LAPORAN PROYEK UAS PEMROGRAMAN BERORIENTASI OBJEK

IMPLEMENTASI KONSEP OOP DALAM PENGEMBANGAN GAME SEDERHANA

Mata Kuliah:

Pemrograman Berorientasi Objek

Oleh:

Anggun Amaylia Abdillah

(24091397123)

PROGRAM STUDI MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

2025

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pemrograman Berorientasi Objek merupakan salah satu paradigma pemrograman yang sangat penting dalam pengembangan perangkat lunak modern. OOP memudahkan proses pembangunan aplikasi berskala besar dengan menyediakan konsep seperti enkapsulasi, pewarisan, dan polimorfisme, sehingga kode menjadi lebih terstruktur, mudah dipelihara, dan dapat dikembangkan kembali.

Dalam mata kuliah Pemrograman Berorientasi Objek, mahasiswa dituntut untuk tidak hanya memahami konsep OOP secara teori, tetapi juga mampu mengimplementasikannya melalui sebuah proyek aplikasi nyata. Game menjadi salah satu bentuk aplikasi yang paling tepat untuk menguji kemampuan tersebut, karena game memiliki banyak entitas, interaksi, dan logika yang memungkinkan penerapan OOP secara penuh.

Python dipilih sebagai bahasa pemrograman utama karena bersifat sederhana, mudah dipelajari, dan memiliki komunitas yang besar. Sedangkan Pygame dipilih sebagai game engine karena menyediakan antarmuka yang lengkap untuk mengelola grafik 2D, animasi, suara, event handling, serta struktur game loop yang dibutuhkan dalam pengembangan game. Pygame sangat cocok dipakai dalam lingkungan akademik dan proyek berukuran menengah karena sifatnya ringan, open-source, dan mendukung penerapan konsep OOP dengan baik.

Game “Elion – The Last Lightkeeper” dirancang untuk menjadi sebuah permainan petualangan 2D (2D top-down adventure) yang tidak hanya menghibur, tetapi juga menjadi contoh nyata bagaimana OOP digunakan dalam sebuah game. Game ini memiliki struktur dunia, sistem musuh, *companion*, *collectible items*, portal, animasi, serta logika tingkat lanjut seperti musuh, *particle effects*, dan *camera shake*. Semua fitur tersebut membutuhkan penerapan OOP secara konsisten dan terstruktur.

Dengan demikian, pengembangan game ini diharapkan dapat memberikan pemahaman yang lebih mendalam mengenai penerapan OOP dalam proyek aplikasi nyata, khususnya dalam konteks pembuatan game 2D menggunakan Python dan Pygame.

1.2 Tujuan Aplikasi

Tujuan dari pengembangan aplikasi ini adalah:

1. Membuat game petualangan 2D berbasis Pygame yang interaktif dan menarik, sehingga dapat menjadi media pembelajaran sekaligus karya kreatif mahasiswa dalam penerapan OOP.
2. Mengimplementasikan konsep-konsep OOP secara nyata, yaitu:
    - *Encapsulation*: Menyembunyikan atribut dan fungsi internal melalui penggunaan atribut private serta getter/setter.
    - *Inheritance*: Menggunakan kelas induk seperti Enemy yang diturunkan ke kelas anak seperti GlimpEnemy dan UmbraEnemy.
    - *Polymorphism*: Mengizinkan kelas turunan untuk menimpa/*override* metode induk seperti take_action() pada setiap musuh.
3. Membangun gameplay yang interaktif, seperti sistem pergerakan *player*, *companion*, musuh dengan perilaku yang berbeda, sistem pengumpulan item, dan portal untuk menyelesaikan level.
4. Menciptakan struktur kode yang bersih, modular, dan *scalable*, sehingga game dapat dikembangkan ke level-level berikutnya atau diperluas dengan fitur tambahan seperti *cutscene, boss fight*, dan efek suara.

1.3 Deskripsi Singkat Game

- **Judul Game:** Elion – The Last Lightkeeper
- **Genre:** *2D Top-Down Fantasy Adventure*
- **Tema & Cerita Singkat:**

  Pemain berperan sebagai Elion, seorang pemuda terakhir dari klan penjaga cahaya (*Lightkeepers)* yang memiliki tugas untuk menjaga keseimbangan antara dunia manusia dan dunia roh. Dunia telah dibungkus kegelapan setelah sumber cahaya purba menghilang. Untuk memulihkan dunia, Elion harus mengumpulkan tiga Lilin Cahaya (*Spirit Light Shards*) yang tersebar di wilayah Spirit Meadow. Dalam petualangannya, Elion ditemani oleh Moku, spirit berbentuk cahaya yang berfungsi sebagai pemberi petunjuk. Elion juga harus menghadapi dua jenis monster: Glimp, penjaga merah yang berpatroli, dan Umbra, makhluk bayangan yang memburu pemain ketika mendekat. Setelah mengumpulkan semua Lilin Cahaya, Elion harus menuju *Lightkeeper Gate*, portal purba yang hanya akan terbuka ketika cahaya telah lengkap. Dari sini, Elion melanjutkan perjalanannya untuk memasuki istana kuno dan menghadapi tantangan selanjutnya.

- **Fitur Utama Game:**
    - Player bergerak secara bebas dalam dunia 2D top-down
    - Companion yang mengikuti player dan memberikan hint
    - Dua jenis musuh dengan perilaku berbeda
    - Sistem pengumpulan item (3 Lilin Cahaya)
    - Portal akhir untuk menyelesaikan level
    - Efek visual (particle, glow, shake)

BAB II

ANALISIS & PERANCANGAN

2.1 OOP yang digunakan

Pada pengembangan Elion – The Last Lightkeeper paradigma Object-Oriented Programming (OOP) digunakan secara eksplisit untuk mengorganisir entitas game, memisahkan tanggung jawab, dan membuat kode yang mudah dikembangkan. Pada implementasi ini diterapkan tiga pilar utama OOP: *Encapsulation, Inheritance*, dan *Polymorphism*. Berikut penjelasan masing-masing beserta contoh nyata dari kode.

2.1.1 Encapsulation (Enkapsulasi)

Menyembunyikan data internal objek dan menyediakan antarmuka (metode/getter/setter) untuk berinteraksi sehingga detail implementasi dapat berubah tanpa merusak kode pemanggil.

Rancangan pada game:

- Atribut privat: banyak atribut kelas dibuat privat (konvensi _underscore) seperti: Player._x, Player._y, Player._inventory, Player._score, Player._lives, Player._vx, Player._vy, Enemy._x, Enemy._y, Gem._collected, dll.
- Getter / method akses: Tersedia method untuk mengakses state tanpa memanipulasi langsung, mis. get_position(), get_rect(), get_lives(), get_score(), is_collected().
- Behavior di dalam kelas: Semua perilaku yang berkaitan disimpan di kelasnya masing-masing — contoh: movement dan handle_input ditangani Player.handle_input(), damage logic pada Player.take_damage(), update animasi gem pada Gem.update().

Memungkinkan pengubahan implementasi internal (mis. mengganti struktur inventory) tanpa mengubah pemanggilan dari kelas lain (mis. Game atau UI). Mempermudah debugging & keamanan state.

2.1.2 Inheritance (Pewarisan)

Kelas anak mewarisi atribut & metode dari kelas induk sehingga mengurangi duplikasi dan menstrukturkan tipe objek secara hierarkis.

Rancangan pada game:

- Enemy adalah kelas dasar (parent) yang menyediakan atribut umum (_x, _y, _vx, _vy, _size, _speed) serta interface take_action(player, dt) dan update(dt).

- GlimpEnemy dan UmbraEnemy adalah kelas turunan yang mewarisi Enemy dan menambahkan/menimpa perilaku:
    - ⇒ GlimpEnemy.take_action() mengimplementasikan patrol waypoint dan wobble animation.
    - ⇒ UmbraEnemy.take_action() mengimplementasikan chase behavior, trail particle, dan aggro range

Menyederhanakan definisi musuh baru serta cukup membuat subclass dan override sedikit metode spesifik tanpa menulis ulang logika umum.

2.1.3 *Polymorphism* (Polimorfisme)

Kemampuan memanggil metode yang sama pada objek berbeda dan mendapatkan perilaku berbeda sesuai tipe objek.

