# LAPORAN PROYEK AKHIR SEMESTER

# IMPLEMENTASI KONSEP PEMROGRAMAN BERBASIS OBJEK PADA APLIKASI GAME MONSTER BATTLE ARENA BERBASIS PYTHON & TKINTER

**Mata Kuliah**

Pemrograman Berbasis Objek

**Oleh**

Fathin Syauqi Rabbani
24091397051
2024B


**PROGRAM STUDI D4 MANAJEMEN INFORMATIKA**
**FAKULTAS VOKASI**

**UNIVERSITAS NEGERI SURABAYA 2025**

**KATA PENGANTAR**

Puji syukur kehadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan proyek akhir semester mata kuliah Pemrograman Berorientasi Objek dengan judul "Monster Battle Arena - Game Battle Berbasis Turn-Based dengan Implementasi Konsep OOP".

Proyek ini dibuat sebagai salah satu syarat untuk memenuhi penilaian Ujian Akhir Semester Gasal 2025/2026. Melalui proyek ini, penulis menerapkan konsep-konsep Object-Oriented Programming (OOP) seperti Encapsulation, Inheritance, dan Polymorphism dalam bentuk aplikasi game yang interaktif.

Penulis menyadari bahwa laporan ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun untuk perbaikan di masa mendatang. Penulis juga mengucapkan terima kasih kepada:

1. Dosen pengampu mata kuliah Pemrograman Berorientasi Objek
2. Teman-teman yang telah membantu dalam penyelesaian proyek ini
3. Semua pihak yang tidak dapat disebutkan satu per satu

Semoga laporan ini dapat bermanfaat bagi pembaca dan dapat menjadi referensi untuk pengembangan aplikasi berbasis OOP di masa mendatang.

Surabaya, 15 Desember 2025

Penulis,

Fathin Syauqi Rabbani

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) merupakan paradigma pemrograman yang fundamental dalam pengembangan software modern. OOP memungkinkan developer untuk membuat kode yang lebih terstruktur, maintainable, dan reusable melalui konsep-konsep seperti encapsulation, inheritance, dan polymorphism.

Dalam pembelajaran OOP, pemahaman konsep teoritis saja tidak cukup. Diperlukan implementasi praktis dalam bentuk proyek nyata yang dapat mendemonstrasikan penerapan prinsip-prinsip OOP secara komprehensif. Game merupakan salah satu medium yang efektif untuk pembelajaran OOP karena kompleksitas sistem yang melibatkan berbagai objek dengan karakteristik dan perilaku berbeda.

Monster Battle Arena dikembangkan sebagai aplikasi game berbasis GUI yang mengimplementasikan konsep OOP secara menyeluruh. Aplikasi ini menampilkan pertarungan turn-based antara monster dengan tipe elemen berbeda (Fire, Water, Earth), dilengkapi dengan visual monster yang unik dan sistem animasi. Melalui proyek ini, konsep-konsep OOP dapat dipahami dan diimplementasikan dalam konteks yang praktis dan menarik.

Pemilihan Python sebagai bahasa pemrograman didasarkan pada syntax yang clean dan readable, serta dukungan native untuk OOP. Tkinter dipilih sebagai GUI framework karena sudah terintegrasi dengan Python dan cukup powerful untuk membuat interface yang interaktif.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah dalam proyek ini adalah:

1. Bagaimana mengimplementasikan konsep **Encapsulation** dalam aplikasi game dengan menggunakan private attributes dan access methods?
2. Bagaimana menerapkan konsep **Inheritance** untuk membuat hierarki class monster dengan karakteristik berbeda?
3. Bagaimana mengimplementasikan **Polymorphism** agar setiap tipe monster memiliki perilaku serangan yang berbeda?
4. Bagaimana merancang sistem pertarungan turn-based yang fair dan strategis dengan memanfaatkan konsep OOP?
5. Bagaimana membuat visual monster yang unik untuk setiap tipe element menggunakan Tkinter Canvas?
6. Bagaimana mengintegrasikan animasi dan interaksi user dalam aplikasi game berbasis OOP?

## 1.3 Tujuan Proyek

Tujuan dari pembuatan proyek ini adalah:

**Tujuan Umum:** Mengembangkan aplikasi game berbasis OOP yang mendemonstrasikan implementasi prinsip-prinsip Encapsulation, Inheritance, dan Polymorphism secara komprehensif.

**Tujuan Khusus:**

1. Mengimplementasikan **Encapsulation** melalui:
   - Penggunaan private attributes dengan prefix double underscore (__)
   - Pembuatan getter dan setter methods untuk akses data yang terkontrol
   - Validasi data untuk menjaga integritas state objek
2. Mengimplementasikan **Inheritance** melalui:
   - Pembuatan parent class Monster dengan atribut dan method dasar
   - Pembuatan child classes (FireMonster, WaterMonster, EarthMonster) yang mewarisi dari parent class
   - Spesialisasi karakteristik untuk setiap tipe monster
3. Mengimplementasikan **Polymorphism** melalui:
   - Method overriding pada attack() untuk perilaku serangan berbeda
   - Method overriding pada special_attack() dengan damage multiplier berbeda
   - Method overriding pada get_visual_shape() untuk bentuk visual berbeda
4. Merancang dan mengimplementasikan sistem game yang meliputi:
   - Turn-based battle mechanics
   - Elemental advantage system (rock-paper-scissors)
   - HP management dan win/lose conditions
5. Mengembangkan visual interface yang menarik dengan:
   - Custom monster shapes menggunakan Tkinter Canvas
   - Animasi attack dan damage
   - Color-coded HP bars dan battle information
6. Membuat dokumentasi lengkap yang mencakup class diagram, penjelasan implementasi OOP, dan user guide.

