LAPORAN PROYEK AKHIR
GAME GALAXY SHOOTER MENGGUNAKAN PYGAME

Mata Kuliah

Pemrograman Berbasis Objek

Oleh

Dyah Hayuningrum Agustin

24091397086

2024C

PROGRAM STUDI D4 MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

TAHUN 2025

DAFTAR ISI

DAFTAR ISI
BAB I PENDAHULUAN.....1
1.1 Latar Belakang.....1
1.2 Tujuan 1
1.3 Ruang Lingkup.....1
BAB II PENGEMBANGAN APLIKASI 2
2.1 Konsep OOP.....2
2.1.1 Encapsulation 2
2.1.2 Inheritance.....3
2.1.3 Polymorphism.....5
2.2 Desain Sistem.....6
2.2.1 Class Diagram.....6
2.2.2 Alur Kerja Game.....8
2.3 Fitur dari Sistem 9
2.4 Evaluasi Sistem.....10
BAB III PENUTUP.....11
3.1 Kesimpulan.....11
3.2 Saran 11

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pemrograman Berorientasi Objek (OOP) merupakan paradigma yang penting dalam pengembangan perangkat lunak. OOP ini memungkinkan perancangan aplikasi yang terstruktur dan mudah dipelihara melalui penerapan konsep – konsep fundamental seperti *encapsulation, inheritance, dan polymorphism*. Hal ini perlu implementasi praktis melalui aplikasi nyata yang bisa mendemonstrasikan penerapan prinsip – prinsip tersebut secara konkret dan komprehensif.

Pesatnya perkembangan teknologi dan meningkatnya minat terhadap permainan digital, pengembangan game sederhana *galaxy shooter* dipilih sebagai proyek akhir semester. Jenis game *space shooter* ini menampilkan berbagai objek dengan karakteristik dan perilaku yang beragam, sehingga sangat tepat untuk menunjukkan penerapan konsep *encapsulation, inheritance* dan *polymorphism*. Selain itu, game ini membutuhkan *enkapsulasi* data yang baik untuk mengatur *state* objek seperti *health*, *score*, dan posisi dengan aman.

1.2 Tujuan

Proyek game sederhana "Galaxy Shooter" ini memiliki beberapa tujuan utama, yaitu:

- Menerapkan dan memahami secara mendalam konsep – konsep OOP.
- Membangun sebuah game sederhana yang berjalan dengan stabil, logika yang jelas, dan mudah dipelihara.
- Melatih kemampuan merancang hierarki kelas yang baik serta mengelola objek – objek game secara dinamis.
- Mendemonstrasikan bahwa OOP dapat digunakan untuk membangun aplikasi game sederhana.

1.3 Ruang Lingkup

Ruang lingkup pengembangan game sederhana ini dibatasi pada aspek-aspek berikut:

Batasan Fungsional :

- Game sederhana berupa single-player game dengan kontrol keyboard.
- Jenis musuh dengan karakteristik berbeda yaitu BasicEnemy, FastEnemy, dan BossEnemy.
- Sistem scoring berdasarkan musuh yang berhasil dikalahkan.

Batasan Non – Fungsional :

- Penggunaan Pygame sebagai dependency utama.
- Mengimplementasikan multiplayer atau leaderboard online.

BAB II

PENGEMBANGAN APLIKASI

2.1 Konsep OOP

Pemrograman Berorientasi Objek adalah paradigma pemrograman yang mengorganisir software design di sekitar konsep sebuah objek yang merepresentasikan entitas dengan data (atribut) dan perilaku (metode). Dalam pengembangan game sederhana *"Galaxy Shooter"* ini, tiga pilar utama OOP diterapkan secara konsisten untuk menghasilkan kode yang terstruktur dan maintainable.

