LAPORAN PROJECT UAS

SPACE DEFENDER

Mata Kuliah

Pemograman Berbasis Objek

Oleh :

Graynaldo Fahrul O.P.M
24091397081
2024C

PROGRAM STUDI D4 MANAJEMEN INFORMATIKA
FAKULTAS VOKASI
UNIVERSITAS NEGERI SURABAYA
TAHUN 2025

DAFTAR ISI

BAB 1: PENDAHULUAN

1.1 Latar Belakang

Object-Oriented Programming (OOP) merupakan paradigma pemrograman yang mengorganisir kode berdasarkan konsep "objek" yang merepresentasikan entitas dalam dunia nyata. OOP menawarkan pendekatan yang lebih terstruktur, modular, dan mudah di-maintain dibanding pemrograman prosedural.

Dalam rangka memenuhi tugas akhir semester mata kuliah Pemrograman Berorientasi Objek, penulis mengembangkan aplikasi game sederhana berjudul "Space Defender" yang mengimplementasikan prinsip-prinsip OOP seperti encapsulation, inheritance, dan polymorphism.

Game dipilih sebagai studi kasus karena game naturally memiliki banyak objek dengan behavior berbeda-beda yang cocok untuk demonstrasi konsep OOP. Space Defender adalah game arcade shooter sederhana dimana pemain mengendalikan pesawat luar angkasa untuk menghindari dan menghancurkan asteroid serta musuh alien.

1.2 Tujuan Project

Tujuan dari project ini adalah:

1. Memahami dan mengimplementasikan prinsip-prinsip OOP dalam aplikasi nyata
2. Mendemonstrasikan penggunaan encapsulation untuk data hiding dan access control
3. Menerapkan inheritance untuk code reusability dan hierarchy design
4. Mengimplementasikan polymorphism untuk flexible dan extensible code
5. Mengembangkan aplikasi yang functional, user-friendly, dan well-documented

1.3 Batasan Masalah

Batasan masalah dalam project ini:

1. Game dikembangkan menggunakan Python dengan library Pygame
2. Game berjalan di desktop environment (tidak di-port ke mobile)
3. Tidak menggunakan database untuk menyimpan high score (data stored in-memory)
4. Fokus pada implementasi OOP, bukan pada advanced game mechanics
5. Graphics menggunakan primitive shapes (polygon, circle, rect) bukan sprite assets

BAB 2: LANDASAN TEORI

2.1 Object-Oriented Programming

Object-Oriented Programming adalah paradigma pemrograman yang berbasis pada konsep "objek" yang berisi data (dalam bentuk fields/attributes) dan kode (dalam bentuk methods/functions). OOP bertujuan untuk meningkatkan flexibility dan maintainability dalam software development.

Keuntungan OOP:

- **Modularity**: Kode terorganisir dalam unit-unit independen (classes)
- **Reusability**: Kode dapat digunakan kembali melalui inheritance
- **Extensibility**: Mudah menambah fitur baru tanpa merusak kode existing
- **Maintainability**: Easier debugging dan maintenance karena structure yang jelas

2.2 Encapsulation

Encapsulation adalah proses bundling data (attributes) dan methods yang bekerja pada data tersebut dalam satu unit (class), serta hiding internal state dari outside access.

Prinsip Encapsulation:

- Data (attributes) dibuat private atau protected
- Access ke data dikontrol melalui public methods (getter/setter)
- Implementation details disembunyikan dari user

Contoh dalam Python:

Class Player:
Private attribute

```
def __init__(self, x, y):
    # Memanggil constructor parent class
    super().__init__(x, y, 50, 40, (0, 255, 0))
    self.__speed = 5
    self.__health = 100
```

Getter method

```
def get_health(self):
    return self.__health
```

Setter method

```
def take_damage(self, damage):
    """Mengurangi health player"""
    self.__health -= damage
    if self.__health < 0:
        self.__health = 0
```

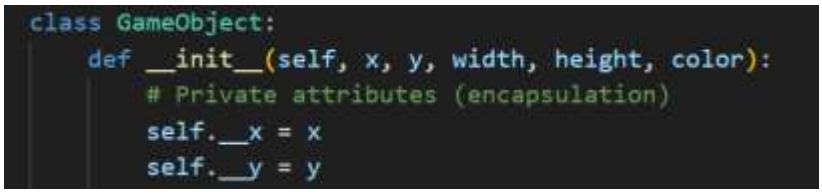
2.3 Inheritance

Inheritance adalah mekanisme dimana sebuah class (child/subclass) dapat mewarisi attributes dan methods dari class lain (parent/superclass). Ini memungkinkan code reusability dan hierarchy design.