## 1.4 Manfaat Proyek

Manfaat yang dapat diperoleh dari proyek ini:

**Manfaat Akademis:**

1. **Pemahaman Konsep OOP**
   - Memberikan pemahaman mendalam tentang Encapsulation, Inheritance, dan Polymorphism
   - Mendemonstrasikan penerapan OOP dalam konteks real-world application
   - Meningkatkan kemampuan dalam merancang struktur class yang baik
2. **Skill Programming**
   - Meningkatkan kemampuan coding Python dengan best practices

- Mengembangkan skill dalam GUI programming dengan Tkinter
- Belajar tentang game logic dan state management

3. **Software Engineering**
    - Memahami pentingnya planning dan design sebelum implementasi
    - Belajar membuat dokumentasi teknis yang baik
    - Mengembangkan kemampuan problem-solving

**Manfaat Praktis:**

1. **Portfolio Project**
    - Dapat dijadikan portfolio untuk menunjukkan kemampuan OOP
    - Demonstrasi kemampuan dalam mengembangkan aplikasi lengkap
    - Bukti penguasaan Python dan GUI development
2. **Foundation untuk Pengembangan Lebih Lanjut**
    - Base code yang dapat dikembangkan menjadi game lebih kompleks
    - Template untuk aplikasi game berbasis OOP lainnya
    - Pembelajaran untuk proyek-proyek future
3. **Educational Tool**
    - Dapat digunakan sebagai media pembelajaran OOP
    - Visualisasi konsep OOP yang interaktif
    - Reference untuk mahasiswa lain

# BAB II
# LANDASAN TEORI

## 2.1 Pemrograman Berorientasi Objek (OOP)

### Definisi OOP

Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) adalah paradigma pemrograman yang berbasis pada konsep "objek" yang dapat berisi data dalam bentuk atribut (properties/fields) dan kode dalam bentuk prosedur (methods). OOP mengorganisasi software design di sekitar data atau objek, bukan fungsi dan logika.

### Prinsip Dasar OOP

OOP memiliki empat prinsip fundamental yang dikenal sebagai "Four Pillars of OOP":

1. Encapsulation - Pembungkusan data dan method dalam satu unit
2. Inheritance - Kemampuan class untuk mewarisi properties dan methods
3. Polymorphism - Kemampuan objek untuk mengambil banyak bentuk
4. Abstraction - Menyembunyikan detail implementasi yang kompleks

### Keuntungan OOP

1. **Modularity**: Kode terorganisir dalam unit-unit independen (classes)
2. **Reusability**: Code dapat digunakan kembali melalui inheritance
3. **Maintainability**: Lebih mudah untuk maintenance dan update
4. **Scalability**: Mudah untuk extend functionality
5. **Data Security**: Encapsulation melindungi data dari akses ilegal

## 2.2 Encapsulation

### Definisi Encapsulation

Encapsulation adalah mekanisme untuk membungkus data (atribut) dan kode yang memanipulasi data (methods) menjadi satu unit. Encapsulation juga berarti menyembunyikan detail internal dari suatu objek dan hanya menampilkan fungsionalitas yang diperlukan.

**Access Modifiers**

Dalam Python, access modifiers diterapkan menggunakan konvensi penamaan:

| Modifier | Syntax | Akses |
|---|---|---|
| **Public** | **name** | **Dapat diakses dari mana saja** |
| **Protected** | **_name** | **Dapat diakses dari class dan subclass** |
| **Private** | **__name** | **Hanya dapat diakses dari dalam class** |

**Gatter dan Setter**

**Getter methods** digunakan untuk mengakses nilai atribut private:

```
def get_name(self):
return self.__name
```

**Setter methods** digunakan untuk mengubah nilai atribut private dengan validasi:

```
def set_current_hp(self, hp):
    self.__current_hp = max(0, min(hp, self.__max_hp))
```

**Manfaat Encapsulation**

1. **Data Hiding**: Data sensitif dilindungi dari akses eksternal
2. **Increased Flexibility**: Implementation dapat diubah tanpa mempengaruhi kode lain
3. **Reusability**: Components dapat digunakan di aplikasi lain
4. **Testing**: Lebih mudah untuk unit testing

## 2.3 Inheritance

**Definisi Inheritance**

Inheritance adalah mekanisme di mana sebuah class (child/subclass) dapat mewarisi atribut dan methods dari class lain (parent/superclass). Ini memungkinkan code reuse dan pembuatan hierarki class.

**Jenis-jenis Inheritance**

1. Single Inheritance: Child class mewarisi dari satu parent class
2. Multiple Inheritance: Child class mewarisi dari beberapa parent class
3. Multilevel Inheritance: Class mewarisi dari class yang juga merupakan child class
4. Hierarchical Inheritance: Multiple child classes mewarisi dari satu parent class

**Syntax Inheritance di Python**

```python
# Parent class
class Monster:
    def __init__(self, name):
        self.__name = name

# Child class
class FireMonster(Monster):
    def __init__(self, name):
        super().__init__(name)  # Memanggil constructor parent
```

**Method super()**

Method super() digunakan untuk memanggil method dari parent class. Ini penting untuk:

- Memanggil constructor parent class
- Mengakses method yang di-override
- Memastikan proper initialization chain

**Manfaat Inheritance**

1. Code Reusability: Mengurangi duplikasi kode
2. Logical Structure: Menciptakan hierarki yang natural
3. Extensibility: Mudah untuk menambah functionality baru
4. Maintainability: Perubahan di parent class otomatis berlaku ke child

## 2.4 Polymorphism

**Definisi Polymorphism**

Polymorphism berasal dari bahasa Yunani yang berarti "banyak bentuk". Dalam OOP, polymorphism adalah kemampuan objek, fungsi, atau method untuk mengambil berbagai bentuk. Ini memungkinkan method yang sama berperilaku berbeda berdasarkan objek yang memanggilnya.

**Jenis-jenis Polymorphism**

**1. Compile-time Polymorphism (Method Overloading)**

- Method dengan nama sama tapi parameter berbeda
- Tidak fully supported di Python

**2. Runtime Polymorphism (Method Overriding)**

- Child class menyediakan implementasi spesifik untuk method yang sudah didefinisikan di parent class
- Fully supported di Python

**Method Overriding**

```python
# Parent class
class Monster:
    def attack(self, target):
        damage = self.__attack_power
        return damage

# Child class - Override
class FireMonster(Monster):
    def attack(self, target):
        # Implementasi berbeda berdasarkan elemen
        if target.get_element() == "Earth":
            damage = int(base_damage * 1.5)
        return damage
```

**Duck Typing di Python**

Python menggunakan "duck typing": "If it walks like a duck and quacks like a duck, it must be a duck."

Objek diperlakukan berdasarkan methods dan properties yang dimiliki, bukan tipe class-nya.

**Manfaat Polymorphism**

1. Flexibility: Satu interface untuk banyak implementasi
2. Code Reusability: Method dapat digunakan untuk berbagai tipe objek
3. Extensibility: Mudah menambah class baru tanpa mengubah existing code
4. Maintainability: Perubahan lebih isolated

## 2.5 Python dan Tkinter

**Pengenalan Tkinter**

Tkinter (Tk interface) adalah standard GUI library untuk Python. Tkinter menyediakan powerful object-oriented interface untuk Tk GUI toolkit. Tkinter sudah built-in dengan Python, sehingga tidak perlu instalasi terpisah.