2.1.1 Encapsulation

Encapsulation diterapkan dengan menyembunyikan data internal objek menggunakan atribut private (__nama__) dan hanya menyediakan akses melalui metode getter atau setter. Adapun implementasi pada game sederhana "Galaxy Shooter" sebagai berikut:

1. Class GameObject

Semua atribut posisi dan status dibuat secara privat dan hanya bisa diakses dan dimodifikasi menggunakan method setter, getter, dan is_active.

```
class GameObject:
    def __init__(self, x, y, width, height):
        self.__x = x
        self.__y = y
        self.__width = width
        self.__height = height
        self.__speed = 5
        self.__is_active = True # Mengecek apakah objek masih aktif
        self.image = None
```

```
# Getter methods
def get_x(self):
    return self.__x
def get_y(self):
    return self.__y
def get_width(self):
    return self.__width
def get_height(self):
    return self.__height
def get_speed(self):
    return self.__speed
def is_active(self):
    return self.__is_active
```

```
# Setter methods
def set_x(self, x):
    self.__x = x
def set_y(self, y):
    self.__y = y
def set_speed(self, speed):
    self.__speed = speed
def set_active(self, status):
    self.__is_active = status
```

2. Class Player

Atribut – Atribut di enkapsulasi dan menggunakan metode take_damage() dan heal() untuk mengubah nilai secara aman.

```
class Player(GameObject): # Mewarisi GameObject
    def __init__(self, x, y):
        super().__init__(x, y, 150, 120)
        self.__health = 100
        self.__max_health = 100
        self.__score = 0
        self.set_speed(7) # Kecepatan Player
```

```
def take_damage(self, damage):
    self.__health -= damage
    if self.__health <= 0:
        self.__health = 0
        self.set_active(False) # Setter untuk status

def heal(self, amount):
    self.__health = min(self.__health + amount, self.__max_health)
```

Penjelasan :

- Health player tidak dapat diubah sembarangan jadi nilai yang tidak masuk akal.
- Setiap perubahan pada health selalu melalui validasi yaitu cek batas dan cek game over.
- Mengatur posisi objek agar terkontrol sehingga tidak keluar dari layer.

2.1.2 Inheritance

Inheritance digunakan untuk membentuk hierarki class sehingga struktur kode menjadi lebih terorganisir dan duplikasi kode dapat diminimalkan. Pada game *Galaxy Shooter*, konsep inheritance diterapkan dalam dua tingkat pewarisan.

Tingkat 1 : Game Object sebagai Superclass.

```
class GameObject:
    def __init__(self, x, y, width, height):
        self.__x = x
        self.__y = y
        self.__width = width
        self.__height = height
        self.__speed = 5
        self.__is_active = True # Mengecek apakah objek masih aktif
        self.image = None
```

- Player, Enemy, Bullet, dan PowerUP Mewarisi Game Object

```
class Player(GameObject):
    def __init__(self, x, y):
        super().__init__(x, y, 150, 120)  # Mewarisi GameObject
        self.__health = 100               # Atribut Khusus
        self.__max_health = 100
        self.__score = 0
        self.set_speed(7) # Kecepatan Player
```

Class Player, Enemy, Bullet, dan PowerUp mewarisi dari GameObject, sehingga secara otomatis mendapat atribut posisi, ukuran, serta metode getter atau setter dan get_rect().

Tingkat 2 : Jenis – Jenis Enemy Mewarisi Subclass Enemy.

Class mewarisi enemy sehingga memiliki perilaku dasar musuh yaitu bergerak ke bawah dan nilai poin dengan tambahan variasi spesifik masing-masing.

```
class Enemy(GameObject): # Semua musuh
    def __init__(self, x, y, width, height, points):
        super().__init__(x, y, width, height)
        self.__points = points
        self.set_speed(random.randint(2, 4)) # Random untuk speed musuh

    def get_points(self):
        return self.__points

    def update(self):
        self.set_y(self.get_y() + self.get_speed())
        if self.get_y() > SCREEN_HEIGHT: # Bergerak ke bawah
            self.set_active(False) # Nonaktif jika keluar layar
```

```
class BasicEnemy(Enemy): # Musuh biasa (tetap)
    def __init__(self, x, y):
        super().__init__(x, y, 40, 40, 10)

    def draw(self, screen):
        if self.image:
            screen.blit(self.image, (self.get_x(), self.get_y()))

class FastEnemy(Enemy): # Musuh dengan speed cepat
    def __init__(self, x, y):
        super().__init__(x, y, 35, 35, 20)
        self.set_speed(random.randint(5, 7))

    def draw(self, screen):
        if self.image:
            screen.blit(self.image, (self.get_x(), self.get_y()))

class BossEnemy(Enemy): # Musuh yang healtnya banyak
    def __init__(self, x, y):
        super().__init__(x, y, 80, 60, 50)
        self.__health = 5
```

2.1.3 Polymorphism

Polymorphism dicapai melalui method overriding yaitu metode dengan nama yang sama memiliki implementasi yang berbeda pada class turunannya.