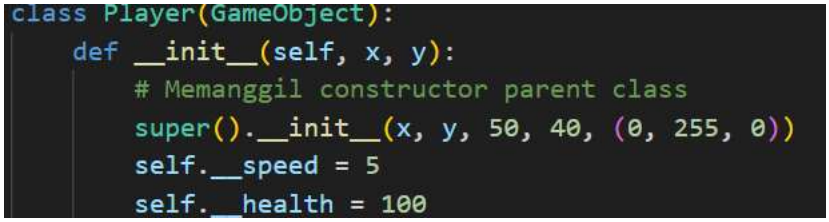
Jenis-jenis Inheritance:

1. **Single Inheritance**: Class mewarisi dari satu parent class
2. **Multi-level Inheritance**: Class mewarisi dari class yang juga mewarisi dari class lain
3. **Multiple Inheritance**: Class mewarisi dari multiple parent classes

Contoh Single Inheritance:



```
class GameObject:
    def __init__(self, x, y, width, height, color):
        # Private attributes (encapsulation)
        self.__x = x
        self.__y = y
```



```
class Player(GameObject):
    def __init__(self, x, y):
        # Memanggil constructor parent class
        super().__init__(x, y, 50, 40, (0, 255, 0))
        self.__speed = 5
        self.__health = 100
```

2.4 Polymorphism

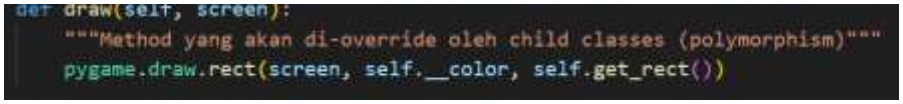
Polymorphism memungkinkan objects dari different classes untuk diperlakukan sebagai objects dari common parent class. Method yang sama dapat memiliki behavior berbeda pada different classes.

Jenis-jenis Polymorphism:

1. **Method Overriding**: Child class provides different implementation untuk method dari parent class
2. **Method Overloading**: Multiple methods dengan nama sama tapi signature berbeda (tidak fully supported di Python)

Contoh Method Overriding:

class GameObject:



```
def draw(self, screen):
    """Method yang akan di-override oleh child classes (polymorphism)"""
    pygame.draw.rect(screen, self.__color, self.get_rect())
```

class Player(GameObject):

```
def draw(self, screen):
    """
    Override method draw dari parent class (POLYMORPHISM)
    Menggambar player sebagai spaceship
    """
    # Body pesawat
    points = [
        (self.get_x() + self.get_width() // 2, self.get_y()),  # Nose
        (self.get_x(), self.get_y() + self.get_height()),      # Left wing
        (self.get_x() + self.get_width(), self.get_y() + self.get_height())  # Right wing
    ]
```

class Enemy(GameObject):

```
def draw(self, screen):
    center_x = self.get_x() + self.get_width() // 2
    center_y = self.get_y() + self.get_height() // 2

    # Gambar asteroid sebagai polygon tidak beraturan
    points = [
        (center_x - 20, center_y - 10),
        (center_x - 15, center_y - 20),
        (center_x + 5, center_y - 20),
        (center_x + 20, center_y - 5),
        (center_x + 15, center_y + 15),
        (center_x - 5, center_y + 20),
        (center_x - 20, center_y + 10)
```

BAB 3: PERANCANGAN SISTEM

3.1 Deskripsi Aplikasi

Space Defender adalah game arcade shooter 2D dimana pemain mengendalikan pesawat luar angkasa untuk bertahan dari serangan asteroid dan alien enemies.