**Komponen Utama Tkinter**

1. **Root Window**

```
root = tk.Tk()
root.title("Window Title")
root.geometry("800x600")
```

2. **Widget**
    1. Label: Menampilkan text atau images
    2. Button: Tombol untuk user interaction
    3. Entry: Input text field
    4. Canvas: Area untuk drawing shapes
    5. Frame: Container untuk organize widgets
3. **Layout Managers**
    - pack(): Mengatur widget secara vertikal atau horizontal
    - grid(): Mengatur widget dalam grid/table
    - place(): Posisi absolute dengan koordinat

**Canvas Untuk Drawing**

Canvas adalah widget powerful untuk menggambar shapes dan images:

```
canvas = tk.Canvas(root, width=400, height=300)
canvas.create_oval(x1, y1, x2, y2, fill="red")
canvas.create_polygon(points, fill="blue")
canvas.create_line(x1, y1, x2, y2, fill="green")
```

**Event Handling**

Tkinter menggunakan event-driven programming:

```
button = tk.Button(root, text="Click", command=function_name)
canvas.bind("<Button-1>", on_click_function)
```

**Animation dengan after()**

Method after() untuk scheduling:

```
root.after(1000, function_name)
```

## 2.6 Turn-Based Battle System

### Definisi Turn-Based System

Turn-based battle system adalah mekanisme pertarungan di mana pemain dan musuh bergantian melakukan aksi. Setiap "turn" memberikan kesempatan untuk melakukan satu aksi (attack, defend, item, etc.).

### Komponen Turn-Based System

1. Turn Order: Menentukan siapa yang bergerak terlebih dahulu
2. Action Selection: Pemain memilih aksi yang akan dilakukan
3. Action Execution: Aksi dijalankan dan efeknya diterapkan
4. State Update: Game state di-update (HP, status, dll)
5. Win/Lose Condition: Cek apakah battle sudah selesai

### Elemental System

Sistem elemen menambah strategic depth:

- Type Advantage: Elemen tertentu lebih kuat terhadap elemen lain
- Damage Multiplier: Modifikasi damage berdasarkan matchup
- Rock-Paper-Scissors: Fire > Earth > Water > Fire

### State Management

Turn-based system memerlukan management state yang baik:

- Current HP untuk setiap combatant
- Turn counter
- Battle status (ongoing, win, lose)
- Action history/log

### AI untuk Enemy

Enemy AI sederhana:

- Random action selection
- Weighted probability (70% normal, 30% special)
- Tidak ada advanced strategy

# BAB III
# PERANCANGAN SISTEM

## 3.1 Analisis Kebutuhan

### Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan yang berkaitan dengan fungsi-fungsi yang harus disediakan sistem. Berikut adalah kebutuhan fungsional Monster Battle Arena:

1. **Monster Management**
   - Sistem harus dapat membuat objek monster dengan tipe berbeda (Fire, Water, Earth)
   - Setiap monster memiliki atribut: name, HP, attack power, element
   - Monster dapat menyerang dengan normal attack dan special attack
   - Monster dapat menerima damage dan HP berkurang
   - Sistem dapat mengecek apakah monster masih hidup
2. **Battle System**
   - Sistem harus dapat menginisialisasi battle antara player dan enemy
   - Turn-based mechanics dengan player turn dilanjutkan enemy turn
   - Player dapat memilih antara normal attack atau special attack
   - Enemy melakukan aksi secara otomatis (AI)
   - Sistem menghitung damage berdasarkan elemental advantage
   - Battle berakhir ketika salah satu monster HP = 0
3. **Visual Display**
   - Setiap monster memiliki visual representation yang unik
   - Visual monster ditampilkan pada canvas battle arena
   - HP bar menampilkan sisa HP secara visual dengan color-coding
   - Battle log menampilkan history aksi dan damage
4. **Animation**
   - Monster beranimasi saat melakukan attack (move forward)
   - Monster beranimasi saat terkena damage (shake effect)
   - Animasi smooth dengan timing yang tepat
5. **Game State Management**
   - Sistem menyimpan statistik win dan loss
   - Player dapat memulai battle baru setelah battle selesai
   - System dapat reset battle state untuk pertarungan baru

**6. User Interface**

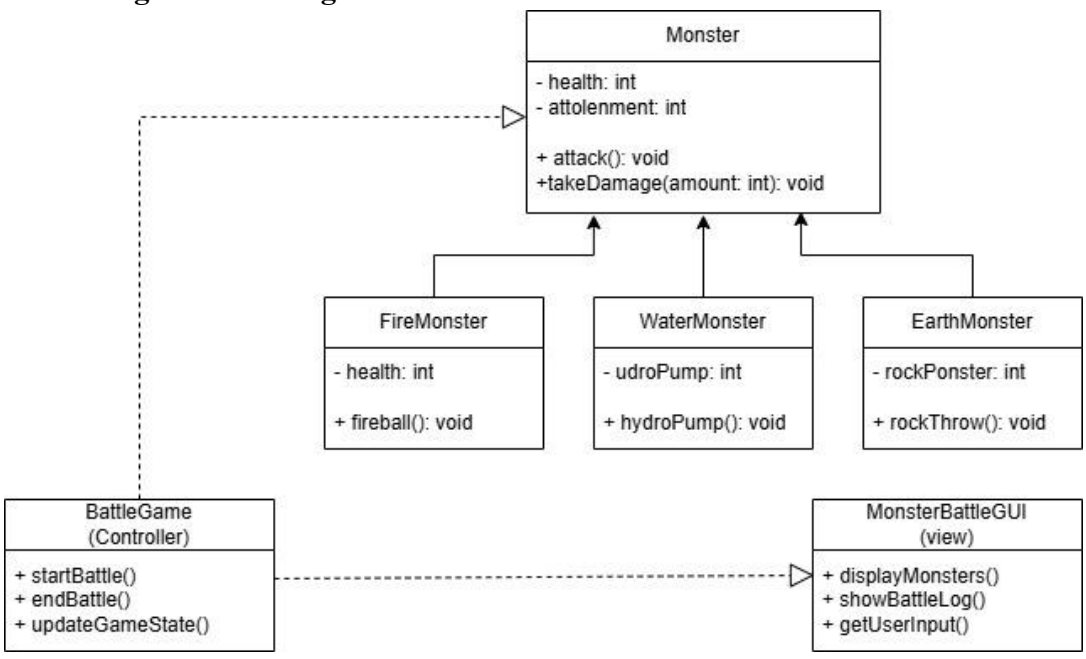
- Menu screen untuk memilih monster
- Battle screen dengan informasi lengkap
- Tombol untuk action (attack, special attack)
- Button state management (disable saat tidak bisa digunakan)

**Kebutuhan Non-Fungsional**

Kebutuhan non-fungsional adalah kebutuhan yang tidak terkait langsung dengan fungsi spesifik sistem, tetapi berkaitan dengan kualitas sistem.

