LAPORAN PROJEK UJIAN AKHIR SEMESTER

MATA KULIAH PEMROGRAMAN BERBASIS OBJEK

"MATH ADVENTURE"

OLEH

MUHAMMAD DANI MALKA MULTAZAM

24091397087

2024C

PROGRAM STUDI MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

2025

BAB 1:

PENDAHULUAN

1.1 Latar Belakang

Pembelajaran matematika sering kali dianggap membosankan oleh siswa. Untuk mengatasi hal ini, diperlukan pendekatan yang lebih interaktif dan menyenangkan. Math Adventure adalah game edukasi yang menggabungkan pembelajaran matematika dengan gameplay yang engaging.

Proyek ini dikembangkan menggunakan bahasa pemrograman Python dengan library Pygame, dan menerapkan konsep Object-Oriented Programming (OOP) secara menyeluruh. Setiap entitas dalam game (player, monster, soal) direpresentasikan sebagai objek dengan atribut dan method yang terorganisir.

1.2 Tujuan Proyek

1. Mengimplementasikan prinsip-prinsip OOP (Encapsulation, Inheritance, Polymorphism) dalam aplikasi nyata
2. Menciptakan game edukasi yang menarik untuk pembelajaran matematika
3. Menerapkan best practices dalam pengembangan software berbasis OOP
4. Membuat aplikasi yang modular, maintainable, dan extensible

1.3 Ruang Lingkup

Fitur Utama:

- Sistem pertarungan berbasis soal matematika (penjumlahan, pengurangan, perkalian, pembagian).
- Timer 30 detik untuk setiap soal.
- Cooldown system untuk gameplay yang smooth.
- Level progression system (1-10).
- Final Boss battle sebagai win condition.
- Multiple monster types dengan karakteristik berbeda.
- Health, lives, dan scoring system.

BAB 2:

KONSEP OOP YANG DIIMPLEMENTASIKAN

2.1 ENCAPSULATION (Enkapsulasi)

Definisi

Encapsulation adalah proses menyembunyikan detail implementasi internal dan hanya menyediakan interface tertentu untuk berinteraksi dengan objek. Ini dilakukan dengan menggunakan access modifier: private (__), protected (_), dan public.

Tujuan Encapsulation

1. Data Protection: Mencegah perubahan data secara langsung yang dapat merusak state objek
2. Controlled Access: Semua perubahan data melalui method yang ter-validasi
3. Maintainability: Dapat mengubah internal implementation tanpa mengubah interface
4. Business Logic: Dapat menambahkan logic otomatis saat set/get data

Implementasi dalam Math Adventure

A. Encapsulation pada Class Character

```python
from abc import ABC

class Character(ABC):
    def __init__(self, name, health, x, y):
        self._name = name
        self.__health = health
        self._x = x
        self._y = y
        self._is_alive = True

    # Getter dan Setter (Encapsulation)
    def get_health(self):
        return self.__health

    def set_health(self, value):
        self.__health = max(0, value)
        if self.__health == 0:
            self._is_alive = False

    def get_position(self):
        return (self._x, self._y)

    def move(self, dx, dy):
        """Method untuk menggerakkan karakter"""
        self._x += dx
        self._y += dy

    def take_damage(self, damage):
        """Method untuk menerima damage"""
        self.__health -= damage
        if self.__health <= 0:
            self.__health = 0
            self._is_alive = False

    def is_alive(self):
        return self._is_alive

    @abstractmethod
    def attack(self):
        pass
```

Penjelasan:

- __health adalah private, tidak bisa diakses langsung: character.__health akan error

- Akses hanya melalui get_health() dan set_health()
- set_health() melakukan validasi: health tidak pernah negatif
- take_damage() otomatis update _is_alive ketika health = 0
- Keuntungan: Data terlindungi, logic terpusat, mudah di-debu

