LAPORAN AKHIR
PROJECT ANT KILLER

Mata Kuliah:
Pemrograman Berorientasi Objek

Oleh:

M.Dzaky Budi Praditya

24091397033

2024B

PROGRAM STUDI D4 MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

2025

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pemrograman Berorientasi Objek (PBO) merupakan paradigma penting dalam pengembangan perangkat lunak modern karena mampu menghasilkan sistem yang terstruktur, modular, dan mudah dikembangkan. Konsep utama PBO seperti Encapsulation, Inheritance, dan Polymorphism tidak cukup dipahami secara teori, tetapi perlu diterapkan langsung melalui sebuah proyek nyata.

Proyek "OOP Ant Killer Game" dikembangkan menggunakan bahasa pemrograman Python dengan library Pygame sebagai media penerapan konsep PBO. Game 2D sederhana ini mensimulasikan permainan membasmi semut yang muncul dan bergerak di layar. Setiap semut dimodelkan sebagai objek dengan atribut dan perilaku tertentu, sehingga konsep Encapsulation dapat diterapkan dengan jelas.

Konsep Inheritance diwujudkan melalui kelas induk Ant yang diturunkan menjadi beberapa kelas seperti AntBiasa, AntPrajurit, dan AntRatu, masing-masing memiliki karakteristik berbeda namun tetap mewarisi sifat dasar yang sama. Sementara itu, Polymorphism diterapkan dengan memungkinkan metode yang sama memiliki perilaku berbeda pada setiap jenis semut.

Selain sebagai media pembelajaran PBO, proyek ini juga melatih kemampuan perancangan logika program, pengelolaan event, serta pengembangan game 2D sederhana menggunakan Pygame. Penggunaan struktur kelas yang jelas membuat alur program lebih mudah dipahami dan dikembangkan di masa depan.

Secara keseluruhan, proyek "OOP Ant Killer Game" dirancang sebagai sarana pembuktian pemahaman Pemrograman Berorientasi Objek melalui implementasi nyata. Dengan menggabungkan konsep PBO dan pengembangan game, proyek ini diharapkan dapat membantu meningkatkan pemahaman konseptual sekaligus keterampilan praktis dalam pemrograman Python.

1.2 Tujuan Proyek

1. Tujuan Umum (General Objectives)
Tujuan umum adalah sasaran utama yang ingin dicapai melalui proyek ini dalam konteks akademis dan pengembangan perangkat lunak:

- Menciptakan Bukti Implementasi Konsep PBO: Menghasilkan sebuah aplikasi fungsional yang secara nyata menerapkan dan mendemonstrasikan pemahaman mendalam terhadap prinsip-prinsip Pemrograman Berorientasi Objek.
- Membangun Arsitektur Perangkat Lunak yang Efisien: Mengembangkan game dengan struktur kode yang modular, mudah dipelihara, dan skalabel menggunakan desain berbasis kelas dan objek.
- Meningkatkan Keterampilan Teknis Pengembangan Game: Menguasai penggunaan library Pygame dalam hal grafis, audio, dan pengelolaan game loop.

2. Tujuan Spesifik (Specific Objectives)

- Mengimplementasikan Inheritance dan Polymorphism: mendefinisikan kelas induk Semut dengan tiga kelas turunan (SemutPrajurit, SemutRatu) yang menampilkan perilaku berbeda (gerakan, pertahanan, spawn) melalui method overriding.
- Menerapkan Encapsulation: Menggunakan atribut private (__health, __posisi_x) dalam kelas Semut dan mengaksesnya hanya melalui metode publik (diserang, get_posisi) untuk menjaga integritas data.
- Mengembangkan Sistem Game State dan Scoring: Mengelola tiga status game (Paused, Running, Game Over) dan mengimplementasikan sistem skor berbasis bobot entitas.
- Menciptakan Fitur Persistensi Data: Menyimpan dan memuat skor tertinggi (high score) secara lokal ke dalam file teks (highscore.txt).
- Mengintegrasikan Audio Interaktif: Berhasil menambahkan sound effect pada saat game dimulai (start_sound) dan saat entitas semut dibasmi (squish_sound).

1.3 Ruang Lingkup Proyek

Proyek ini mencakup:

1. Pengembangan sistem entitas semut dengan tiga kelas turunan (Semut, SemutPrajurit, SemutRatu).
2. Implementasi pergerakan otonom yang berbeda untuk setiap jenis semut.
3. Implementasi sistem interaksi pengguna (klik mouse) untuk membasmi semut.
4. Integrasi sistem sound effect sederhana (start game dan squish).
5. Pengelolaan status game (running, paused, game over) dan pencatatan high score lokal.

