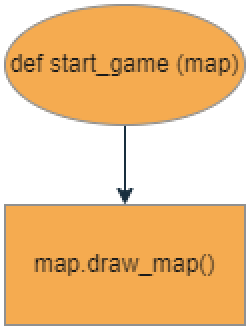


Program Utama

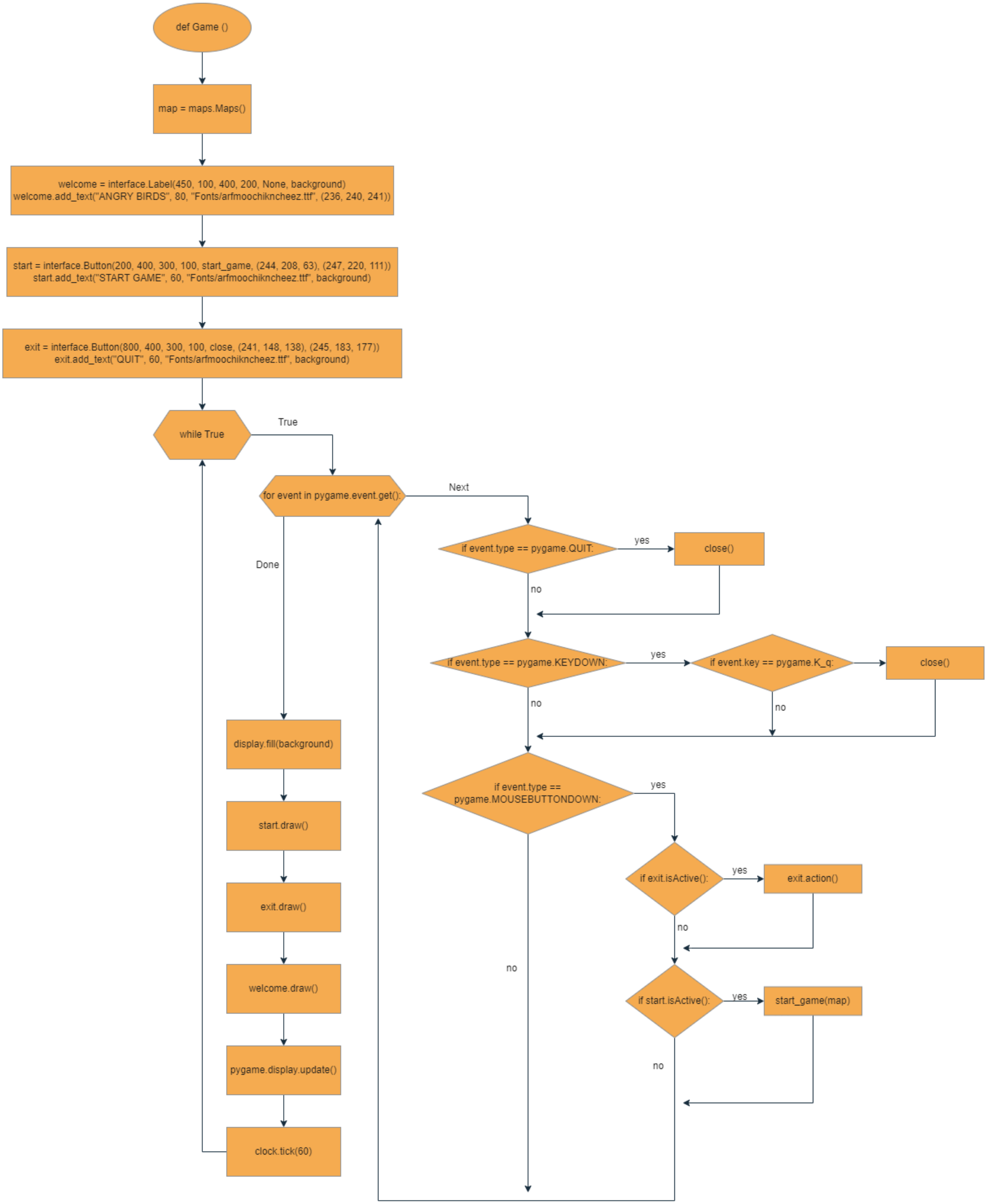
1. Pendefinisian close ()



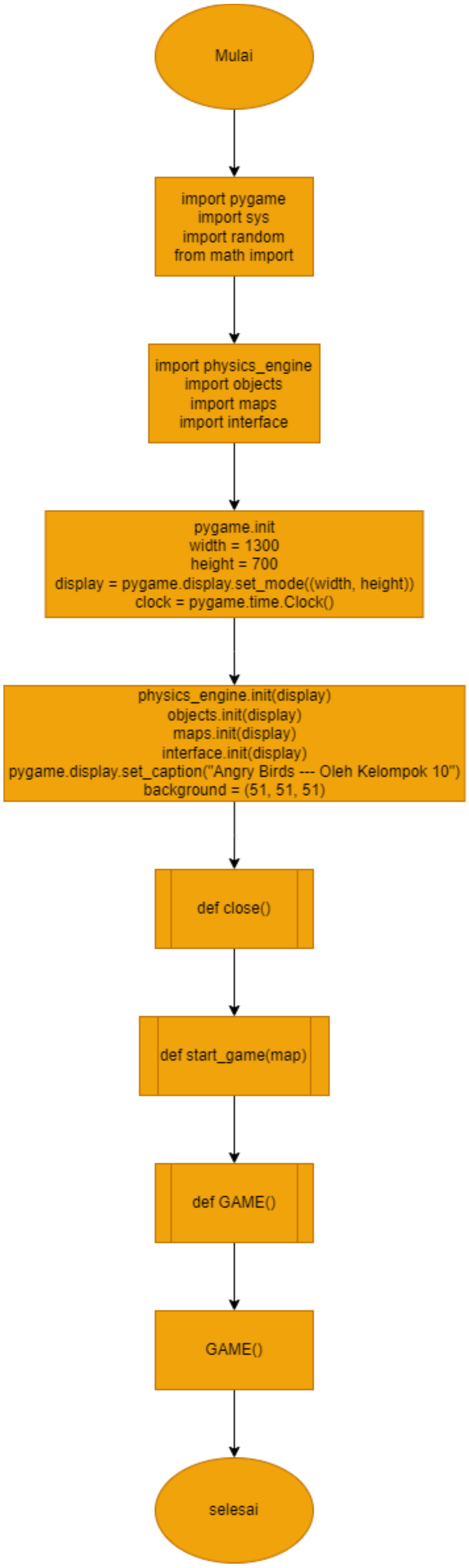
2. Pendefinisian Start_game(map)



3. Pendefinisian Game()


def Game ()
map = maps.Maps()
welcome = interface.Label(450, 100, 400, 200, None, background)
welcome.add_text("ANGRY BIRDS", 80, "Fonts/arfmoochikncheez.ttf", (236, 240, 241))
start = interface.Button(200, 400, 300, 100, start_game, (244, 208, 63), (247, 220, 111))
start.add_text("START GAME", 60, "Fonts/arfmoochikncheez.ttf", background)
exit = interface.Button(800, 400, 300, 100, close, (241, 148, 138), (245, 183, 177))
exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", background)
while True
True
for event in pygame.event.get():
Next
if event.type == pygame.QUIT:
yes
close()
no
Done
if event.type == pygame.KEYDOWN:
yes
if event.key == pygame.K_q:
close()
no
no
display.fill(background)
if event.type == pygame.MOUSEBUTTONDOWN:
yes
start.draw()
if exit.isActive():
yes
exit.action()
exit.draw()
no
no
welcome.draw()
if start.isActive():
yes
start_game(map)
pygame.display.update()
no
clock.tick(60)

4. Program Utama Angry Bird

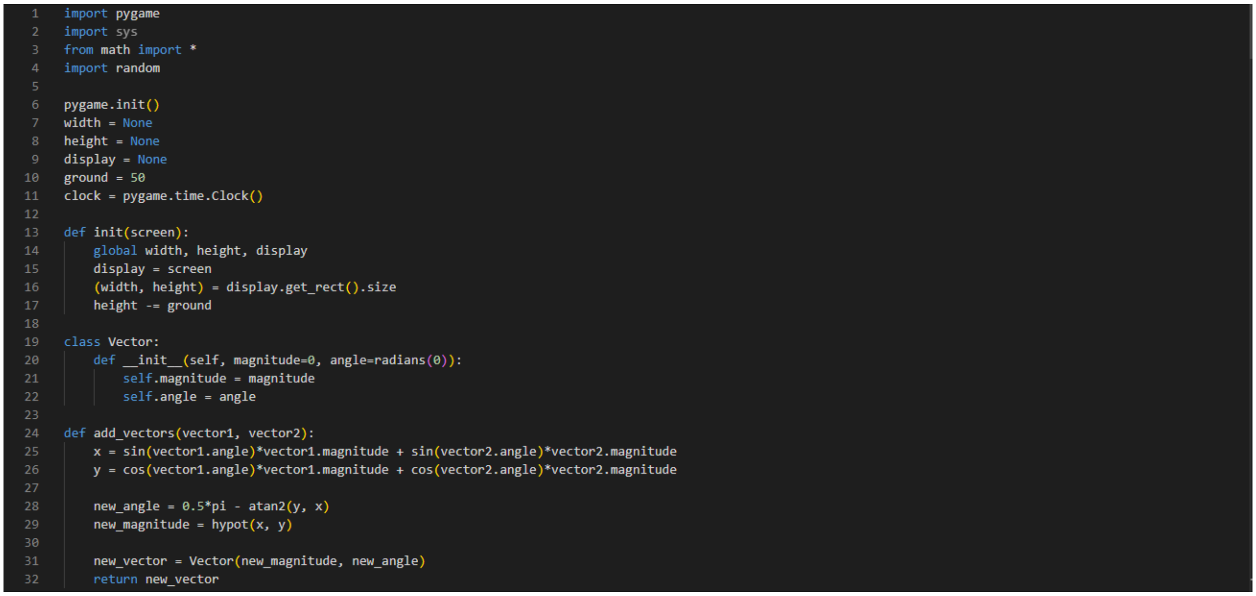


BAB IV

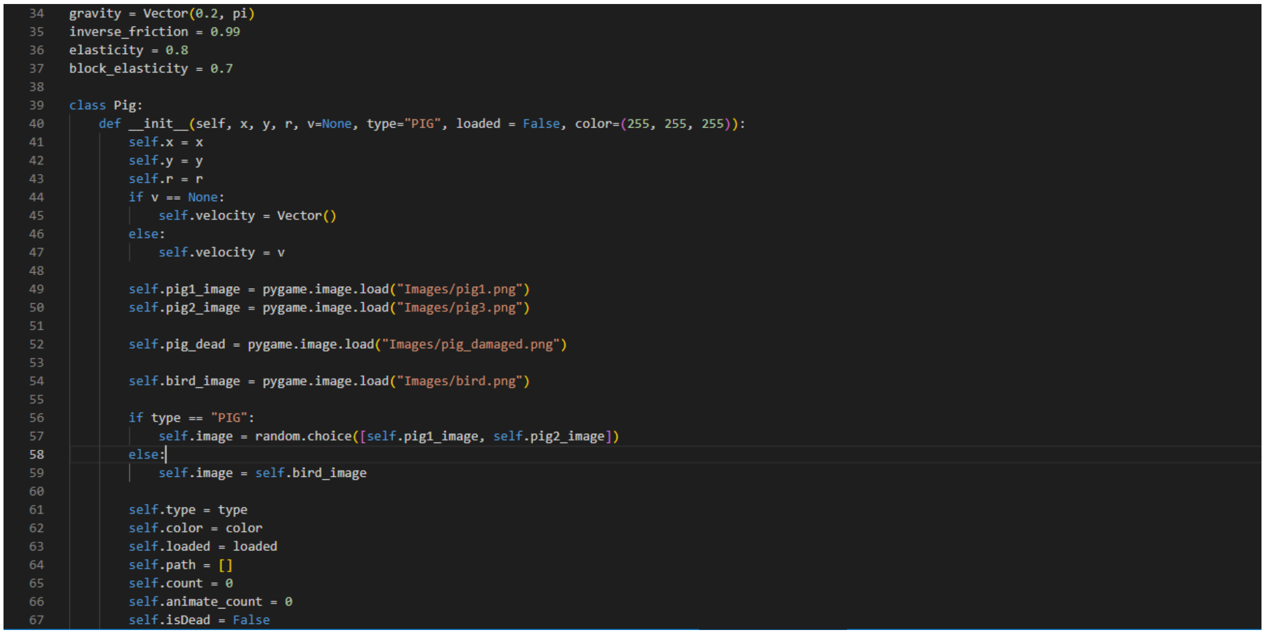
HASIL PEMBAHASAN

4.1. Source Code

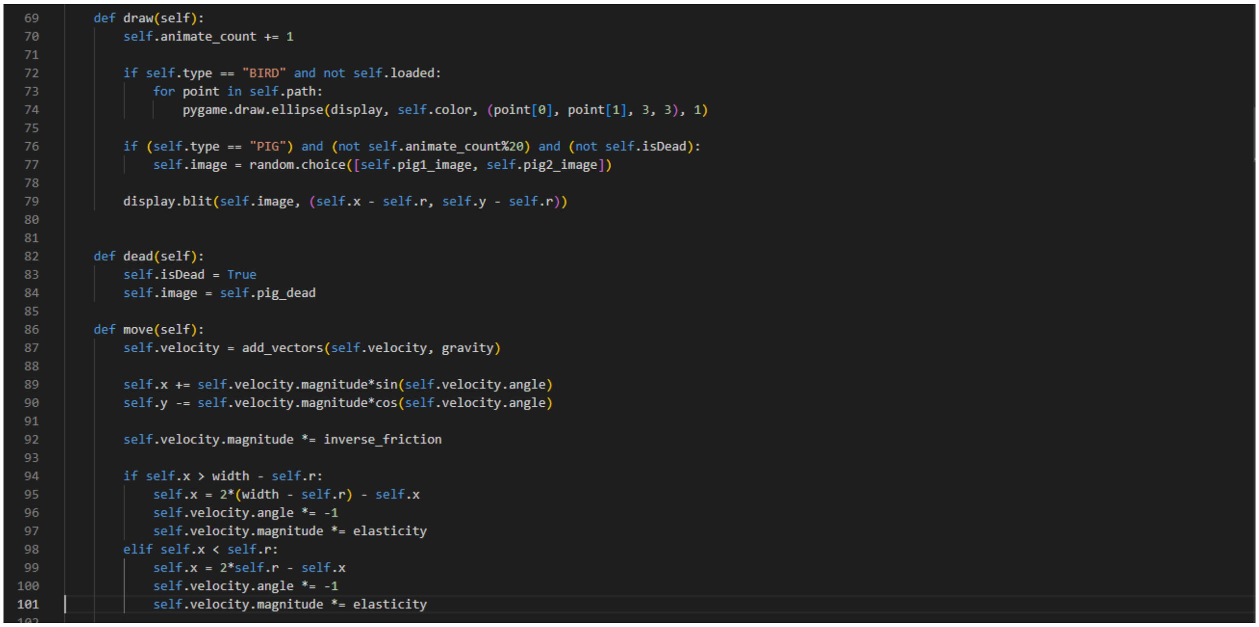
- Physics_engine



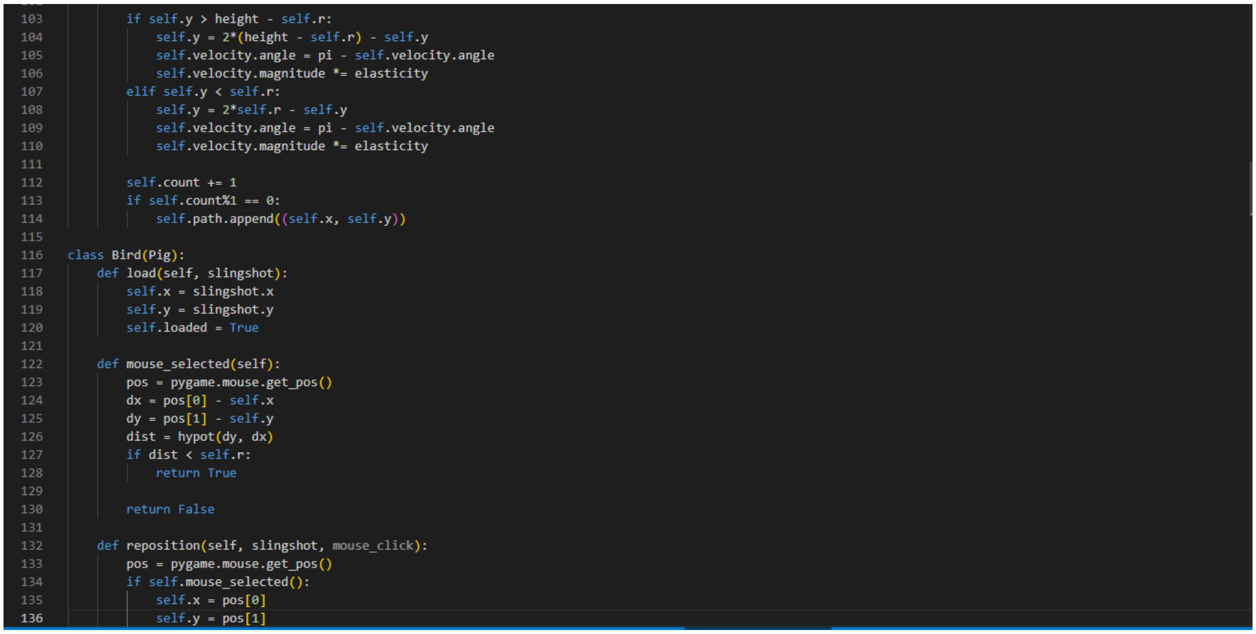
```python
import pygame
import sys
from math import *
import random

pygame.init()
width = None
height = None
display = None
ground = 50
clock = pygame.time.Clock()

def init(screen):
    global width, height, display
    display = screen
    (width, height) = display.get_rect().size
    height -= ground

class Vector:
    def __init__(self, magnitude=0, angle=radians(0)):
        self.magnitude = magnitude
        self.angle = angle

def add_vectors(vector1, vector2):
    x = sin(vector1.angle)*vector1.magnitude + sin(vector2.angle)*vector2.magnitude
    y = cos(vector1.angle)*vector1.magnitude + cos(vector2.angle)*vector2.magnitude

    new_angle = 0.5*pi - atan2(y, x)
    new_magnitude = hypot(x, y)

    new_vector = Vector(new_magnitude, new_angle)
    return new_vector
```