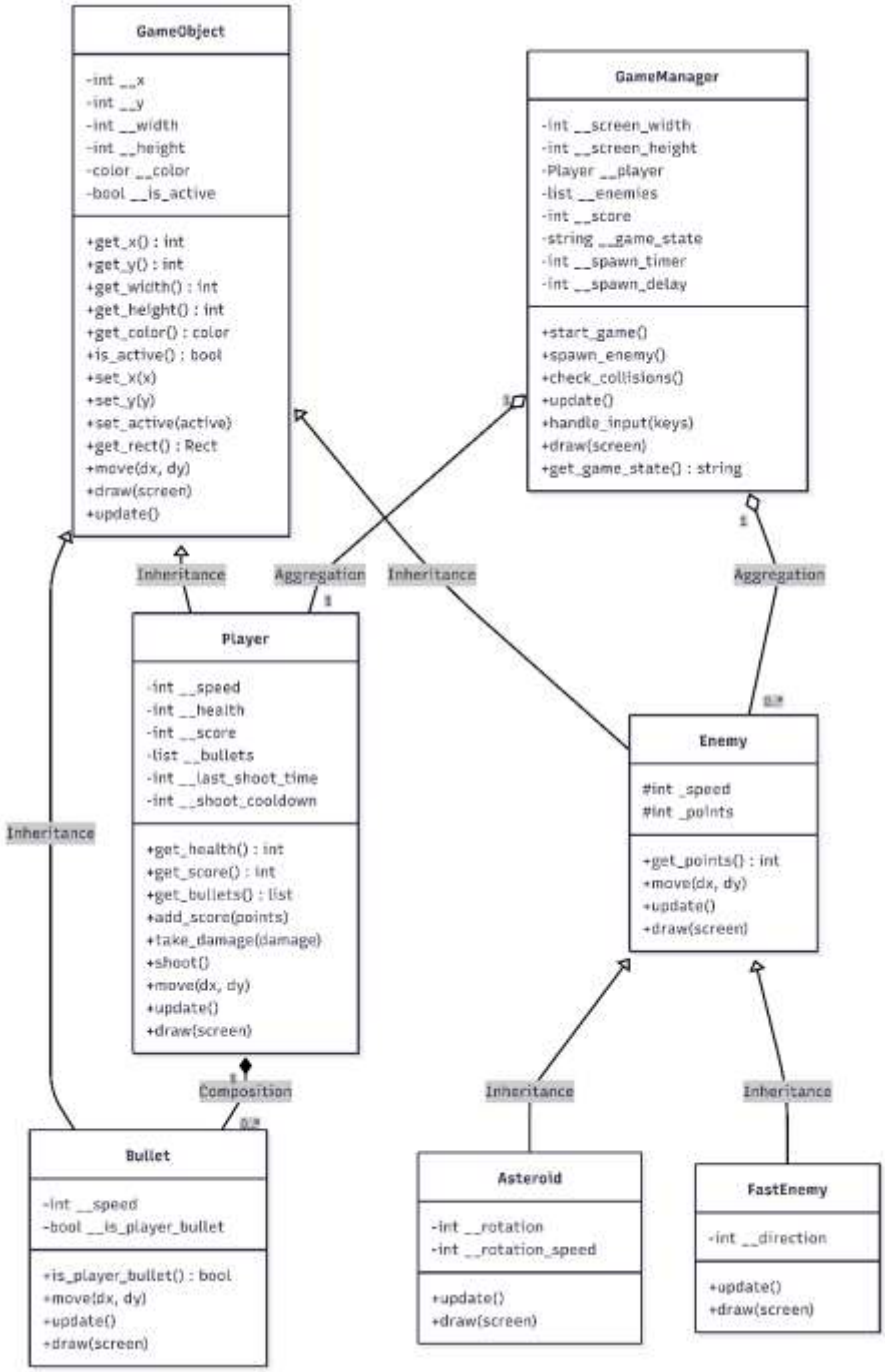
Gameplay:

- Player dapat bergerak ke segala arah menggunakan keyboard (WASD atau Arrow Keys)
- Player dapat menembak peluru dengan menekan SPACE
- Asteroid dan enemy muncul dari atas layar dan bergerak ke bawah
- Player mendapat poin setiap berhasil menghancurkan enemy
- Player memiliki health bar yang berkurang ketika tertabrak enemy
- Game over ketika health habis

Game States:

1. **Menu State**: Layar awal dengan instruksi
2. **Playing State**: Gameplay berlangsung
3. **Game Over State**: Menampilkan final score dengan opsi restart