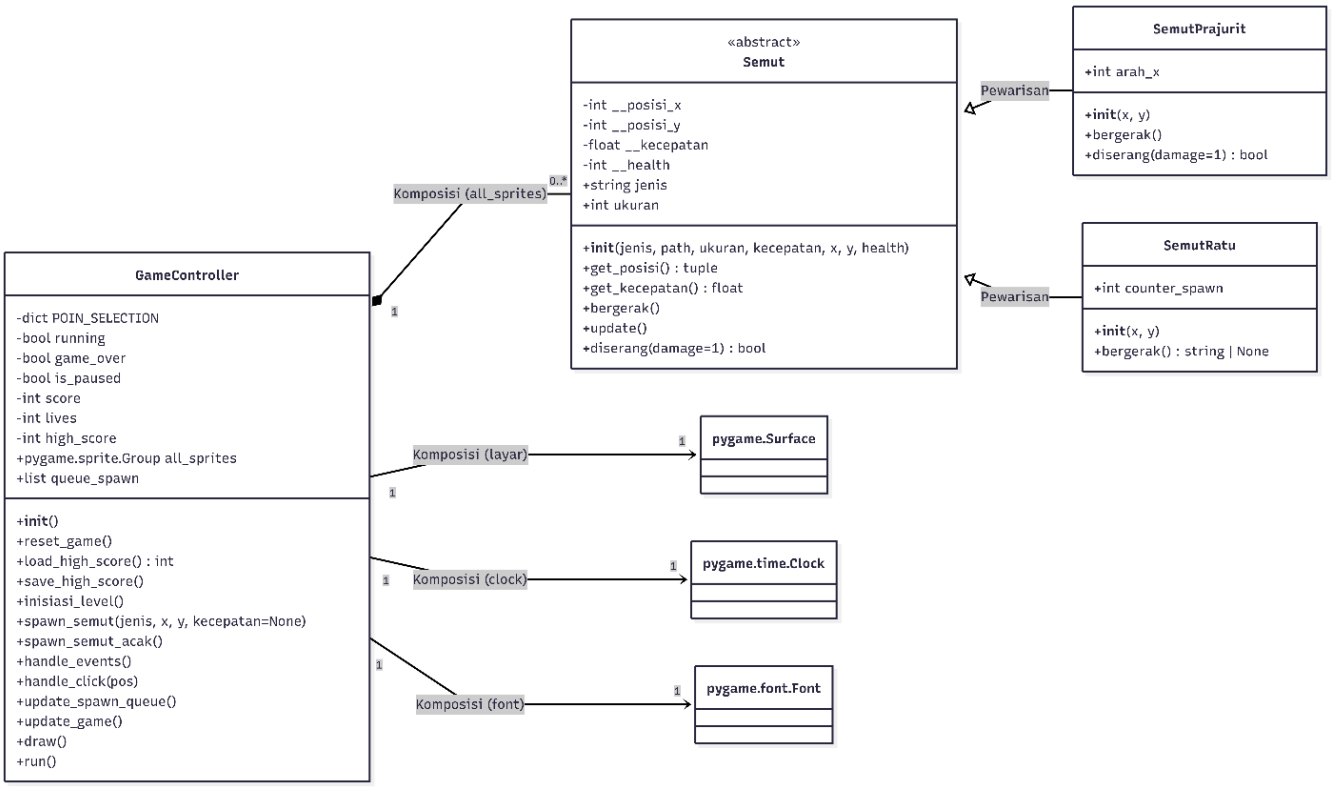
1.4 Metodologi Pengembangan

Metodologi yang digunakan adalah Prototyping Iteratif dengan fokus pada pendekatan Top-Down Design (desain atas-bawah).

1. Perancangan Konsep PBO: Mendefinisikan kelas induk (Semut) dan kelas turunan (SemutPrajurit, SemutRatu) serta merencanakan relasi pewarisan.
2. Inisiasi Pygame: Menyiapkan lingkungan Pygame dan kelas kontrol utama (GameController).
3. Pengembangan Inti (Iterasi 1): Mengimplementasikan pergerakan dan mekanisme klik dasar pada kelas Semut.
4. Pengembangan Lanjutan (Iterasi 2): Mengimplementasikan Inheritance dan Polymorphism pada SemutPrajurit dan SemutRatu, serta menambahkan game loop dan scoring.
5. Penyempurnaan (Iterasi 3): Menambahkan fitur high score, sound effect, dan game state (pause/game over).
6. Pengujian dan Dokumentasi: Melakukan pengujian fungsional dan menyusun laporan.

BAB II
PERANCANGAN APLIKASI

2.1 Diagram Class & Penjelasan



Penjelasan:
Diagram kelas ini memvisualisasikan arsitektur perangkat lunak dari game, dengan fokus utama pada penerapan prinsip-prinsip Pemrograman Berorientasi Objek (PBO): Pewarisan, Polimorfisme, Komposisi, dan Enkapsulasi.

1. Kelas Induk Abstrak: Semut (<<abstract>>)
   Kelas Semut berfungsi sebagai fondasi konseptual (blueprint) untuk semua entitas yang bergerak dalam permainan. Meskipun ditandai sebagai <<abstract>>, ia menyediakan implementasi dasar yang akan diwarisi.

- Enkapsulasi Data: Data internal penting semut dilindungi dengan mendeklarasikannya sebagai atribut privat (ditandai dengan __ seperti __posisi_x, __posisi_y, __kecepatan, dan __health). Perlindungan ini memastikan bahwa nilai-nilai tersebut hanya dapat diubah melalui metode publik yang terkontrol, seperti diserang(damage: 1) atau bergerak(),

menjaga integritas logika game.

- Perilaku Dasar: Kelas ini mendefinisikan fungsionalitas inti yang dibagi oleh semua semut: get_posisi() untuk membaca lokasi, bergerak() untuk pergerakan lurus ke bawah standar, update() untuk menjalankan pergerakan setiap frame, dan diserang() untuk mengurangi kesehatan dan menentukan apakah entitas mati.

2. Kelas Turunan (Pewarisan / Inheritance)
   Kelas Semut Prajurit dan Semut Ratu memperoleh semua atribut dan metode dari Semut melalui hubungan Pewarisan.

- SemutPrajurit: Kelas ini menambahkan atribut spesifik +arah_x: int untuk mengontrol pergerakan horizontal. Ia mendemonstrasikan Polimorfisme dengan meng-override metode bergerak() untuk menciptakan pola gerakan zigzag yang unik. Selain itu, metode diserang() juga di-override untuk mengimplementasikan logika armor yang membuat semut Prajurit lebih sulit dibunuh.
- SemutRatu: Kelas ini menambahkan +counter_spawn: int sebagai mekanisme internalnya. Ia menunjukkan Polimorfisme yang paling kompleks pada metode bergerak(), di mana alih-alih hanya menggerakkan diri, metode ini juga memicu sinyal spawn ("SPAWN") pada interval waktu tertentu, menghasilkan semut baru dan mengembalikan string atau None.

3. Kelas Kontrol Sentral: GameController
   Kelas GameController adalah unit utama yang mengendalikan seluruh jalannya permainan, mulai dari game loop hingga manajemen sumber daya.