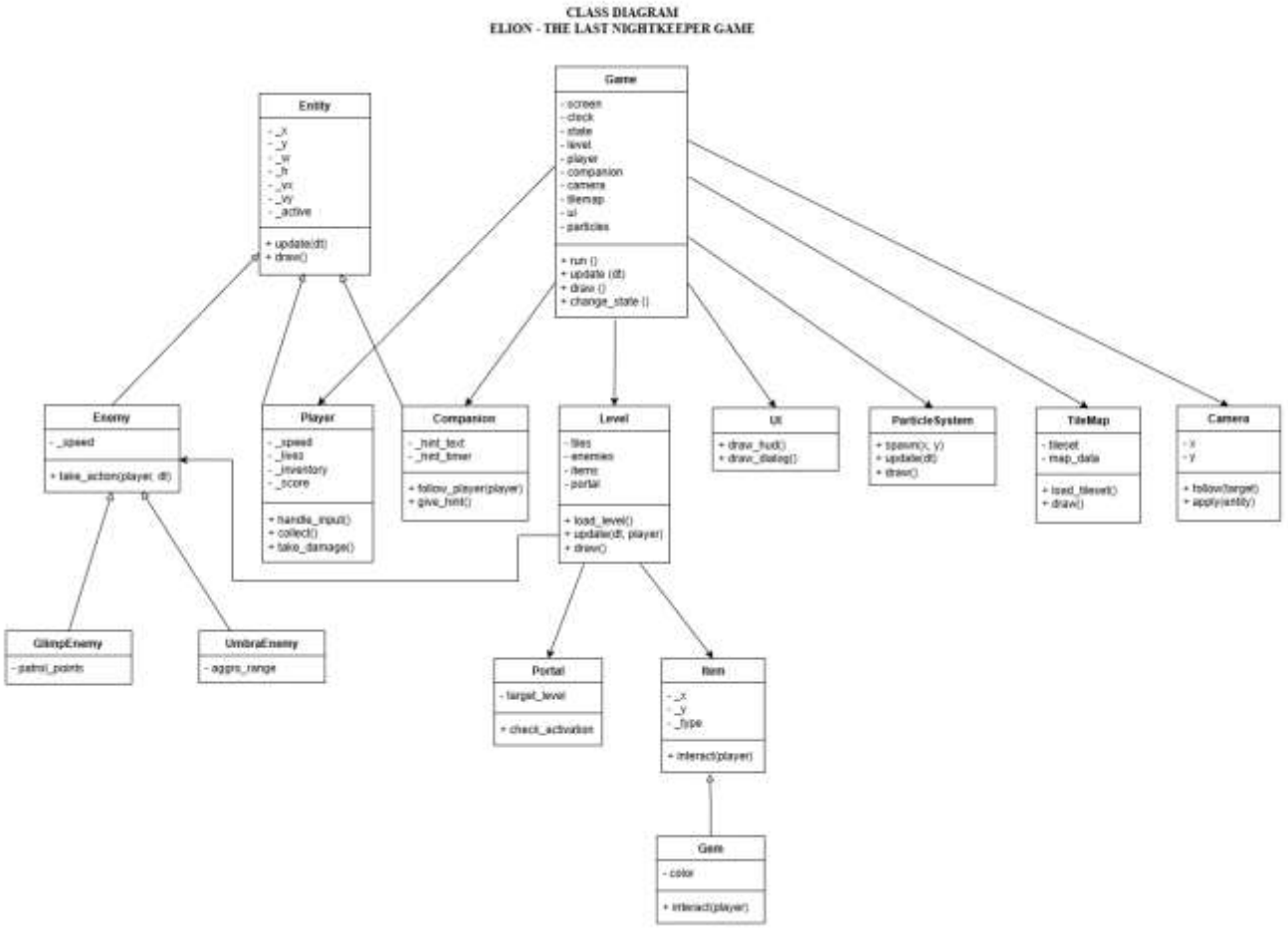
Rancangan pada game:

- Kode update umum yang memanggil loop seperti *for enemy in self.enemies:* meskipun tipe enemy bisa GlimpEnemy atau UmbraEnemy. Pemanggilan take_action mengeluarkan perilaku berbeda sesuai kelas.
- Pada rendering atau collision, koleksi homogennya (*list of Enemy*) diproses dengan cara yang sama meski hasilnya berbeda.

Mempermudah penambahan variasi perilaku, memperkecil *conditional branching* berdasarkan tipe di tempat lain.

2.2 Class Diagram

Pada tahap perancangan aplikasi, pembuatan class diagram menjadi bagian penting untuk menggambarkan struktur program secara menyeluruh. Class diagram memberikan visualisasi hubungan antar kelas, atribut yang dimiliki, serta fungsi-fungsi utama yang digunakan dalam sistem. Dalam pengembangan game Elion – The Last Lightkeeper, class diagram digunakan untuk memastikan bahwa konsep Object-Oriented Programming (OOP) diterapkan secara konsisten, terutama pada aspek encapsulation, inheritance, dan polymorphism.



Penjelasan Class Diagram

Class diagram pada game **Elion – The Last Lightkeeper** terdiri dari beberapa kelompok kelas utama, yaitu:

1. **Kelas Dasar (Parent Class)**

   Entity:

   Kelas Entity berfungsi sebagai kelas induk abstrak bagi hampir semua objek dinamis dalam game, seperti Player, Enemy, dan Companion, untuk menyediakan fondasi umum dan menghindari duplikasi kode. Ini menunjukkan penerapan inheritance yang efisien, di mana subclass mewarisi atribut dan metode dasar sambil menambahkan perilaku spesifik.

2. **Kelas Pemain dan Companion**

   Player:

   Kelas Player merepresentasikan karakter utama, Elion, sebagai objek sentral yang dikendalikan pengguna. Ini adalah subclass dari Entity, sehingga mewarisi atribut dasar sambil menambahkan fitur interaktif. Player memiliki metode untuk menerima input (handle_input()), mengumpulkan item (collect()), dan menerima damage

(take_damage()).

Companion:

Kelas Companion mengelola Moku, pendamping yang memberikan dukungan naratif dan mengikuti Player. Ini juga subclass dari Entity, memastikan konsistensi pergerakan.

3. **Kelas Musuh**

Enemy (Parent):

Enemy adalah kelas dasar musuh yang memiliki atribut kecepatan (_speed) dan metode abstrak take_action(). Semua musuh menggunakan metode ini untuk menentukan AI atau perilaku serangan mereka.

GlimpEnemy:

Musuh tipe patrolling, bergerak mengikuti titik-titik rute yang sudah ditentukan (_patrol_points). Perilakunya menggambarkan polymorphism dengan mengimplementasikan take_action() versi sendiri.

UmbraEnemy:

Musuh yang mengejar player menggunakan radius agresi (_aggro_range). Menggunakan logika AI yang berbeda, tetap dengan prinsip polymorphism.

4. **Kelas Item**

Item (Parent):

Item adalah kelas dasar untuk seluruh objek yang bisa diambil. Menyimpan posisi dan tipe, serta metode abstrak interact().

Gem (Child):

Gem merupakan turunan Item.Gem memiliki atribut warna dan metode interact() yang menambahkannya ke inventori Player. Gem diperlukan untuk membuka portal ke level berikutnya.

5. **Kelas Level**

Level:

Level berfungsi sebagai pengatur elemen-elemen di satu area permainan: tile, musuh, item, dan portal. Level memiliki metode load_level() untuk memuat konfigurasi level, update() untuk memperbarui objek dalam level, dan draw() untuk merender seluruh elemen. Kelas ini menjadi pusat manajemen lingkungan permainan.

6. **Kelas Pendukung Lingkungan**

TileMap:

Digunakan untuk memuat dan menampilkan tileset sebagai dasar peta permainan. TileMap menyimpan data peta dan metode untuk meload tiles dari spritesheet.

Camera:

Berfungsi mengikuti pergerakan Player agar dunia game dapat digulir (scrolling). Camera mengaplikasikan offset posisi menggunakan apply().

Portal:

Objek yang memindahkan Player ke level berikutnya jika syarat terpenuhi. Portal memiliki atribut target_level dan metode check_activation() untuk mengatur perpindahan level.