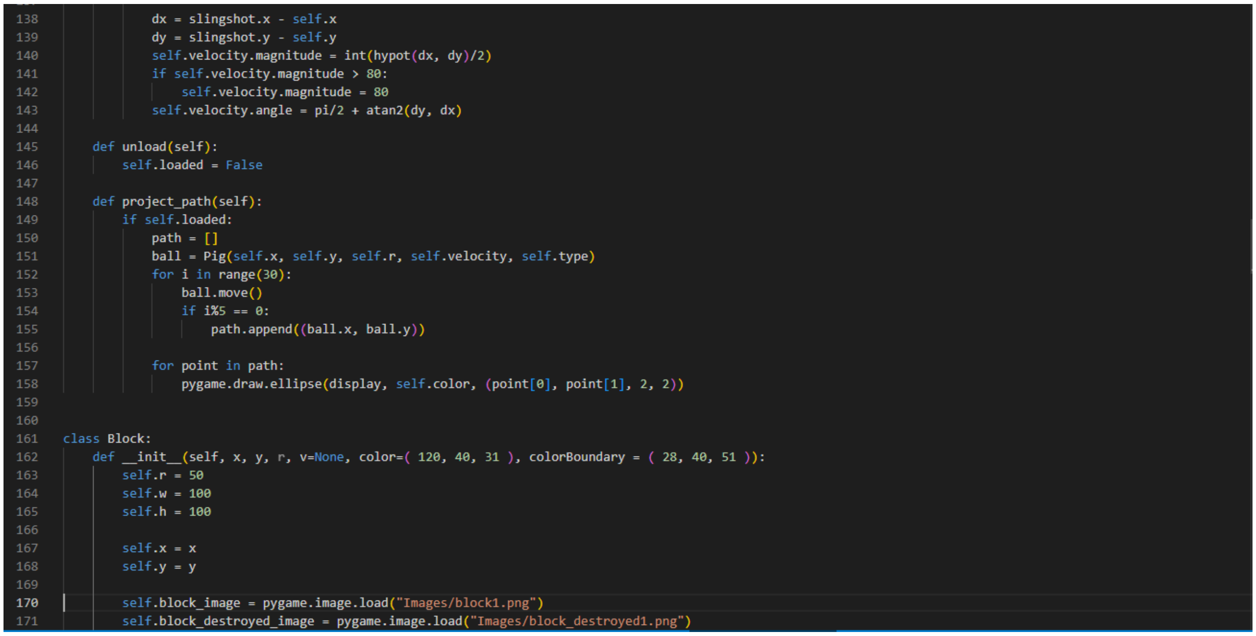
```python
gravity = Vector(0.2, pi)
inverse_friction = 0.99
elasticity = 0.8
block_elasticity = 0.7

class Pig:
    def __init__(self, x, y, r, v=None, type="PIG", loaded = False, color=(255, 255, 255)):
        self.x = x
        self.y = y
        self.r = r
        if v == None:
            self.velocity = Vector()
        else:
            self.velocity = v

        self.pig1_image = pygame.image.load("Images/pig1.png")
        self.pig2_image = pygame.image.load("Images/pig3.png")

        self.pig_dead = pygame.image.load("Images/pig_damaged.png")

        self.bird_image = pygame.image.load("Images/bird.png")

        if type == "PIG":
            self.image = random.choice([self.pig1_image, self.pig2_image])
        else:
            self.image = self.bird_image

        self.type = type
        self.color = color
        self.loaded = loaded
        self.path = []
        self.count = 0
        self.animate_count = 0
        self.isDead = False
```



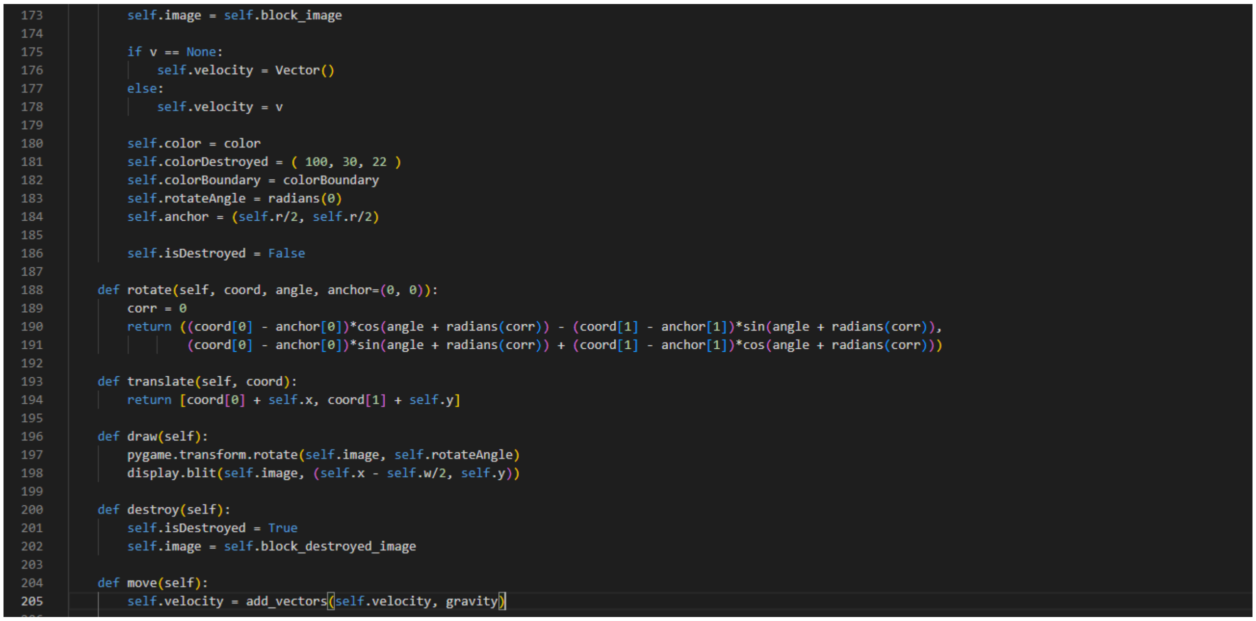
```
    def draw(self):
        self.animate_count += 1

        if self.type == "BIRD" and not self.loaded:
            for point in self.path:
                pygame.draw.ellipse(display, self.color, (point[0], point[1], 3, 3), 1)

        if (self.type == "PIG") and (not self.animate_count%20) and (not self.isDead):
            self.image = random.choice([self.pig1_image, self.pig2_image])

        display.blit(self.image, (self.x - self.r, self.y - self.r))


    def dead(self):
        self.isDead = True
        self.image = self.pig_dead

    def move(self):
        self.velocity = add_vectors(self.velocity, gravity)

        self.x += self.velocity.magnitude*sin(self.velocity.angle)
        self.y -= self.velocity.magnitude*cos(self.velocity.angle)

        self.velocity.magnitude *= inverse_friction

        if self.x > width - self.r:
            self.x = 2*(width - self.r) - self.x
            self.velocity.angle *= -1
            self.velocity.magnitude *= elasticity
        elif self.x < self.r:
            self.x = 2*self.r - self.x
            self.velocity.angle *= -1
            self.velocity.magnitude *= elasticity
```



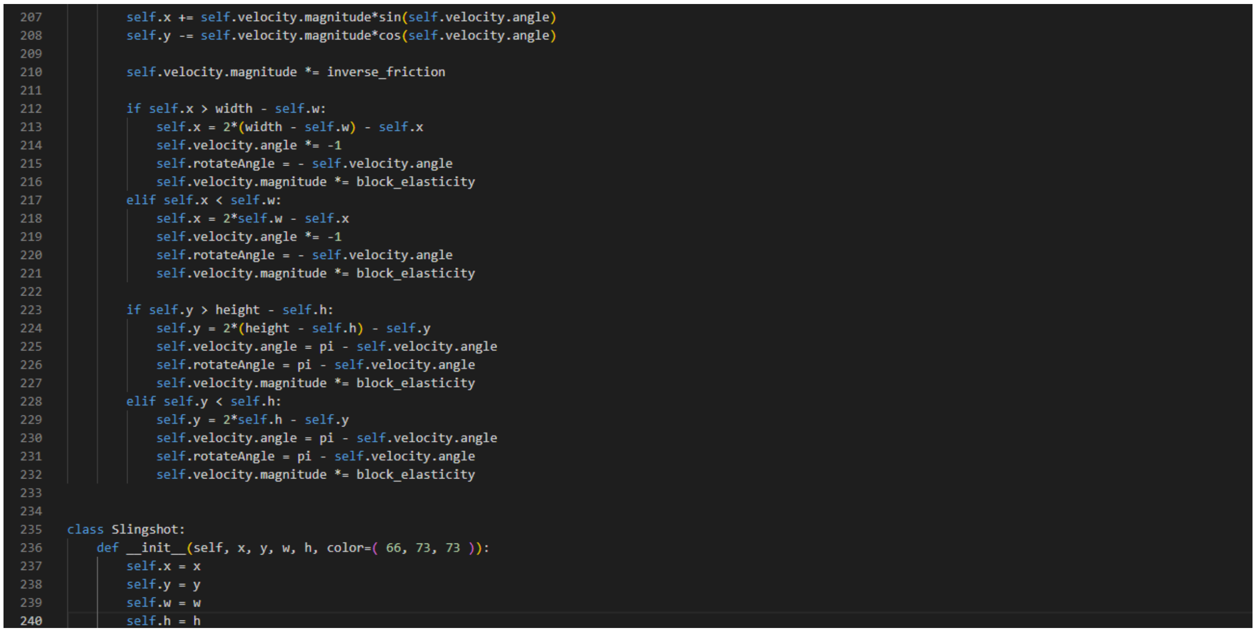
```
        if self.y > height - self.r:
            self.y = 2*(height - self.r) - self.y
            self.velocity.angle = pi - self.velocity.angle
            self.velocity.magnitude *= elasticity
        elif self.y < self.r:
            self.y = 2*self.r - self.y
            self.velocity.angle = pi - self.velocity.angle
            self.velocity.magnitude *= elasticity

        self.count += 1
        if self.count%1 == 0:
            self.path.append((self.x, self.y))

class Bird(Pig):
    def load(self, slingshot):
        self.x = slingshot.x
        self.y = slingshot.y
        self.loaded = True

    def mouse_selected(self):
        pos = pygame.mouse.get_pos()
        dx = pos[0] - self.x
        dy = pos[1] - self.y
        dist = hypot(dy, dx)
        if dist < self.r:
            return True

        return False

    def reposition(self, slingshot, mouse_click):
        pos = pygame.mouse.get_pos()
        if self.mouse_selected():
            self.x = pos[0]
            self.y = pos[1]
```



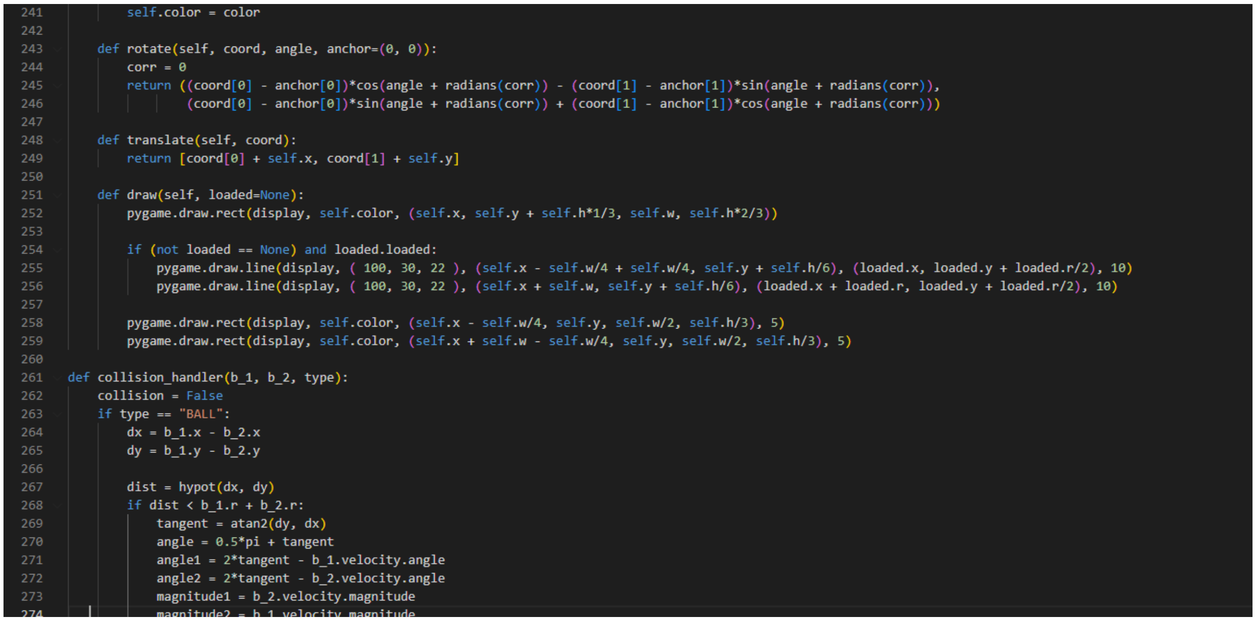
```python
            dx = slingshot.x - self.x
            dy = slingshot.y - self.y
            self.velocity.magnitude = int(hypot(dx, dy)/2)
            if self.velocity.magnitude > 80:
                self.velocity.magnitude = 80
            self.velocity.angle = pi/2 + atan2(dy, dx)

    def unload(self):
        self.loaded = False

    def project_path(self):
        if self.loaded:
            path = []
            ball = Pig(self.x, self.y, self.r, self.velocity, self.type)
            for i in range(30):
                ball.move()
                if i%5 == 0:
                    path.append((ball.x, ball.y))

            for point in path:
                pygame.draw.ellipse(display, self.color, (point[0], point[1], 2, 2))


class Block:
    def __init__(self, x, y, r, v=None, color=( 120, 40, 31 ), colorBoundary = ( 28, 40, 51 )):
        self.r = 50
        self.w = 100
        self.h = 100

        self.x = x
        self.y = y

        self.block_image = pygame.image.load("Images/block1.png")
        self.block_destroyed_image = pygame.image.load("Images/block_destroyed1.png")
```



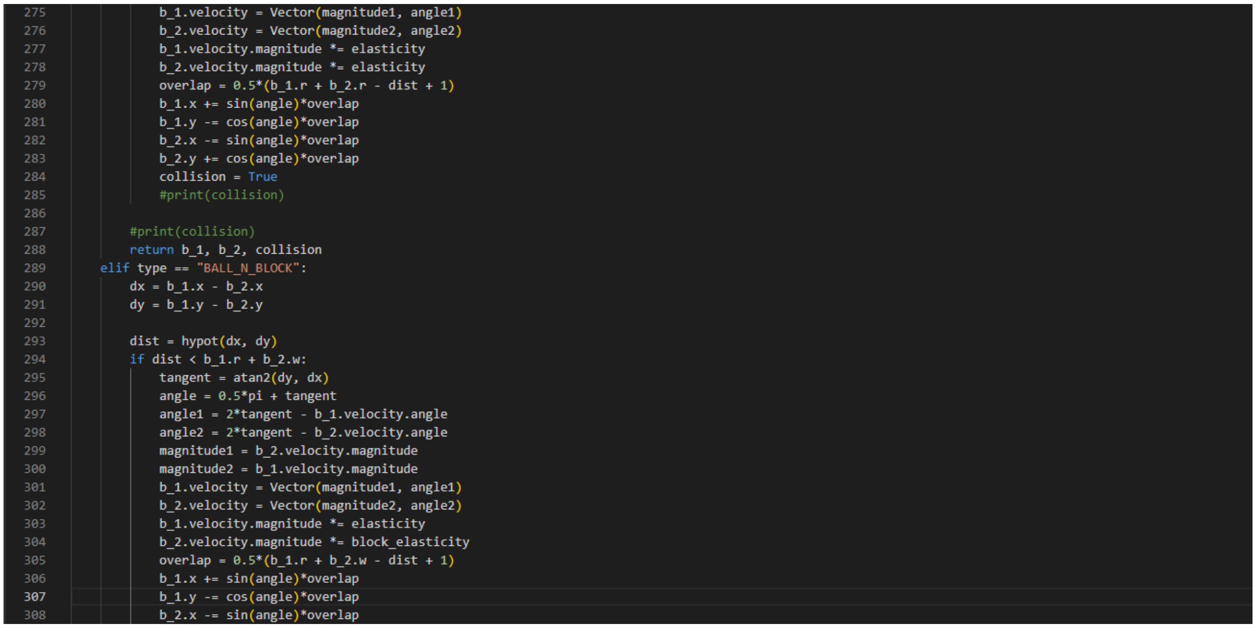
```python
        self.image = self.block_image

        if v == None:
            self.velocity = Vector()
        else:
            self.velocity = v

        self.color = color
        self.colorDestroyed = ( 100, 30, 22 )
        self.colorBoundary = colorBoundary
        self.rotateAngle = radians(0)
        self.anchor = (self.r/2, self.r/2)

        self.isDestroyed = False

    def rotate(self, coord, angle, anchor=(0, 0)):
        corr = 0
        return ((coord[0] - anchor[0])*cos(angle + radians(corr)) - (coord[1] - anchor[1])*sin(angle + radians(corr)),
                (coord[0] - anchor[0])*sin(angle + radians(corr)) + (coord[1] - anchor[1])*cos(angle + radians(corr)))

    def translate(self, coord):
        return [coord[0] + self.x, coord[1] + self.y]

    def draw(self):
        pygame.transform.rotate(self.image, self.rotateAngle)
        display.blit(self.image, (self.x - self.w/2, self.y))

    def destroy(self):
        self.isDestroyed = True
        self.image = self.block_destroyed_image

    def move(self):
        self.velocity = add_vectors(self.velocity, gravity)
```