3.2 Class Diagram



Penjelasan Class Diagram:

1. **GameObject (Base Class)**
   - Parent class untuk semua game objects
   - Menyimpan posisi, size, dan color
   - Menyediakan basic methods: move(), draw(), update()
2. **Player (extends GameObject)**
   - Merepresentasikan pesawat player
   - Menambah attributes: health, score, bullets
   - Override methods untuk behavior khusus player

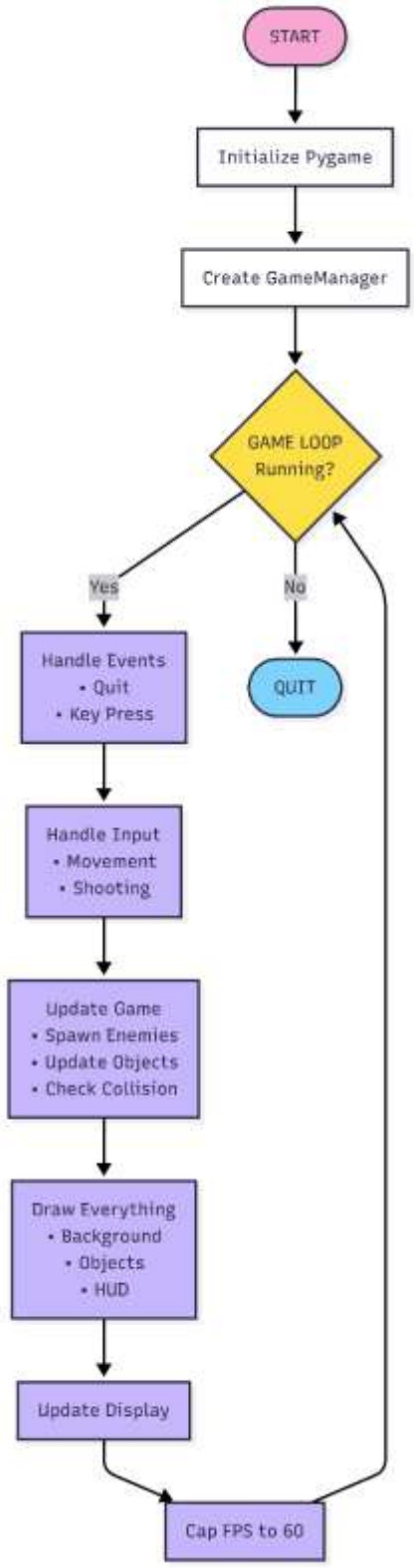
3. **Bullet (extends GameObject)**
    - Merepresentasikan peluru
    - Memiliki speed dan direction
    - Auto-deactivate ketika keluar screen
4. **Enemy (extends GameObject)**
    - Base class untuk semua enemies
    - Memiliki speed dan point value
    - Template untuk different enemy types
5. **Asteroid (extends Enemy)**
    - Enemy type pertama
    - Bergerak lurus dengan rotation
    - Memberikan 10 points
6. **FastEnemy (extends Enemy)**
    - Enemy type kedua
    - Bergerak zigzag dengan speed lebih cepat
    - Memberikan 20 points
7. **GameManager**
    - Orchestrator class
    - Manage game state, spawning, collision
    - Tidak inherit dari GameObject karena bukan visual object

Relationships:

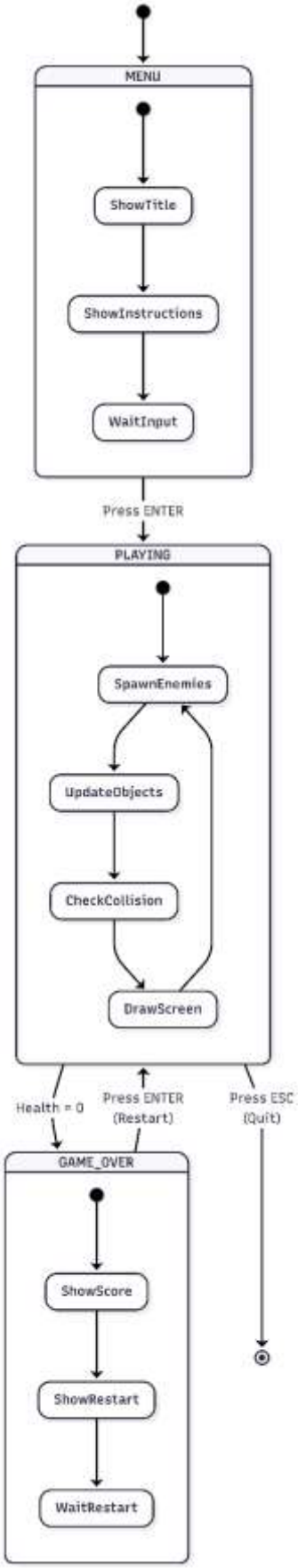
- **Inheritance**: Indicated by arrow with hollow triangle
- **Composition**: Player contains Bullets (strong ownership)
- **Aggregation**: GameManager references Player and Enemies (weak ownership)

3.3 Flowchart Aplikasi

Main Game Loop:

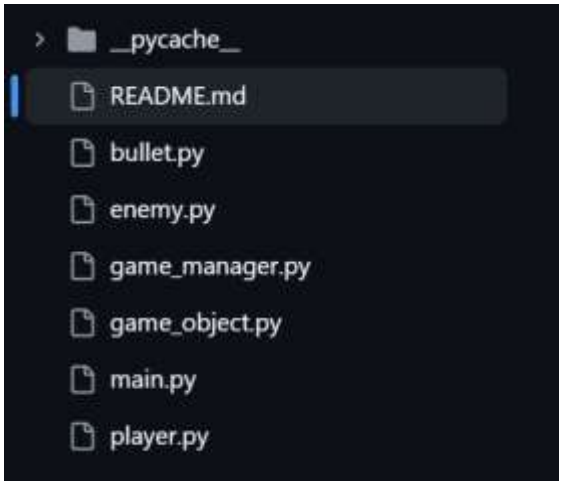


Game State Flow:



BAB 4: IMPLEMENTASI

4.1 Struktur Project



Penjelasan File:

1. **main.py**:
    - Entry point aplikasi
    - Initialize Pygame dan GameManager
    - Main game loop
2. **game_object.py**:
    - Base class untuk semua game objects
    - Implementasi encapsulation dengan private attributes
3. **player.py**:
    - Player class yang inherit dari GameObject
    - Demonstrasi inheritance dan polymorphism
4. **enemy.py**:
    - Enemy base class dan derived classes (Asteroid, FastEnemy)
    - Demonstrasi multi-level inheritance
5. **bullet.py**:
    - Bullet class untuk projectiles
    - Simple inheritance example
6. **game_manager.py**:
    - Game logic coordinator
    - Composition dengan Player dan Enemies

4.2 Implementasi Encapsulation

Konsep yang Diimplementasikan:

Encapsulation diterapkan di class game object dengan menjadikan semua attributes sebagai private (menggunakan double underscore __).

Code Implementation:

```python
class GameObject:
    def __init__(self, x, y, width, height, color):
        # Private attributes (encapsulation)
        self.__x = x
        self.__y = y
        self.__width = width
        self.__height = height
        self.__color = color
        self.__is_active = True
```

Getter Methods untuk Read Access:

```python
    # Getter methods (encapsulation)
    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y

    def get_width(self):
        return self.__width
```

Setter Methods untuk Write Access:

```python
    # Setter methods (encapsulation)
    def set_x(self, x):
        self.__x = x

    def set_y(self, y):
        self.__y = y

    def set_active(self, active):
        self.__is_active = active
```

Keuntungan Approach Ini:

- Data integrity terjaga - tidak bisa diubah sembarangan
- Bisa menambah validation di setter jika diperlukan
- Internal representation bisa diubah tanpa mempengaruhi external code
- Memudahkan debugging karena akses terkontrol

Contoh Penggunaan:

```
# SALAH - tidak bisa langsung akses private attribute
player = Player(100, 100)
print(player.__x)  # AttributeError
```

```
# BENAR - menggunakan getter
print(player.get_x())  # Works!

# SALAH - tidak bisa langsung set private attribute
player.__x = 200  # Creates new attribute, doesn't modify original

# BENAR - menggunakan setter
player.set_x(200)  # Works!
```