- Method Update()

```
class Enemy(GameObject): # Semua musuh
    def __init__(self, x, y, width, height, points):
        super().__init__(x, y, width, height)
        self.__points = points
        self.set_speed(random.randint(2, 4)) # Random untuk speed musuh

    def get_points(self):
        return self.__points

    def update(self):
        self.set_y(self.get_y() + self.get_speed())
        if self.get_y() > SCREEN_HEIGHT: # Bergerak ke bawah
            self.set_active(False) # Nonaktif jika keluar layar
```

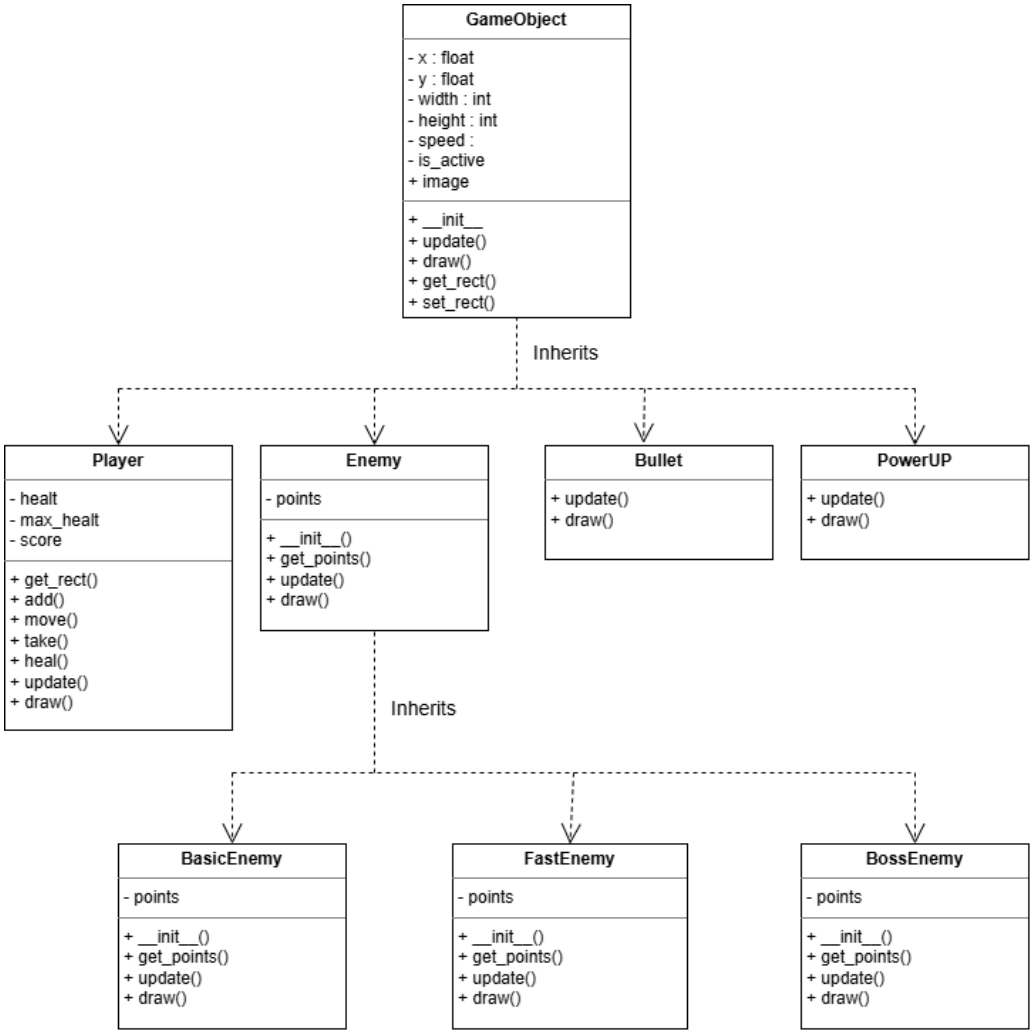
```
class Bullet(GameObject):
    def __init__(self, x, y):
        super().__init__(x, y, 20, 30)
        self.set_speed(10)

    def update(self):
        self.set_y(self.get_y() - self.get_speed())
        if self.get_y() < 0:
            self.set_active(False)

    def draw(self, screen):
        if self.image:
            screen.blit(self.image, (self.get_x(), self.get_y()))

class PowerUp(GameObject):
    def __init__(self, x, y):
        super().__init__(x, y, 30, 30)
        self.set_speed(3)

    def update(self):
        self.set_y(self.get_y() + self.get_speed())
        if self.get_y() > SCREEN_HEIGHT:
            self.set_active(False)
```

Penjelasan :

- Pada class bullet, method update berguna agar peluru bergerak ke atas.
- Pada class enemy, method update berguna agar musuh bergerak ke bawah.