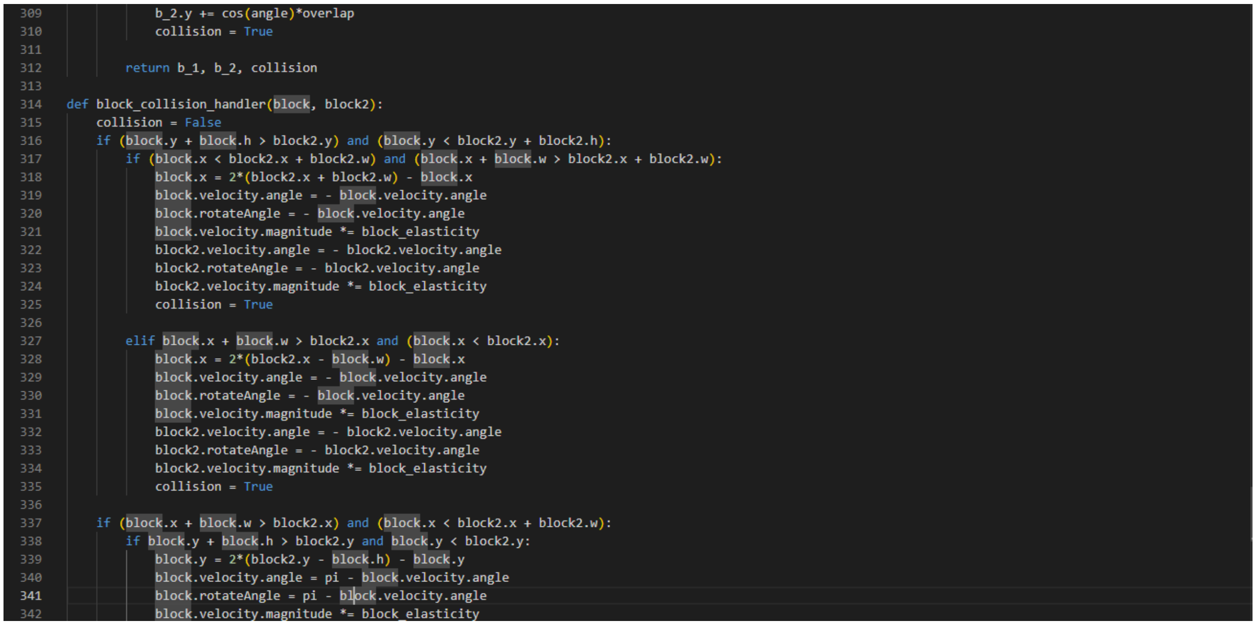
```python
        self.x += self.velocity.magnitude*sin(self.velocity.angle)
        self.y -= self.velocity.magnitude*cos(self.velocity.angle)

        self.velocity.magnitude *= inverse_friction

        if self.x > width - self.w:
            self.x = 2*(width - self.w) - self.x
            self.velocity.angle *= -1
            self.rotateAngle = - self.velocity.angle
            self.velocity.magnitude *= block_elasticity
        elif self.x < self.w:
            self.x = 2*self.w - self.x
            self.velocity.angle *= -1
            self.rotateAngle = - self.velocity.angle
            self.velocity.magnitude *= block_elasticity

        if self.y > height - self.h:
            self.y = 2*(height - self.h) - self.y
            self.velocity.angle = pi - self.velocity.angle
            self.rotateAngle = pi - self.velocity.angle
            self.velocity.magnitude *= block_elasticity
        elif self.y < self.h:
            self.y = 2*self.h - self.y
            self.velocity.angle = pi - self.velocity.angle
            self.rotateAngle = pi - self.velocity.angle
            self.velocity.magnitude *= block_elasticity


class Slingshot:
    def __init__(self, x, y, w, h, color=( 66, 73, 73 )):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
```



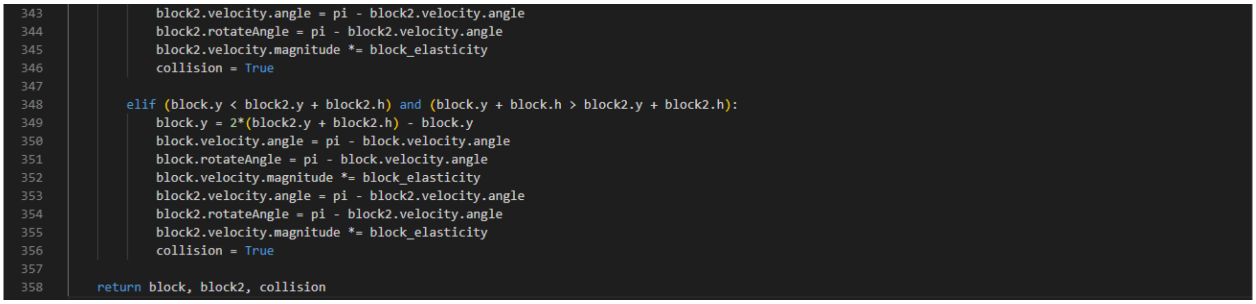
```python
        self.color = color

    def rotate(self, coord, angle, anchor=(0, 0)):
        corr = 0
        return ((coord[0] - anchor[0])*cos(angle + radians(corr)) - (coord[1] - anchor[1])*sin(angle + radians(corr)),
                (coord[0] - anchor[0])*sin(angle + radians(corr)) + (coord[1] - anchor[1])*cos(angle + radians(corr)))

    def translate(self, coord):
        return [coord[0] + self.x, coord[1] + self.y]

    def draw(self, loaded=None):
        pygame.draw.rect(display, self.color, (self.x, self.y + self.h*1/3, self.w, self.h*2/3))

        if (not loaded == None) and loaded.loaded:
            pygame.draw.line(display, ( 100, 30, 22 ), (self.x - self.w/4 + self.w/4, self.y + self.h/6), (loaded.x, loaded.y + loaded.r/2), 10)
            pygame.draw.line(display, ( 100, 30, 22 ), (self.x + self.w, self.y + self.h/6), (loaded.x + loaded.r, loaded.y + loaded.r/2), 10)

        pygame.draw.rect(display, self.color, (self.x - self.w/4, self.y, self.w/2, self.h/3), 5)
        pygame.draw.rect(display, self.color, (self.x + self.w - self.w/4, self.y, self.w/2, self.h/3), 5)

def collision_handler(b_1, b_2, type):
    collision = False
    if type == "BALL":
        dx = b_1.x - b_2.x
        dy = b_1.y - b_2.y

        dist = hypot(dx, dy)
        if dist < b_1.r + b_2.r:
            tangent = atan2(dy, dx)
            angle = 0.5*pi + tangent
            angle1 = 2*tangent - b_1.velocity.angle
            angle2 = 2*tangent - b_2.velocity.angle
            magnitude1 = b_2.velocity.magnitude
            magnitude2 = b_1.velocity.magnitude
```



```
            b_1.velocity = Vector(magnitude1, angle1)
            b_2.velocity = Vector(magnitude2, angle2)
            b_1.velocity.magnitude *= elasticity
            b_2.velocity.magnitude *= elasticity
            overlap = 0.5*(b_1.r + b_2.r - dist + 1)
            b_1.x += sin(angle)*overlap
            b_1.y -= cos(angle)*overlap
            b_2.x -= sin(angle)*overlap
            b_2.y += cos(angle)*overlap
            collision = True
            #print(collision)

        #print(collision)
        return b_1, b_2, collision
    elif type == "BALL_N_BLOCK":
        dx = b_1.x - b_2.x
        dy = b_1.y - b_2.y

        dist = hypot(dx, dy)
        if dist < b_1.r + b_2.w:
            tangent = atan2(dy, dx)
            angle = 0.5*pi + tangent
            angle1 = 2*tangent - b_1.velocity.angle
            angle2 = 2*tangent - b_2.velocity.angle
            magnitude1 = b_2.velocity.magnitude
            magnitude2 = b_1.velocity.magnitude
            b_1.velocity = Vector(magnitude1, angle1)
            b_2.velocity = Vector(magnitude2, angle2)
            b_1.velocity.magnitude *= elasticity
            b_2.velocity.magnitude *= block_elasticity
            overlap = 0.5*(b_1.r + b_2.w - dist + 1)
            b_1.x += sin(angle)*overlap
            b_1.y -= cos(angle)*overlap
            b_2.x -= sin(angle)*overlap
```

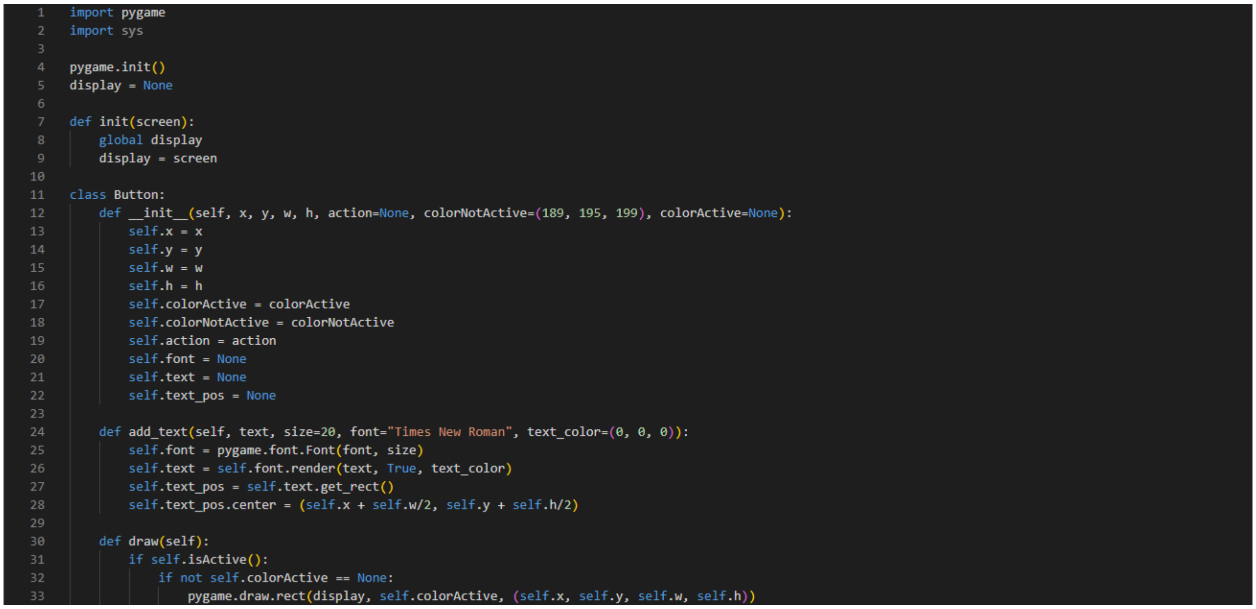


```
            b_2.y += cos(angle)*overlap
            collision = True

        return b_1, b_2, collision

def block_collision_handler(block, block2):
    collision = False
    if (block.y + block.h > block2.y) and (block.y < block2.y + block2.h):
        if (block.x < block2.x + block2.w) and (block.x + block.w > block2.x + block2.w):
            block.x = 2*(block2.x + block2.w) - block.x
            block.velocity.angle = - block.velocity.angle
            block.rotateAngle = - block.velocity.angle
            block.velocity.magnitude *= block_elasticity
            block2.velocity.angle = - block2.velocity.angle
            block2.rotateAngle = - block2.velocity.angle
            block2.velocity.magnitude *= block_elasticity
            collision = True

        elif block.x + block.w > block2.x and (block.x < block2.x):
            block.x = 2*(block2.x - block.w) - block.x
            block.velocity.angle = - block.velocity.angle
            block.rotateAngle = - block.velocity.angle
            block.velocity.magnitude *= block_elasticity
            block2.velocity.angle = - block2.velocity.angle
            block2.rotateAngle = - block2.velocity.angle
            block2.velocity.magnitude *= block_elasticity
            collision = True

    if (block.x + block.w > block2.x) and (block.x < block2.x + block2.w):
        if block.y + block.h > block2.y and block.y < block2.y:
            block.y = 2*(block2.y - block.h) - block.y
            block.velocity.angle = pi - block.velocity.angle
            block.rotateAngle = pi - block.velocity.angle
            block.velocity.magnitude *= block_elasticity
```

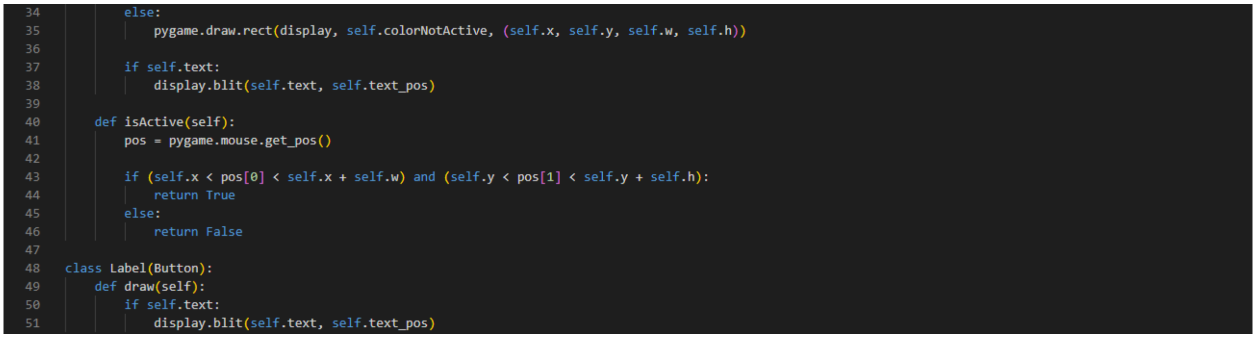


```
            block2.velocity.angle = pi - block2.velocity.angle
            block2.rotateAngle = pi - block2.velocity.angle
            block2.velocity.magnitude *= block_elasticity
            collision = True

        elif (block.y < block2.y + block2.h) and (block.y + block.h > block2.y + block2.h):
            block.y = 2*(block2.y + block2.h) - block.y
            block.velocity.angle = pi - block.velocity.angle
            block.rotateAngle = pi - block.velocity.angle
            block.velocity.magnitude *= block_elasticity
            block2.velocity.angle = pi - block2.velocity.angle
            block2.rotateAngle = pi - block2.velocity.angle
            block2.velocity.magnitude *= block_elasticity
            collision = True

    return block, block2, collision
```