4.3 Implementasi Inheritance

Single Inheritance: Player extends GameObject

```
class Player(GameObject):
    def __init__(self, x, y):
        # Memanggil constructor parent class
        super().__init__(x, y, 50, 40, (0, 255, 0))
        self.__speed = 5
        self.__health = 100
        self.__score = 0
        self.__bullets = []
```

Penjelasan:

- Player mewarisi semua attributes dan methods dari GameObject
- super().__init__() memanggil constructor parent untuk initialize inherited attributes
- Player menambah attributes baru yang spesifik untuk player

Multi-level Inheritance: Asteroid extends Enemy extends GameObject

```
# Level 1: Enemy extends GameObject
class Enemy(GameObject):
    def __init__(self, x, y, width, height, color, speed, points):
        super().__init__(x, y, width, height, color)
        self._speed = speed  # Protected attribute
        self._points = points  # Protected attribute
# Level 2: Asteroid extends Enemy
class Asteroid(Enemy):
    def __init__(self, x, y):
        # Asteroid besar
        super().__init__(x, y, 40, 40, (150, 75, 0), 2, 10)
        self.__rotation = 0
        self.__rotation_speed = random.randint(1, 5)
```

Hierarchy:

GameObject (Level 0)
↓
Enemy (Level 1) - adds speed and points
↓
Asteroid (Level 2) - adds rotation

Keuntungan Multi-level Inheritance:

- Code reusability maksimal
- Clear hierarchy
- Asteroid mendapat semua functionality dari GameObject dan Enemy
- Mudah menambah enemy type baru dengan extend Enemy

4.4 Implementasi Polymorphism

Method Overriding - Method `draw()`

Setiap class meng-override method `draw()` dengan implementation berbeda:

1. GameObject (Base Implementation):

```
def draw(self, screen):
    """Method yang akan di-override oleh child classes (polymorphism)"""
    pygame.draw.rect(screen, self.__color, self.get_rect())
```

2. Player (Override dengan Spaceship Shape):

```
def draw(self, screen):
    """
    Override method draw dari parent class (POLYMORPHISM)
    Menggambar player sebagai spaceship
    """
    # Body pesawat
    points = [
        (self.get_x() + self.get_width() // 2, self.get_y()),  # Nose
        (self.get_x(), self.get_y() + self.get_height()),      # Left wing
        (self.get_x() + self.get_width(), self.get_y() + self.get_height())  # Right wing
    ]
    pygame.draw.polygon(screen, self.get_color(), points)
    # Cockpit
    pygame.draw.circle(screen, (0, 200, 255),
                       (self.get_x() + self.get_width() // 2,
                        self.get_y() + self.get_height() // 2), 8)
```

3. Asteroid (Override dengan Irregular Shape):

```
def draw(self, screen):
    center_x = self.get_x() + self.get_width() // 2
    center_y = self.get_y() + self.get_height() // 2

    # Gambar asteroid sebagai polygon tidak beraturan
    points = [
        (center_x - 20, center_y - 10),
        (center_x - 15, center_y - 20),
        (center_x + 5, center_y - 20),
        (center_x + 20, center_y - 5),
        (center_x + 15, center_y + 15),
        (center_x - 5, center_y + 20),
        (center_x - 20, center_y + 10)
```

4. FastEnemy (Override dengan UFO Shape):

```
def draw(self, screen):
    center_x = self.get_x() + self.get_width() // 2
    center_y = self.get_y() + self.get_height() // 2

    # Body UFO
    pygame.draw.ellipse(screen, self.get_color(),
                        (self.get_x(), center_y, self.get_width(), 15))

    # Dome UFO
    pygame.draw.arc(screen, (255, 100, 100),
                    (self.get_x() + 5, self.get_y(), 20, 20), 0, 3.14, 2)
```

Runtime Polymorphism:

#Di GameManager
objects = [player, asteroid1, asteroid2, fast_enemy, bullet1, bullet2]

#Polymorphic call - setiap object draw dirinya dengan cara berbeda
for obj in objects:
  obj.draw(screen) # Calls appropriate draw() based on actual type

Keuntungan:

- Uniform interface - semua objects dipanggil dengan cara sama
- Flexible - mudah menambah object type baru
- Extensible - tidak perlu modify existing code untuk add new types
- Runtime decision - behavior ditentukan saat program berjalan

Method Overriding - Method `update()`

1. GameObject (Default):

```
    def update(self):
        """Method yang akan di-override oleh child classes (polymorphism)"""
        pass
```

2. Player (Update Bullets):

```
def update(self):
    """Override method update dari parent class (POLYMORPHISM)"""
    # Update semua bullets
    for bullet in self.__bullets[:]:
        bullet.update()
        if not bullet.is_active():
            self.__bullets.remove(bullet)
```

3. Asteroid (Move and Rotate):

```
def update(self):
    """Override method update dengan behavior khusus (POLYMORPHISM)"""
    super().update()
    self.__rotation += self.__rotation_speed
```

4. FastEnemy (Zigzag Movement):

```
def update(self):
    """Override method update dengan behavior khusus (POLYMORPHISM)"""
    super().update()
    self.__rotation += self.__rotation_speed
```

BAB 5: HASIL DAN PEMBAHASAN

5.1 Fitur Aplikasi

Fitur Utama:

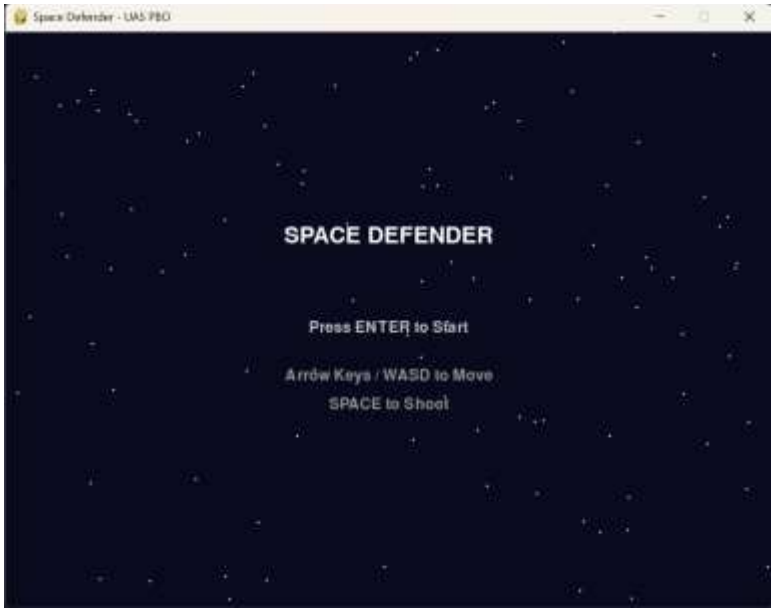
1. **Player Control**
    - Movement: 8-directional movement dengan WASD atau Arrow Keys
    - Shooting: Press SPACE untuk menembak dengan cooldown system
    - Speed: Consistent movement speed untuk control yang responsive
2. **Enemy System**
    - Multiple Enemy Types:
        - Asteroid: Moves straight down dengan rotation animation
        - FastEnemy: Moves dalam zigzag pattern dengan speed lebih cepat
    - Spawn System: Progressive difficulty - spawn rate meningkat seiring waktu
    - Point System: Different enemies memberikan different points (Asteroid: 10, FastEnemy: 20)
3. **Collision Detection**
    - Bullet-Enemy Collision: Bullet menghancurkan enemy dan add points
    - Player-Enemy Collision: Player take damage dan enemy destroyed
    - Accurate Hitbox: Menggunakan pygame.Rect() untuk precise collision
4. **Game States**
    - Menu State: Welcome screen dengan instruksi
    - Playing State: Active gameplay
    - Game Over State: Display final score dengan restart option
5. **UI/HUD**
    - Score Counter: Real-time score display
    - Health Display: Text dan visual health bar
    - Progressive Health Bar: Color-coded (green when healthy, red when low)
6. **Visual Effects**
    - Custom Shapes: Spaceship, asteroid, UFO designs
    - Laser Glow: Bullets memiliki glow effect
    - Starfield Background: Animated stars untuk depth
    - Rotation Animation: Asteroids rotate while moving

Fitur Tambahan (Kreativitas):