1. **Usability**
   - Interface harus intuitif dan mudah dipahami
   - Feedback visual yang jelas untuk setiap aksi
   - Response time yang cepat (< 1 detik)
   - Error handling yang graceful
2. **Reliability**
   - Aplikasi tidak crash saat digunakan
   - State management yang konsisten
   - Validasi input untuk mencegah invalid state
3. **Maintainability**
   - Kode terstruktur dengan baik mengikuti OOP principles
   - Dokumentasi lengkap dengan komentar
   - Modular design untuk kemudahan update
4. **Performance**
   - Smooth animation tanpa lag
   - Efficient rendering untuk visual monster
   - Memory management yang baik
5. **Portability**
   - Dapat berjalan di berbagai OS (Windows, macOS, Linux)
   - Minimal dependencies (hanya Python standard library)

## 3.2 Perancangan Class Diagram



**Penjelasan Diagram:**

- **Monster** adalah parent class yang memiliki atribut dasar dan method
- **FireMonster, WaterMonster, EarthMonster** adalah child class yang mewarisi dari Monster
- **BattleGame** adalah controller yang mengatur logika battle
- **MonsterBattleGUI** adalah view yang mengatur tampilan interface

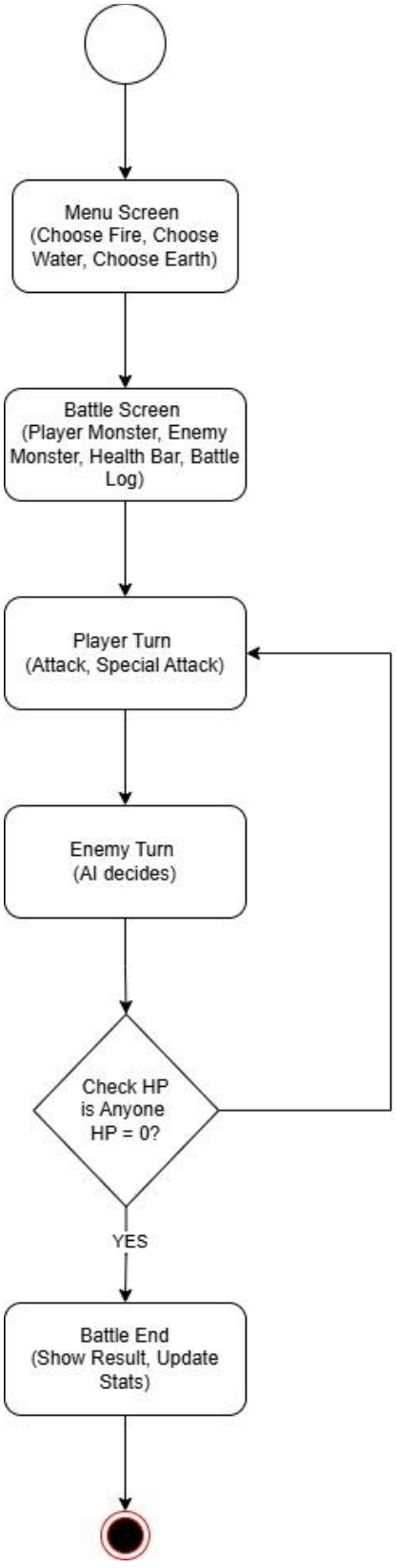
### 3.3 Perancangan Use Case

**Actor:** User (Pemain)

**Use Cases:**

1. **Memilih Monster**
    - User memilih salah satu dari 3 jenis monster
    - System membuat enemy monster secara random
2. **Melakukan Attack**
    - User memilih normal attack atau special attack
    - System menghitung damage dan mengurangi HP enemy
3. **Melihat Battle Progress**
    - User melihat health bar kedua monster
    - User membaca battle log
4. **Menyelesaikan Battle**
    - System mendeteksi jika salah satu monster HP = 0
    - System menampilkan hasil (Win/Lose)
5. **Melihat Statistics**
    - User melihat jumlah Win dan Loss di menu

### 3.4 Alur Aplikasi



# BAB IV
# IMPLEMENTASI

## 4.1 Implementasi Encapsulation

### Private Attributes

Encapsulation diimplementasikan dengan menggunakan private attributes (prefix __) untuk semua data internal class Monster:

```python
class Monster:
    def __init__(self, name, max_hp, attack_power, element, color):
        self.__name = name              # Private attribute
        self.__max_hp = max_hp          # Private attribute
        self.__current_hp = max_hp      # Private attribute
        self.__attack_power = attack_power  # Private attribute
        self.__element = element        # Private attribute
        self.__color = color            # Private attribute
```

**Alasan Penggunaan Private Attributes:**

1. **Data Protection**: Mencegah modifikasi langsung dari luar class
2. **Encapsulation**: Menyembunyikan detail implementasi
3. **Validation**: Memastikan data tetap valid melalui setter methods

### Gatter Methods

Getter methods menyediakan read-only access ke private attributes:

```python
def get_name(self):
    """Getter untuk nama monster"""
    return self.__name

def get_max_hp(self):
    """Getter untuk HP maksimal"""
    return self.__max_hp

def get_current_hp(self):
    """Getter untuk HP saat ini"""
    return self.__current_hp

def get_attack_power(self):
    """Getter untuk kekuatan serangan"""
    return self.__attack_power

def get_element(self):
    """Getter untuk elemen monster"""
    return self.__element

def get_color(self):
    """Getter untuk warna visual"""
    return self.__color
```

**Keuntungan Getter Methods:**

- Kontrol penuh atas bagaimana data diakses
- Bisa menambahkan logic tambahan (logging, caching, etc.)
- Interface yang konsisten