- **Interface**

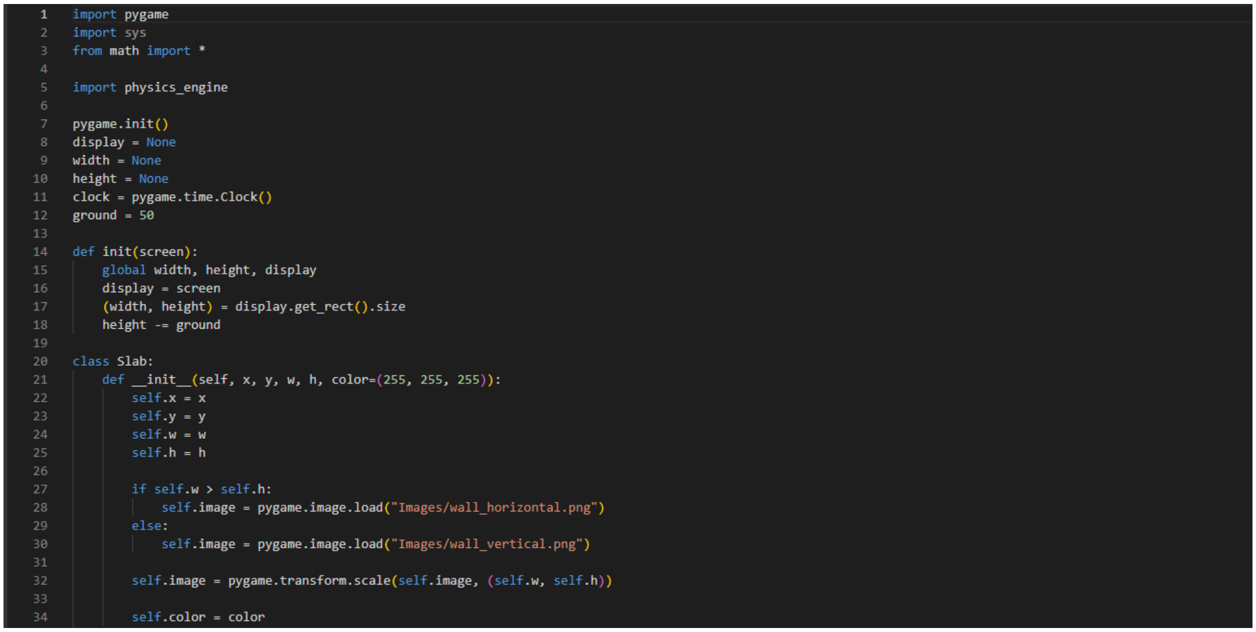


```python
import pygame
import sys

pygame.init()
display = None

def init(screen):
    global display
    display = screen

class Button:
    def __init__(self, x, y, w, h, action=None, colorNotActive=(189, 195, 199), colorActive=None):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.colorActive = colorActive
        self.colorNotActive = colorNotActive
        self.action = action
        self.font = None
        self.text = None
        self.text_pos = None

    def add_text(self, text, size=20, font="Times New Roman", text_color=(0, 0, 0)):
        self.font = pygame.font.Font(font, size)
        self.text = self.font.render(text, True, text_color)
        self.text_pos = self.text.get_rect()
        self.text_pos.center = (self.x + self.w/2, self.y + self.h/2)

    def draw(self):
        if self.isActive():
            if not self.colorActive == None:
                pygame.draw.rect(display, self.colorActive, (self.x, self.y, self.w, self.h))
```

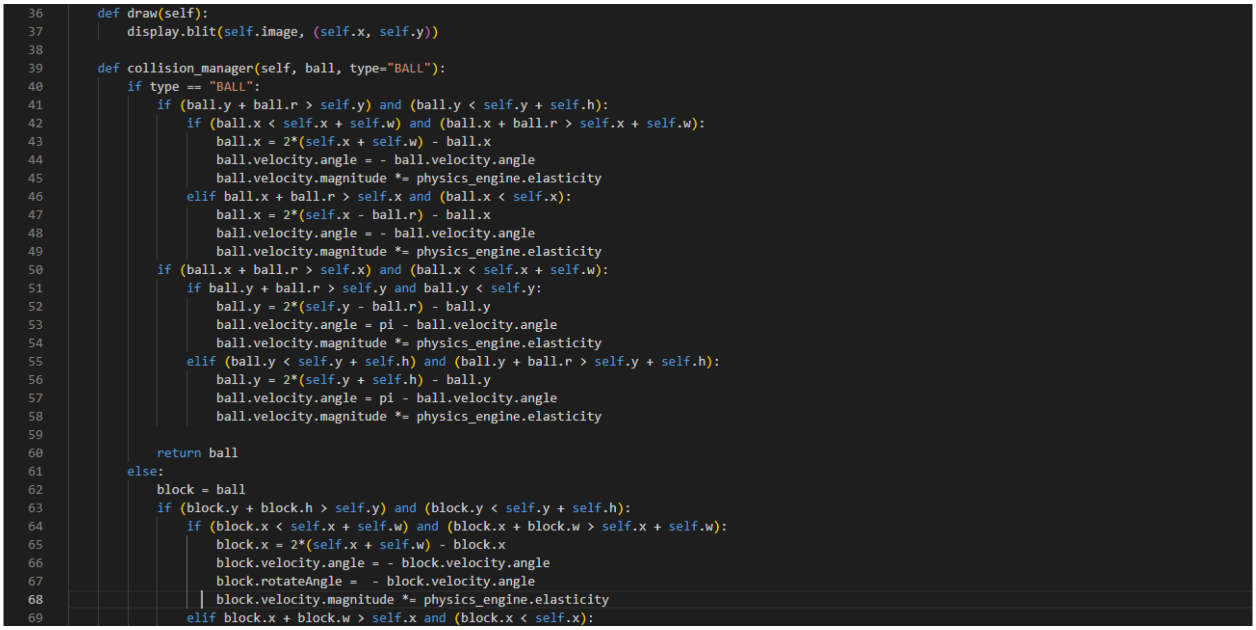


```python
            else:
                pygame.draw.rect(display, self.colorNotActive, (self.x, self.y, self.w, self.h))

        if self.text:
            display.blit(self.text, self.text_pos)

    def isActive(self):
        pos = pygame.mouse.get_pos()

        if (self.x < pos[0] < self.x + self.w) and (self.y < pos[1] < self.y + self.h):
            return True
        else:
            return False

class Label(Button):
    def draw(self):
        if self.text:
            display.blit(self.text, self.text_pos)
```

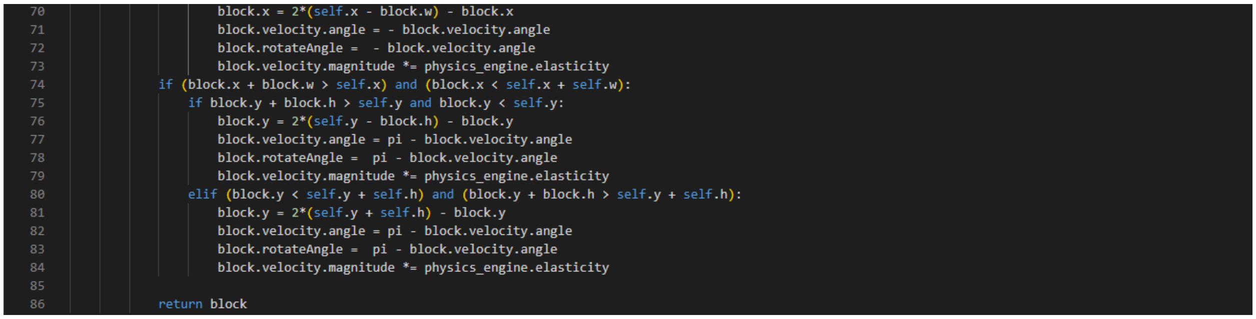
- **Objects**



```python
import pygame
import sys
from math import *

import physics_engine

pygame.init()
display = None
width = None
height = None
clock = pygame.time.Clock()
ground = 50

def init(screen):
    global width, height, display
    display = screen
    (width, height) = display.get_rect().size
    height -= ground

class Slab:
    def __init__(self, x, y, w, h, color=(255, 255, 255)):
        self.x = x
        self.y = y
        self.w = w
        self.h = h

        if self.w > self.h:
            self.image = pygame.image.load("Images/wall_horizontal.png")
        else:
            self.image = pygame.image.load("Images/wall_vertical.png")

        self.image = pygame.transform.scale(self.image, (self.w, self.h))

        self.color = color
```

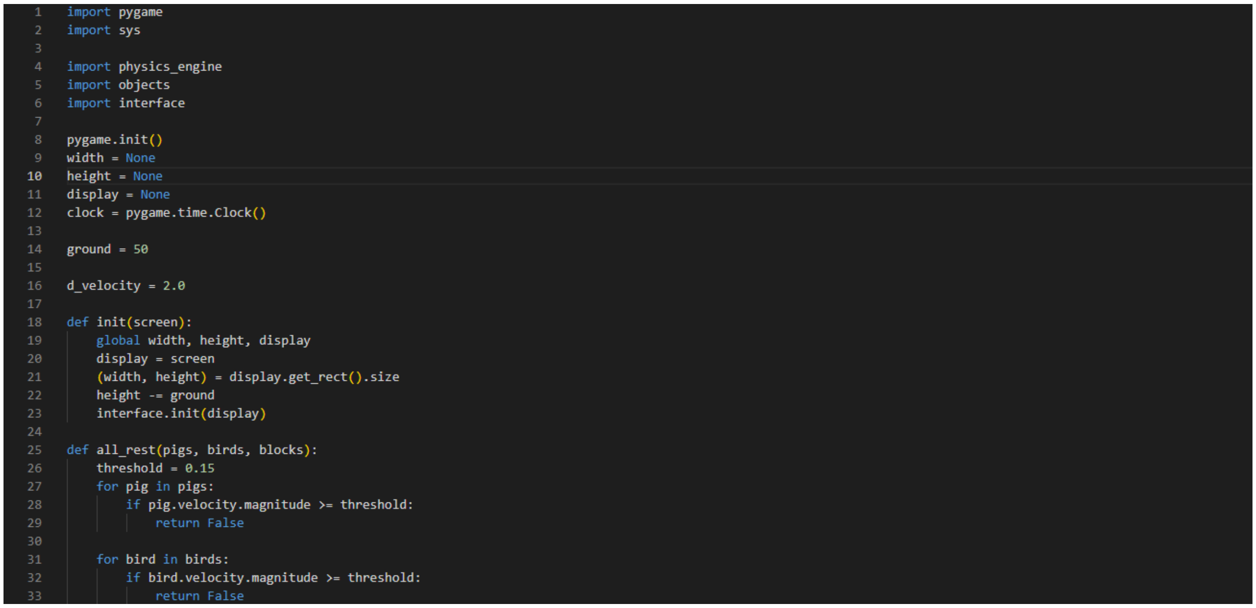


```python
    def draw(self):
        display.blit(self.image, (self.x, self.y))

    def collision_manager(self, ball, type="BALL"):
        if type == "BALL":
            if (ball.y + ball.r > self.y) and (ball.y < self.y + self.h):
                if (ball.x < self.x + self.w) and (ball.x + ball.r > self.x + self.w):
                    ball.x = 2*(self.x + self.w) - ball.x
                    ball.velocity.angle = - ball.velocity.angle
                    ball.velocity.magnitude *= physics_engine.elasticity
                elif ball.x + ball.r > self.x and (ball.x < self.x):
                    ball.x = 2*(self.x - ball.r) - ball.x
                    ball.velocity.angle = - ball.velocity.angle
                    ball.velocity.magnitude *= physics_engine.elasticity
            if (ball.x + ball.r > self.x) and (ball.x < self.x + self.w):
                if ball.y + ball.r > self.y and ball.y < self.y:
                    ball.y = 2*(self.y - ball.r) - ball.y
                    ball.velocity.angle = pi - ball.velocity.angle
                    ball.velocity.magnitude *= physics_engine.elasticity
                elif (ball.y < self.y + self.h) and (ball.y + ball.r > self.y + self.h):
                    ball.y = 2*(self.y + self.h) - ball.y
                    ball.velocity.angle = pi - ball.velocity.angle
                    ball.velocity.magnitude *= physics_engine.elasticity

            return ball
        else:
            block = ball
            if (block.y + block.h > self.y) and (block.y < self.y + self.h):
                if (block.x < self.x + self.w) and (block.x + block.w > self.x + self.w):
                    block.x = 2*(self.x + self.w) - block.x
                    block.velocity.angle = - block.velocity.angle
                    block.rotateAngle =  - block.velocity.angle
                    block.velocity.magnitude *= physics_engine.elasticity
                elif block.x + block.w > self.x and (block.x < self.x):
```

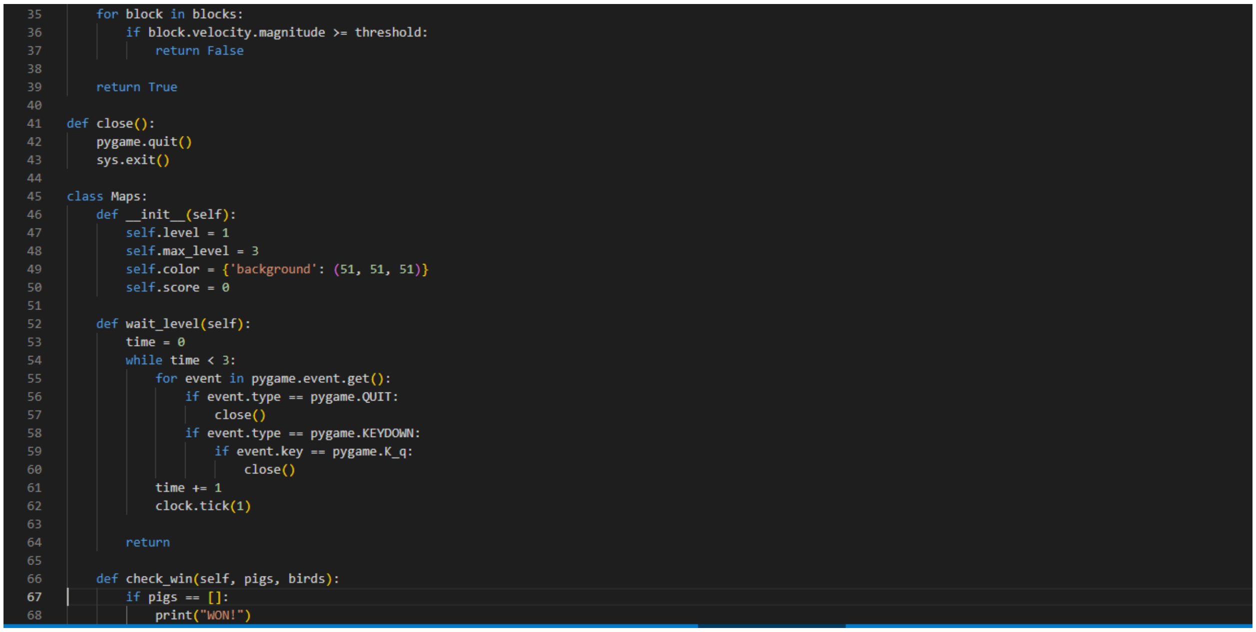


```python
                    block.x = 2*(self.x - block.w) - block.x
                    block.velocity.angle = - block.velocity.angle
                    block.rotateAngle =  - block.velocity.angle
                    block.velocity.magnitude *= physics_engine.elasticity
            if (block.x + block.w > self.x) and (block.x < self.x + self.w):
                if block.y + block.h > self.y and block.y < self.y:
                    block.y = 2*(self.y - block.h) - block.y
                    block.velocity.angle = pi - block.velocity.angle
                    block.rotateAngle =  pi - block.velocity.angle
                    block.velocity.magnitude *= physics_engine.elasticity
                elif (block.y < self.y + self.h) and (block.y + block.h > self.y + self.h):
                    block.y = 2*(self.y + self.h) - block.y
                    block.velocity.angle = pi - block.velocity.angle
                    block.rotateAngle =  pi - block.velocity.angle
                    block.velocity.magnitude *= physics_engine.elasticity

            return block
```