7. **Sistem Antarmuka dan Efek**

UI:

UI menangani tampilan HUD seperti nyawa, jumlah gem, level, dan timer. Metode draw_hud() memastikan informasi penting selalu terlihat.

ParticleSystem:

Menghasilkan efek visual seperti cahaya, debu, atau magic particle. Memiliki metode spawn(), update(), dan draw().

8. **Kelas Game (Controller Utama)**

Game:

Kelas Game berfungsi sebagai pengontrol utama dan pusat koordinasi untuk seluruh sistem permainan Elion – The Last Lightkeeper. Tugas intinya adalah mengatur alur state machine permainan, mengelola transisi antara status-status penting seperti Menu, Playing, Pause, Win, dan Game Over. Selain itu, kelas ini bertanggung jawab penuh untuk mengelola dan mengkoordinasikan semua objek kunci yang membentuk lingkungan permainan, termasuk Level, Player, Companion, data visual TileMap, elemen interaktif UI, efek visual ParticleSystem, dan sistem pandangan Camera.

Logika inti permainan diatur dalam metode *run()*, yang menangani game loop utama, sementara metode *update()* dan *draw()* secara berkesinambungan mengelola pemrosesan logika dan rendering grafis untuk setiap frame, memastikan keseluruhan sistem berjalan secara sinkron.

2.3 Perancangan Gameplay / Level

Bagian ini menjelaskan rancangan gameplay dan struktur level dalam game *Elion – The Last Lightkeeper*. Setiap level dirancang dengan tujuan pembelajaran, mekanika permainan, serta peningkatan tingkat kesulitan secara bertahap. Perancangan level juga disesuaikan dengan implementasi Object-Oriented Programming (OOP) yang telah dibahas pada subbab sebelumnya.

2.3.1. WORLD MAP – Tahap Edukasi Awal

- Tujuan utama:

  Memberikan pengantar konsep OOP kepada pemain sebelum permainan utama dimulai, serta membangun konteks cerita perjalanan Elion menuju Lightkeeper Castle.

- Layout & Visual:
  - Tampilan peta dunia (world map) dengan sudut pandang top-down statis.
  - Terdapat 3 node lokasi yang saling terhubung menggunakan jalur cahaya.
- Objek & Interaksi:
  - Node lokasi berbentuk ikon interaktif (clickable).
  - Codex Panel muncul ketika node diklik.
  - Tombol “Mulai Petualangan” akan aktif setelah seluruh Codex dibaca.
- Sistem Edukasi (codex):
  - Setiap lokasi memiliki satu Codex:
    - *Spirit Meadow* → Encapsulation
    - *Crimson Mountain* → Inheritance & Polymorphism
    - *Lightkeeper Castle* → Arsitektur Game & Class Diagram
- Transisi:
  - Setelah semua panel codex dibaca, lanjut ke opening cutscene → ke level 1.

2.3.2. LEVEL 1 – Spirit Meadow

- Tujuan utama:

  Memperkenalkan kontrol dasar (movement), companion, pengumpulan gem, dan bahaya musuh patrol.

- Layout & Visual:
    - Area terbuka berukuran sedang.
    - Beberapa pohon & batu (tilemap objects).
    - *Spawn point player* di pojok kiri bawah.
    - Portal kecil di pojok kanan atas.
- Objek & Penempatan:
    - Gem (positions): (400,300), (200,150), (520,90)
    - 1–2 Patrol Enemies (GlimpEnemy) dengan waypoint rectangle: [(200,200), (600,200), (600,400), (200,400)]
    - Companion spawn dekat player (spawn_x - 40)
- Musuh:
    - GlimpEnemy berpatroli sepanjang *waypoints*. Jika player mendekat dalam radius kecil (<100 px) → alert glow (visual) tetapi tidak langsung chase.
    - Collision: jika player.get_rect().colliderect(enemy.get_rect()) → player.take_damage().
- Pickup & Interaksi:
    - Ketika player bertabrakan dengan Gem → Gem.collect() → player.collect_gem() → particle + sound.
    - Gem disimpan di player._inventory (encapsulation), HUD diupdate.
- Portal & Gate:
    - Portal tetap tidak aktif sampai player.get_gem_count() >= 3.
    - Sesudah gem lengkap → portal glow aktif → player boleh masuk → Game akan meng-increment level atau memulai Level 2.
- Balancing parameter awal:
    - Player speed: 150 px/detik
    - Patrol enemy speed: 80 px/detik
    - Jumlah nyawa awal: 3
    - Nilai score per gem: +100 score
- Edge cases & implementasi:
    - Jika player kehilangan hidup (lives == 0) → Game.state = GAMEOVER.
    - Pastikan gem spawn tidak tersembunyi di dalam object (tile collision detection).

2.3.3. LEVEL 2 – Crimson Mountain

- Tujuan utama:

  Meningkatkan tantangan permainan melalui musuh agresif serta memperkenalkan mekanisme bertahan hidup dan strategi menghindar.

- Layout & Visual:
  - Area sempit dan lebih gelap dibanding level sebelumnya.
  - Dominasi warna merah gelap dan ungu untuk menekankan atmosfer bahaya.
  - Jalur eksplorasi lebih terbatas dan menuntut ketelitian.

- Objek & Penempatan:
  - 2–3 musuh bertipe *FlareWolf*.
  - Spirit Gem dijaga oleh musuh.
  - Portal terletak di area paling akhir level.

- Musuh:
  - Flare Wolf:
    - Patrol small area; ketika melihat player dalam radius 200 px → dash (burst speed) arahnya ke player.
    - Single hit kill (balancing: cepat tapi sedikit HP).
    - Implementasi: raycast kecil atau direct vector normalized * dash_duration * speed.
  - Forest Guardian:
    - HP: 3
    - Perilaku fase:
      - ⇒ Phase 1 (HP 3): slow walk, melee collision deals damage.
      - ⇒ Phase 2 (HP 2): periodically shoots slow projectiles toward player (projectile speed ~120 px/s).
      - ⇒ Phase 3 (HP 1): aggressive movement, short-range charge, increased speed.
    - Boss dapat memberikan feedback visual tiap hit: flash, particle burst, camera.shake().

- Mekanikan Serangan Pemain (Level 2 aktif):
  - Spirirt Burst projectile:
    - ⇒ Kecepatan: 250–300 px/s
    - ⇒ Lifetime: 1.2 s

⇒ Cooldown: 0.5 s

⇒ Damage: 1 per hit (Boss membutuhkan 3 hits)

⇒ Implementasi: class Projectile dengan update(), draw(), dan collides_with(enemy).

- Arena Logic:
  - Boss mulai aktif saat player memasuki arena rectangle (trigger zone).
  - Jika player keluar dari arena, boss tetap aktif (desain preferensi), atau dipause kalau ingin reset.
- Balancing & Tuning:
  - Jarak safe zone minimal di narrow path (1.5x player width).
  - Buat projectiles boss lambat namun mematikan jika tidak dihindari.
- Keberhasilan: Setelah boss HP == 0 → drop key / portal unlock → pemain lanjut ke Level

2.3.4. LEVEL 3 – Lightkeeper Castle

- Tujuan utama:

  Menguji kemampuan combat pemain (Spirit Burst attack), mengatur positioning, menangkal projectile boss, dan menunjukkan first "boss-like" encounter Menghadirkan pengalaman klimaks: menaruh 3 gems di altar, mengaktifkan ritual, dan menampilkan ending cinematic.
- Layout & Visual:
  - Interior castle: ruang utama (circular altar area) dikelilingi pilar tinggi.
  - Lantai marmer bercahaya, lighting spotlight dari atas (spotlight tecnica via overlay surface).
  - Beberapa ruang kecil (koridor) yang dapat berisi decorative spirits (non-hostile).
- Objek & Interaksi:
  - Spirit Altar (center): objek besar yang menerima gem
    - Ketika player berada di radius altar dan menekan E atau kondisi auto-trigger, inventory member of gems akan "melayang" ke altar satu per satu (animasi).
    - Altar menampung 3 gems → ketika lengkap, mulai ritual sequence.
  - Trigger:

- Step 1: setiap gem mengorbit altar (animation)
- Step 2: cahaya meningkat (overlay white fade), particle spawn massive
- Step 3: Spirit Tree grows (sprite scale up / procedural drawing)
- Step 4: ending text + music → return to menu or credits.