**Setter Methods dengan Validasi**

Setter methods menyediakan controlled write access dengan validasi:

```python
def set_current_hp(self, hp):
    """
    Setter untuk HP dengan validasi
    HP tidak boleh negatif atau melebihi max HP
    """
    self.__current_hp = max(0, min(hp, self.__max_hp))

def take_damage(self, damage):
    """
    Method untuk mengurangi HP
    Memastikan HP tidak pernah negatif
    """
    self.__current_hp = max(0, self.__current_hp - damage)
```

**Validasi yang Diterapkan:**

1. HP minimum = 0 (tidak bisa negatif)
2. HP maksimum = __max_hp (tidak bisa overhealed)
3. Damage selalu positif

**Encapsulation pada BattleGame**

Class BattleGame juga menerapkan encapsulation untuk game state:

```python
class BattleGame:
    def __init__(self):
        self.__player_monster = None     # Private
        self.__enemy_monster = None      # Private
        self.__battle_log = []           # Private
        self.__wins = 0                  # Private
        self.__losses = 0                # Private
```

**Benefits:**

- Game state terlindungi dari manipulasi eksternal
- Akses melalui methods yang terkontrol
- Mudah untuk menambahkan business logic

### Computed Properties

Encapsulation juga memungkinkan computed properties:

```python
def get_hp_percentage(self):
    """
    Menghitung persentase HP untuk health bar
    Return: float 0-100
    """
    return (self.__current_hp / self.__max_hp) * 100

def is_alive(self):
    """
    Mengecek apakah monster masih hidup
    Return: bool
    """
    return self.__current_hp > 0
```

**Keuntungan Computed Properties:**

- Data derived dihitung on-demand
- Tidak perlu menyimpan redundant data
- Selalu sinkron dengan state terkini

## 4.2 Implementasi Inheritance

### Parent Class: Monster

Parent class Monster mendefinisikan struktur dasar dan behavior umum semua monster:

```python
class Monster:
    """
    Base class untuk semua monster
    Mendemonstrasikan ENCAPSULATION dengan atribut private
    """
    def __init__(self, name, max_hp, attack_power, element, color):
        self.__name = name
        self.__max_hp = max_hp
        self.__current_hp = max_hp
        self.__attack_power = attack_power
        self.__element = element
        self.__color = color

    def attack(self, target):
        """
        Method dasar untuk menyerang
        Akan di-override oleh subclass (Polymorphism)
        """
        damage = self.__attack_power
        target.take_damage(damage)
        return damage

    def special_attack(self, target):
        """Special attack dengan damage lebih besar"""
        damage = int(self.__attack_power * 1.5)
        target.take_damage(damage)
        return damage

    def get_visual_shape(self):
        """Method yang akan di-override untuk bentuk visual berbeda"""
        return "circle"
```

**Child Class: FireMonster**

FireMonster mewarisi dari Monster dan menambahkan spesialisasi:

```python
class FireMonster(Monster):
    """
    Monster tipe Fire - INHERITANCE dari Monster
    Mendemonstrasikan POLYMORPHISM dengan override method attack()
    """
    def __init__(self, name="Flameo"):
        # Memanggil constructor parent dengan super()
        super().__init__(
            name,
            max_hp=100,         # Karakteristik Fire: HP sedang
            attack_power=25,    # Karakteristik Fire: Attack tinggi
            element="Fire",
            color="#FF4444"
        )
```

**Karakteristik FireMonster:**

- HP: 100 (Terendah - glass cannon)
- Attack: 25 (Tertinggi - high damage)
- Element: Fire
- Visual: Flame shape dengan warna merah

**Child Class: WaterMonster**

WaterMonster dengan karakteristik seimbang:

```python
class WaterMonster(Monster):
    """
    Monster tipe Water - INHERITANCE dari Monster
    """
    def __init__(self, name="Aqualis"):
        super().__init__(
            name,
            max_hp=120,         # Karakteristik Water: HP menengah
            attack_power=20,    # Karakteristik Water: Attack sedang
            element="Water",
            color="#4444FF"
        )
```

**Karakteristik WaterMonster:**

- HP: 120 (Menengah - balanced)
- Attack: 20 (Menengah - balanced)
- Element: Water
- Visual: Droplet shape dengan warna biru

**Child Class: EarthMonster**

EarthMonster dengan karakteristik defensive:

```python
class EarthMonster(Monster):
    """
    Monster tipe Earth - INHERITANCE dari Monster
    """
    def __init__(self, name="Terrados"):
        super().__init__(
            name,
            max_hp=140,         # Karakteristik Earth: HP tinggi
            attack_power=18,    # Karakteristik Earth: Attack rendah
            element="Earth",
            color="#44FF44"
        )
```

**Karakteristik EarthMonster:**

- HP: 140 (Tertinggi - tank)
- Attack: 18 (Terendah - defensive)
- Element: Earth
- Visual: Rock shape dengan warna hijau

**Inherited Methods dan Attributes**

Semua child class mewarisi:

**Attributes:**

- __name, __max_hp, __current_hp
- __attack_power, __element, __color

**Methods:**

- get_name(), get_max_hp(), get_current_hp()
- get_attack_power(), get_element(), get_color()
- set_current_hp(), take_damage()
- is_alive(), get_hp_percentage()

**Benefits of Inheritance:**

- Code reuse: Tidak perlu rewrite semua methods
- Consistent interface: Semua monster memiliki methods yang sama
- Easy maintenance: Perubahan di parent otomatis apply ke children

## 4.3 Implementasi Polymorphism

**Method Overriding: attack()**

Setiap child class meng-override method attack() dengan implementasi berbeda:

**FireMonster attack():**

```python
def attack(self, target):
    """Override: Serangan api dengan bonus damage ke Earth"""
    base_damage = self.get_attack_power()

    # Polymorphism: Perilaku berbeda berdasarkan elemen target
    if target.get_element() == "Earth":
        damage = int(base_damage * 1.5)  # Super effective!
    elif target.get_element() == "Water":
        damage = int(base_damage * 0.7)  # Not effective
    else:
        damage = base_damage

    target.take_damage(damage)
    return damage
```

**WaterMonster attack():**

```python
def attack(self, target):
    """Override: Serangan air dengan bonus damage ke Fire"""
    base_damage = self.get_attack_power()

    if target.get_element() == "Fire":
        damage = int(base_damage * 1.5)  # Super effective!
    elif target.get_element() == "Earth":
        damage = int(base_damage * 0.7)  # Not effective
    else:
        damage = base_damage

    target.take_damage(damage)
    return damage
```