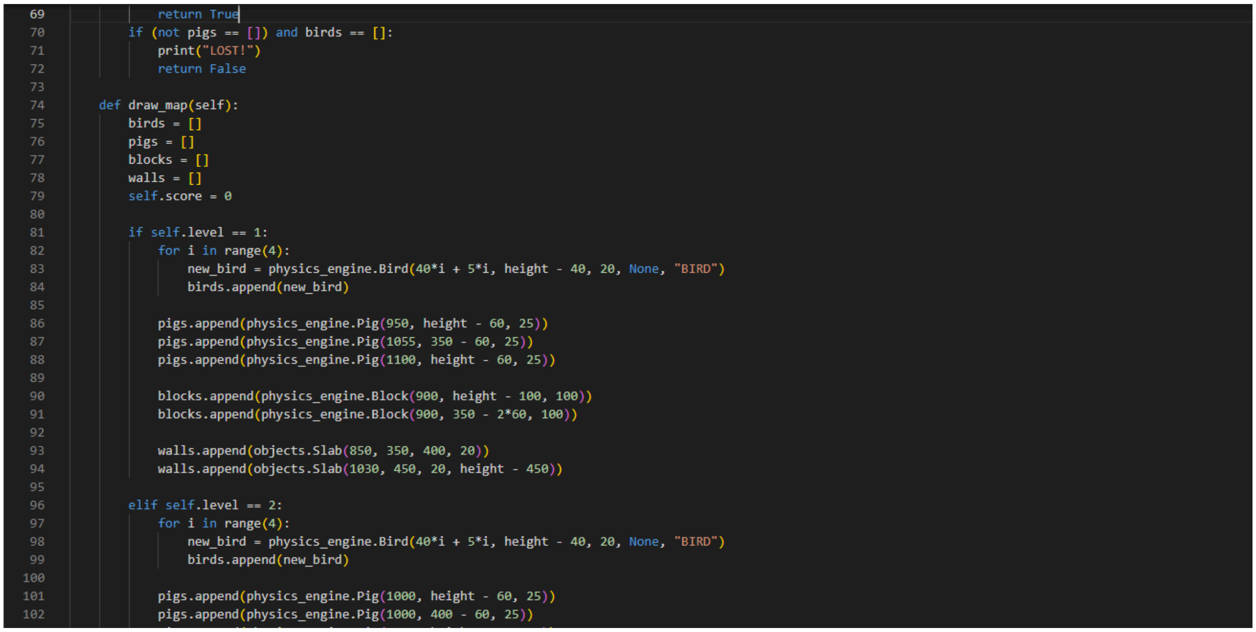
- ## Maps



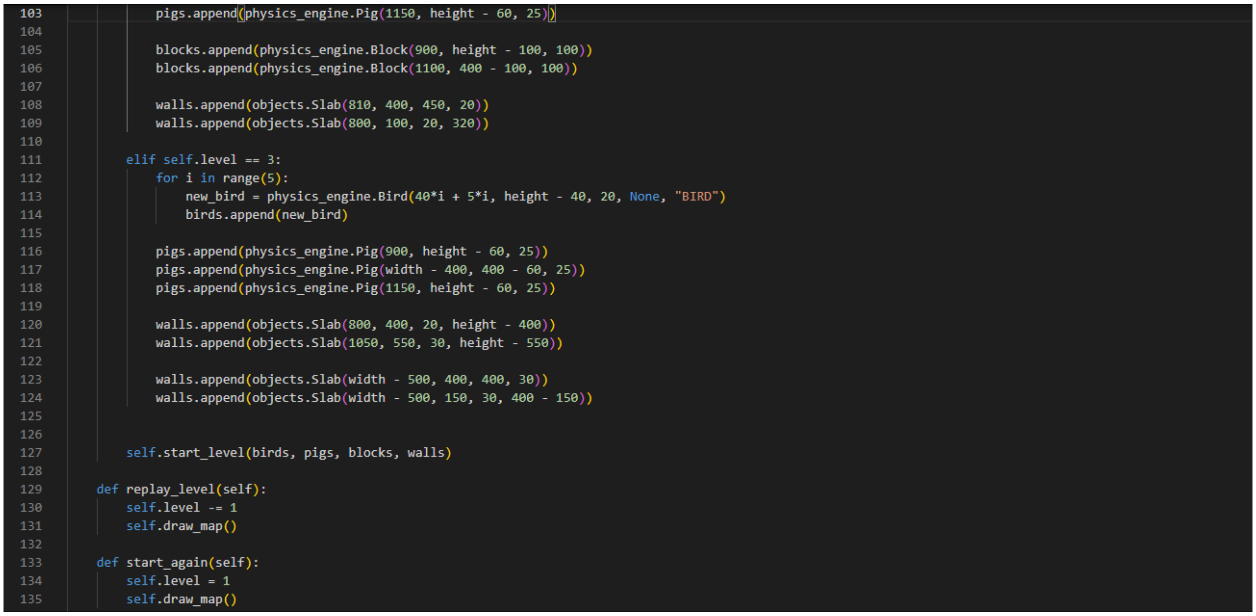
```python
import pygame
import sys

import physics_engine
import objects
import interface

pygame.init()
width = None
height = None
display = None
clock = pygame.time.Clock()

ground = 50

d_velocity = 2.0

def init(screen):
    global width, height, display
    display = screen
    (width, height) = display.get_rect().size
    height -= ground
    interface.init(display)

def all_rest(pigs, birds, blocks):
    threshold = 0.15
    for pig in pigs:
        if pig.velocity.magnitude >= threshold:
            return False

    for bird in birds:
        if bird.velocity.magnitude >= threshold:
            return False
```



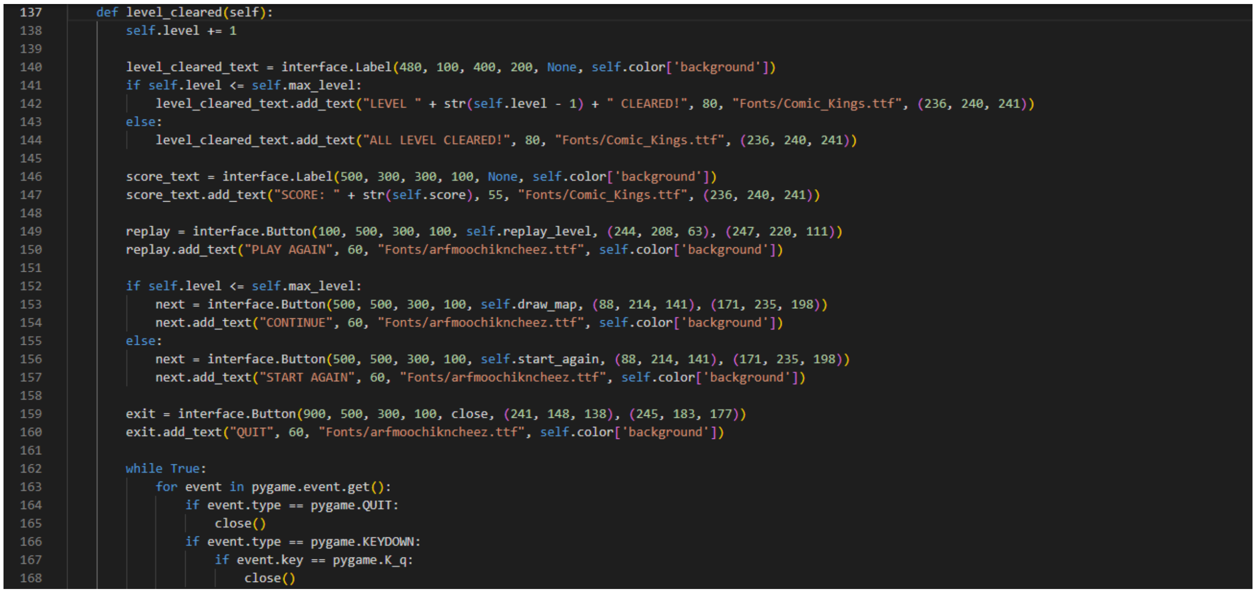
```python
    for block in blocks:
        if block.velocity.magnitude >= threshold:
            return False

    return True

def close():
    pygame.quit()
    sys.exit()

class Maps:
    def __init__(self):
        self.level = 1
        self.max_level = 3
        self.color = {'background': (51, 51, 51)}
        self.score = 0

    def wait_level(self):
        time = 0
        while time < 3:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    close()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        close()
            time += 1
            clock.tick(1)

        return

    def check_win(self, pigs, birds):
        if pigs == []:
            print("WON!")
```



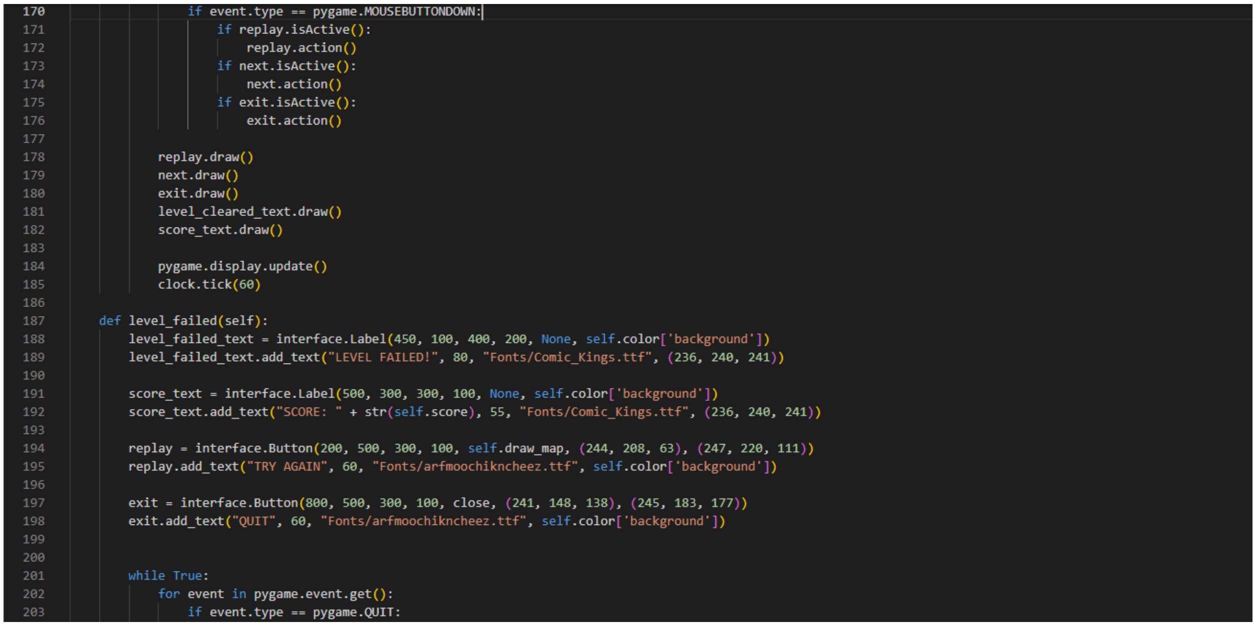
```python
            return True
        if (not pigs == []) and birds == []:
            print("LOST!")
            return False

    def draw_map(self):
        birds = []
        pigs = []
        blocks = []
        walls = []
        self.score = 0

        if self.level == 1:
            for i in range(4):
                new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
                birds.append(new_bird)

            pigs.append(physics_engine.Pig(950, height - 60, 25))
            pigs.append(physics_engine.Pig(1055, 350 - 60, 25))
            pigs.append(physics_engine.Pig(1100, height - 60, 25))

            blocks.append(physics_engine.Block(900, height - 100, 100))
            blocks.append(physics_engine.Block(900, 350 - 2*60, 100))

            walls.append(objects.Slab(850, 350, 400, 20))
            walls.append(objects.Slab(1030, 450, 20, height - 450))

        elif self.level == 2:
            for i in range(4):
                new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
                birds.append(new_bird)

            pigs.append(physics_engine.Pig(1000, height - 60, 25))
            pigs.append(physics_engine.Pig(1000, 400 - 60, 25))
```



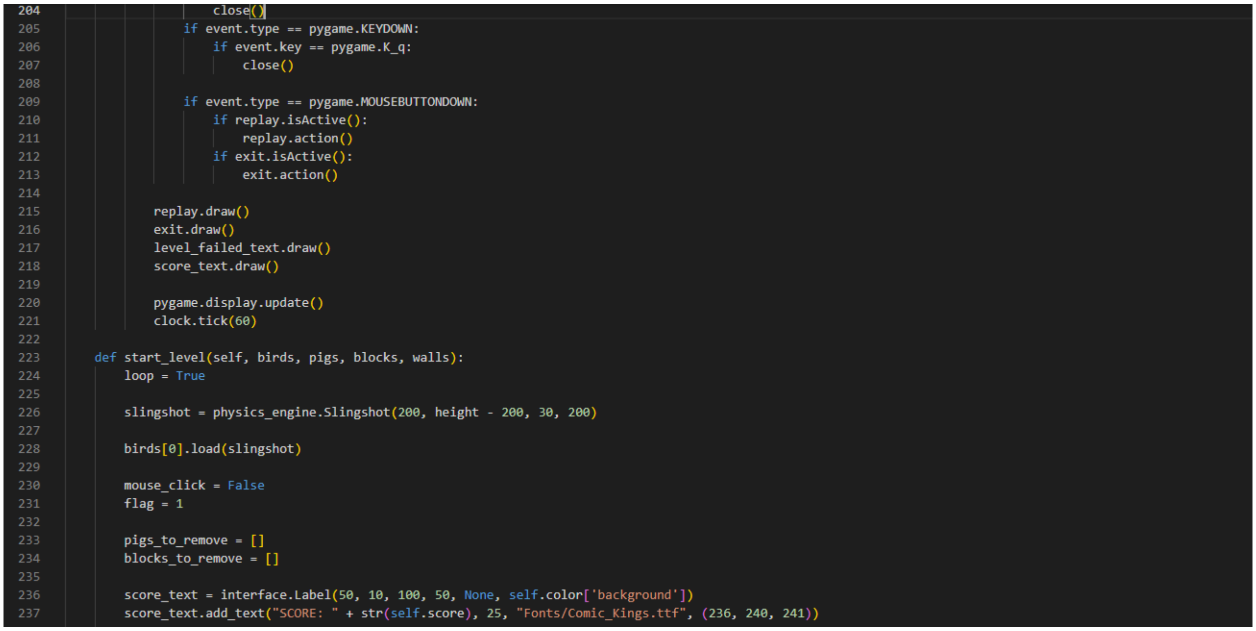
```python
            pigs.append(physics_engine.Pig(1150, height - 60, 25))

            blocks.append(physics_engine.Block(900, height - 100, 100))
            blocks.append(physics_engine.Block(1100, 400 - 100, 100))

            walls.append(objects.Slab(810, 400, 450, 20))
            walls.append(objects.Slab(800, 100, 20, 320))

        elif self.level == 3:
            for i in range(5):
                new_bird = physics_engine.Bird(40*i + 5*i, height - 40, 20, None, "BIRD")
                birds.append(new_bird)

            pigs.append(physics_engine.Pig(900, height - 60, 25))
            pigs.append(physics_engine.Pig(width - 400, 400 - 60, 25))
            pigs.append(physics_engine.Pig(1150, height - 60, 25))

            walls.append(objects.Slab(800, 400, 20, height - 400))
            walls.append(objects.Slab(1050, 550, 30, height - 550))

            walls.append(objects.Slab(width - 500, 400, 400, 30))
            walls.append(objects.Slab(width - 500, 150, 30, 400 - 150))


        self.start_level(birds, pigs, blocks, walls)

    def replay_level(self):
        self.level -= 1
        self.draw_map()

    def start_again(self):
        self.level = 1
        self.draw_map()
```



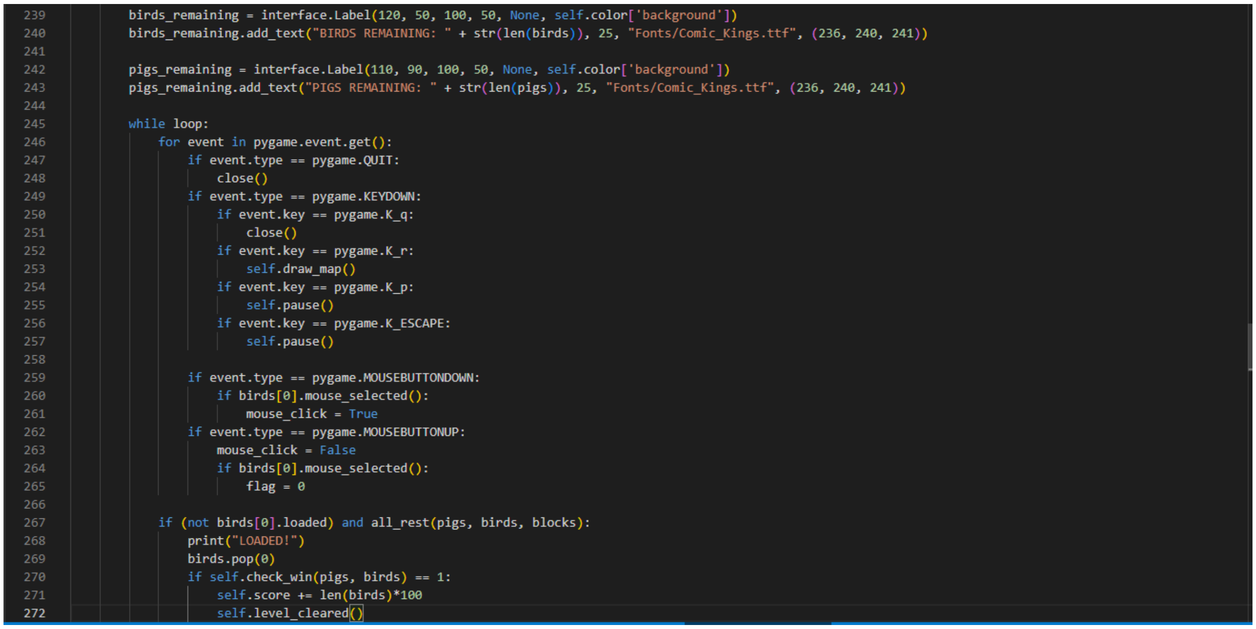
```python
    def level_cleared(self):
        self.level += 1

        level_cleared_text = interface.Label(480, 100, 400, 200, None, self.color['background'])
        if self.level <= self.max_level:
            level_cleared_text.add_text("LEVEL " + str(self.level - 1) + " CLEARED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))
        else:
            level_cleared_text.add_text("ALL LEVEL CLEARED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))

        score_text = interface.Label(500, 300, 300, 100, None, self.color['background'])
        score_text.add_text("SCORE: " + str(self.score), 55, "Fonts/Comic_Kings.ttf", (236, 240, 241))

        replay = interface.Button(100, 500, 300, 100, self.replay_level, (244, 208, 63), (247, 220, 111))
        replay.add_text("PLAY AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])

        if self.level <= self.max_level:
            next = interface.Button(500, 500, 300, 100, self.draw_map, (88, 214, 141), (171, 235, 198))
            next.add_text("CONTINUE", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])
        else:
            next = interface.Button(500, 500, 300, 100, self.start_again, (88, 214, 141), (171, 235, 198))
            next.add_text("START AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])

        exit = interface.Button(900, 500, 300, 100, close, (241, 148, 138), (245, 183, 177))
        exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])

        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    close()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        close()
```