B. Encapsulation pada Class Player

```python
from Class.Character import Character
class Player(Character):
    def __init__(self, name, x=100, y=400):
        super().__init__(name, health=100, x=x, y=y)
        self.__score = 0  # Private attribute
        self.__level = 1
        self.__lives = 3
        self.__speed = 5

    # Getters (Encapsulation)
    def get_score(self):
        return self.__score

    def get_level(self):
        return self.__level

    def get_lives(self):
        return self.__lives

    def get_speed(self):
        return self.__speed

    # Methods
    def answer_question(self, is_correct):
        """Method untuk memproses jawaban"""
        if is_correct:
            self.gain_score(10 * self.__level)
            return True
        else:
            self.take_damage(20)
            return False

    def gain_score(self, points):
        """Method untuk menambah score"""
        self.__score += points
        # Level up setiap 100 points
        if self.__score >= self.__level * 100:
            self.level_up()

    def level_up(self):
        """Method untuk naik level"""
        self.__level += 1
        self.set_health(self.get_health() + 20)
        print(f"Level Up! Now level {self.__level}")

    def lose_life(self):
        """Method untuk kehilangan nyawa"""
        self.__lives -= 1
        if self.__lives > 0:
            self.set_health(100)  # Reset health

    def attack(self):
        return 15 * self.__level

    def move_with_keys(self, keys):
        if keys[0]:  # UP
            self.move(0, -self.__speed)
        if keys[1]:  # DOWN
            self.move(0, self.__speed)
        if keys[2]:  # LEFT
            self.move(-self.__speed, 0)
        if keys[3]:  # RIGHT
            self.move(self.__speed, 0)
```

Penjelasan Encapsulation:

1. Data Protection:
    - __score, __level, __lives adalah private
    - Tidak bisa diubah dengan: player.__score = 9999
    - Harus pakai method: player.gain_score(10)
2. Controlled Access:
    - Setiap perubahan melalui method yang ter-validasi
    - gain_score() otomatis trigger level_up() kalau cukup points
    - lose_life() otomatis reset health

3. Business Logic:
    - Level up logic terpusat di satu tempat
    - Kalau mau ubah rumus level up, cukup edit 1 method
    - Tidak ada duplikasi logic di berbagai tempat

C. Encapsulation pada Class Monster

```python
from Class.Character import Character
import random

class Monster(Character)
    def __init__(self, name, difficulty, x, y):
        health = 30 + (difficulty * 20)
        super().__init__(name, health, x, y)
        self.__difficulty = difficulty  # Private (Encapsulation)
        self.__reward_points = 10 * difficulty
        self.__speed = 2 + difficulty

    def get_difficulty(self):
        return self.__difficulty

    def get_reward_points(self):
        return self.__reward_points

    def attack(self):
        """Override method attack (Polymorphism)"""
        return 10 + (self.__difficulty * 5)

    def move_towards_player(self, player_x, player_y):
        """AI sederhana untuk mengejar player"""
        if self._x < player_x:
            self._x += self.__speed
        elif self._x > player_x:
            self._x -= self.__speed

        if self._y < player_y:
            self._y += self.__speed
        elif self._y > player_y:
            self._y -= self.__speed

    def get_question(self):
        """Generate soal berdasarkan difficulty"""
        if self.__difficulty == 1:
            # Penjumlahan sederhana
            a = random.randint(1, 20)
            b = random.randint(1, 20)
            question = f"{a} + {b} = ?"
            answer = a + b
        elif self.__difficulty == 2:
            # Pengurangan dan perkalian
            a = random.randint(1, 12)
            b = random.randint(1, 12)
            question = f"{a} × {b} = ?"
            answer = a * b
        else:
            # Pembagian
            b = random.randint(2, 10)
            answer = random.randint(2, 15)
            a = answer * b
            question = f"{a} ÷ {b} = ?"

        return question, answer


class Boss(Monster):
    def __init__(self, name, difficulty, x, y):
        super().__init__(name, difficulty + 2, x, y)
        self.__special_ability = "Multi-Question"
        self.__special_cooldown = 0

    def attack(self):
        base_damage = super().attack()
        return base_damage * 2

    def special_attack(self):
        if self.__special_cooldown == 0:
            self.__special_cooldown = 5
            return True  # Trigger multiple questions
        return False

    def update_cooldown(self):
        if self.__special_cooldown > 0:
            self.__special_cooldown -= 1

    def get_question(self):
        # Soal kombinasi operasi
        a = random.randint(5, 20)
        b = random.randint(2, 10)
        c = random.randint(1, 10)

        operations = [
            (f"({a} + {b}) × {c} = ?", (a + b) * c),
            (f"{a} × {b} - {c} = ?", a * b - c),
            (f"({a} - {b}) × {c} = ?", (a - b) * c)
        ]

        question, answer = random.choice(operations)
        return question, answer
```