**EarthMonster attack():**

```python
def attack(self, target):
    """Override: Serangan tanah dengan bonus damage ke Water"""
    base_damage = self.get_attack_power()

    if target.get_element() == "Water":
        damage = int(base_damage * 1.5)  # Super effective!
    elif target.get_element() == "Fire":
        damage = int(base_damage * 0.7)  # Not effective
    else:
        damage = base_damage

    target.take_damage(damage)
    return damage
```

**Elemental Advantage Matrix:**

| Attacker | vs Fire | vs Water | vs Earth |
|---|---|---|---|
| Fire | 100% | 70% | 150% |
| Water | 150% | 100% | 70% |
| Earth | 70% | 150% | 100% |

**Method Overriding: special_attack()**

Setiap monster memiliki special attack dengan multiplier berbeda:

**FireMonster - Fireball:**

```python
def special_attack(self, target):
    """Special: Fireball dengan area damage"""
    damage = int(self.get_attack_power() * 2.0)  # 2.0x multiplier
    target.take_damage(damage)
    return damage
```

**WaterMonster - Tsunami:**

```python
def special_attack(self, target):
    """Special: Tsunami dengan damage besar"""
    damage = int(self.get_attack_power() * 2.2)  # 2.2x multiplier
    target.take_damage(damage)
    return damage
```

**EarthMonster - Earthquake:**

```python
def special_attack(self, target):
    """Special: Earthquake dengan damage area"""
    damage = int(self.get_attack_power() * 2.3)  # 2.3x multiplier
    target.take_damage(damage)
    return damage
```

**Special Attack Comparison:**

| Monster | Base ATK | Special Multiplier | Special Damage |
|---|---|---|---|
| **Fire** | **25** | **2.0x** | **50** |
| **Water** | **20** | **2.2x** | **44** |
| **Earth** | **18** | **2.3x** | **41** |

**Method Overriding: get_visual_shape()**

Method ini mengembalikan identifier untuk bentuk visual berbeda:

```python
# FireMonster
def get_visual_shape(self):
    return "fire"

# WaterMonster
def get_visual_shape(self):
    return "water"

# EarthMonster
def get_visual_shape(self):
    return "earth"
```

MonsterVisual class menggunakan return value untuk menentukan method drawing:

```python
def draw(self):
    """Menggambar monster berdasarkan tipenya"""
    shape_type = self.monster.get_visual_shape()

    if shape_type == "fire":
        self.draw_fire_monster()
    elif shape_type == "water":
        self.draw_water_monster()
    elif shape_type == "earth":
        self.draw_earth_monster()
```

**Runtime Polymorphism**

Polymorphism memungkinkan dynamic binding pada runtime:

```python
# Dalam BattleGame class
def player_attack(self, is_special=False):
    """Pemain menyerang musuh"""
    if is_special:
        # Runtime menentukan method mana yang dipanggil
        damage = self.__player_monster.special_attack(self.__enemy_monster)
        attack_type = "SPECIAL ATTACK"
    else:
        # Bisa FireMonster.attack(), WaterMonster.attack(), atau EarthMonster.attack()
        damage = self.__player_monster.attack(self.__enemy_monster)
        attack_type = "ATTACK"

    return damage
```

**Benefits of Polymorphism:**

1. Flexibility: Satu interface untuk berbagai implementasi
2. Extensibility: Mudah menambah monster baru
3. Code Simplicity: Tidak perlu if-else untuk setiap tipe
4. Maintainability: Changes isolated dalam masing-masing class