1. **Progressive Difficulty**
    - Spawn delay berkurang setiap 5 enemies spawned
    - Minimum spawn delay untuk cap difficulty
2. **Multiple Control Schemes**
    - Support WASD dan Arrow Keys
    - Flexible untuk different user preferences
3. **Smooth Animations**
    - 60 FPS gameplay untuk smooth experience
    - Rotation animations untuk asteroids

5.2 Screenshot Aplikasi

[Screenshot 1: Menu Screen]



- Menampilkan title "SPACE DEFENDER"
- Instructions untuk control
- "Press ENTER to Start"

[Screenshot 2: Gameplay - Early Game]

- Player di posisi awal
- Beberapa asteroids spawning
- Score: 30
- Health bar penuh (100)

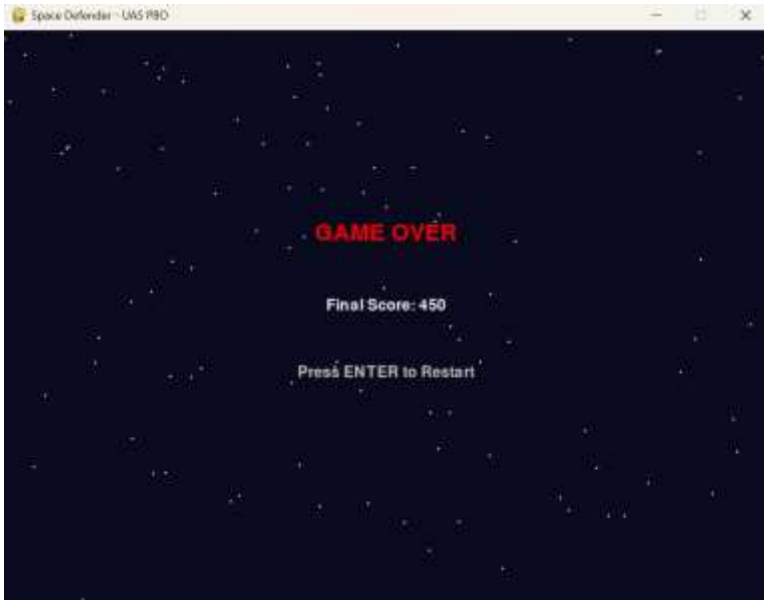
[Screenshot 3: Gameplay - Mid Game]

- Multiple enemies (asteroids dan UFOs)
- Player shooting bullets
- Score lebih tinggi: 170
- Health bar berkurang ke 60

[Screenshot 4: Gameplay - Intense Moment]

- Screen penuh dengan enemies
- Multiple bullets dari player
- Health bar critical (20)
- High score: 450

[Screenshot 5: Game Over Screen]



- "GAME OVER" text
- Final Score displayed
- "Press ENTER to Restart"

BAB 6: KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan development dan testing project "Space Defender", dapat disimpulkan:

1. **Implementasi OOP Berhasil**
    - Encapsulation berhasil diterapkan dengan private attributes dan getter/setter methods
    - Inheritance hierarchy terstruktur dengan baik dari GameObject ke derived classes
    - Polymorphism berfungsi dengan method overriding di draw(), update(), dan move()
2. **Code Quality**
    - Code modular dan terorganisir dalam separate files
    - Easy to understand dan maintain berkat clear structure
    - Extensible - mudah menambah enemy types atau features baru
3. **Aplikasi Functional**
    - Game berjalan smooth tanpa major bugs
    - All features working as intended
    - User interface intuitive dan responsive
4. **Learning Outcomes**
    - Pemahaman mendalam tentang OOP principles
    - Praktik best practices dalam code organization
    - Experience dengan game development fundamentals

DAFTAR PUSTAKA

1. Deitel, Paul J., dan Harvey Deitel. *Python for Programmers*. Pearson, 2019.
2. Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
3. Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.
4. Pygame Documentation. "Pygame Documentation." https://www.pygame.org/docs/. Diakses Desember 2024.
5. Python Software Foundation. "Python Documentation." https://docs.python.org/3/. Diakses Desember 2024.
6. Sweigart, Al. *Invent Your Own Computer Games with Python*. No Starch Press, 2016.
7. Lecture Notes: Pemrograman Berorientasi Objek, [Nama Universitas], 2024.