➢ Mekanik Ending:

- save_score() dipanggil, hasil waktu/score tersimpan.
- Kamera zoom: implementasi sederhana dengan men-scale render surface atau menyesuaikan camera.smoothing & target toward altar center and decreasing camera.width/height for zoom feel.
- Cutscene timing: sequence timed by deltatime, gunakan CutsceneManager / state machine untuk transisi.

2.3.5. ENDING REFLECTION – Penutup Edukasi

➢ Tujuan utama:

Memberikan penutup naratif sekaligus refleksi pembelajaran OOP.

➢ Layout & Visual:

- Layar hitam dengan partikel cahaya.
- Teks refleksi muncul perlahan: Encapsulation, Inheritance, dan Polymorphism. Berfungsi untuk menegaskan bahwa pemain telah memahami konsep OOP melalui pengalaman bermain.

BAB III

IMPLEMENTASI APLIKASI

3.1 Struktur File Program

Aplikasi game Elion – The Last Lightkeeper diimplementasikan sepenuhnya dalam satu file utama, yaitu elion_pygame.py. Pendekatan ini dipilih untuk mengoptimalkan kemudahan pengembangan dan pemeliharaan, terutama dalam konteks proyek skala kecil, di mana modularitas file terpisah tidak diperlukan. Meskipun hanya satu file, struktur kode tetap terorganisir dengan baik melalui penggunaan kelas-kelas OOP yang mandiri, sehingga menghindari kekacauan dan mematuhi prinsip single responsibility principle (SRP) dalam OOP. File ini mencakup sekitar 5194 baris kode yang dibagi menjadi bagian-bagian logis sebagai berikut:

1. Konstanta Global dan Pengaturan Game

   Bagian awal file berisi variabel-variabel global yang mendefinisikan parameter dasar permainan, termasuk resolusi layar (screen resolution), skema warna, dimensi entitas (sprite), nilai kecepatan pergerakan, dan enumerasi (GameState) untuk mengelola status permainan.

2. Kelas-Kelas OOP yang Mengatur Sistem Game

   Segmen terbesar yang menampung seluruh definisi kelas yang membentuk Seluruh logika permainan dienkapsulasi dalam kelas-kelas yang didefinisikan secara berurutan dalam file dan entitas permainan, mulai dari sistem partikel, kamera, karakter utama, musuh, hingga manajemen peta.

3. Game Loop Utama

   Bagian ini berisi logika eksekusi permainan, mencakup inisialisasi (initialization) objek Pygame, siklus pembaruan logika (update logic), urutan penggambaran (rendering) frame, dan penanganan event masukan (input handling) dari pengguna.

3.2 Penjelasan Kelas

Penjelasan setiap kelas dijelaskan secara rinci, termasuk atribut utama, metode kunci, dan peranannya dalam ekosistem game. Penjelasan difokuskan pada implementasi OOP dalam satu file, di mana kelas-kelas saling merujuk tanpa impor eksternal selain Pygame.

A. Sistem Partikel

Aplikasi diawali dengan pembentukan kelas **ParticleData** dan **ParticleSystem**, yang digunakan untuk menghasilkan berbagai efek visual seperti percikan cahaya pada gem, efek serangan musuh, maupun animasi portal. `ParticleData` berfungsi sebagai struktur data sederhana yang menyimpan posisi, kecepatan, warna, serta durasi hidup partikel.

Atribut utama:

- _x, _y: Koordinat posisi awal (float).
- _vx, _vy: Komponen kecepatan vektor (float) untuk pergerakan.
- _life: Durasi hidup partikel dalam frame (int), yang berkurang setiap update.
- _color: Tupel RGB untuk warna partikel (tuple[int]).
- _active: Flag boolean yang menandai apakah partikel sedang digunakan.
- Fungsi: Partikel ini diinisialisasi dalam pool dan diaktifkan saat diperlukan, memastikan efisiensi dalam rendering.

Sementara itu, *ParticleSystem* bertanggung jawab untuk mengatur seluruh partikel tersebut, mulai dari proses penciptaan, pembaruan posisi, hingga proses menggambar partikel ke layar. Sistem partikel menggunakan pendekatan object pooling sehingga lebih efisien dan tidak membebani memori.

Atribut Utama:

- _particles: Daftar instance ParticleData (list[ParticleData]).
- _pool_size: Ukuran pool tetap (int, misalnya 100).
- Metode utama:
- emit(x, y, vx, vy, life, color): Mengaktifkan partikel baru dari pool dengan parameter tertentu.
- update(): Memperbarui posisi dan life setiap partikel aktif, serta menonaktifkan yang habis masa hidupnya.
- draw(screen): Menggambar partikel aktif sebagai lingkaran kecil menggunakan pygame.draw.circle().
- Penggunaan: Kelas ini dipanggil oleh Player (untuk efek liontin), Gem (kilauan), Portal (efek spiritual), dan Enemy (damage/serangan), sehingga mendukung reuse kode.

B. Sistem Kamera

Kelas **Camera** digunakan untuk mengatur tampilan dunia berdasarkan posisi pemain. Kamera mengikuti gerakan Elion secara halus dengan teknik interpolasi, dan juga menyediakan efek tambahan seperti *screen shake* ketika pemain menerima serangan atau mengambil gem. Dalam game ini, kamera menjadi komponen penting untuk menghasilkan pengalaman bermain yang lebih dinamis dan sinematik.

Atribut utama:

- _target: Referensi ke objek target (misalnya, Player).
- _offset_x, _offset_y: Perpindahan kamera (float).
- _shake_intensity, _shake_duration: Parameter untuk efek getar (float, int).
- Metode utama:
- set_target(target): Menetapkan objek yang diikuti.
- update(dt): Memperbarui offset menggunakan interpolasi linier (lerp) untuk follow halus, serta mengaplikasikan shake jika aktif.
- get_offset(): Mengembalikan tuple offset untuk rendering.
- Fitur: Efek shake diaktifkan pada event seperti damage, pengambilan gem, atau kemenangan, dengan durasi dan intensitas yang dapat disesuaikan.

C. Player dan Companion

Kelas **Player** berfungsi untuk merepresentasikan karakter utama bernama Elion. Seluruh atribut penting seperti posisi, kecepatan, jumlah nyawa, inventori, dan skor disembunyikan menggunakan mekanisme encapsulation melalui penamaan atribut privat. Player memiliki kemampuan untuk bergerak bebas, mengambil item, menerima damage, menampilkan animasi idle, serta menghasilkan efek cahaya dari liontin roh yang dibawanya. Perilaku-perilaku tersebut diatur melalui method seperti `handle_input`, `update`, dan `collect_gem`, yang semuanya terkapsulasi dalam class ini.

Atribut utama:

- _x, _y: Posisi (float).
- _vx, _vy: Kecepatan (float).
- _inventory: Daftar gem yang dikumpul (list[Gem]).
- _score: Nilai skor (int).
- _lives: Jumlah nyawa (int, awal 3).
- _invincible_timer: Timer kekebalan setelah damage (float).
- Metode utama:

- handle_input(keys): Memproses input keyboard (WASD) untuk pergerakan.
- update(dt, tilemap, enemies, gems, portal): Memperbarui posisi, deteksi collision, dan interaksi.
- draw(screen, camera): Menggambar sprite dengan efek glow jika memiliki gem.
- collect_gem(gem): Menambah inventory, skor, dan memicu efek shake serta hint.
- take_damage(): Mengurangi nyawa jika tidak invincible, memicu partikel damage.
- Fitur: Animasi idle (bobbing), partikel liontin, dan getter seperti get_position() untuk akses aman.

Elion ditemani oleh **Companion**, seekor beruang kecil bernama Moku. Companion ini mengikuti pemain menggunakan teknik *linear interpolation*, sehingga pergerakannya terlihat halus dan natural. Selain itu, Moku berfungsi sebagai **Mentor Spirit**, yaitu pembimbing yang menyampaikan edukasi dan refleksi secara kontekstual selama permainan berlangsung. Kelas Companion juga memiliki animasi naik-turun (bobbing effect) serta efek glow saat mengeluarkan hint, sehingga membuat interaksi antar karakter menjadi lebih hidup.