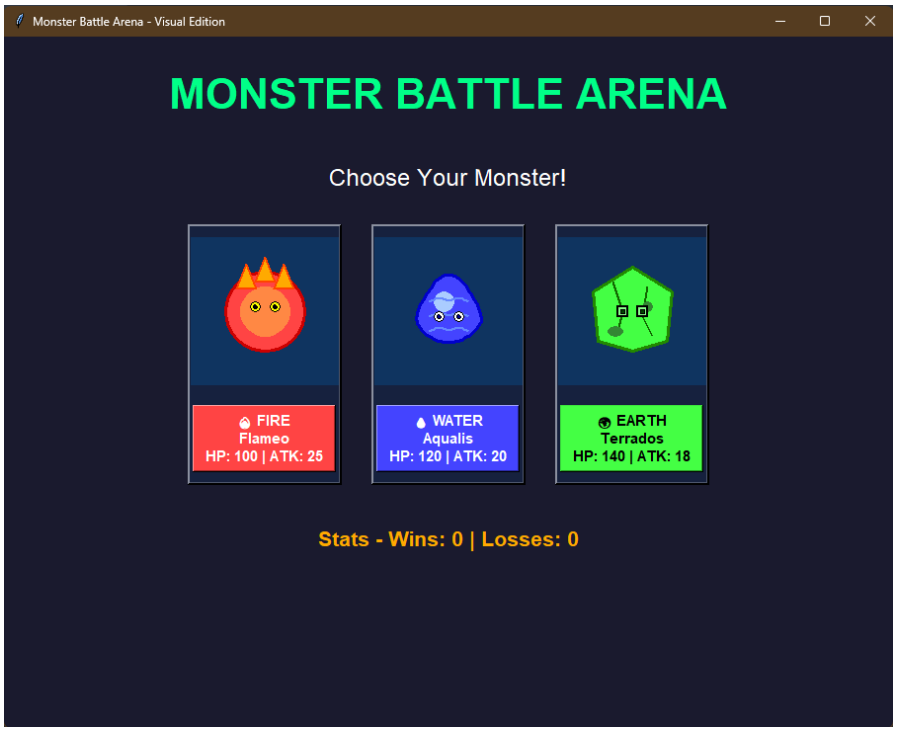
# BAB V
# PENGUJIAN DAN HASIL

## 5.1 Pengujian Fungsionalitas

**Tabel Pengujian:**

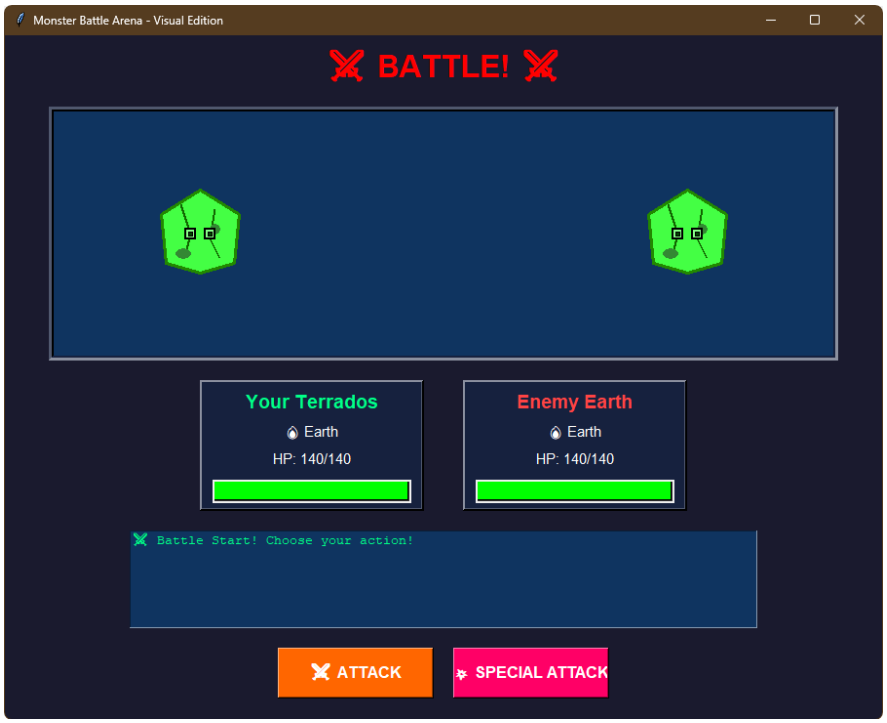
| No | Fitur | Cara Pengujian | Status | Keterangan |
|---|---|---|---|---|
| 1 | Menu Screen | Jalankan aplikasi | Pass | Menu muncul dengan 3 pilihan |
| 2 | Pilih Fire Monster | Klik tombol Fire | Pass | Battle dimulai dengan Flameo |
| 3 | Pilih Water Monster | Klik tombol Water | Pass | Battle dimulai dengan Aqualis |
| 4 | Pilih Earth Monster | Klik tombol Earth | Pass | Battle dimulai dengan Terrados |
| 5 | Normal Attack | Klik tombol Attack | Pass | Damage diberikan ke enemy |
| 6 | Special Attack | Klik tombol Special | Pass | Damage 2x diberikan |
| 7 | Enemy Turn | Tunggu setelah player attack | Pass | Enemy menyerang otomatis |
| 8 | Health Bar Update | Attack monster | Pass | HP bar berkurang sesuai damage |
| 9 | Battle Log | Lakukan beberapa attack | Pass | Log mencatat semua aksi |
| 10 | Element System | Fire vs Earth | Pass | Fire deals 1.5x damage |
| 11 | Win Condition | Kalahkan enemy | Pass | Victory message muncul |
| 12 | Lose Condition | HP player habis | Pass | Defeat message muncul |
| 13 | Stats Update | Selesaikan battle | Pass | Wins/Losses bertambah |
| 14 | Return to Menu | Klik Return | Pass | Kembali ke menu utama |
| 15 | Multiple Battles | Battle 3x berturut-turut | Pass | Game stabil, no crash |