Penjelasan:

- Difficulty menentukan health, reward, dan tipe soal
- Logic pembuatan soal **tersembunyi** di dalam method

- External code hanya perlu call get_question(), tidak perlu tahu detail

2.2 INHERITANCE (Pewarisan)

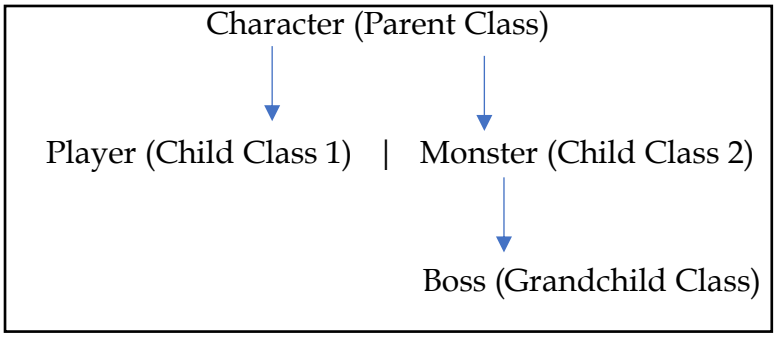
Definisi

Inheritance adalah mekanisme yang memungkinkan sebuah class (child class) mewarisi atribut dan method dari class lain (parent class). Ini memungkinkan code reuse dan membuat hierarki class yang logis.

Tujuan Inheritance

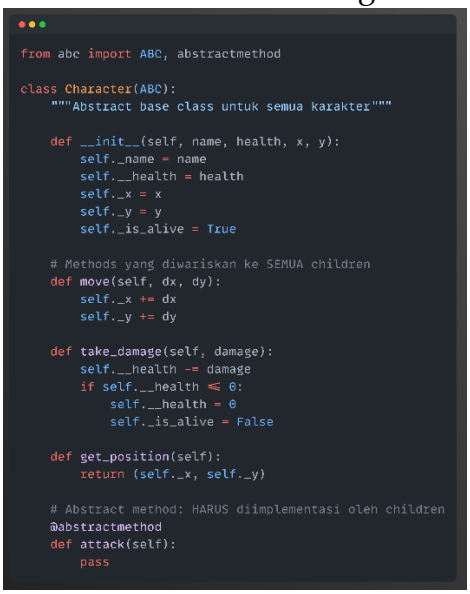
1. Code Reuse: Tidak perlu tulis ulang code yang sama
2. Extensibility: Mudah menambah fitur baru via subclass
3. Logical Hierarchy: Struktur yang jelas dan mudah dipahami
4. Maintainability: Perubahan di parent otomatis diwariskan ke children

Hierarki Class dalam Math Adventure



Implementasi Multi-Level Inheritance

- Level 1: Character sebagai Base Class



```python
from abc import ABC, abstractmethod

class Character(ABC):
    """Abstract base class untuk semua karakter"""

    def __init__(self, name, health, x, y):
        self._name = name
        self.__health = health
        self._x = x
        self._y = y
        self._is_alive = True

    # Methods yang diwariskan ke SEMUA children
    def move(self, dx, dy):
        self._x += dx
        self._y += dy

    def take_damage(self, damage):
        self.__health -= damage
        if self.__health <= 0:
            self.__health = 0
            self._is_alive = False

    def get_position(self):
        return (self._x, self._y)

    # Abstract method: HARUS diimplementasi oleh children
    @abstractmethod
    def attack(self):
        pass
```

- Level 2A: Player extends Character

```python
class Player(Character):
    """Player class mewarisi Character"""

    def __init__(self, name, x=100, y=400):
        # Memanggil constructor parent
        super().__init__(name, health=100, x=x, y=y)

        # Atribut tambahan khusus Player
        self.__score = 0
        self.__level = 1
        self.__lives = 3

    # Implementasi abstract method (POLYMORPHISM)
    def attack(self):
        return 15 * self.__level

    # Method baru khusus Player
    def gain_score(self, points):
        self.__score += points

    def move_with_keys(self, keys):
        """Menggunakan move() yang diwarisi dari parent"""
        if keys[0]:  # UP
            self.move(0, -5)
        # ... dst
```

Yang Diwarisi dari Parent:

- move(dx, dy) - digunakan dalam move_with_keys()