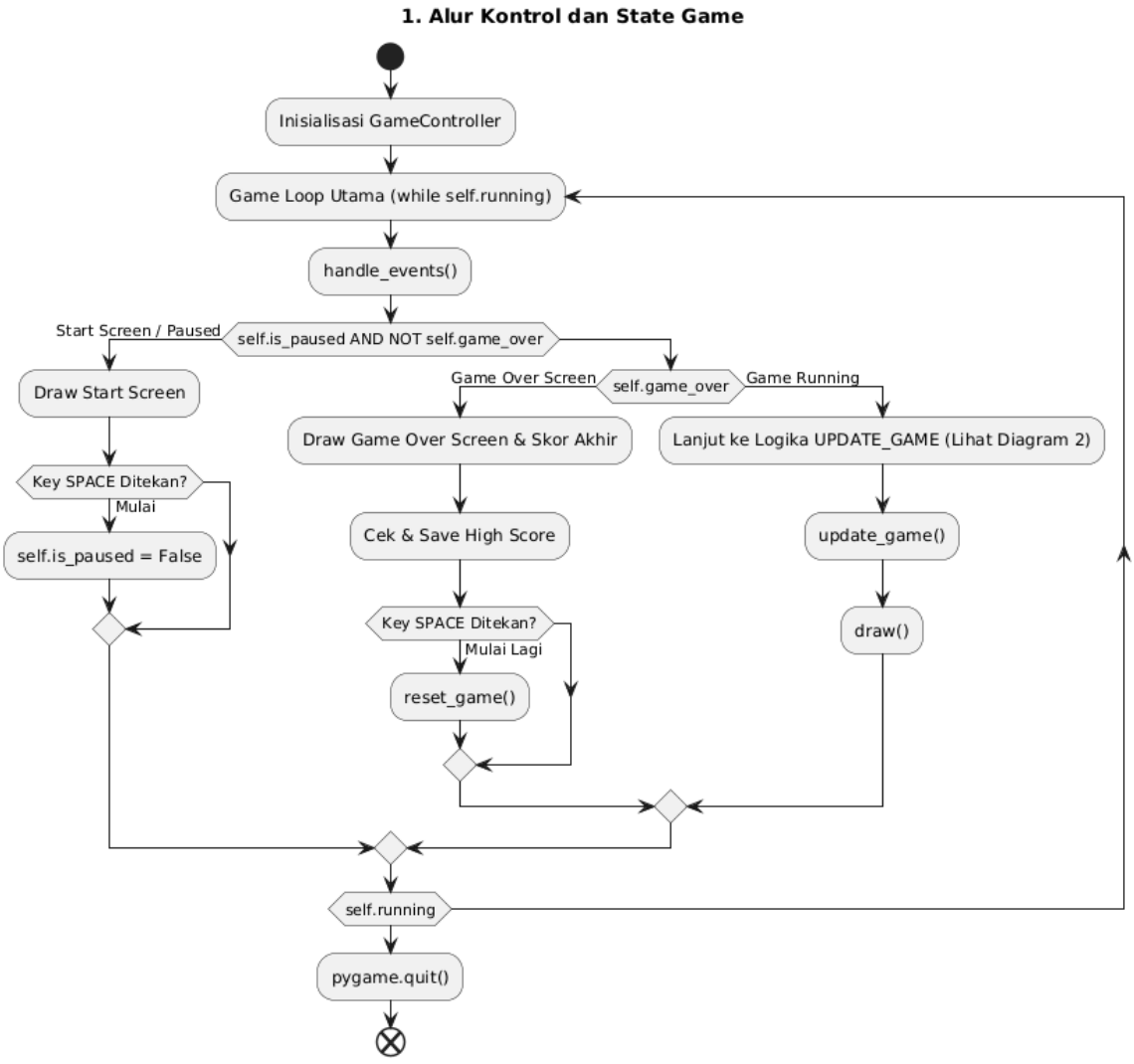
- Manajemen Status: Kelas ini menyimpan semua variabel status game seperti running, game_over, is_paused, score, lives, dan high_score, yang memungkinkan kontrol terhadap alur permainan.
- Aksi Utama: Metode utamanya termasuk run() (menjalankan game loop), handle_events() (mengelola input mouse/keyboard), update_game() (memperbarui logika game dan entitas), dan draw() (menggambar semua elemen ke layar).
- Manajemen Entitas: Memiliki metode khusus seperti spawn_semut(), update_spawn_queue(), dan handle_click() yang berfungsi sebagai jembatan antara logika kontrol dan logika entitas semut.

4. Relasi Komposisi (Composition)
   GameController memiliki hubungan Komposisi (ditunjukkan oleh garis dengan belah ketupat hitam) dengan beberapa objek Pygame esensial, yang berarti Controller bertanggung jawab penuh atas penciptaan dan pengelolaan objek-objek tersebut selama masa hidupnya:

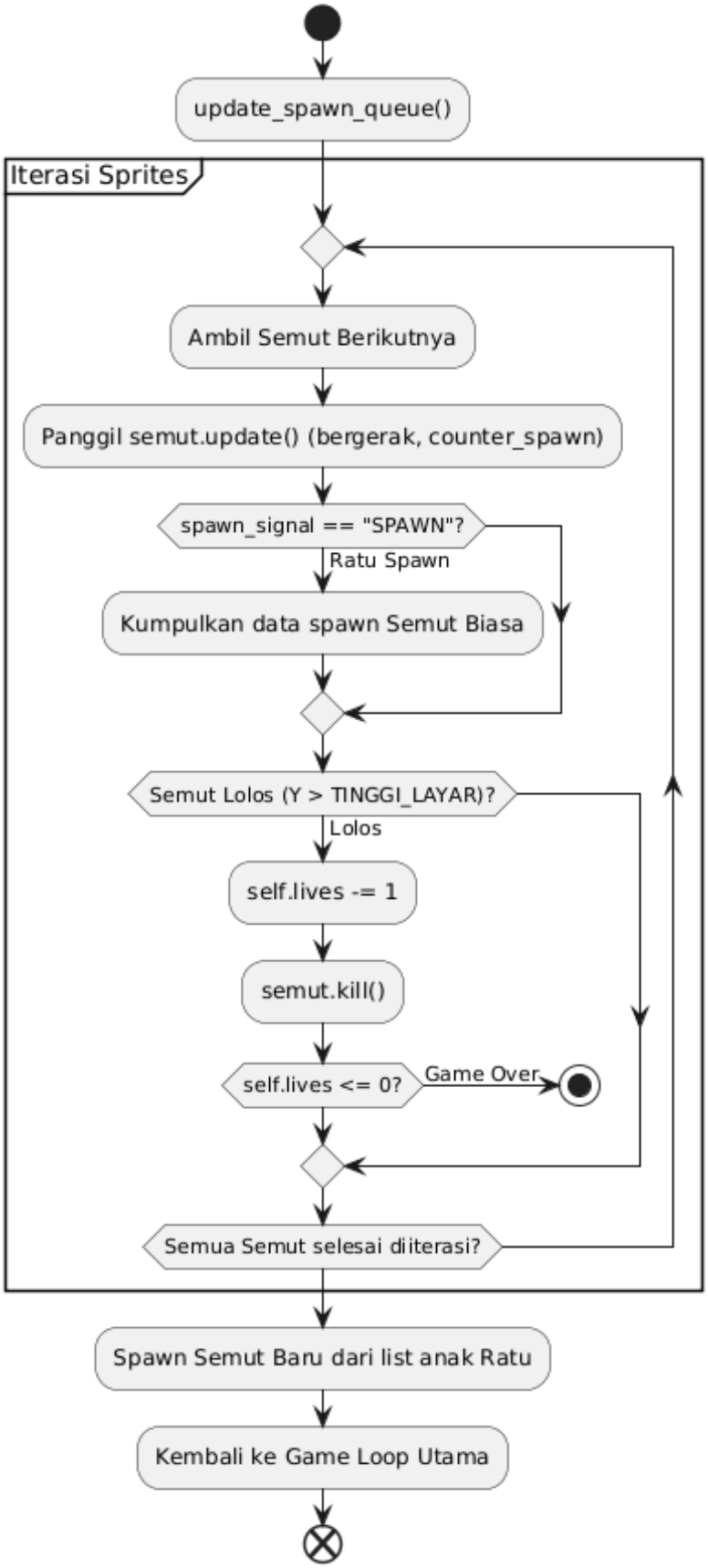
- Komposisi dengan Entitas (all_sprites): GameController menampung dan mengelola sekelompok objek Semut (melalui pygame.sprite.Group all_sprites), yang merupakan inti dari game loop.

- Komposisi dengan Sumber Daya Pygame: Ia mengontrol objek pygame.Surface (layar) untuk rendering grafis, pygame.time.Clock (clock) untuk sinkronisasi waktu dan FPS, dan pygame.font.Font (font) untuk menampilkan UI dan teks.