Atribut Utama:

- _x, _y: Posisi (float).
- _hint_timer: Timer untuk hint (float).
- Metode utama:
- follow_player(player_pos, dt): Mengikuti pemain menggunakan lerp untuk pergerakan halus.
- give_hint(): Menampilkan hint acak melalui UI jika cooldown selesai.
- draw(screen, camera): Menggambar sprite dengan efek glow saat memberikan hint.
- can_give_hint(): Memeriksa ketersediaan hint.
- Fitur: Animasi bobbing dan aktivasi hint otomatis saat gem diambil.

Sebagai Mentor Spirit, Companion memiliki kemampuan untuk:

- Memberikan hint gameplay, seperti petunjuk jumlah gem yang tersisa
- Menyampaikan penjelasan singkat OOP dalam bentuk dialog ringan
- Memberikan refleksi naratif setelah pemain menyelesaikan suatu tahap

Implementasi mentor ini dirancang agar tidak memotong gameplay. Hint dan pesan edukasi muncul secara dinamis berdasarkan kondisi permainan, misalnya:

- Setelah pemain mengambil sebuah Spirit Gem
- Ketika mendekati Portal
- Saat memasuki area berbahaya atau menghadapi musuh tertentu

Dari sisi OOP, Companion tetap menerapkan encapsulation melalui atribut privat dan method khusus seperti give_hint() dan can_give_hint(). Hal ini menunjukkan bahwa fitur edukasi tetap dibangun di atas struktur OOP yang konsisten dan terorganisir.

D. Sistem Musuh

Untuk musuh, game ini menggunakan satu parent class, yaitu **Enemy**, yang menjadi basis dari semua tipe musuh.

Atribut Utama:

- _x, _y: Posisi (float).
- _speed: Kecepatan (float).
- _rect: Objek pygame.Rect untuk collision.
- Metode utama:
- take_action(player_pos): Metode abstrak yang dioverride di subclass.
- update(dt): Memperbarui posisi berdasarkan aksi.
- draw(screen, camera): Menggambar sprite dasar.

Dua kelas turunan dibuat dari kelas ini, yaitu **GlimpEnemy** dan **UmbraEnemy**, yang membuktikan penerapan inheritance dan polymorphism dalam program. **GlimpEnemy** merupakan musuh tipe patroli yang bergerak mengikuti rute waypoint tertentu. Ia dapat masuk mode waspada jika pemain mendekat, serta memiliki animasi tubuh jelly yang bergetar. Musuh ini menggunakan metode override: take_action (player_pos) untuk beralih ke mode chase jika dalam jarak, sehingga menunjukkan penerapan polymorphism yang sangat jelas.

Fitur Logika GlimpEnemy:

- Berpatroli mengikuti waypoint
- Memiliki efek wobble
- Menjadi alert ketika player dekat
- Memberikan glow merah saat mendekat.

Sebaliknya, **UmbraEnemy** memiliki perilaku menyerang pemain secara aktif dengan algoritma pengejaran. Saat mengejar, tubuhnya memanjang, mata menyala, dan ia meninggalkan jejak asap ungu. Musuh ini menggunakan metode override: take_action (player_pos) untuk menghitung vector ke pemain untuk pengejaran, sehingga menunjukkan penerapan polymorphism yang sangat jelas

Fitur Logika UmbraEnemy:

- Mengejar player bila masuk aggro range
- Tubuh memanjang (stretch) saat agresif
- Meninggalkan jejak asap ungu
- Mata bersinar kuning.

E. TileMap

Lingkungan permainan dikendalikan oleh kelas **TileMap**, yang menghasilkan peta 2D berukuran luas. TileMap membagi dunia menjadi tiga lapisan, yaitu lapisan dasar berupa rumput, lapisan objek bawah seperti batang pohon, dan lapisan objek atas seperti kanopi pohon. Pembagian layer ini memungkinkan player dan musuh dapat berjalan “di bawah” pohon, sehingga memberikan kedalaman visual pada game.

Atribut utama:

- _layers: Daftar surface pre-rendered (list[pygame.Surface]).
- Metode utama:
  - __init__(map_data): Memuat data peta dan pre-render layer (base, below, above).
  - draw(screen, camera, layer_type): Menggambar layer spesifik dengan offset.
  - Fitur: Teknik pre-render untuk efisiensi, memungkinkan karakter bergerak di antara layer.

F. Sistem World Map & Codex Edukasi

Sistem World Map yang berfungsi sebagai tahap awal sebelum pemain memasuki gameplay utama. World Map ini berperan sebagai penghubung naratif sekaligus mekanisme edukatif yang mempersiapkan pemain secara konseptual. World Map divisualisasikan sebagai peta perjalanan menuju istana cahaya, yang terdiri dari beberapa node lokasi utama, yaitu:

- Spririt Meadow (Hutan Cahaya)
- Crimson Mountain (Gunung Merah)
- Lightkeeper Castle (Istana Cahaya)

Setiap node pada World Map dapat diinteraksikan oleh pemain menggunakan input keyboard atau mouse. Ketika sebuah node dipilih, game akan menampilkan panel Codex edukasi yang berisi informasi pembelajaran terkait konsep Object-Oriented Programming (OOP) yang diterapkan dalam game.

Dari sisi implementasi, World Map diatur sebagai state permainan terpisah (misalnya WORLD_MAP) yang dikelola oleh kelas Game. Selama berada pada state ini, pemain belum dapat mengendalikan karakter, sehingga fokus utama diarahkan pada eksplorasi informasi dan pemahaman alur cerita.

G. Sistem Codex Edukasi Berbasis Konsep OOP

Setiap lokasi pada World Map memiliki **Codex Edukasi**, yaitu panel informasi yang berfungsi sebagai media pembelajaran ringan mengenai konsep OOP. Codex ini dirancang agar tetap selaras dengan tema game dan tidak terasa seperti materi teori yang terpisah dari permainan.

Isi codex pada masing-masing lokasi meliputi:

- Spririt Meadow (Hutan Cahaya) → Penjelasan encapsulation
- Crimson Mountain (Gunung Merah) → Penjelasan Inhertance
- Lightkeeper Castle (Istana Cahaya) → Penjelasan Polymorphism dan refleksi seluruh game.

Codex ditampilkan dalam bentuk overlay UI dengan latar semi-transparan, teks naratif, dan ilustrasi sederhana. Pemain diwajibkan membaca seluruh Codex pada setiap lokasi sebelum dapat melanjutkan ke gameplay utama. Mekanisme ini diimplementasikan dengan sistem validasi sederhana (flag), sehingga tombol “Lanjutkan” hanya aktif jika seluruh Codex telah diakses.

Keberadaan Codex ini menjadikan game Elion – The Last Lightkeeper tidak hanya sebagai game hiburan, tetapi juga sebagai media pembelajaran kontekstual, di mana konsep OOP dipahami melalui contoh nyata dalam game itu sendiri.

H. Sistem Item

Item permainan direpresentasikan oleh kelas **Gem**, yaitu permata cahaya yang harus dikumpulkan pemain. Setiap gem memiliki warna, jenis, serta animasi pulse yang membuatnya bersinar lembut. Ketika pemain menyentuh gem, ia akan menghilang, menambahkan skor, mengaktifkan efek cahaya pada player, serta memicu hint dari companion.

Atribut Utama:

- _type: Jenis gem (str).
- _pulse_timer: Timer animasi (float).
- Metode utama:
- update(dt): Memperbarui pulse dan partikel.
- draw(screen, camera): Menggambar dengan glow dan kilau.
- Fitur: Saat diambil, naikkan skor, shake, hint, dan aura pemain.

Setelah ketiga gem dikumpulkan, pemain dapat mengakses **Portal**, yaitu gerbang cahaya yang menjadi tujuan akhir level. Portal memiliki animasi lingkaran cahaya, titik partikel yang berputar, serta efek visual yang semakin intens ketika pemain mendekat.
Atribut Utama:

- _active: Flag aktivasi (bool).
- Metode utama:
- update(player_gems): Aktif jika 3 gem, perbarui animasi orbiting.
- draw(screen, camera): Gambar ring glow dan partikel.
- Fitur: Responsif terhadap kedekatan pemain, aktif hanya jika syarat terpenuhi.