- take_damage(damage) - langsung bisa dipakai
- get_position() - untuk collision detection
- _x, _y, __health - atribut dasar
- is_alive() - untuk check status

Yang Ditambahkan:

- __score, __level, __lives - atribut baru
- gain_score(), level_up() - method baru
- move_with_keys() - method khusus player
- attack() - implementasi abstract method

- Level 2B: Monster extends Character

```python
class Monster(Character):
    """Monster class mewarisi Character"""

    def __init__(self, name, difficulty, x, y):
        health = 30 + (difficulty * 20)
        super().__init__(name, health, x, y)

        self.__difficulty = difficulty
        self.__reward_points = 10 * difficulty
        self.__speed = 2 + difficulty

    def attack(self):
        """Override abstract method"""
        return 10 + (self.__difficulty * 5)

    def move_towards_player(self, player_x, player_y):
        """AI: kejar player menggunakan move() dari parent"""
        if self._x < player_x:
            self.move(self.__speed, 0)
```

Penjelasan:
Class Monster mewarisi semua atribut dan method dasar dari Character, seperti:

- position (x, y)
- health dan status hidup
- move() dan take_damage()

Class ini menambahkan perilaku khusus:

- difficulty sebagai pembeda tingkat kesulitan
- reward sebagai poin yang diperoleh player
- move_towards_player() untuk AI sederhana mengejar player

Inheritance memungkinkan Monster fokus pada logika musuh tanpa menulis ulang code dasar karakter

- Level 3: Boss extends Monster (Multi-level Inheritance)

```python
class Boss(Monster):
    def __init__(self, name, difficulty, x, y):
        # Call Monster constructor dengan difficulty lebih tinggi
        super().__init__(name, difficulty + 2, x, y)

        # Atribut tambahan khusus Boss
        self.__special_ability = "Multi-Question"
        self.__special_cooldown = 0

    def attack(self):
        """
        Override attack dari Monster (POLYMORPHISM)
        Boss damage 2x lipat
        """
        base_damage = super().attack()  # Panggil parent attack
        return base_damage * 2

    def special_attack(self):
        """Special ability hanya Boss yang punya"""
        if self.__special_cooldown == 0:
            self.__special_cooldown = 5
            return True
        return False

    def get_question(self):
        """Override untuk soal lebih kompleks"""
        a = random.randint(5, 20)
        b = random.randint(2, 10)
        c = random.randint(1, 10)

        operations = [
            (f"({a} + {b}) × {c} = ?", (a + b) * c),
            (f"{a} × {b} - {c} = ?", a * b - c)
        ]

        return random.choice(operations)
```

Penjelasan:

Class Boss merupakan turunan dari Monster, sehingga otomatis mewarisi:

- Seluruh atribut Character
- Seluruh atribut Monster (difficulty, reward, AI)

Boss menambahkan fitur eksklusif:

- Special attack dengan damage lebih besar
- Cooldown agar serangan tidak spam
- Override get_question() untuk soal matematika yang lebih kompleks

Multi-level inheritance: Character → Monster → Boss

Keuntungan:

- Code reuse maksimal
- Boss tetap konsisten sebagai "monster" namun dengan kemampuan lanjutan

2.3 POLYMORPHISM (Polimorfisme)

Definisi

Polymorphism memungkinkan method atau objek untuk memiliki bentuk atau perilaku yang berbeda berdasarkan konteks penggunaannya. Dalam OOP, polymorphism dicapai melalui:

- Method Overriding: Child class mengubah implementasi method dari parent
- Abstract Methods: Parent class mendefinisikan interface, child implement detail

Tujuan Polymorphism

1. Flexibility: Satu interface, banyak implementasi
2. Extensibility: Mudah tambah tipe baru tanpa ubah existing code
3. Code Simplification: Kode lebih general dan reusable

Implementasi Polymorphism

a. Method Overriding: attack()

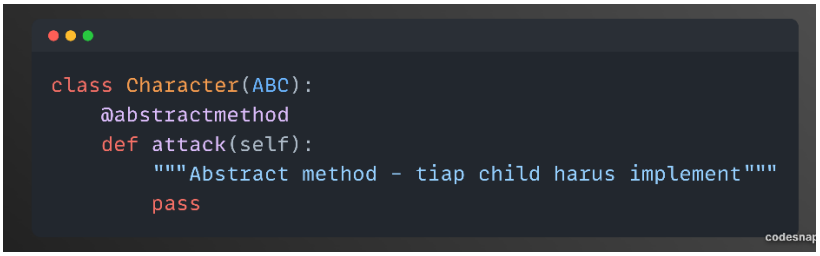
   Method attack() didefinisikan sebagai abstract method di class Character.