2.2 Alur Aplikasi / Flowchart



2. Detail Game Loop (UPDATE_GAME Logic)



1. Alur Kontrol dan State Game (Diagram 1)
Alur ini menunjukkan bagaimana GameController mengelola status permainan dari awal hingga akhir.
Setelah inisialisasi Pygame dan GameController selesai, program memasuki Game Loop Utama (repeat...while (self.running)). Di setiap frame, Game Loop melakukan hal berikut:

1. Event Handling: Memproses semua input dari pengguna (klik, tombol, dll.) melalui handle_events().
2. Pengecekan State: Sistem kemudian memeriksa apakah permainan sedang dalam kondisi statis (tidak berjalan):
    - Start Screen / Paused: Jika self.is_paused adalah True, program menggambar layar awal. Program akan tetap di sini sampai pemain menekan SPACE, yang kemudian mengubah self.is_paused menjadi False.
    - Game Over Screen: Jika self.game_over adalah True, layar Game Over ditampilkan. Pada saat ini, High Score akan dicek dan disimpan ke file. Pemain harus menekan SPACE untuk memanggil reset_game() dan memulai ulang siklus.
3. Game Running: Jika tidak dalam kondisi statis, flow berlanjut ke Logika UPDATE_GAME dan kemudian ke draw(), menampilkan visual permainan.

Jika self.running menjadi False (biasanya karena pemain menutup jendela), program keluar dari loop dan memanggil pygame.quit() untuk mengakhiri aplikasi.

2. Detail Game Loop (UPDATE_GAME Logic) (Diagram 2)
Ini adalah inti logika game yang berjalan ketika pemain berinteraksi.
Flow dimulai dengan update_spawn_queue(), di mana sistem memeriksa timer untuk melepaskan semut yang telah diantri sebelumnya ke dalam layar.
Selanjutnya, program memasuki Partition "Iterasi Sprites" untuk memperbarui status setiap semut di layar:

1. Pembaruan Polimorfik: Untuk setiap semut, metode semut.update() dipanggil.
    - Fungsi ini bersifat polimorfik: Semut biasa hanya bergerak, sementara SemutRatu (Ratu Semut) mengembalikan sinyal "SPAWN" secara periodik. Jika sinyal ini diterima, data Semut Biasa baru akan dikumpulkan untuk di-spawn nanti.
2. Pengecekan Lolos: Setelah bergerak, posisi semut diperiksa terhadap batas bawah layar.
    - Jika semut Lolos, self.lives dikurangi 1 dan semut di-kill.
3. Game Over Check (Lives): Segera setelah nyawa dikurangi, sistem memeriksa apakah self.lives <= 0.
    - Jika Ya, flow berhenti secara paksa (stop), menghentikan iterasi dan mengirim sinyal Game Over ke Game Loop utama.
    - Jika Tidak, iterasi berlanjut ke semut berikutnya.

Setelah iterasi selesai (repeat while (Semua Semut selesai diiterasi?)), program

menjalankan langkah terakhir:

- Spawn Anak Ratu: Semua data semut baru yang dikumpulkan dari sinyal "SPAWN" Ratu ditambahkan ke dalam all_sprites.

2.3 Deskripsi Kelas & Tanggung Jawab

Perancangan aplikasi ini menggunakan empat kelas utama yang saling berinteraksi, menciptakan sebuah hierarki melalui pewarisan.

- Kelas Induk: Semut Kelas ini adalah fondasi untuk semua musuh dalam permainan. Tanggung jawab utamanya adalah manajemen posisi, kecepatan, dan health. Kelas ini mendefinisikan perilaku dasar seperti bergerak() (gerakan vertikal ke bawah) dan diserang() (logika pengurangan health dan penghancuran objek). Atribut krusial seperti posisi (__posisi_x, __posisi_y), kecepatan, dan health dienkapsulasi untuk memastikan modifikasi hanya terjadi melalui metode terkontrol.
- Kelas Turunan: SemutPrajurit Kelas ini mewarisi semua sifat dari Semut. Tugas spesifiknya adalah mengimplementasikan perilaku pergerakan zigzag dengan menambahkan komponen horizontal pada metode bergerak(). Selain itu, melalui polimorfisme, kelas ini memodifikasi metode diserang() untuk memberikan ilusi "Armor" dengan health awal yang lebih tinggi (2).
- Kelas Turunan: SemutRatu Kelas ini mewakili musuh jenis boss. Peran utamanya adalah memicu event spawn. Melalui override pada metode update(), Ratu tidak hanya bergerak sangat lambat, tetapi juga secara berkala mengirimkan sinyal "SPAWN" yang kemudian diproses oleh GameController untuk menghasilkan Semut Biasa di sekitarnya.
- Kelas Kontrol Utama: GameController Kelas ini adalah orkestrator seluruh permainan. Tanggung jawab utamanya meliputi pengelolaan game loop, event handling (seperti klik mouse untuk menyerang), inisialisasi dan manajemen sprite group (all_sprites), kontrol game state (is_paused, game_over), sistem skor, dan persistensi High Score.

2.4 Teknologi, Framework, & Library yang Digunakan

Proyek OOP Ant Killer Game dikembangkan secara eksklusif menggunakan Python 3.x. Python dipilih karena sintaksisnya yang bersih dan dukungan bawaan yang sangat kuat untuk paradigma Pemrograman Berorientasi Objek (PBO), yang merupakan tujuan utama dari proyek ini.

1. Framework Utama: Pygame
Pygame adalah library Python yang berfungsi sebagai framework utama untuk pengembangan game 2D. Perannya sangat krusial dalam mengimplementasikan semua komponen yang berinteraksi dengan tampilan dan waktu:

- Manajemen Game Loop dan Waktu: Pygame menyediakan objek pygame.time.Clock yang memastikan game loop utama berjalan pada frame rate yang stabil (60 FPS), esensial untuk pergerakan semut yang mulus.
- Grafis dan Sprite: Pygame memungkinkan pemuatan, penskalaan, dan manipulasi objek visual (gambar ant_biasa.png, wooden_background.png). Kelas Semut mewarisi dari pygame.sprite.Sprite untuk memanfaatkan sistem rendering dan collision detection yang efisien.
- Event Handling: Pygame menyediakan sistem antrian event yang memungkinkan GameController.handle_events() untuk menangkap semua input pengguna, seperti:
    - pygame.QUIT (menutup jendela).
    - pygame.KEYDOWN (menekan tombol SPACE untuk start/restart).
    - pygame.MOUSEBUTTONDOWN (klik mouse untuk mekanisme squish).
- Audio (Pygame Mixer): Modul ini digunakan untuk memuat dan memutar efek suara (start sound, squish sound), menambahkan umpan balik audio yang penting bagi pengalaman bermain game.