I. User Interface

Antarmuka pengguna (HUD) dan tampilan menu game diatur oleh kelas **UI**. Kelas ini menggambar nyawa, skor, timer, jumlah gem, hint bubble, menu utama, serta layar kemenangan. Seluruh elemen UI dirancang agar informatif dan mudah dibaca, namun tetap estetis sesuai tema fantasy-game.
Atribut Utama:

- _font: Objek font Pygame.
- Metode utama:
    - draw_hud(screen, lives, gems, score, time): Gambar ikon nyawa, gem, skor, timer.
    - draw_hint(screen, hint_text): Tampilkan kotak hint.
    - draw_menu(screen): Menu utama.
    - draw_win(screen): Layar kemenangan.

J. Game Controller

Terakhir, seluruh mekanisme game dikendalikan oleh kelas Game sebagai pusat logika aplikasi. Game bertanggung jawab terhadap pengaturan state (menu, playing, win),

pembaruan semua objek setiap frame, pemrosesan input pemain, rendering berurutan berdasarkan layer, serta menjalankan loop utama permainan. Kelas Game juga membuat seluruh objek lain seperti Player, Companion, Enemy, Portal, TileMap, dan ParticleSystem. Dengan demikian, class Game berfungsi sebagai otak utama yang mengoordinasikan seluruh sistem dalam game sehingga berjalan dengan harmonis.

Atribut Utama:

- _state: Enumerasi GameState.
- Referensi ke objek lain (player, enemies, dll.).
- Metode utama:
  - __init__(): Inisialisasi semua objek.
  - update(dt): Perbarui berdasarkan state (misalnya, update player dan musuh saat PLAYING).
  - draw(screen): Rendering berurutan: background, tilemap below, gem, enemy, player/companion, particles, tilemap above, UI.
  - handle_events(events): Proses input untuk transisi state.
  - Fitur: Mengelola transisi seperti dari MENU ke PLAYING, atau ke WIN saat portal diaktifkan.
- Mata bersinar kuning.

K. Sistem Refleksi Ending

Sebagai penutup permainan, ditambahkan sebuah Sistem Refleksi Ending yang muncul setelah pemain berhasil menyelesaikan seluruh misi dan mencapai akhir permainan. Sistem ini dirancang untuk memberikan pengalaman reflektif, bukan sekadar layar kemenangan biasa.

Ending Reflection ditampilkan dalam bentuk:

- Layar menghitam secara perlahan (fade out)
- Efek partikel cahaya yang bergerak pelan
- Teks reflektif yang merangkum perjalanan pemain dan pembelajaran OOP

Dengan adanya Refleksi Ending, game tidak hanya memberikan kepuasan visual, tetapi juga menutup pengalaman bermain dengan pesan edukatif dan naratif yang kuat, sehingga meninggalkan kesan mendalam bagi pemain.

BAB IV

CARA PENGGUNAAN APLIKASI

4.1 Kontrol

Pada permainan *Elion – The Last Lightkeeper*, kontrol dirancang sederhana, intuitif, dan responsif agar pemain dapat beradaptasi tanpa memerlukan tutorial panjang. Seluruh kontrol menggunakan keyboard, dan setiap aksi langsung dipetakan pada method tertentu di dalam kelas **Player**, sehingga interaksi pemain sepenuhnya terhubung dengan sistem OOP di dalam game.

1) Kontrol Gerakan

   Pemain dapat menggerakkan karakter utama ‘Elion’, ke empat arah menggunakan dua alternatif tombol: WASD atau Arrow Keys.

   Fungsi ini ditangani oleh method handle_input() pada class Player.

   - W / Arrow Up → Bergerak ke atas
   - S / Arrow Down → Bergerak ke bawah
   - A / Arrow Left → Bergerak ke kiri
   - D / Arrow Right → Bergerak ke kanan

   Gerakan dapat dikombinasikan (misalnya W + A), sehingga memungkinkan gerakan diagonal. Sistem juga melakukan normalisasi kecepatan saat diagonal, sehingga kecepatan tidak menjadi dua kali lebih cepat.

2) Kontrol Interaksi dan Menu

   - ENTER
     ⇒ Memulai permainan dari menu utama
     ⇒ Melanjutkan ke level berikutnya (jika ada)
     ⇒ Mengonfirmasi pilihan pada layar kemenangan (Win Screen)
   - ESC
     ⇒ Keluar dari permainan
     ⇒ Menutup aplikasi kapan saja
     ⇒ Tombol Enter dan ESC diproses pada fungsi handle_events() di kelas Game, yang menangani seluruh event input dari Pygame.

3) Kontrol Serangan

Saat sudah mencapai level 2 fitur pertempuran ditambahkan, pemain dapat melakukan serangan menggunakan:

- Space → Melakukan serangan Light Burst

  Fitur ini hanya aktif setelah pemain memperoleh kemampuan khusus pada level 2. Saat tombol ini ditekan, objek serangan dipanggil dan digunakan untuk mengurangi HP musuh atau menghentikan serangan tertentu.

4.2 Cara Menang

Pada permainan Elion – The Last Lightkeeper, mekanisme kemenangan dirancang untuk memberikan pengalaman eksplorasi, pemecahan masalah, serta penghindaran musuh yang menantang namun tetap mudah dipahami. Pemain tidak hanya bergerak dari titik awal menuju akhir, tetapi harus menyelesaikan serangkaian tujuan yang membentuk inti dari alur permainan.

Sistem kemenangan dalam game ini juga menunjukkan bagaimana logika permainan, deteksi tabrakan, state machine, dan manajemen objek saling terintegrasi melalui konsep OOP yang digunakan. Dengan demikian, cara menang dalam game bukan hanya sekadar "tujuan akhir permainan", tetapi merupakan representasi dari cara seluruh sistem bekerja secara harmonis menggunakan teknik pemrograman berorientasi objek.

Untuk menyelesaikan permainan dan mencapai kondisi WIN, pemain harus melalui beberapa tahapan utama mulai dari eksplorasi, pengumpulan item penting, menghindari ancaman, hingga akhirnya berhasil mengaktifkan portal suci. Deskripsi lengkap dari mekanisme kemenangan tersebut dijelaskan sebagai berikut:

4.2.1 Mengumpulkan 3 Spirit Gems

Tahapan pertama menuju kemenangan adalah mengumpulkan **tiga Spirit Gems** yang tersebar di dalam area permainan. Setiap gem memiliki warna, lokasi, dan efek visual yang berbeda sebagai penanda bahwa item tersebut merupakan objek penting.

a) Mekanik Pengumpulan

Pemain mengendalikan karakter Elion menggunakan input keyboard (WASD atau Arrow Keys) untuk mendekati gem. Deteksi interaksi dilakukan melalui pemeriksaan tabrakan antara bounding box pemain (pygame.Rect dari Player) dan bounding box gem (pygame.Rect dari Gem). Saat tabrakan terdeteksi dalam metode update() kelas Game, sistem memanggil:

- gem.collect(): Menandai gem sebagai tidak aktif dan menghapusnya dari daftar gem aktif di Game._gems menggunakan operasi list removal (misalnya, self._gems.remove(gem)).
- player.collect_gem(gem_type, gem_color): Menambahkan gem ke inventori pemain (_inventory.append(gem_type)), meningkatkan skor (_score += GEM_SCORE_VALUE, di mana GEM_SCORE_VALUE adalah konstanta global seperti 100), dan memicu efek visual seperti aura glow pada pemain.

Proses ini memastikan irreversibilitas pengumpulan, menghindari duplikasi, dan mempertahankan integritas state permainan.

b) Tanda Visual Bahwa Gem sudah Diambil

Setelah pengumpulan, gem dihilangkan secara instan dari loop rendering di Game.draw() untuk mencegah redraw yang tidak perlu. Efek visual pasca-pengumpulan mencakup:

- Emisi partikel berwarna sesuai gem (misalnya, hijau untuk gem hijau) melalui ParticleSystem.emit() dengan parameter seperti life=30, color=gem_color.
- Companion (Moku) memicu hint dialog melalui companion.give_hint(), menampilkan teks seperti “Nice! 2 more to go!” atau “All gems collected! Find the portal!” di UI hint box, dengan durasi tampilan 5 detik.