   Setiap turunan mengimplementasikannya secara berbeda:

   - Player: attack() dipicu saat menjawab soal dengan benar
   - Monster: attack() mengurangi health player
   - Boss: attack() memiliki damage lebih besar atau special effect

   Polymorphism memungkinkan GameManager memanggil attack() tanpa perlu tahu tipe objeknya.

   - Parent Class (abstrctack)



- Child Class-Player

```python
class Player(Character):
    def attack(self):
        """Player attack based on level"""
        return 15 * self.__level
```

- Child Class-Monster

```python
class Monster(Character):
    def attack(self):
        """Monster attack based on difficulty"""
        return 10 + (self.__difficulty * 5)
```

- Grandchild Class- Boss

```python
class Boss(Monster):
    def attack(self):
        """Boss attack 2x Monster"""
        base_damage = super().attack()  # Call parent
        return base_damage * 2
```

b. Method Overriding: get_question()

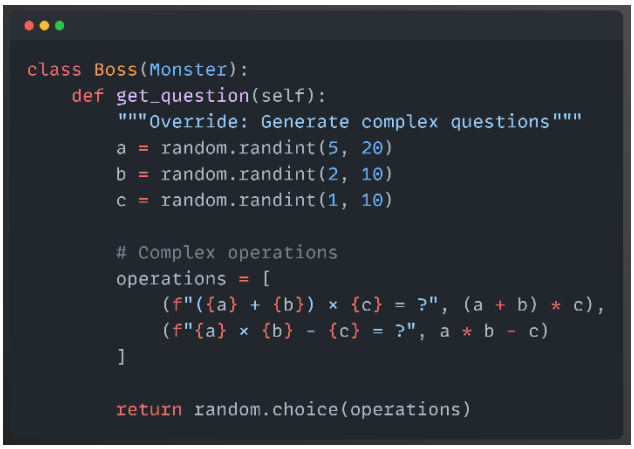
- Monster (Basic Questions):

```python
class Monster(Character):
    def get_question(self):
        """Generate basic math questions"""
        if self.__difficulty == 1:
            # Easy: Addition
            a = random.randint(1, 20)
            b = random.randint(1, 20)
            return f"{a} + {b} = ?", a + b
        elif self.__difficulty == 2:
            # Medium: Multiplication
            # ...
```

Penjelasan:

- Soal disesuaikan dengan difficulty
- Operasi tunggal (tambah, kali, bagi)

- Boss (Complex Questions):



```python
class Boss(Monster):
    def get_question(self):
        """Override: Generate complex questions"""
        a = random.randint(5, 20)
        b = random.randint(2, 10)
        c = random.randint(1, 10)

        # Complex operations
        operations = [
            (f"({a} + {b}) × {c} = ?", (a + b) * c),
            (f"{a} × {b} - {c} = ?", a * b - c)
        ]

        return random.choice(operations)
```

Penjelasan:

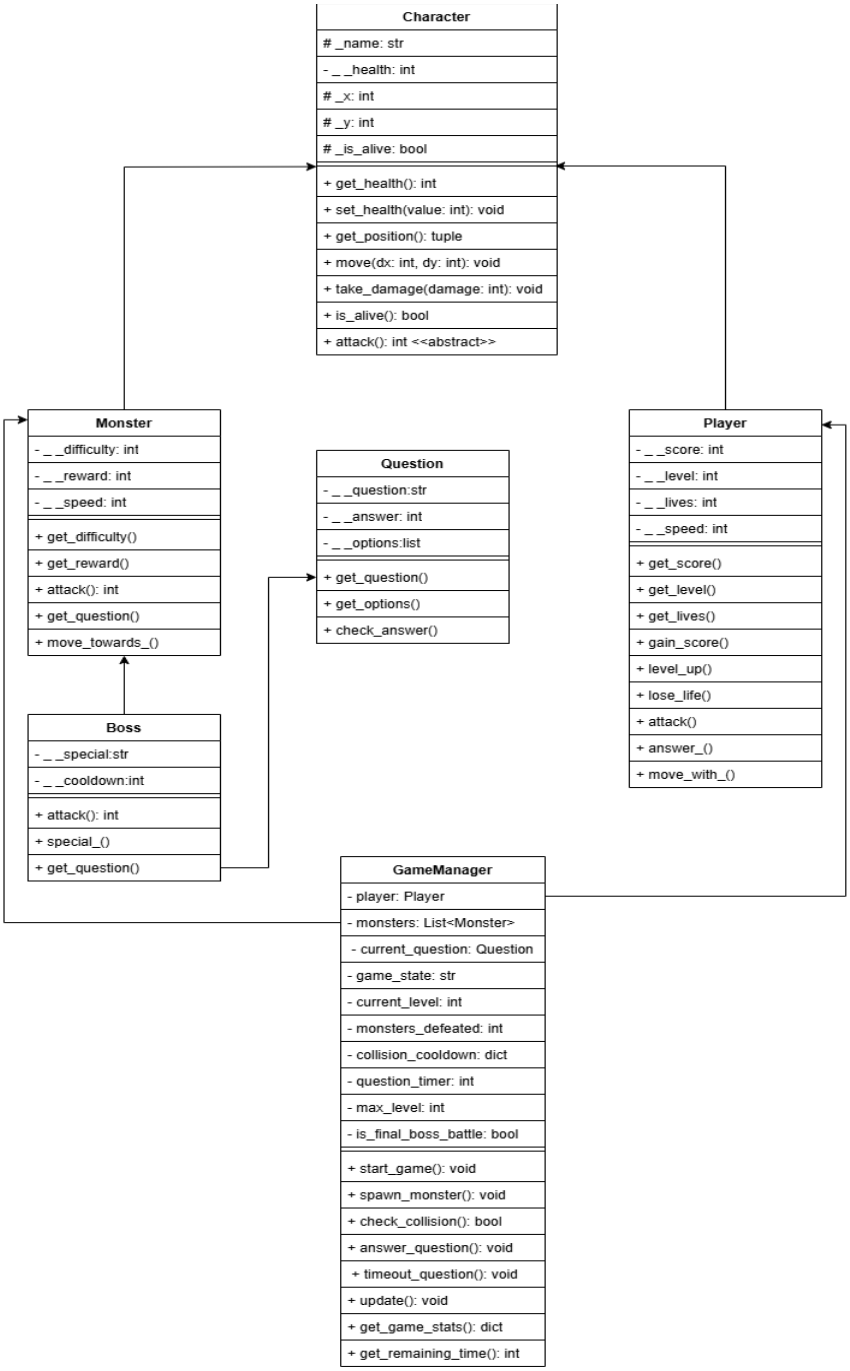
- Soal kombinasi beberapa operasi
- Tingkat kesulitan lebih tinggi

Meskipun method yang dipanggil sama (get_question()), perilaku berbeda tergantung objek.

BAB 3:

DIAGRAM CLASS DAN PERANCANGAN SISTEM

3.1 Class Diagram



3.2 Hubungan Antar Class

1. Pewarisan (Inheritance) Hierarki
   Bagian inti dari diagram ini adalah struktur hierarki karakter.
   - Character (Abstract Class)
     - Peran: Ini adalah "blueprint" atau kerangka dasar. Kelas ini bersifat Abstrak (<<abstract>>), artinya tidak bisa membuat objek langsung dari kelas ini. Ia hanya berfungsi sebagai orang tua bagi kelas lain.
     - Atribut: Menyimpan data umum yang dimiliki karakter yang ada di game, seperti name (nama), health (nyawa), position (x, y), dan status hidup (is_alive). Simbol # menandakan protected (bisa diakses oleh anak kelas), dan - atau __ menandakan private.
     - Method: Memiliki fungsi dasar seperti bergerak (move), menerima serangan (take_damage), dan satu fungsi abstrak attack() yang wajib diimplementasikan ulang (override) oleh anak-anak kelasnya dengan cara yang berbeda.
   - Player (Anak dari Character)
     - Peran: Mewakili karakter yang dikendalikan pengguna.
     - Fitur Tambahan: Karena mewarisi Character, ia punya nyawa dan posisi. Namun, Player menambahkan fitur khusus seperti score (skor), lives (jumlah kesempatan hidup), dan level.
     - Logika: Pemain bisa naik level (level_up), menjawab pertanyaan (answer_), dan bergerak menggunakan input (move_with_keys).
   - Monster (Anak dari Character)
     - Peran: Mewakili musuh biasa.
     - Fitur Tambahan: Memiliki tingkat kesulitan (difficulty) dan hadiah jika dikalahkan (reward).
     - Logika: Monster bergerak mendekati pemain secara otomatis (move_towards_player) dan memiliki data pertanyaan (get_question), yang mengindikasikan pemain mungkin harus menjawab pertanyaan saat bertemu monster.
   - Boss (Anak dari Monster)
     - Peran: Musuh yang lebih kuat, merupakan turunan dari Monster (sehingga juga turunan dari Character).