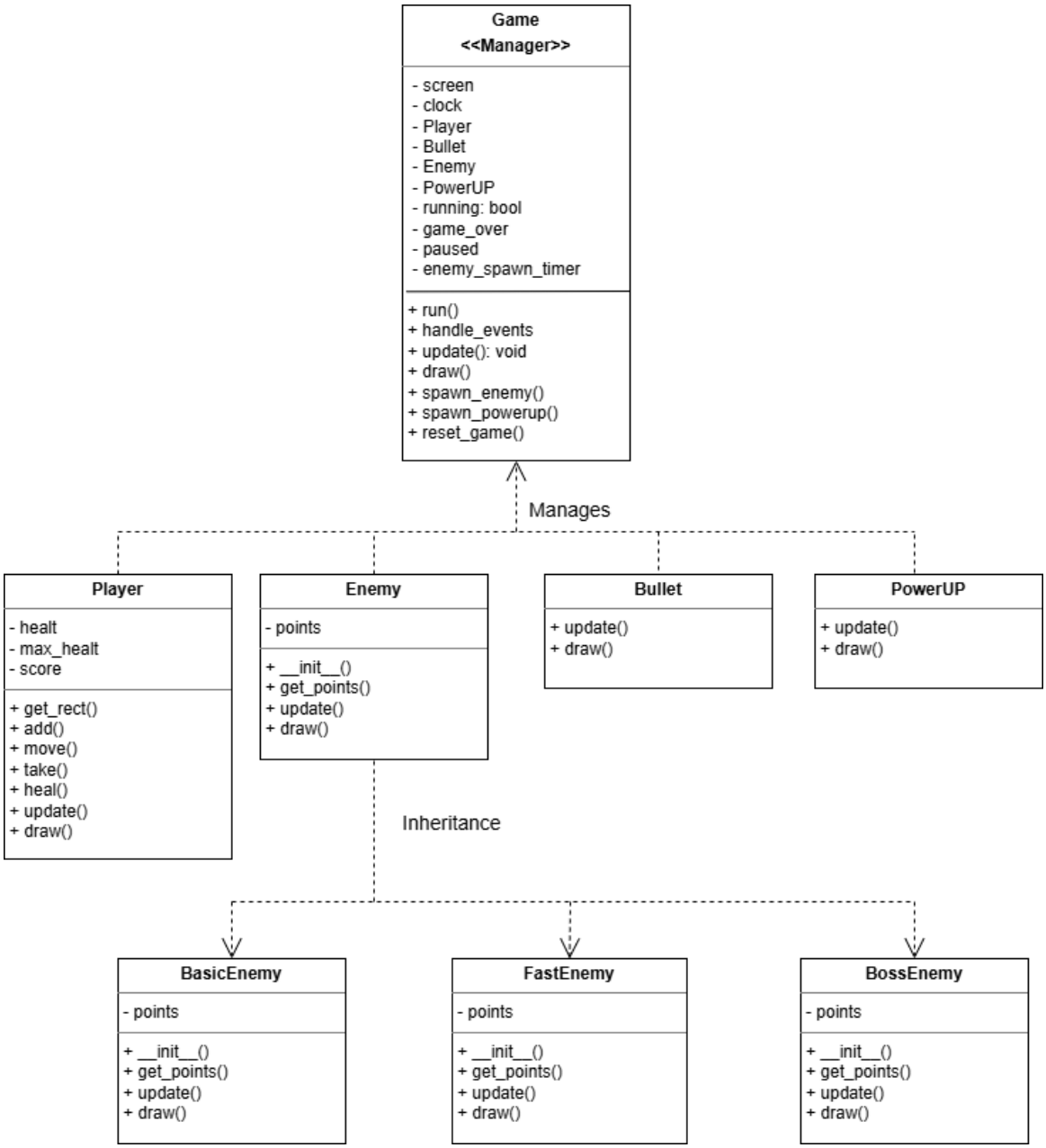
2.2 Desain Sistem

2.2.1 Class Diagram



Penjelasan :