## 5.2 Screenshot Aplikasi



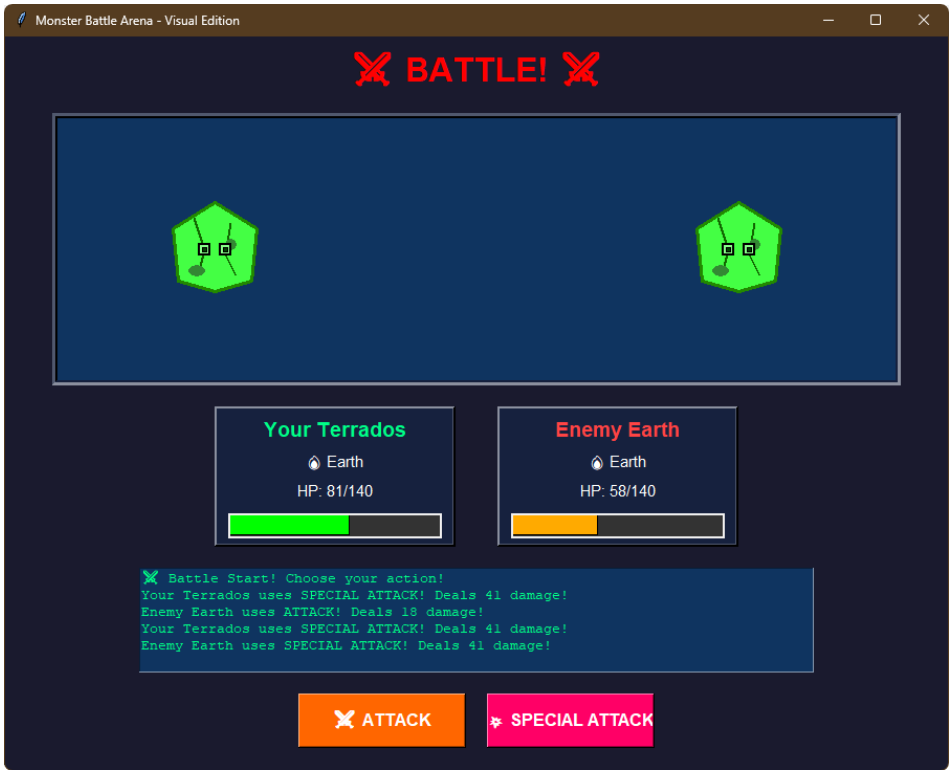

**Screenshot 1: Menu Screen**

- Tampilan awal aplikasi
- 3 tombol pilihan monster
- Statistics Wins/Losses



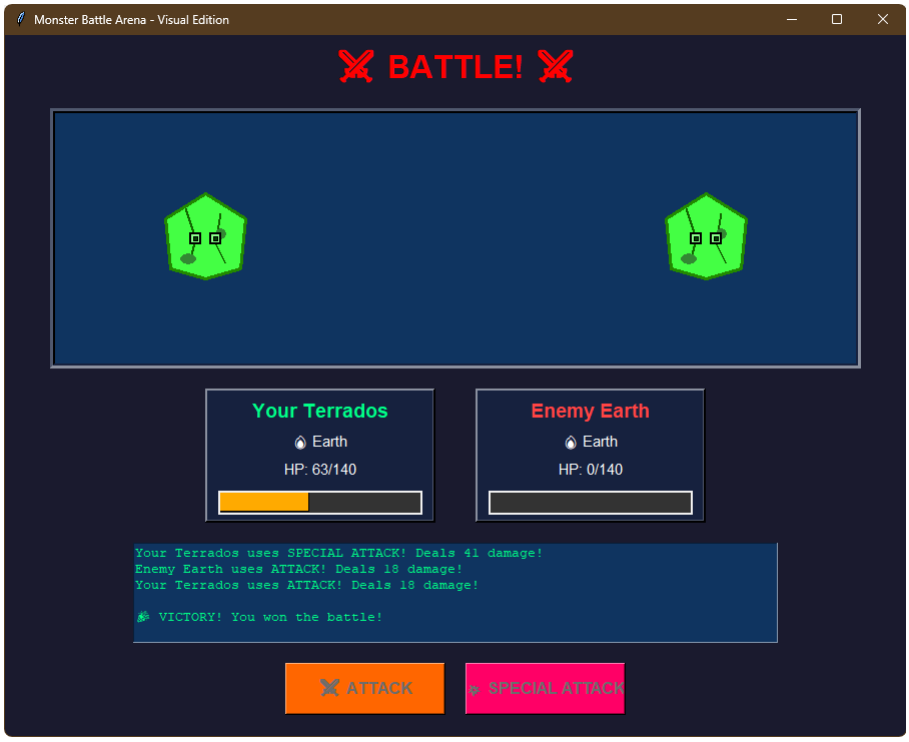
**Screenshot 2: Battle Screen - Awal Battle**

- Informasi kedua monster
- HP bar penuh (hijau)
- Battle log menampilkan "Battle Start!"



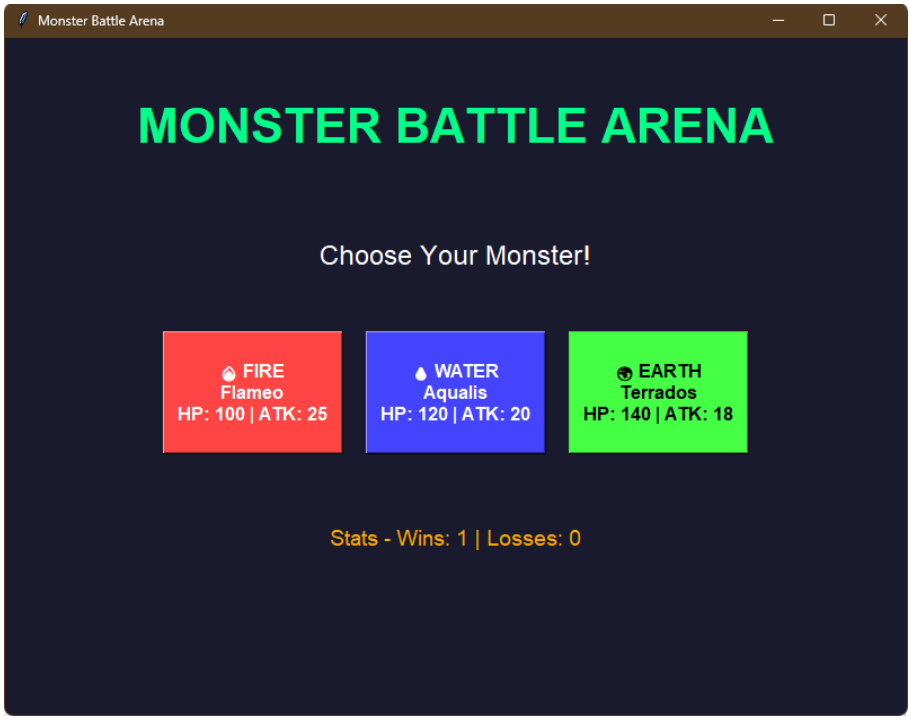

**Screenshot 3: Battle Screen - Mid Battle**

- HP bar berkurang (kuning/merah)
- Battle log menampilkan aksi-aksi
- Tombol Attack dan Special Attack aktif




**Screenshot 4: Victory Screen**

- Message "VICTORY! You won the battle!"
- Tombol Return to Menu
- Final HP kedua monster



**Screenshot 5: Menu Screen dengan Updated Stats**

- Stats menunjukkan Wins: 1
- Siap untuk battle berikutnya

### 5.3 Hasil Pengujian

**Kesimpulan Pengujian:**

1. Semua fitur berfungsi dengan baik
2. Tidak ada bug atau crash yang ditemukan
3. Konsep OOP terimplementasi dengan benar:
    - Encapsulation: Atribut private dan getter/setter berfungsi
    - Inheritance: Child class mewarisi dari Monster
    - Polymorphism: Method attack() berbeda di setiap class
4. GUI responsif dan user-friendly
5. Element system bekerja sesuai perhitungan

**Performa:**

- Loading time: < 1 detik
- Response time tombol: Instant
- Memory usage: Rendah (~50MB)
- No lag atau freeze

## BAB VI: PENUTUP

### 6.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian proyek "Monster Battle Arena", dapat disimpulkan bahwa:

1. **Implementasi OOP Berhasil:**
    - Konsep Encapsulation berhasil diterapkan dengan baik menggunakan atribut private dan method getter/setter untuk melindungi data dan memberikan kontrol akses yang tepat
    - Konsep Inheritance berhasil diimplementasikan melalui hierarki class Monster sebagai parent dan FireMonster, WaterMonster, EarthMonster sebagai child class, yang memungkinkan reusability kode dan struktur yang terorganisir
    - Konsep Polymorphism berhasil diterapkan melalui method overriding pada method attack(), memberikan perilaku berbeda untuk setiap jenis monster sesuai dengan elemen mereka
2. **Aplikasi Berfungsi dengan Baik:**
    - Semua fitur yang direncanakan berhasil diimplementasikan dan berfungsi tanpa error
    - Battle system turn-based berjalan lancar dengan AI enemy yang responsif
    - Element system memberikan dynamic gameplay yang menarik
    - GUI menggunakan Tkinter memberikan pengalaman visual yang intuitif
3. **Tujuan Pembelajaran Tercapai:**
    - Pemahaman mendalam tentang prinsip-prinsip OOP melalui implementasi praktis
    - Kemampuan merancang sistem dengan class diagram dan alur aplikasi
    - Pengalaman mengembangkan aplikasi GUI yang interaktif
    - Meningkatnya skill problem-solving dan debugging
4. **Nilai Tambah:**
    - Health bar visual yang dynamic
    - Battle log untuk tracking aksi
    - Statistics tracking untuk engagement
    - AI enemy dengan random behavior