    - Fitur Tambahan: Memiliki serangan spesial (special_attack) dan waktu jeda (cooldown). Ini adalah contoh pewarisan bertingkat (*multilevel inheritance*).

2. Komponen Logika Game
    - Question
        - Peran: Kelas mandiri yang menangani logika kuis atau soal.
        - Isi: Menyimpan teks pertanyaan, jawaban yang benar, dan pilihan jawaban.
        - Fungsi: Memvalidasi apakah jawaban pemain benar atau salah (check_answer).
3. Pengendali Utama (Controller)
    - GameManager
        - Peran: "Otak" dari seluruh permainan. Kelas ini yang mengatur alur game dari awal sampai akhir.
        - Hubungan (Association): Diagram menunjukkan panah uses yang berarti GameManager mengelola:
            - 1 Objek Player.
            - Banyak Objek Monster (List<Monster>).
            - Objek Question saat ini.
        - Tugas Utama:
            - start_game(): Memulai permainan.
            - spawn_monster(): Memunculkan musuh baru.
            - check_collision(): Mengecek apakah pemain menabrak monster.
            - update(): Dijalankan terus menerus (game loop) untuk memperbarui posisi dan waktu.

BAB 4:
FITUR DAN CARA PENGGUNAAN APLIKASI

4.1 Fitur Utama

4.1.1 Sistem Pertarungan Berbasis Matematika

Mekanisme:

- Player bertemu monster → Soal matematika muncul
- 4 pilihan jawaban (multiple choice)
- Jawaban benar → Monster mati, player dapat score
- Jawaban salah → Health berkurang, monster tetap hidup

Tipe Soal Berdasarkan Difficulty:

- Difficulty 1 (Easy): Penjumlahan 1-20
- Difficulty 2 (Medium): Perkalian 1-12
- Difficulty 3 (Hard): Pembagian dengan hasil bulat
- Boss: Kombinasi operasi (e.g., "(5 + 3) × 2 = ?")

4.1.2 Timer System (30 Detik)

Implementasi:

- Setiap soal memiliki batas waktu 30 detik
- Countdown otomatis dimulai saat soal muncul
- Timer ditampilkan dalam bentuk:
    - Progress bar visual
    - Text countdown: "Time: 25s"

Visual Timer Bar:

- Hijau : Waktu aman
- Kuning : Waktu mulai menipis
- Merah : Waktu hampir habis

Auto-Timeout:

- Jika waktu habis sebelum player menjawab
- Dianggap sebagai jawaban salah
- Health berkurang sesuai damage monster
- Game langsung lanjut (tidak stuck)

4.1.3 Cooldown System

Tujuan: Mencegah spam collision dengan monster yang sama

Mekanisme:

- Setelah collision (jawab benar/salah), ada cooldown 2 detik
- Player punya waktu untuk kabur dari monster
- Monster tidak bisa trigger collision lagi selama cooldown aktif
- Setiap monster punya cooldown independent

4.1.4 Multiple Monster Types

A. Monster Biasa

Difficulty	Damage	Reward (Poin)
1 (Easy)	15	10
2 (Medium)	20	20
3 (Hard)	25	30

- Karakteristik:
  - Spawn setiap 3 detik
  - Maksimal 3 monster di layar
  - AI sederhana: mengejar player
  - Speed berdasarkan difficulty

B. Boss

- Spawn Condition: Setiap 5 monster defeated.
- Karakteristik:
  - Damage: 2x monster biasa
  - Reward: 2x monster biasa
  - Soal: Kombinasi operasi matematika
  - Visual: Lebih besar dari monster biasa

C. Final Boss

- Spawn Condition:
  - Saat player mencapai Level 10
  - Hanya spawn 1 kali per game
- Karakteristik:
  - Damage: 240 per hit
  - Soal: Kombinasi operasi kompleks
  - Win Condition: Defeat Final Boss = WIN!