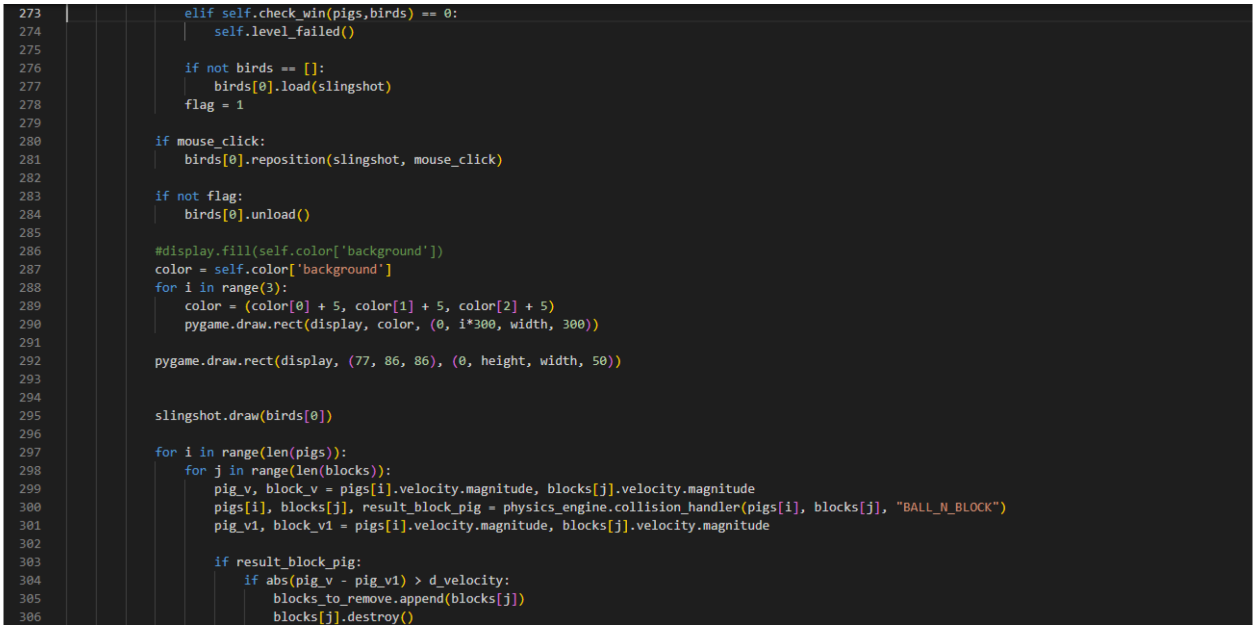
```
                if event.type == pygame.MOUSEBUTTONDOWN:
                    if replay.isActive():
                        replay.action()
                    if next.isActive():
                        next.action()
                    if exit.isActive():
                        exit.action()

            replay.draw()
            next.draw()
            exit.draw()
            level_cleared_text.draw()
            score_text.draw()

            pygame.display.update()
            clock.tick(60)

    def level_failed(self):
        level_failed_text = interface.Label(450, 100, 400, 200, None, self.color['background'])
        level_failed_text.add_text("LEVEL FAILED!", 80, "Fonts/Comic_Kings.ttf", (236, 240, 241))

        score_text = interface.Label(500, 300, 300, 100, None, self.color['background'])
        score_text.add_text("SCORE: " + str(self.score), 55, "Fonts/Comic_Kings.ttf", (236, 240, 241))

        replay = interface.Button(200, 500, 300, 100, self.draw_map, (244, 208, 63), (247, 220, 111))
        replay.add_text("TRY AGAIN", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])

        exit = interface.Button(800, 500, 300, 100, close, (241, 148, 138), (245, 183, 177))
        exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", self.color['background'])


        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
```



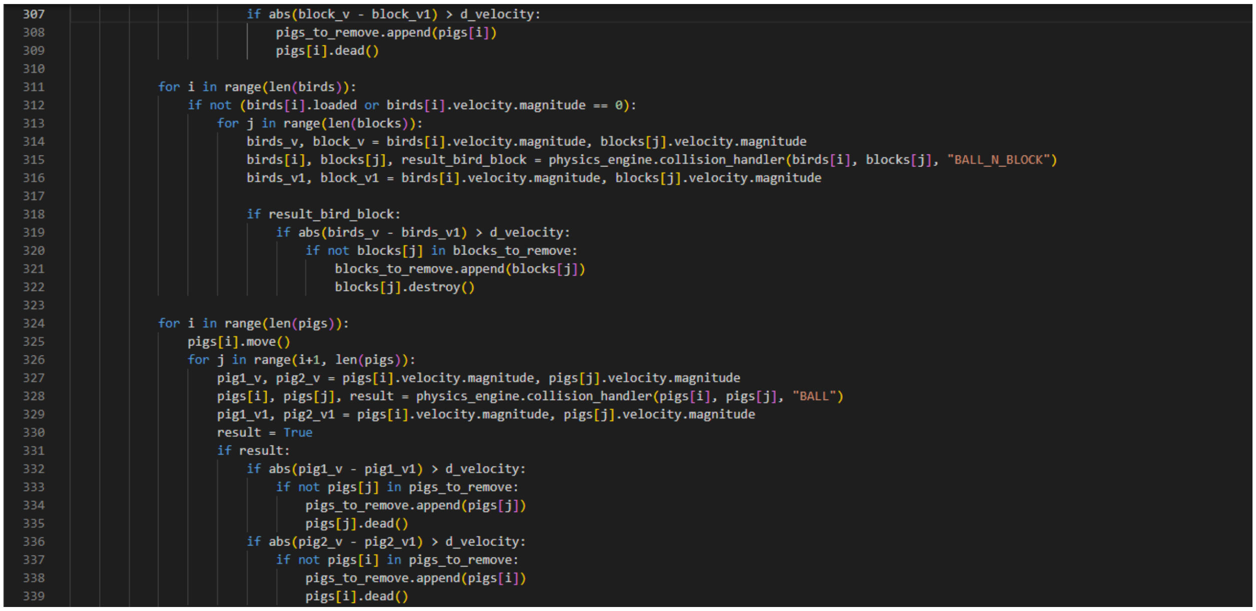
```
                    close()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        close()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    if replay.isActive():
                        replay.action()
                    if exit.isActive():
                        exit.action()

            replay.draw()
            exit.draw()
            level_failed_text.draw()
            score_text.draw()

            pygame.display.update()
            clock.tick(60)

    def start_level(self, birds, pigs, blocks, walls):
        loop = True

        slingshot = physics_engine.Slingshot(200, height - 200, 30, 200)

        birds[0].load(slingshot)

        mouse_click = False
        flag = 1

        pigs_to_remove = []
        blocks_to_remove = []

        score_text = interface.Label(50, 10, 100, 50, None, self.color['background'])
        score_text.add_text("SCORE: " + str(self.score), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
```



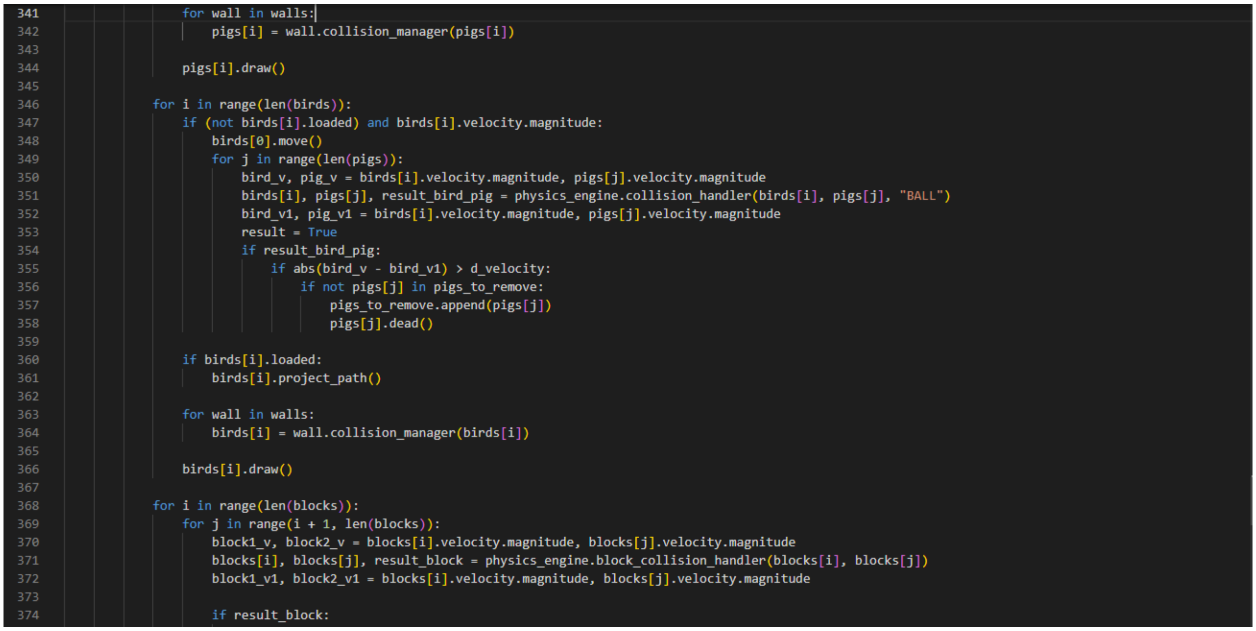
```python
        birds_remaining = interface.Label(120, 50, 100, 50, None, self.color['background'])
        birds_remaining.add_text("BIRDS REMAINING: " + str(len(birds)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))

        pigs_remaining = interface.Label(110, 90, 100, 50, None, self.color['background'])
        pigs_remaining.add_text("PIGS REMAINING: " + str(len(pigs)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))

        while loop:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    close()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        close()
                    if event.key == pygame.K_r:
                        self.draw_map()
                    if event.key == pygame.K_p:
                        self.pause()
                    if event.key == pygame.K_ESCAPE:
                        self.pause()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    if birds[0].mouse_selected():
                        mouse_click = True
                if event.type == pygame.MOUSEBUTTONUP:
                    mouse_click = False
                    if birds[0].mouse_selected():
                        flag = 0

            if (not birds[0].loaded) and all_rest(pigs, birds, blocks):
                print("LOADED!")
                birds.pop(0)
                if self.check_win(pigs, birds) == 1:
                    self.score += len(birds)*100
                    self.level_cleared()
```



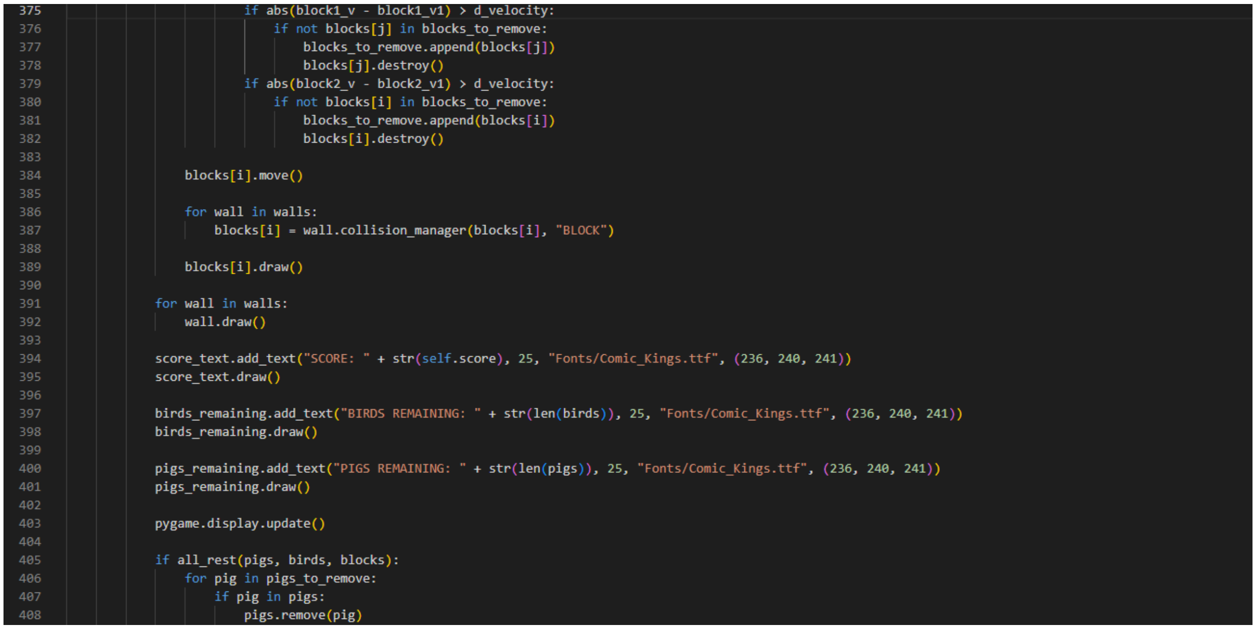
```python
                elif self.check_win(pigs,birds) == 0:
                    self.level_failed()

                if not birds == []:
                    birds[0].load(slingshot)
                flag = 1

            if mouse_click:
                birds[0].reposition(slingshot, mouse_click)

            if not flag:
                birds[0].unload()

            #display.fill(self.color['background'])
            color = self.color['background']
            for i in range(3):
                color = (color[0] + 5, color[1] + 5, color[2] + 5)
                pygame.draw.rect(display, color, (0, i*300, width, 300))

            pygame.draw.rect(display, (77, 86, 86), (0, height, width, 50))


            slingshot.draw(birds[0])

            for i in range(len(pigs)):
                for j in range(len(blocks)):
                    pig_v, block_v = pigs[i].velocity.magnitude, blocks[j].velocity.magnitude
                    pigs[i], blocks[j], result_block_pig = physics_engine.collision_handler(pigs[i], blocks[j], "BALL_N_BLOCK")
                    pig_v1, block_v1 = pigs[i].velocity.magnitude, blocks[j].velocity.magnitude

                    if result_block_pig:
                        if abs(pig_v - pig_v1) > d_velocity:
                            blocks_to_remove.append(blocks[j])
                            blocks[j].destroy()
```



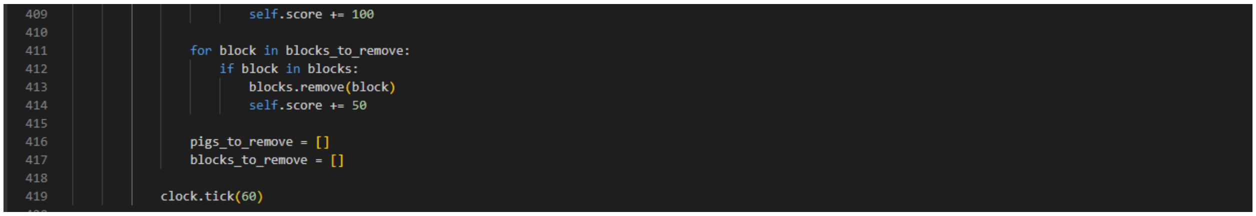
```python
                        if abs(block_v - block_v1) > d_velocity:
                            pigs_to_remove.append(pigs[i])
                            pigs[i].dead()

            for i in range(len(birds)):
                if not (birds[i].loaded or birds[i].velocity.magnitude == 0):
                    for j in range(len(blocks)):
                        birds_v, block_v = birds[i].velocity.magnitude, blocks[j].velocity.magnitude
                        birds[i], blocks[j], result_bird_block = physics_engine.collision_handler(birds[i], blocks[j], "BALL_N_BLOCK")
                        birds_v1, block_v1 = birds[i].velocity.magnitude, blocks[j].velocity.magnitude

                        if result_bird_block:
                            if abs(birds_v - birds_v1) > d_velocity:
                                if not blocks[j] in blocks_to_remove:
                                    blocks_to_remove.append(blocks[j])
                                    blocks[j].destroy()

            for i in range(len(pigs)):
                pigs[i].move()
                for j in range(i+1, len(pigs)):
                    pig1_v, pig2_v = pigs[i].velocity.magnitude, pigs[j].velocity.magnitude
                    pigs[i], pigs[j], result = physics_engine.collision_handler(pigs[i], pigs[j], "BALL")
                    pig1_v1, pig2_v1 = pigs[i].velocity.magnitude, pigs[j].velocity.magnitude
                    result = True
                    if result:
                        if abs(pig1_v - pig1_v1) > d_velocity:
                            if not pigs[j] in pigs_to_remove:
                                pigs_to_remove.append(pigs[j])
                                pigs[j].dead()
                        if abs(pig2_v - pig2_v1) > d_velocity:
                            if not pigs[i] in pigs_to_remove:
                                pigs_to_remove.append(pigs[i])
                                pigs[i].dead()
```



```python
                for wall in walls:
                    pigs[i] = wall.collision_manager(pigs[i])

                pigs[i].draw()

            for i in range(len(birds)):
                if (not birds[i].loaded) and birds[i].velocity.magnitude:
                    birds[0].move()
                    for j in range(len(pigs)):
                        bird_v, pig_v = birds[i].velocity.magnitude, pigs[j].velocity.magnitude
                        birds[i], pigs[j], result_bird_pig = physics_engine.collision_handler(birds[i], pigs[j], "BALL")
                        bird_v1, pig_v1 = birds[i].velocity.magnitude, pigs[j].velocity.magnitude
                        result = True
                        if result_bird_pig:
                            if abs(bird_v - bird_v1) > d_velocity:
                                if not pigs[j] in pigs_to_remove:
                                    pigs_to_remove.append(pigs[j])
                                    pigs[j].dead()

                if birds[i].loaded:
                    birds[i].project_path()

                for wall in walls:
                    birds[i] = wall.collision_manager(birds[i])

                birds[i].draw()

            for i in range(len(blocks)):
                for j in range(i + 1, len(blocks)):
                    block1_v, block2_v = blocks[i].velocity.magnitude, blocks[j].velocity.magnitude
                    blocks[i], blocks[j], result_block = physics_engine.block_collision_handler(blocks[i], blocks[j])
                    block1_v1, block2_v1 = blocks[i].velocity.magnitude, blocks[j].velocity.magnitude

                    if result_block:
```