2. Standard Python Libraries
Beberapa modul bawaan (Standard Library) Python juga memainkan peran penting dalam fungsionalitas dan utilitas proyek:

- os (Operating System Interface):
    - Peran Utama: Interaksi dengan sistem berkas (file system).
    - Implementasi: Digunakan untuk memeriksa keberadaan file aset (gambar dan suara) sebelum dimuat, dan yang paling penting, untuk membaca dan menulis data High Score pada file highscore.txt. Ini memastikan persistensi skor tertinggi antar sesi permainan.
- random (Pseudorandom Number Generation):
    - Peran Utama: Menambahkan variabilitas dan ketidakpastian dalam gameplay.
    - Implementasi: Digunakan untuk:

        - Menentukan posisi X acak di mana semut akan muncul dari atas layar.
        - Menghitung kecepatan awal yang berbeda-beda untuk Semut biasa (random.uniform(0.5, 1.5)).
        - Menentukan peluang spawn acak setelah antrian awal habis.
- math (Mathematical Functions):
    - Peran Utama: Menyediakan fungsi matematika tingkat lanjut.
    - Implementasi: Meskipun implementasi zigzag pada SemutPrajurit menggunakan operator modulo (%) sederhana, library ini tersedia untuk fungsionalitas lanjutan yang mungkin diperlukan (misalnya, jika diterapkan pergerakan berbasis trigonometri seperti sinus atau kosinus).

3. Struktur Data Bawaan Python

Selain libraries di atas, Python juga dimanfaatkan untuk struktur data yang mendukung konsep OOP dan logika permainan:

- List (sebagai Queue): Digunakan dalam GameController.queue_spawn untuk menyimpan urutan semut yang akan dimunculkan di awal permainan, mengatur kesulitan tahap awal.
- Dictionary: Digunakan dalam GameController.POIN_SELECTION untuk memetakan jenis semut ("Biasa", "Prajurit", "Ratu") ke nilai skor yang sesuai, sehingga memudahkan penambahan atau perubahan bobot skor di masa depan.

2.5 Prinsip Desain yang Digunakan (OOP)

Konsep OOP diterapkan secara ketat untuk memastikan arsitektur yang kuat.

1. Encapsulation (Enkapsulasi): Diterapkan pada kelas Semut dengan melindungi atribut internal yang krusial (__health, __posisi_y) menggunakan konvensi double underscore. Data ini hanya dapat diakses melalui method publik (get_posisi()) atau dimodifikasi melalui method terkontrol (diserang()), yang penting untuk menjaga integritas logika kematian semut.
2. Inheritance (Pewarisan): Digunakan untuk menciptakan hierarki musuh yang efisien. SemutPrajurit dan SemutRatu mewarisi fungsionalitas visual dan inisialisasi dari kelas induk Semut menggunakan super().__init__(), menghindari penulisan ulang kode untuk pemuatan sprite dan rect.
3. Polymorphism (Polimorfisme): Prinsip ini memungkinkan GameController memperlakukan semua objek sebagai Semut (menggunakan antarmuka yang sama), sementara objek itu sendiri merespons secara berbeda:
    - Method Overriding terjadi pada bergerak() untuk SemutPrajurit (zigzag).
    - Method Overriding terjadi pada update() untuk SemutRatu (logika spawning).

4. Abstraction (Abstraksi): Detail implementasi pergerakan zigzag (SemutPrajurit) atau mekanisme spawning (SemutRatu) di-abstract dari GameController. Kontroler hanya perlu memanggil semut.update() tanpa perlu tahu detail internal setiap jenis semut.

2.6 Perancangan Basis Data (ERD & Struktur Tabel)

1. Justifikasi Arsitektur Persistensi Data

Proyek OOP Ant Killer Game adalah aplikasi mandiri dengan fokus utama pada logika game loop dan demonstrasi konsep Pemrograman Berorientasi Objek. Berdasarkan sifatnya, aplikasi ini tidak memerlukan arsitektur basis data yang kompleks, baik itu Relasional (seperti MySQL atau PostgreSQL) maupun Non-Relasional (seperti MongoDB atau Redis).
Alasan utama di balik keputusan ini adalah bahwa kebutuhan persistensi data sangat minimalis: hanya ada satu nilai tunggal yang harus disimpan dan diambil, yaitu High Score pemain.

2. Mekanisme Penyimpanan dan Implementasi

Untuk mencapai persistensi data tunggal ini, digunakanlah mekanisme File I/O Dasar Python.
File Target

- Nama File: highscore.txt
- Lokasi: Disimpan di direktori yang sama dengan executable game, membuatnya mudah diakses.

Implementasi Fungsi

- Fungsi load_high_score(): Bertanggung jawab untuk membaca konten dari highscore.txt. Fungsi ini mencakup error handling menggunakan blok try-except untuk mengatasi dua skenario kegagalan:
    1. File belum ada (IOError atau FileNotFoundError): Akan mengembalikan nilai default 0.
    2. Konten file rusak atau tidak valid (ValueError): Akan mengeluarkan peringatan dan mengembalikan nilai default 0.
- Fungsi save_high_score(): Dipanggil hanya ketika score pemain saat ini melebihi high_score yang tersimpan. Fungsi ini akan menimpa konten file dengan nilai integer skor tertinggi yang baru, memastikan bahwa hanya skor terbaik yang dipertahankan.

Struktur Data yang Disimpan

- Format: File tersebut hanya menampung satu baris teks.
- Isi Data: Nilai integer yang mewakili skor maksimum. Misalnya, jika high score adalah 150, file highscore.txt hanya akan berisi: 150.

3. Analisis Kebutuhan Pemodelan Data

Karena sifat data yang sangat sederhana, pemodelan data formal menjadi tidak relevan.

- Entity-Relationship Diagram (ERD): Pembuatan ERD tidak diperlukan karena tidak ada entitas data yang berbeda (seperti Pemain, Level, atau

Musuh), dan oleh karena itu, tidak ada relasi yang perlu dimodelkan (misalnya, One-to-Many atau Many-to-Many).

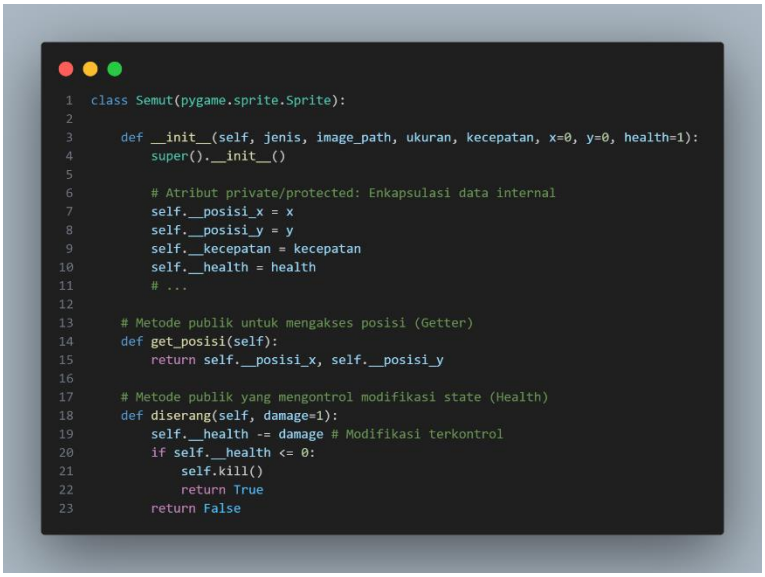
- Struktur Tabel Formal: Penggunaan tabel dengan skema, kolom, dan tipe data (seperti pada basis data SQL) akan menimbulkan overhead yang tidak perlu. Solusi File I/O langsung lebih ringan dan cepat untuk tujuan single-value persistence ini.