4.1.5 Level Progression System

Level Up Mechanism:

- Player dimulai di Level 1
- Gain score untuk level up
- Requirement: 100 × current_level
- Level 1 → 2: 100 score
- Level 2 → 3: 200 score (total 300)
- Level 3 → 4: 300 score (total 600)

Level Up Bonus:

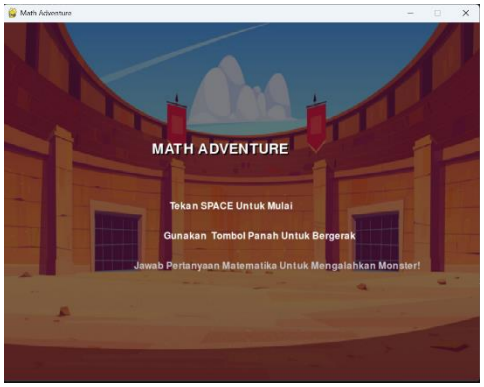
- Health +20
- Monster spawn lebih sulit **Wave System:**
- Setiap 10 monster defeated → Current level + 1
- Difficulty monster meningkat

Level Maksimal: 10

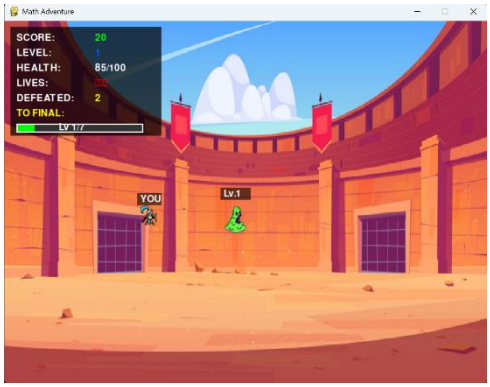
- Setelah Level 10, Final Boss spawn
- Defeat Final Boss = WIN

BAB 5:
SCREENSHOT PROGRAM

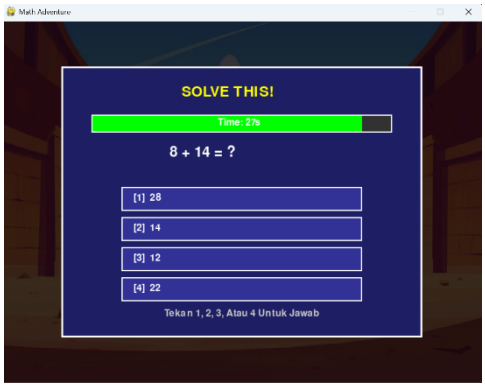
5.1 Menu Screen



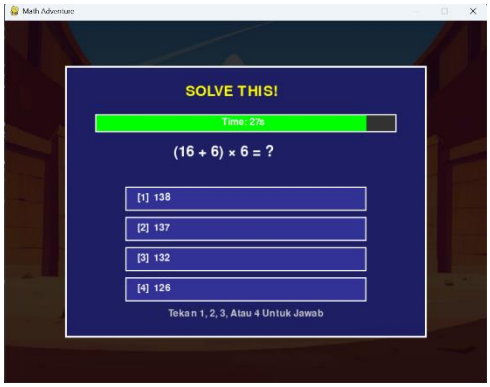
5.2 Playing State - Gameplay Normal



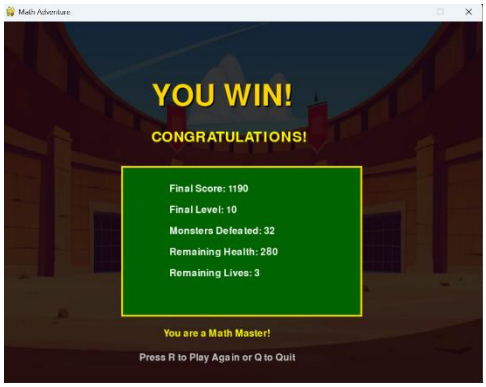
5.3 Question Screen



5.5 Boss Battle Screen



5.8 Win Screen



5.9 Game Over Screen