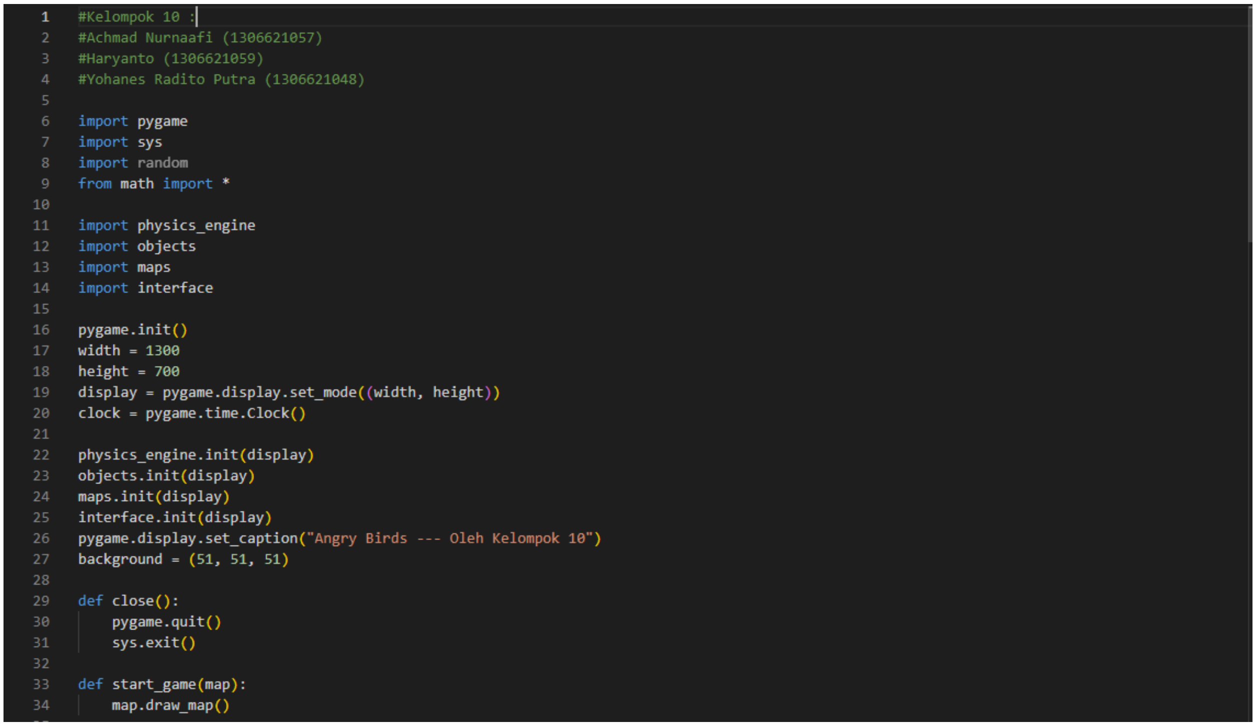


```
                        if abs(block1_v - block1_v1) > d_velocity:
                            if not blocks[j] in blocks_to_remove:
                                blocks_to_remove.append(blocks[j])
                                blocks[j].destroy()
                        if abs(block2_v - block2_v1) > d_velocity:
                            if not blocks[i] in blocks_to_remove:
                                blocks_to_remove.append(blocks[i])
                                blocks[i].destroy()

                blocks[i].move()

                for wall in walls:
                    blocks[i] = wall.collision_manager(blocks[i], "BLOCK")

                blocks[i].draw()

            for wall in walls:
                wall.draw()

            score_text.add_text("SCORE: " + str(self.score), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
            score_text.draw()

            birds_remaining.add_text("BIRDS REMAINING: " + str(len(birds)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
            birds_remaining.draw()

            pigs_remaining.add_text("PIGS REMAINING: " + str(len(pigs)), 25, "Fonts/Comic_Kings.ttf", (236, 240, 241))
            pigs_remaining.draw()

            pygame.display.update()

            if all_rest(pigs, birds, blocks):
                for pig in pigs_to_remove:
                    if pig in pigs:
                        pigs.remove(pig)
```

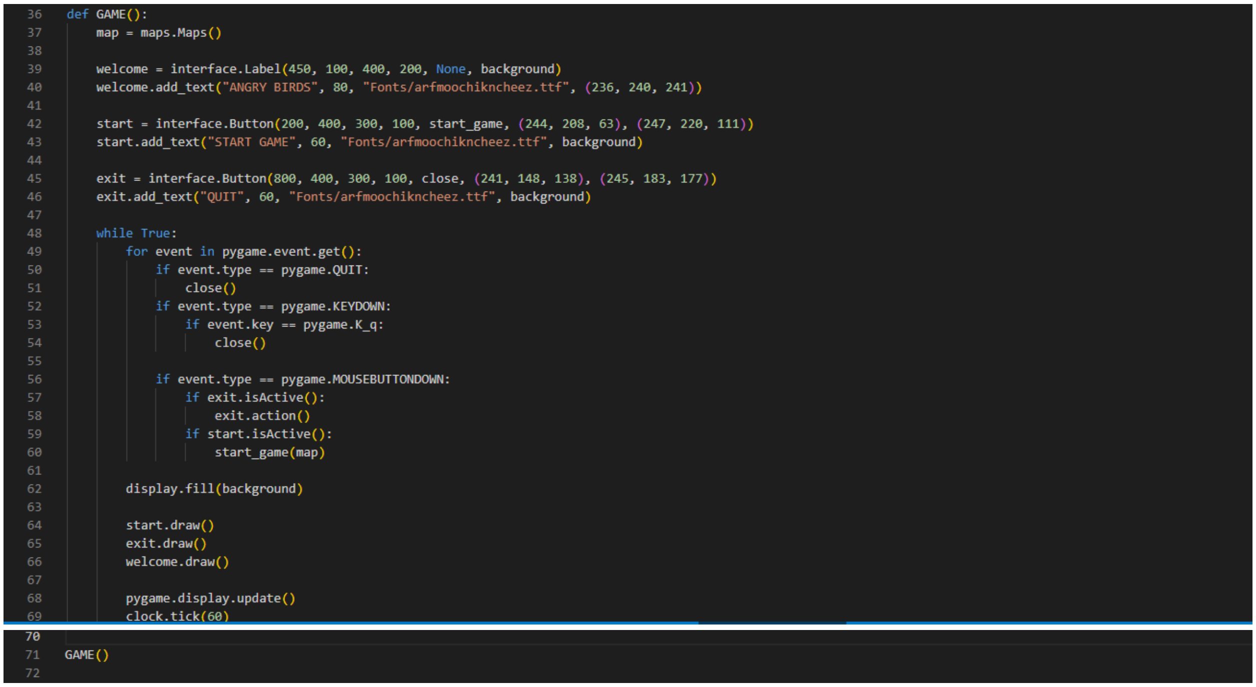


```
                        self.score += 100

                for block in blocks_to_remove:
                    if block in blocks:
                        blocks.remove(block)
                        self.score += 50

                pigs_to_remove = []
                blocks_to_remove = []

            clock.tick(60)
```

- Program Utama



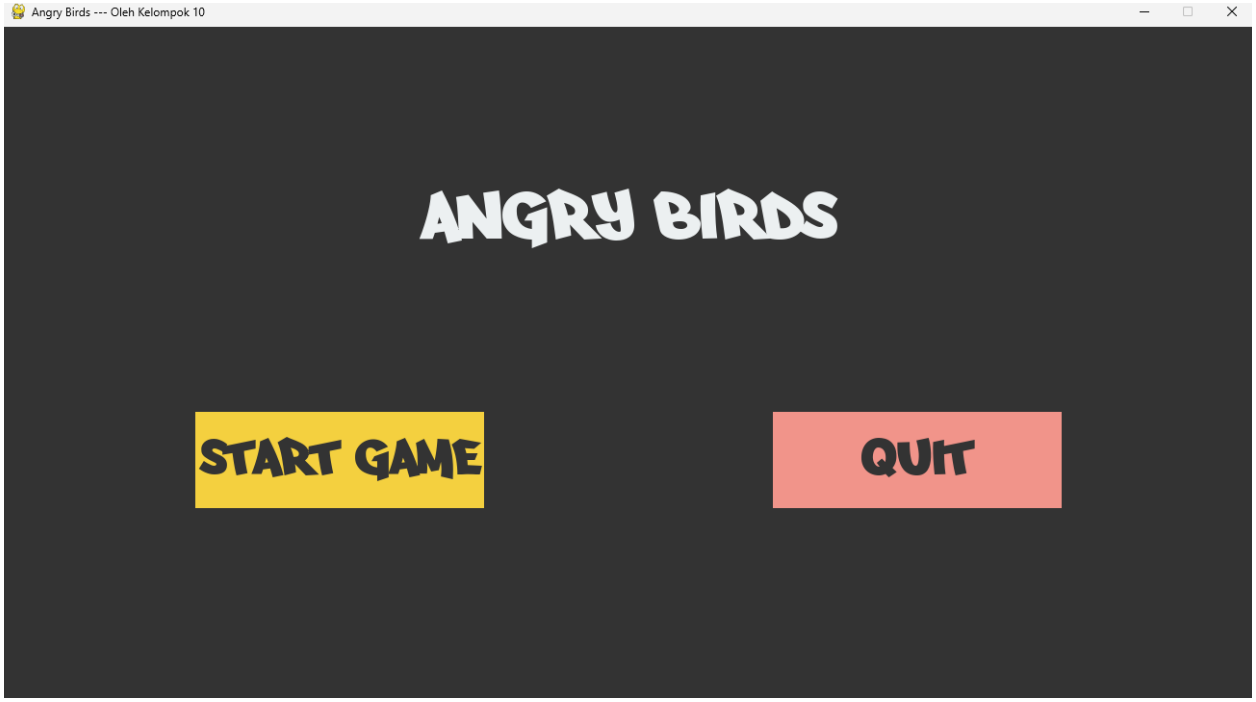
```
#Kelompok 10 :
#Achmad Nurnaafi (1306621057)
#Haryanto (1306621059)
#Yohanes Radito Putra (1306621048)

import pygame
import sys
import random
from math import *

import physics_engine
import objects
import maps
import interface

pygame.init()
width = 1300
height = 700
display = pygame.display.set_mode((width, height))
clock = pygame.time.Clock()

physics_engine.init(display)
objects.init(display)
maps.init(display)
interface.init(display)
pygame.display.set_caption("Angry Birds --- Oleh Kelompok 10")
background = (51, 51, 51)

def close():
    pygame.quit()
    sys.exit()

def start_game(map):
    map.draw_map()
```



```
def GAME():
    map = maps.Maps()

    welcome = interface.Label(450, 100, 400, 200, None, background)
    welcome.add_text("ANGRY BIRDS", 80, "Fonts/arfmoochikncheez.ttf", (236, 240, 241))

    start = interface.Button(200, 400, 300, 100, start_game, (244, 208, 63), (247, 220, 111))
    start.add_text("START GAME", 60, "Fonts/arfmoochikncheez.ttf", background)

    exit = interface.Button(800, 400, 300, 100, close, (241, 148, 138), (245, 183, 177))
    exit.add_text("QUIT", 60, "Fonts/arfmoochikncheez.ttf", background)

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                close()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    close()

            if event.type == pygame.MOUSEBUTTONDOWN:
                if exit.isActive():
                    exit.action()
                if start.isActive():
                    start_game(map)

        display.fill(background)

        start.draw()
        exit.draw()
        welcome.draw()

        pygame.display.update()
        clock.tick(60)

GAME()
```