Efek ini memberikan feedback langsung, meningkatkan rasa pencapaian pemain.

c) Feedback HUD

Antarmuka pengguna (HUD) secara dinamis memperbarui progres melalui UI.draw_hud(), menampilkan indikator seperti:

- “Gems: 1/3” (untuk satu gem terkumpul).
- “Gems: 2/3” (untuk dua gem).
- “Gems: 3/3” (untuk lengkap), dengan teks berwarna hijau untuk menekankan kesuksesan.
- Pembaruan ini dilakukan setiap frame berdasarkan player.get_gem_count(), memungkinkan pemantauan real-time tanpa mengganggu gameplay.

d) Implementasi OOP di Balik Mekanik Ini

Setiap gem diinstansiasi sebagai objek dari kelas Gem, dengan atribut seperti _type (misalnya, 'hijau', 'biru', 'kuning') dan metode collect() yang mengenkapsulasi logika internal. Interaksi dengan pemain menunjukkan polimorfisme, di mana collect_gem() pada Player menangani berbagai jenis gem secara seragam. Daftar gem dikelola dalam Game._gems sebagai list objek, dengan penghapusan menggunakan list comprehension untuk efisiensi (misalnya, self._gems = [g for g in self._gems if not g.is_collected()]). Tahapan ini tidak hanya mempromosikan eksplorasi tetapi juga mengintegrasikan risiko dari musuh, memaksa pemain untuk merencanakan rute secara taktis.

4.2.2 Menghindari Musuh dan Bertahan Hidup

Tahapan ini menekankan elemen survival, di mana pemain harus menavigasi ancaman musuh sambil melanjutkan pengumpulan gem. Desain ini meningkatkan ketegangan permainan, memastikan bahwa kemenangan memerlukan keterampilan adaptasi dan pengamatan lingkungan.

a) Jenis-jenis Musuh

- **GlimpEnemy** (Musuh Patroli): Bergerak mengikuti waypoint yang didefinisikan dalam atribut _waypoints (list koordinat), dengan animasi wobble (perubahan skala sprite secara sinusoidal). Deteksi pemain memicu mode alert melalui take_action(), mengubah kecepatan menjadi lebih tinggi.
- **UmbraEnemy** (Musuh Pengejar): Fokus pada pursuit, menghitung vektor ke posisi pemain jika dalam aggro range (misalnya, 300 pixel), dengan efek smoke trail via ParticleSystem dan mata menyala (perubahan warna sprite).

b) Cara Musuh Mengancam Pemain

Tabrakan antara rect musuh dan rect pemain memicu player.take_damage(), yang mengurangi _lives (awalnya 3), mengatur _invincible_timer ke nilai positif (misalnya, 2 detik), dan memanggil Camera.shake(intensity=5, duration=0.5). Efek visual termasuk partikel merah (ParticleSystem.emit(color=(255,0,0))).

c) Konsekuensi Kalah

Jika _lives == 0, state bertransisi ke GameState.GAMEOVER dalam Game.update(), menghentikan semua update gameplay dan menampilkan layar

game over melalui UI.draw_gameover(). Pemain diarahkan kembali ke menu utama via input ENTER, memerlukan restart dari awal untuk mempertahankan tantangan.

d) Cara Bertahan Hidup

Strategi meliputi menghindari waypoint musuh, memanfaatkan objek map (seperti pohon di TileMap) sebagai penghalang visual, pergerakan zig-zag untuk mengelabui UmbraEnemy, dan prioritas pengumpulan gem cepat untuk meminimalkan eksposur. Tidak ada mekanisme serangan; fokus pada evasion.

e) Implementasi OOP

Kelas Enemy sebagai parent menyediakan metode take_action() yang dioverride di subclass (GlimpEnemy untuk patrol, UmbraEnemy untuk chase), menunjukkan polimorfisme. Enkapsulasi pada Player melindungi _lives dan _invincible_timer, diakses via getter. Tahapan ini memastikan keseimbangan antara progres dan risiko, mencegah kemenangan yang terlalu mudah.

4.2.3 Mengaktifkan dan Masuk ke Portal

Setelah seluruh gem terkumpul, pemain dapat mencari dan mendekati portal untuk memicu transisi menuju kemenangan.

a) Portal Awalnya Non-Aktif

Dalam kondisi default (_active = False), portal dirender dengan pulse brightness lambat (menggunakan math.sin(time.time()) untuk animasi) dan tanpa partikel intens. Interaksi diblokir.

b) Portal Menjadi Aktif

Saat player.get_gem_count() == 3 dicek dalam Game.update(), _active = True, meningkatkan kecepatan pulse, intensitas cahaya (alpha blending), dan emisi partikel cyan via ParticleSystem. Companion memicu hint seperti “All gems collected! The gate is waiting!” melalui give_hint().

c) Cara Masuk ke Portal

Tabrakan portal.get_rect().colliderect(player.get_rect()) memicu transisi ke WIN state jika aktif.

d) Efek Visual Setelah Masuk

Memicu shake kamera intens (Camera.shake(intensity=10, duration=1.0)), emisi puluhan partikel dari pusat portal, dan penghentian input pemain. Jika BGM diimplementasikan (via pygame.mixer), musik berubah ke tema kemenangan. Portal berfungsi sebagai trigger visual untuk ending.

4.2.4 Memicu WIN State

Ketika pemain berhasil memasuki portal setelah memenuhi semua syarat (3 gem lengkap), game memasuki WIN State, yaitu kondisi akhir yang menandakan bahwa pemain telah menyelesaikan level/seluruh permainan.

a) Perubahan State

Kode mengubah self._state = GameState.WIN dalam Game.update(), menghentikan update musuh dan pemain, serta memicu animasi partikel khusus (misalnya, confetti-like effects).

b) Tampilan Layar Kemenangan

Pada WIN State, layar permainan menampilkan Win Screen yang dirancang sebagai klimaks visual dan naratif. Elemen yang ditampilkan meliputi:

- Judul kemenangan (misalnya “Light Restored” atau “Victory”)
- Statistik permainan, seperti waktu penyelesaian dan skor akhir
- Efek partikel cahaya yang melambangkan keberhasilan

Seluruh elemen ini digambar melalui kelas UI menggunakan method khusus seperti draw_win(), yang terpisah dari HUD biasa untuk menjaga keteraturan kode.

c) Refleksi Ending sebagai Penutup Edukatif

Refleksi Ending yaitu layar refleksi dengan latar gelap dan partikel cahaya yang perlahan memudar. Pada layar ini, pemain disajikan pesan naratif dan edukatif, yaitu:

- Refleksi perjalanan Elion sebagai Lightkeeper
- Ringkasan konsep OOP yang telah dipelajari selama permainan
- Keterkaitan antara perjalanan karakter dan struktur kelas dalam program

Ending Reflection berfungsi sebagai penutup pengalaman bermain sekaligus penguatan tujuan edukatif game, sehingga pemain tidak hanya merasakan kepuasan menyelesaikan permainan, tetapi juga memahami nilai pembelajaran di baliknya.

d) Menutup Permainan

Setelah Refleksi Ending selesai, pemain dapat menekan tombol ENTER untuk kembali ke menu utama. Mekanisme ini memungkinkan pemain untuk:

- Memainkan ulang game
- Mengulang pembelajaran melalui Codex
- Mencoba strategi permainan yang berbeda

Dengan demikian, WIN State tidak hanya menandai akhir permainan, tetapi juga membuka peluang replay dan refleksi, selaras dengan tujuan game sebagai media pembelajaran interaktif berbasis OOP.

4.3 Alur Gameplay dan Edukasi

Game Elion – The Last Lightkeeper tidak hanya dirancang sebagai permainan hiburan berbasis petualangan 2D, tetapi juga sebagai media pembelajaran kontekstual yang memperkenalkan konsep Object-Oriented Programming (OOP) secara implisit dan naratif. Integrasi unsur edukasi dilakukan tanpa memutus alur permainan (non-intrusive learning), sehingga pemain tetap merasa bermain game, bukan membaca materi teori secara kaku.

Pendekatan edukasi ini diwujudkan melalui tiga mekanisme utama, yaitu World Map interaktif, Codex edukasi, dan Companion sebagai mentor spiritual, yang seluruhnya terintegrasi ke dalam sistem gameplay dan state machine game.