- **GameObject** menjadi superclass yang berisi atribut dan fungsi umum untuk semua objek game, seperti posisi, ukuran, status aktif, serta method update() dan draw().
- **Player, Enemy, Bullet, dan PowerUp** merupakan turunan dari GameObject, sehingga seluruh objek game memiliki perilaku dasar yang sama dan dapat dikelola dengan cara yang seragam.
- **Player** adalah karakter yang dikendalikan pemain, memiliki health, dan score serta dapat bergerak, menerima damage, dan memulihkan health.
- **Enemy** mewakili musuh yang memberi poin dan menjadi class bagi beberapa jenis musuh.
- **BasicEnemy, FastEnemy, dan BossEnemy** adalah variasi musuh dengan tingkat kesulitan berbeda, tetapi tetap menggunakan struktur dasar yang sama dari Enemy.
- **Bullet** berfungsi sebagai peluru serangan, sedangkan **PowerUp** sebagai item pendukung.



Penjelasan:

- Game berperan sebagai Game Manager yaitu class utama yang mengatur jalannya permainan, mulai dari proses input, update logika game, hingga menampilkan objek ke layar.
- Class Game menyimpan dan mengelola objek Player, Enemy, Bullet, dan PowerUp, serta mengatur kondisi permainan seperti status berjalan, game over, dan waktu kemunculan musuh.

2.2.2 Alur Kerja Game

Alur kerja game sederhana "*Galaxy Shooter"* dirancang secara terstruktur agar setiap proses permainan berjalan secara berurutan dan mudah dipahami. Alur kerja terdiri dari tahap inisialisasi, proses utama permainan, hingga kondisi game over.

1. Inisialisasi Game

Sistem melakukan proses persiapan sebelum permainan dimulai. Proses ini meliputi:

- Inisialisasi library Pygame beserta modul pendukung seperti tampilan dan suara.
- Pembuatan window permainan dengan ukuran layar yang telah ditentukan.
- Pemanggilan dan pemuatan aset permainan, meliputi gambar (player, musuh, peluru, power-up) serta efek suara dan musik latar.
- Pembuatan objek utama permainan, yaitu player, daftar musuh (*enemy*), peluru (*bullet*), power-up, serta latar belakang berupa animasi bintang.
- Inisialisasi variabel status permainan seperti *running* dan *game over*.

2. Main Loop

Permainan masuk ke *main loop* yang berjalan secara terus-menerus selama game masih aktif. Pada bagian ini, alur permainan terbagi menjadi tiga proses utama, yaitu:

- handle_events() berfungsi untuk menangani seluruh input dari pemain dan sistem, seperti:
  - Pergerakan player menggunakan keyboard.
  - Aksi menembak peluru.
  - Keluar dari permainan saat game over.
- update() digunakan untuk memperbarui seluruh logika permainan, antara lain:
  - Memperbarui posisi player, musuh, peluru, dan power-up.
  - Memunculkan musuh dan power-up secara berkala.
  - Menggerakkan latar belakang bintang.
  - Melakukan pengecekan tabrakan (*collision detection*) antara peluru dengan musuh, musuh dengan player, serta power-up dengan player.
  - Memperbarui nilai skor dan health player sesuai kondisi permainan.
- draw() bertugas menampilkan seluruh elemen visual ke layar, meliputi:
  - Menggambar latar belakang permainan.
  - Menampilkan seluruh objek game (player, musuh, peluru, dan power-up).
  - Menampilkan HUD (score dan health bar).
  - Menampilkan overlay game over jika kondisi tersebut terpenuhi.

3. Kondisi Game Over

Permainan memasuki kondisi *game over* ketika nilai health player mencapai nol. Pada kondisi ini:

- Pergerakan dan pembaruan objek permainan dihentikan.
- Sistem menampilkan layar *game over* yang berisi informasi skor akhir.
- Pemain diberikan pilihan untuk menekan tombol Q untuk keluar dari aplikasi.