Kesimpulannya, mekanisme File I/O memberikan solusi persistensi yang paling efektif, cepat, dan ringkas, sangat sesuai dengan ruang lingkup proyek game edukasi yang fokus pada implementasi prinsip-prinsip PBO daripada arsitektur backend data.

BAB III IMPLEMENTASI KONSEP OOP

3.1 Implementasi Encapsulation

Enkapsulasi dicapai dengan membatasi akses langsung ke atribut internal objek. Dalam Python, konvensi penamaan dengan double underscore (__) digunakan untuk menandai atribut yang dimaksudkan sebagai private atau protected. Akses dan modifikasi atribut ini harus melalui metode publik (getter/setter) yang didefinisikan dalam kelas.
Kode Relevan:



```python
class Semut(pygame.sprite.Sprite):

    def __init__(self, jenis, image_path, ukuran, kecepatan, x=0, y=0, health=1):
        super().__init__()

        # Atribut private/protected: Enkapsulasi data internal
        self.__posisi_x = x
        self.__posisi_y = y
        self.__kecepatan = kecepatan
        self.__health = health
        # ...

    # Metode publik untuk mengakses posisi (Getter)
    def get_posisi(self):
        return self.__posisi_x, self.__posisi_y

    # Metode publik yang mengontrol modifikasi state (Health)
    def diserang(self, damage=1):
        self.__health -= damage # Modifikasi terkontrol
        if self.__health <= 0:
            self.kill()
            return True
        return False
```

Penjelasan:

- Atribut seperti self.__health dan self.__posisi_x tidak diakses secara langsung dari luar kelas.
- Perubahan health hanya dilakukan melalui metode diserang(damage), yang juga mencakup logika penting (pengecekan kematian semut dan pemanggilan self.kill()). Hal ini menjaga konsistensi state objek.

3.2 Implementasi Inheritance

Pewarisan digunakan untuk membangun hierarki kelas di mana kelas-kelas turunan (SemutPrajurit, SemutRatu) mewarisi semua sifat dan perilaku dasar dari kelas induk (Semut).
Kode Relevan (Kelas SemutPrajurit dan SemutRatu):
Kedua kelas turunan memanggil konstruktor induk (super().__init__()) untuk inisialisasi dasar Pygame dan atribut yang dienkapsulasi, namun mengaplikasikan nilai unik untuk atributnya (ukuran, kecepatan, dan health).

```python
# C. KELAS TURUNAN: SemutPrajurit
class SemutPrajurit(Semut):
    def __init__(self, x, y):
        # Memanggil konstruktor kelas induk dengan atribut spesifik:
        # health=2 (Armor), ukuran=53, kecepatan=2
        super().__init__(jenis="Prajurit", image_path=ANT_PRAJURIT_IMAGE,
                         ukuran=53, kecepatan=2, x=x, y=y, health=2)
        self.arah_x = 1

# D. KELAS TURUNAN: SemutRatu
class SemutRatu(Semut):
    def __init__(self, x, y):
        # Memanggil konstruktor kelas induk dengan atribut spesifik:
        # health=5 (Boss), ukuran=90, kecepatan=0.5
        super().__init__(jenis="Ratu", image_path=ANT_RATU_IMAGE,
                         ukuran=90, kecepatan=0.5, x=x, y=y, health=5)
        self.counter_spawn = 0
```

Penjelasan:

- Kedua kelas secara otomatis mendapatkan properti seperti self.image, self.rect, dan metode dasar seperti update() dan diserang() (sebelum di-override).
- Reusability kode tercapai karena logika inisialisasi gambar dan sprite hanya ditulis sekali di kelas Semut.

3.3 Implementasi Polymorphism

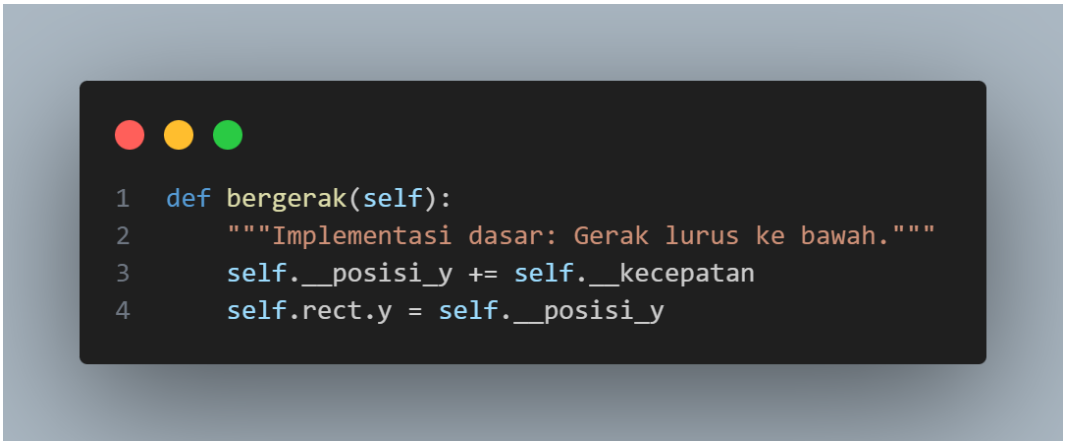
Polimorfisme memungkinkan metode dengan nama yang sama memiliki implementasi yang berbeda di setiap kelas turunan, memberikan perilaku yang spesifik.

1) Polimorfisme pada Metode bergerak()

Metode ini menentukan algoritma pergerakan untuk setiap jenis semut.