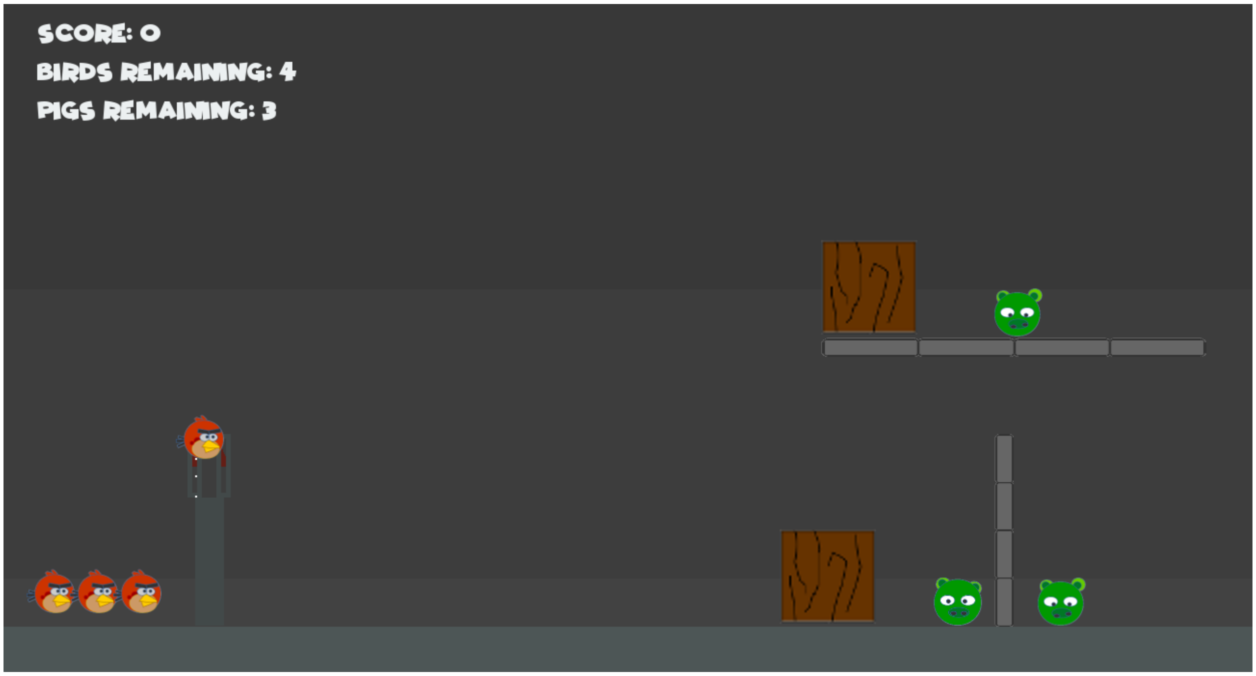
4.2. Screen Capture Hasil

- Tampilan Awal

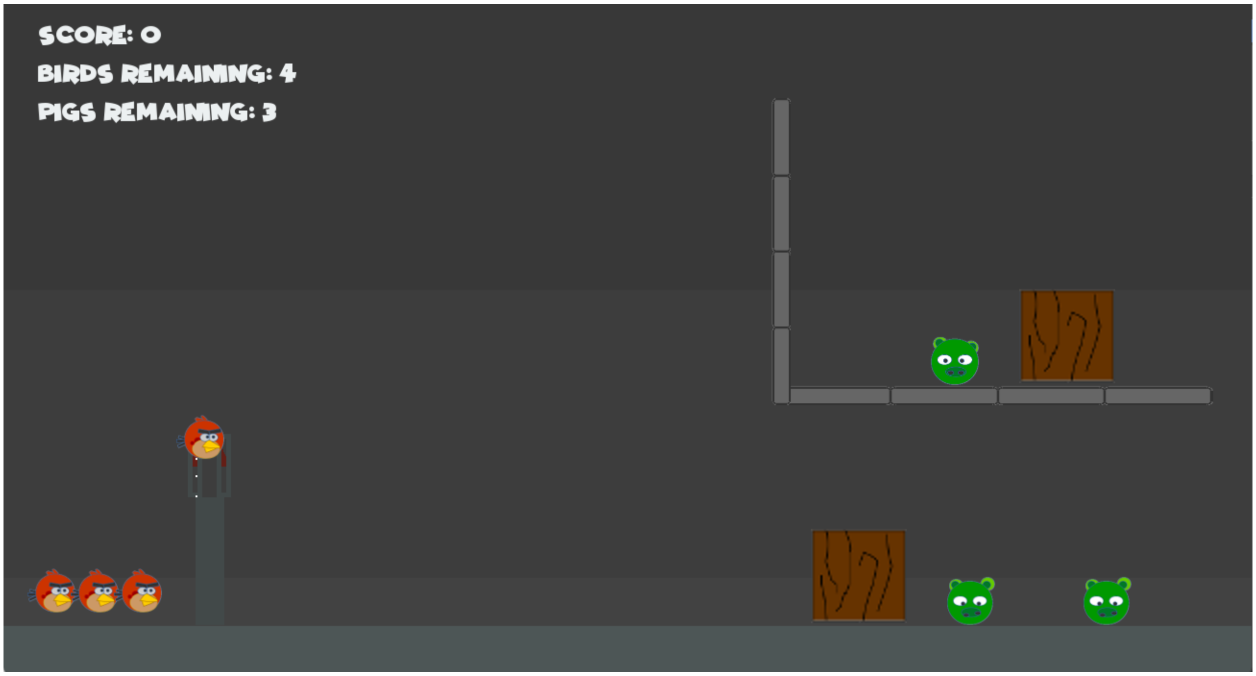


- Tampilan Saat Permainan Dimulai

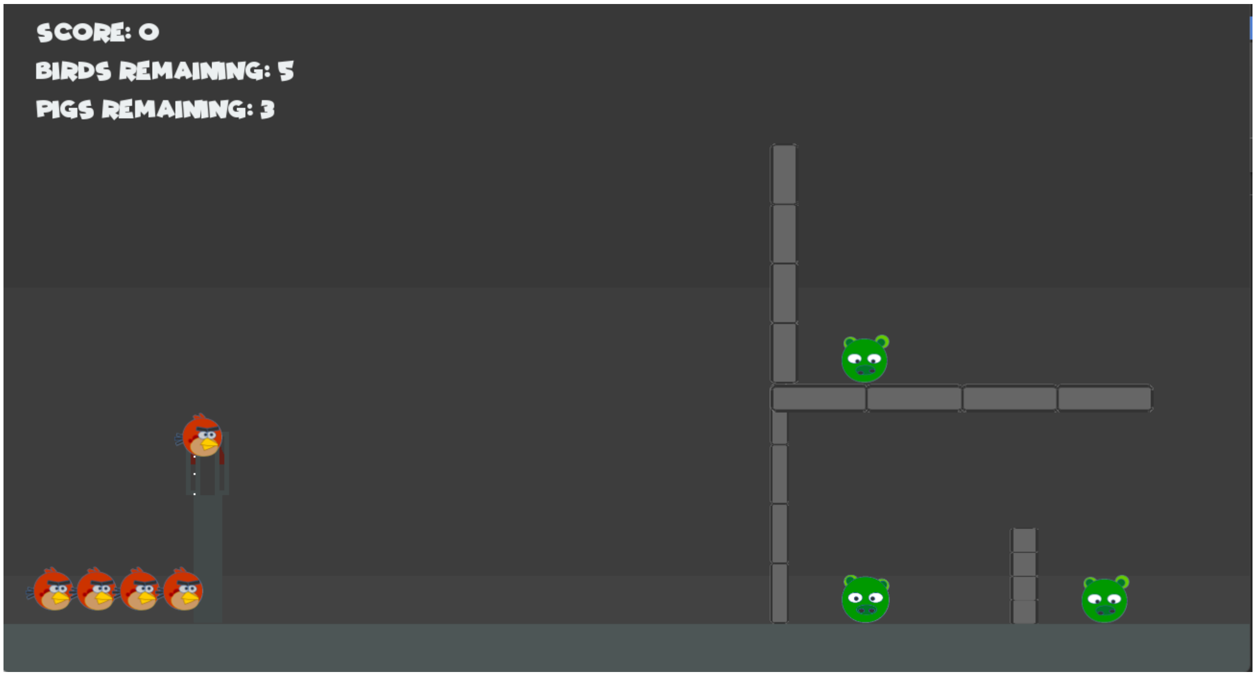
Level 1



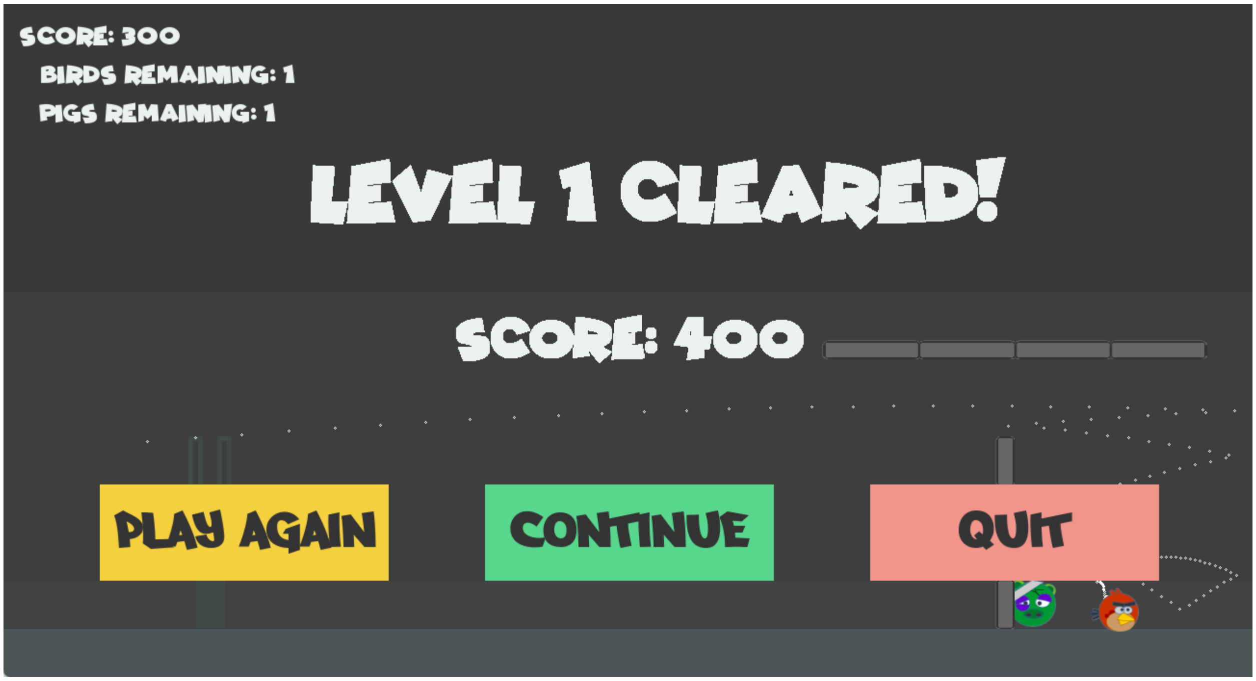
Level 2



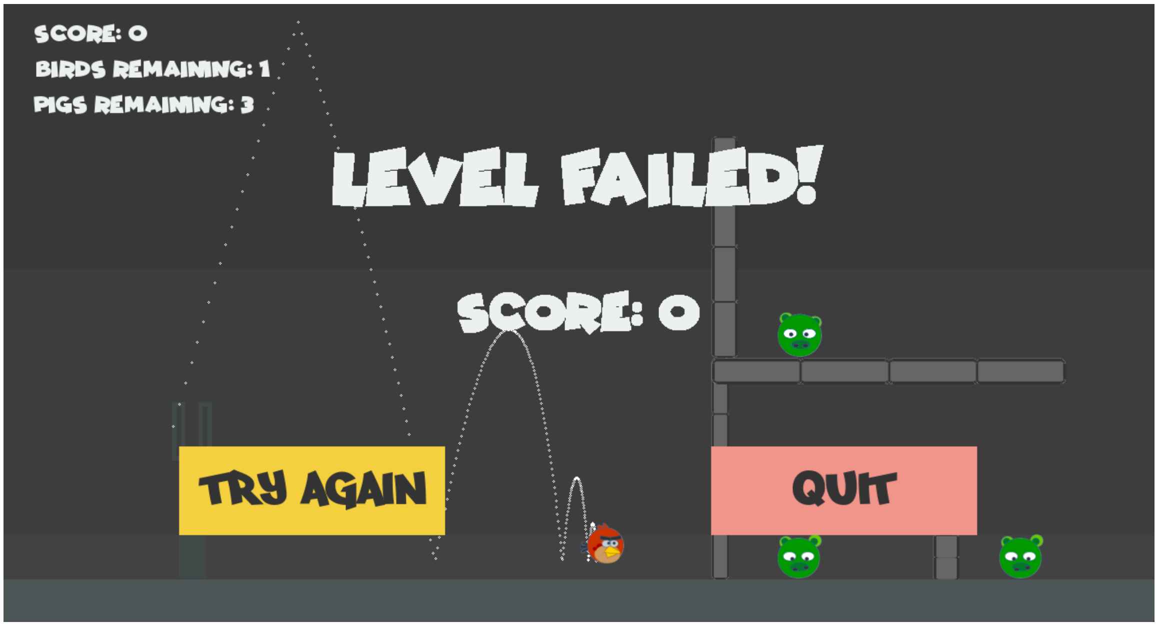
Level 3



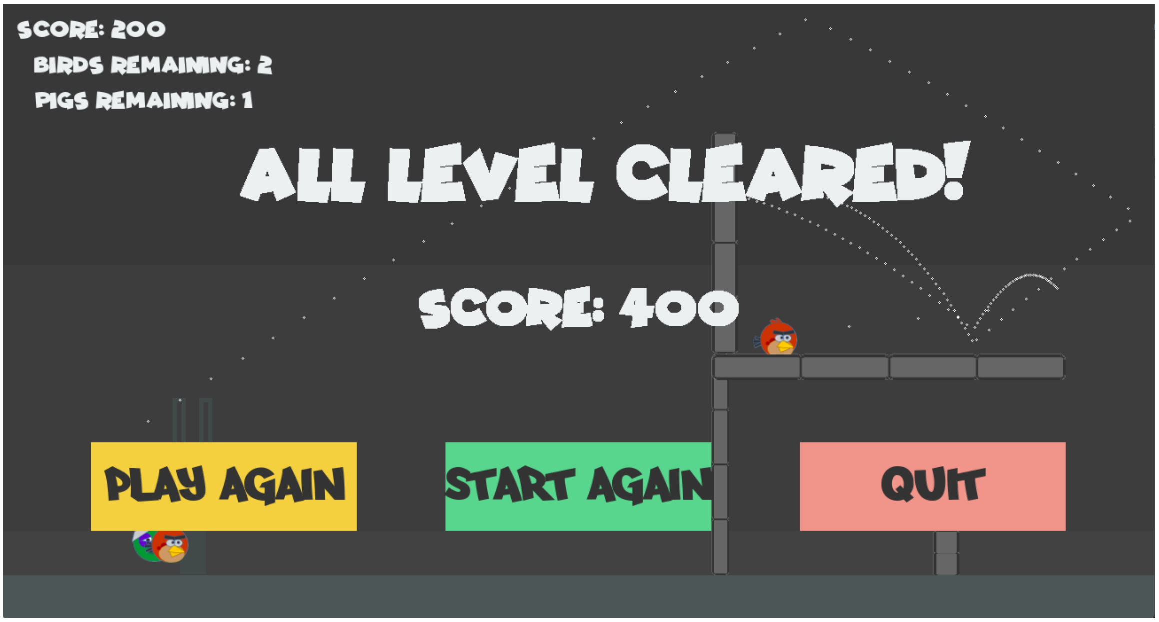
- **Tampilan Bila Menang**



- **Tampilan Bila Kalah**



- **Tampilan Bila menyelesaikan Permainan**



4.3. Pembahasan

Pada game angry birds, cara memainkannya dengan melontarkan burung ke udara. Semua gerakan burung yang dilontarkan ke udara membentuk gerak parabola. Ini bisa digunakan sebagai media pembelajaran topik gerak parabola. cara memainkannya Burung-burung dilontarkan dengan katapel untuk memusnahkan babi-babi. Pada setiap level, cuma tersedia beberapa ekor burung untuk memusnahkan semua babi. Agar burung meluncur lebih cepat, karet katapel perlu ditarik lebih panjang. Setelah dilepas, burung yang semula diam menjadi bergerak. Istilah fisikanya, kecepatan awal burung nol, setelah lepas dari ketapel kecepatan burung tidak nol lagi.

BAB V
KESIMPULAN & SARAN

5.1. Kesimpulan

Proyek kali ini mencakup peristiwa gerak parabola dan elastisitas hukum Hooke yang darinya kami mengembangkan konsep permainan Angry Birds dengan ketapel, yang pelurunya harus mengenai target babi di setiap level.Dimana Level rintangan yang harus di lewati menjadi lebih sulit. setiap levelnya.

5.2. Saran

Dalam proyek simulasi pygame ini seharusnya masih mampu menonjolkan konsep fisika yang terjadi di dalam game, seperti menampilkan kecepatan dan kekuatan peluncuran saat karakter angry bird dilontarkan. Serta kurangnya variasi tampilan seperti suara latar dan efek suara.

DAFTAR PUSTAKA

Elisa & Claudya, Y., 2016. PENENTUAN KONSTANTA PEGAS DENGAN CARA STATIS DAN DINAMIS. *Jurnal Fisika Edukasi,* 3(1), pp. 1-57.

Rismalasari, D., 2013. *GAME ANGRY BIRDS DAN PROGRAM TRACKER SEBAGAI MEDIA PEMBELAJARAN FISIKA PADA TOPIK GERAK PARABOLA*, Salatiga: Fakultas Sains dan Matematika Universitas Kristen Satya Wacana.

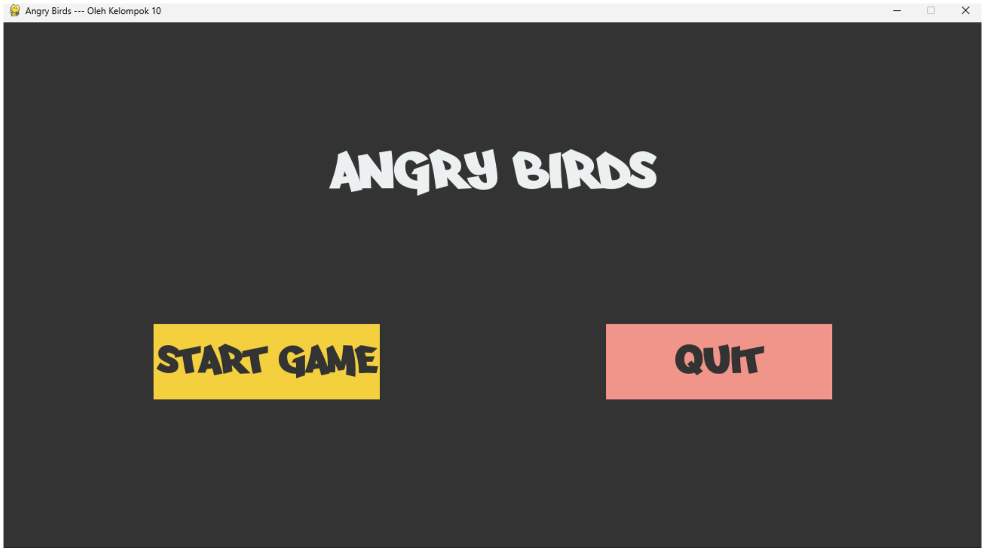
Satria, 2018. *Perancangan dan Implementasi Prototype Penyeimbang Mobil Pada Saat Drifting.*, Bandung: UNIKOM.

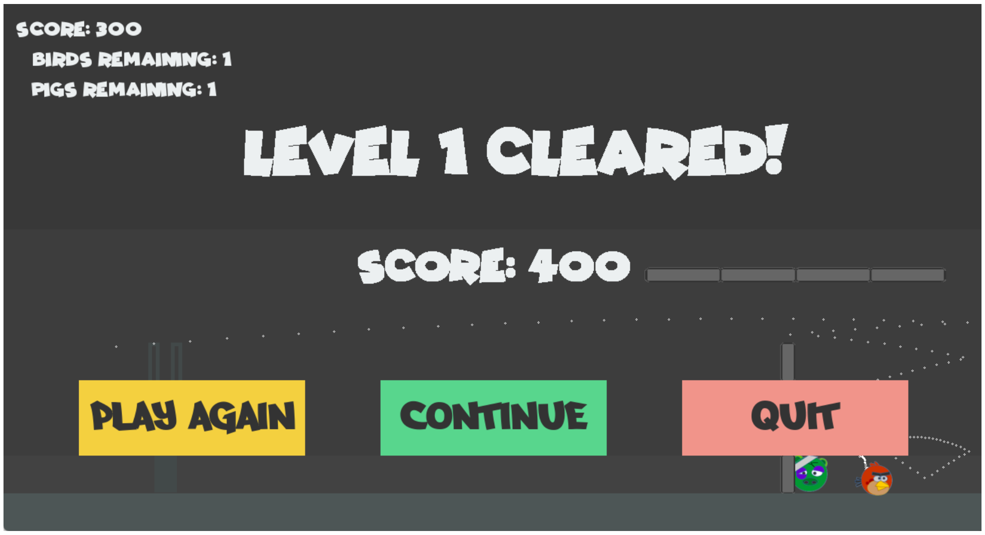
https://jatinmandav.wordpress.com/2018/05/25/angry-birds-in-python-using-pygame/

LAMPIRAN-LAMPIRAN

https://drive.google.com/drive/folders/15VL8jFiZhP0q9QL6cixHU8j_0IrNAC7z?usp=sharing

https://www.canva.com/design/DAFU7vzdWyo/cwlspEzTFoxpMWRWN6GnDQ/view?utm_content=DAFU7vzdWyo&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink#9


Angry Birds --- Oleh Kelompok 10
ANGRY BIRDS
START GAME
QUIT


SCORE: 300
BIRDS REMAINING: 1
PIGS REMAINING: 1
LEVEL 1 CLEARED!
SCORE: 400
PLAY AGAIN
CONTINUE
QUIT