2.3 Fitur dari Sistem

1. Pergerakan Player

   Player dapat bergerak secara bebas ke segala arah yaitu ke atas, bawah, kiri, dan kanan. Pergerakan dikendalikan menggunakan tombol arah (arrow keys) pada keyboard, sehingga kontrol permainan bersifat intuitif dan mudah digunakan.

2. Sistem Penembakan

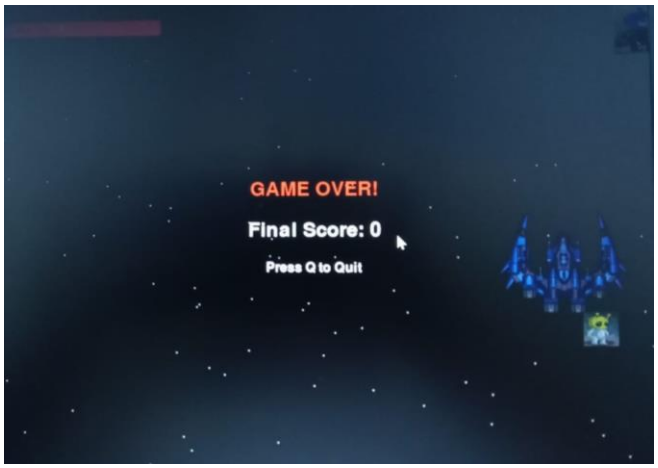
   Player dapat menembakkan peluru dengan menekan tombol SPACE. Peluru akan bergerak ke arah atas layar dan akan dinonaktifkan secara otomatis ketika keluar dari area permainan atau mengenai musuh.

3. Variasi Jenis Musuh

   Permainan menghadirkan tiga jenis musuh dengan karakteristik yang berbeda, yaitu:

   - Basic Enemy yaitu musuh standar dengan kecepatan normal.
   - Fast Enemy yaitu musuh dengan kecepatan gerak yang lebih tinggi.
   - Boss Enemy yaitu musuh dengan ukuran lebih besar dan tingkat ketahanan yang lebih tinggi.

4. Power-Up Kesehatan
   Power-up kesehatan muncul secara berkala dan bergerak ke arah bawah layar. Ketika power-up diambil oleh player, nilai health akan bertambah dan membantu pemain bertahan lebih lama dalam permainan.
5. HUD (Head-Up Display)
   Informasi permainan ditampilkan melalui HUD yang terdiri dari:
   - Score yang menunjukkan total poin yang diperoleh pemain.
   - Health bar yang menampilkan kondisi kesehatan player secara visual.
6. Efek Suara dan Musik Latar
   Permainan dilengkapi dengan efek suara untuk aksi menembak dan pengambilan power-up, serta musik latar yang diputar secara berulang untuk meningkatkan suasana permainan.
7. Game Over
   Ketika health player mencapai nol, permainan akan masuk ke kondisi *game over* dan menampilkan opsi untuk memulai ulang atau keluar dari permainan.



2.4 **Evaluasi Sistem**

Beberapa tantangan yang dihadapi ketika pembuatan game sederhana ini antara lain:

- Collision detection kurang akurat. Perbaikan yang bisa dilakukan dengan konversi koordinat ke tipe integer pada get_rect().
- Terdapat asset yang tidak ditemukan. Perbaikan untuk menanganinya yaitu menggunakan *try-except* agar program tetap berjalan.

BAB III
PENUTUP

3.1 Kesimpulan

Proyek Galaxy Shooter berhasil dibuat sebagai game space shooter sederhana yang stabil dan interaktif menggunakan Python dan Pygame. Proyek ini telah menerapkan secara efektif ketiga konsep utama OOP:

- Encapsulation: Atribut privat dengan akses melalui getter atau setter dan metode khusus.
- Inheritance untuk hierarki kelas yang baik dengan GameObject sebagai superclass bagi semua objek game.
- Polymorphism menggunakan overriding update() dan draw() sehingga semua objek diproses secara seragam.

3.2 Saran

Untuk mengembangkan **Galaxy Shooter** menjadi game yang lebih kompleks, profesional, dan kaya fitur bisa menggunakan library Python lain selain Pygame. Adapun hal yang bisa dikembangkan antara lain:

1. Menambah sistem level atau wave dengan kesulitan bertahap.
2. Variasikan senjata dan power-up (triple shot, shield, dll).
3. Membuat menu utama dan pengaturan suara atau kontrol.
4. Menambah animasi ledakan dan particle effect.