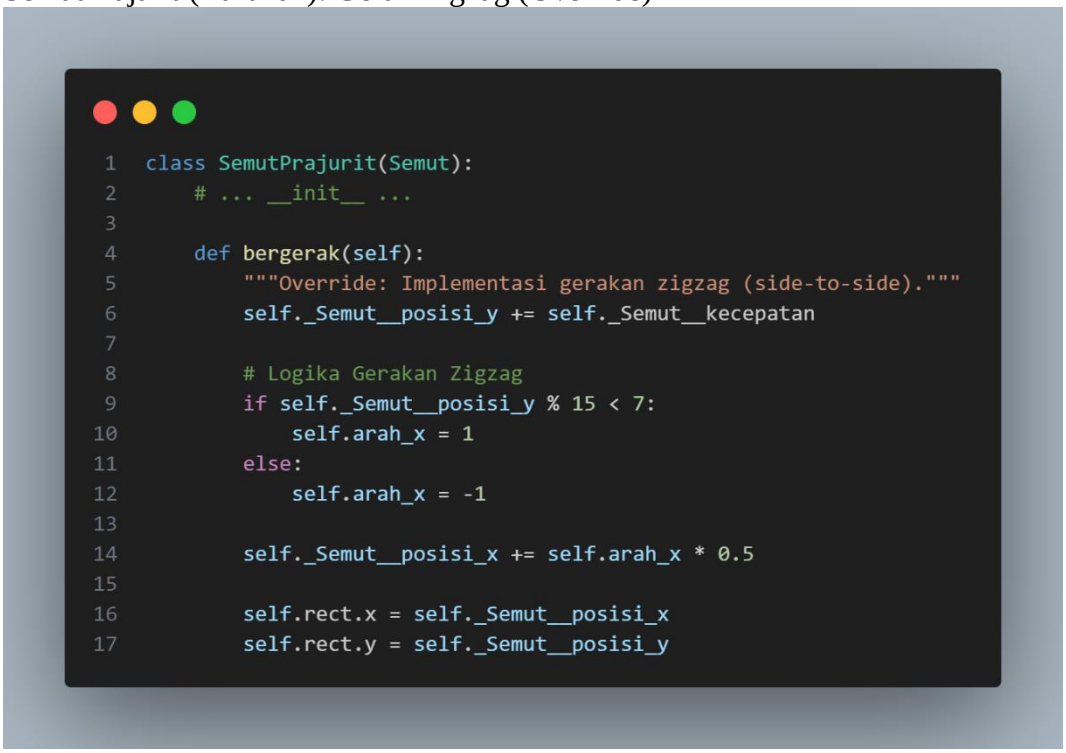
Penjelasan :
Semut Biasa (Induk) hanya bergerak lurus ke bawah. SemutPrajurit menimpa metode ini untuk menambahkan logika yang menyebabkan pergerakan zigzag (sisi ke sisi) di sepanjang sumbu X, membuatnya lebih sulit untuk diklik.

Semut (Induk): Gerak Lurus Ke Bawah



```python
def bergerak(self):
    """Implementasi dasar: Gerak lurus ke bawah."""
    self.__posisi_y += self.__kecepatan
    self.rect.y = self.__posisi_y
```

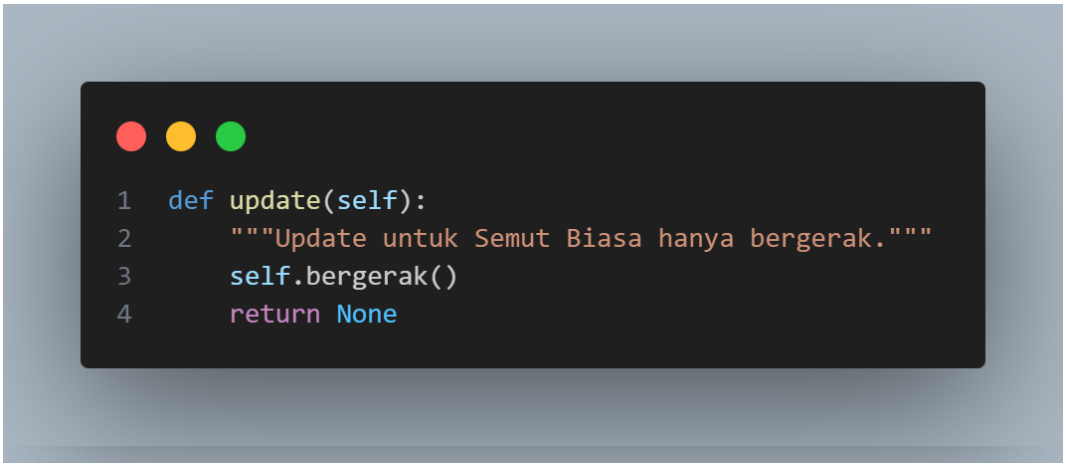
SemutPrajurit (Turunan): Gerak Zigzag (Override)



```python
class SemutPrajurit(Semut):
    # ... __init__ ...

    def bergerak(self):
        """Override: Implementasi gerakan zigzag (side-to-side)."""
        self._Semut__posisi_y += self._Semut__kecepatan

        # Logika Gerakan Zigzag
        if self._Semut__posisi_y % 15 < 7:
            self.arah_x = 1
        else:
            self.arah_x = -1

        self._Semut__posisi_x += self.arah_x * 0.5

        self.rect.x = self._Semut__posisi_x
        self.rect.y = self._Semut__posisi_y
```

2) Polimorfisme pada Metode diserang()

Metode ini menentukan bagaimana semut bereaksi saat diklik.
Penjelasan :
SemutPrajurit menimpa metode diserang() untuk mengimplementasikan logika armor. Prajurit memiliki *health* awal 2. Dengan menimpa metode ini, serangan pertama hanya mengeluarkan pesan armor dan mengurangi *health* menjadi 1 melalui super().diserang(), sedangkan serangan kedua baru membunuhnya. Semut lain hanya menggunakan logika kematian dasar.



```python
def update(self):
    """Update untuk Semut Biasa hanya bergerak."""
    self.bergerak()
    return None
```

Semut (Induk): Logika Kesehatan Dasar
SemutPrajurit (Turunan): Logika Armor (Override)

3) Polimorfisme pada Metode update()

Metode ini adalah game loop tick untuk setiap objek.
Penjelasan
Semut Ratu menimpa metode update() untuk memberikan fungsionalitas unik, yaitu memicu sinyal spawn ("SPAWN") setiap 120 frame sekali. Semut biasa dan Prajurit hanya melakukan pergerakan.

Semut (Induk): Update Dasar

```python
class SemutPrajurit(Semut):
    # ... __init__ ...

    def diserang(self, damage=1):
        """Override: Menambahkan logika armor."""
        if self._Semut__health > 1:
            print("[Prajurit] Armor menahan serangan!")
        # Memanggil implementasi metode induk untuk menyelesaikan pengurangan health
        return super().diserang(damage)
```

SemutRatu (Turunan): Update + Sinyal Spawn (Override)

```python
class SemutRatu(Semut):
    # ... __init__ ...

    def update(self):
        """Override update: Selain bergerak, Ratu juga memicu sinyal spawn."""
        self.bergerak()

        self.counter_spawn += 1
        if self.counter_spawn % 120 == 0:
            return "SPAWN" # Mengirim sinyal unik
        return None
```

4) Penggunaan Polymorphic di GameController

Polimorfisme memungkinkan *GameController* memperlakukan semua objek secara seragam (dengan memanggil update() atau diserang()), meskipun perilaku aktual objek tersebut berbeda-beda.

Bukti Kode Panggilan Polymorphic

```
# Dalam GameController.update_game()
for semut in list(self.all_sprites):

    # Panggilan polymorphic: Behavior ditentukan oleh kelas objek (Ratu, Prajurit, Biasa)
    spawn_signal = semut.update()

    # ... (Logika menangani sinyal spawn_signal) ...


# Dalam GameController.handle_click()
for semut in list(self.all_sprites):
    if semut.rect.collidepoint(x, y):

        # Panggilan polymorphic: Behavior diserang() ditentukan oleh kelas objek
        is_dead = semut.diserang(damage=1)

        # ... (Logika skor) ...
        return
```

BAB IV
HASIL & PEMBAHASAN

4.1 Penjelasan Fitur & Kaitannya Dengan OOP

Fitur	Konsep OOP Terkait	Penjelasan
Variasi Semut	**Inheritance & Polymorphism**	Setiap jenis semut (Biasa, Prajurit, Ratu) adalah objek dari kelas turunan yang berbeda, masing-masing dengan statistik dan perilaku unik, diwariskan dari kelas induk Semut.
Sistem Armor	**Polymorphism (Override)**	Metode diserang() pada SemutPrajurit di-override untuk memberikan pertahanan tambahan (hanya mati setelah 2 kali klik), berbeda dengan SemutBiasa yang mati sekali klik.
Spawn Anak Ratu	**Inheritance & Polymorphism**	SemutRatu memiliki logika spawn yang tertanam dalam metode bergerak() yang di-override, menunjukkan perilaku unik yang tidak dimiliki kelas turunan lain.
Manajemen Status	**Encapsulation & Composition**	Kelas GameController menyembunyikan detail game loop dan mengelola state (misalnya self.is_paused, self.game_over) serta komposisi objek Semut.

4.2 Fitur Utama Aplikasi

1. **Tiga Jenis Entitas Semut**: Biasa (cepat, health 1), Prajurit (lambat, health 2, gerak zigzag), dan Ratu (sangat lambat, health 5, spawner).
2. **Sistem Scoring**: Poin berbeda untuk setiap jenis semut yang dibunuh (Biasa: 1.0, Prajurit: 3.0, Ratu: 10.0).
3. **Game State Management**: Mampu beralih antara Start Screen (Paused), Game Running, dan Game Over.

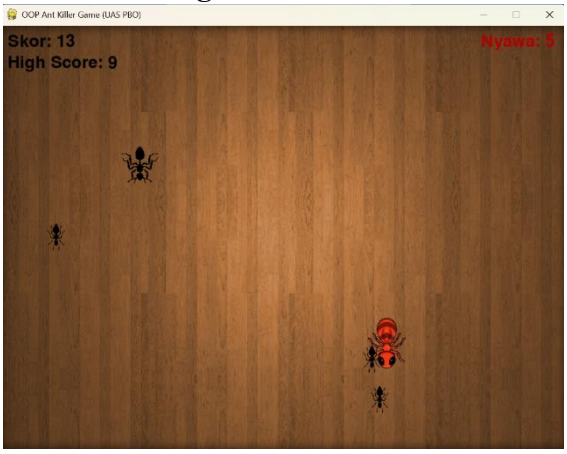
4. **High Score Persistence**: Skor tertinggi disimpan dan dimuat dari file highscore.txt.
5. **Sound Effect**: Umpan balik audio saat game dimulai (start_sound) dan saat semut dibasmi (squish_sound).

4.3 Tampilan Aplikasi dan Penjelasan

1. **Start Screen**

- Menampilkan judul dan perintah untuk memulai ("TEKAN SPACE UNTUK MULAI"). Status self.is_paused adalah True.

2. **Game Running**



- Tampilan utama dengan latar belakang kayu.
- Objek Semut bergerak ke bawah.
- UI di pojok atas menunjukkan Skor, High Score, dan Nyawa (lives).

3. **Game Over Screen**

- Muncul ketika self.lives mencapai 0.
- Menampilkan pesan "GAME OVER" dan Skor Akhir.
- Pemain dapat menekan SPACE untuk memanggil reset_game().

BAB V PENUTUPAN

5.1 Kesimpulan

Proyek “OOP Ant Killer Game” berhasil mencapai tujuan utamanya sebagai sarana implementasi praktis dari konsep Pemrograman Berorientasi Objek (PBO) dalam pengembangan perangkat lunak. Melalui proyek ini, konsep-konsep PBO tidak hanya dipahami secara teoritis, tetapi juga diterapkan secara nyata dalam bentuk aplikasi game 2D yang fungsional, terstruktur, dan mudah dikembangkan. Setiap bagian dari sistem dirancang untuk mencerminkan prinsip-prinsip dasar PBO secara konsisten.

Penerapan konsep Inheritance terlihat jelas melalui pemodelan berbagai jenis semut, seperti SemutPrajurit dan SemutRatu, yang diturunkan dari satu kelas dasar yaitu Semut. Dengan pendekatan ini, atribut dan perilaku umum dapat didefinisikan pada kelas induk, sementara karakteristik khusus masing-masing jenis semut dikelola pada kelas turunan. Hal ini membuat kode menjadi lebih efisien, mengurangi duplikasi, serta memudahkan penambahan jenis semut baru di masa mendatang.

Konsep Polymorphism memungkinkan setiap jenis semut memiliki implementasi perilaku yang berbeda meskipun menggunakan metode yang sama, seperti bergerak() dan diserang(). Perbedaan perilaku ini menjadi inti dari desain modular, karena sistem game dapat memperlakukan semua objek semut secara seragam tanpa perlu mengetahui tipe spesifiknya. Dengan demikian, logika permainan menjadi lebih fleksibel dan mudah dipelihara.

Selain itu, Encapsulation diterapkan untuk melindungi data internal setiap entitas, seperti nilai health, posisi, dan status semut. Data tersebut hanya dapat diakses atau diubah melalui metode yang telah ditentukan, sehingga integritas data tetap terjaga dan risiko kesalahan logika dapat diminimalkan. Pendekatan ini juga meningkatkan keterbacaan serta keamanan kode.

Dari sisi arsitektur, penggunaan Composition pada kelas GameController berperan penting dalam mengoordinasikan seluruh komponen game, mulai dari pengelolaan objek semut, input pengguna, hingga status permainan. Dengan sistem yang terpusat dan terorganisasi dengan baik, keseluruhan alur game menjadi lebih mudah dikelola. Hasil akhirnya adalah sebuah aplikasi game 2D yang tidak hanya berjalan dengan baik, tetapi juga dirancang sesuai dengan kaidah Pemrograman Berorientasi Objek yang baik dan benar.

5.2 Tantangan Dan Saran

Tantangan
Selama pengembangan proyek OOP Ant Killer Game, tantangan utama yang dihadapi adalah merancang struktur kelas yang tepat agar konsep Encapsulation, Inheritance, dan Polymorphism dapat diterapkan secara optimal. Selain itu, pengelolaan game loop, state permainan (paused, running, game over), serta sinkronisasi event input dan pergerakan objek di Pygame memerlukan logika yang cermat agar permainan berjalan stabil dan responsif.

Saran
Untuk pengembangan selanjutnya, game ini dapat ditingkatkan dengan menambahkan level bertahap, variasi musuh, serta peningkatan visual dan animasi. Dari sisi arsitektur, penerapan design pattern seperti *Factory Method* dapat membuat sistem spawning lebih fleksibel. Selain itu, mekanisme penyimpanan data dapat dikembangkan menggunakan format yang lebih terstruktur, serta dokumentasi dan pengujian kode dapat diperluas agar aplikasi lebih mudah dikembangkan di masa depan.

LINK GITHUB :
https://github.com/pradityaa09/M.DZAKY-BUDI-PRADITYA_24091397033_UASPBO