4.3.1 World Map sebagai Tahap Pra-Gameplay

Sebelum pemain memasuki gameplay utama, sistem game menampilkan World Map, yaitu peta dunia yang merepresentasikan perjalanan Elion menuju istana cahaya (Lightkeeper Castle). World Map ini terdiri dari beberapa node lokasi yang saling terhubung, antara lain:

- Spirit Meadow (Level 1 – Forest Area)
- Crimson Mountain (Level 2 – Dangerous Zone)
- Lightkeeper Castle (Level 3 – Final Area)

Setiap node pada World Map dapat diklik oleh pemain menggunakan input keyboard atau mouse. Namun, node gameplay tidak langsung dapat dimainkan sebelum pemain menyelesaikan tahap edukasi yang menyertainya.

Secara teknis, World Map diatur sebagai state tersendiri (GameState.WORLD_MAP) yang memisahkan fase edukasi dari fase gameplay, sehingga struktur state game tetap rapi dan terenkapsulasi dengan baik.

4.3.2 Codex Edukasi pada Setiap Lokasi

Setiap lokasi pada World Map memiliki Codex Panel, yaitu panel informasi yang berisi materi edukasi terkait konsep OOP yang digunakan di dalam game. Codex ini ditampilkan dalam bentuk panel UI semi-transparan dengan teks singkat, ilustrasi ikon, dan penjelasan kontekstual yang mudah dipahami.

Isi Codex per lokasi:

- Spirit Meadow

  Menjelaskan konsep Encapsulation, seperti bagaimana atribut Player disembunyikan (_x, _y, _lives) dan hanya dapat diakses melalui method getter.
- Crimson Mountain

  Menjelaskan konsep Inheritance, dengan contoh kelas Enemy sebagai parent class dan GlimpEnemy serta UmbraEnemy sebagai child class.
- Lightkeeper Castle

  Menjelaskan konsep Polymorphism, khususnya bagaimana method take_action() memiliki perilaku berbeda tergantung tipe musuh, meskipun dipanggil dengan cara yang sama.

Pemain diwajibkan membaca seluruh Codex di setiap lokasi sebelum tombol "Lanjutkan Perjalanan" dapat diaktifkan. Mekanisme ini memastikan bahwa unsur edukasi benar-benar tersampaikan tanpa memaksa pemain membaca materi panjang di luar konteks permainan.

4.3.3 Peran Companion sebagai Mentor Edukatif

Di dalam gameplay utama, Companion (Moku) tidak hanya berfungsi sebagai karakter pendamping visual, tetapi juga berperan sebagai mentor spiritual yang memberikan edukasi ringan selama permainan berlangsung.

Companion akan:

- Memberikan hint naratif ketika pemain mengambil gem
- Mengingatkan tujuan permainan
- Menyampaikan refleksi singkat terkait mekanik game yang baru saja dialami pemain

Sebagai contoh, setelah pemain berhasil mengumpulkan gem, Companion dapat menampilkan dialog seperti:

"Seperti class yang menyimpan data dengan aman, gem ini juga tersimpan rapi dalam inventori milikmu."

Dengan pendekatan ini, edukasi OOP disisipkan secara natural melalui dialog, tanpa menghentikan kontrol pemain atau memunculkan layar pembelajaran terpisah.

4.3.4 Transisi ke Opening dan Gameplay Utama

Setelah seluruh Codex edukasi pada World Map dibaca, pemain menekan tombol konfirmasi untuk melanjutkan. Sistem kemudian menampilkan Opening Cutscene berupa teks sinematik singkat yang menjelaskan latar cerita Elion sebagai penjaga cahaya terakhir. Setelah opening selesai, state game berpindah ke GameState.PLAYING, dan gameplay utama dimulai. Transisi ini menandai bahwa pemain telah:

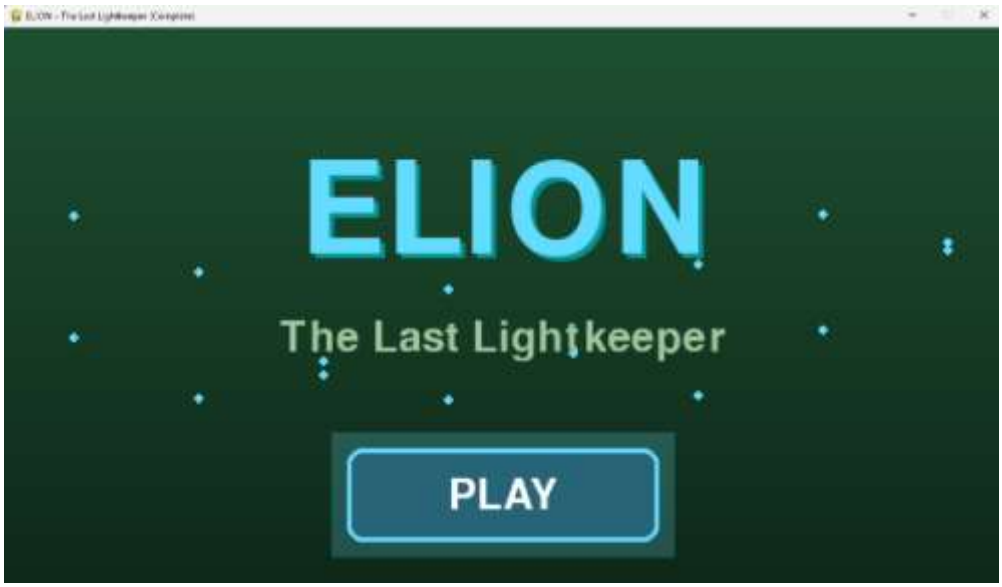
1. Memahami konteks cerita
2. Menerima materi edukasi OOP dasar
3. Siap untuk menerapkan pemahaman tersebut melalui pengalaman bermain

BAB V

DOKUMENTASI VISUAL APLIKASI

5.1 Tampilan Main Menu

Screenshot ini menunjukkan tampilan awal ketika pemain membuka permainan. Main Menu berfungsi sebagai gerbang utama sebelum permainan dimulai.



5.2 Tampilan World Map dan Codex Edukasi

Screenshot ini menunjukkan tampilan world map setelah pemain membuka permainan. Tujuannya untuk membuktikan adanya integrasi unsur edukasi OOP sebelum gameplay dimulai.


Spirit Forest – Menjaga Kekuatan Dalam
ENCAPSULATION
Sejak zaman kuno, Lightkeeper menyimpan cahaya di dalam dirinya.
Tidak semua orang boleh mengakses kekuatan tersebut.
Hanya ritual tertentu yang dapat membuka rahasia ini.
Di hutan ini, ELION belajar:
Menyimpan atribut dengan aman (_x, _inventory)
Mengakses melalui method khusus (get_position())
Melindungi data dari perubahan sembarangan
Tutup Codex
ENCAPSULATION:


Crimson Mountain – Warisan Kekuatan
INHERITANCE
Makhluk bayangan berasal dari sumber yang sama,
namun berkembang menjadi wujud berbeda.
Di gunung berapi ini, ELION menemukan:
Enemy sebagai kelas dasar
GlimpEnemy & UmbraEnemy sebagai turunan
Masing-masing mewarisi atribut dasar
Namun memiliki perilaku yang unik
Tutup Codex
INHERITANCE:


Lightkeeper Castle – Satu Panggilan, Banyak Bentuk
POLYMORPHISM
Satu perintah dapat menghasilkan tindakan berbeda
tergantung siapa yang menerimanya.
Di kastil akhir, ELION menyadari:
Method take_action() dipanggil sama
Hasil berbeda untuk tiap enemy
Fleksibilitas dalam merespon
Konsistensi dalam antarmuka
Tutup Codex
POLYMORPHISM:

5.3 Tampilan Player dan Companion

Screenshot ini menampilkan karakter utama Elion beserta companion spirit bernama Moku.

5.4 Tampilan Musuh GlimpEnemy

Screenshot ini menunjukkan musuh tipe Glimp Enemy, yaitu musuh yang bergerak secara patroli.

5.5 Tampilan Gem

Screenshot ini memperlihatkan Spirit Gem yang memancarkan efek cahaya (glow).

5.6 Tampilan Portal

Screenshot ini menunjukkan Portal yang menjadi gerbang akhir untuk menyelesaikan level.

5.7 Tampilan HUD

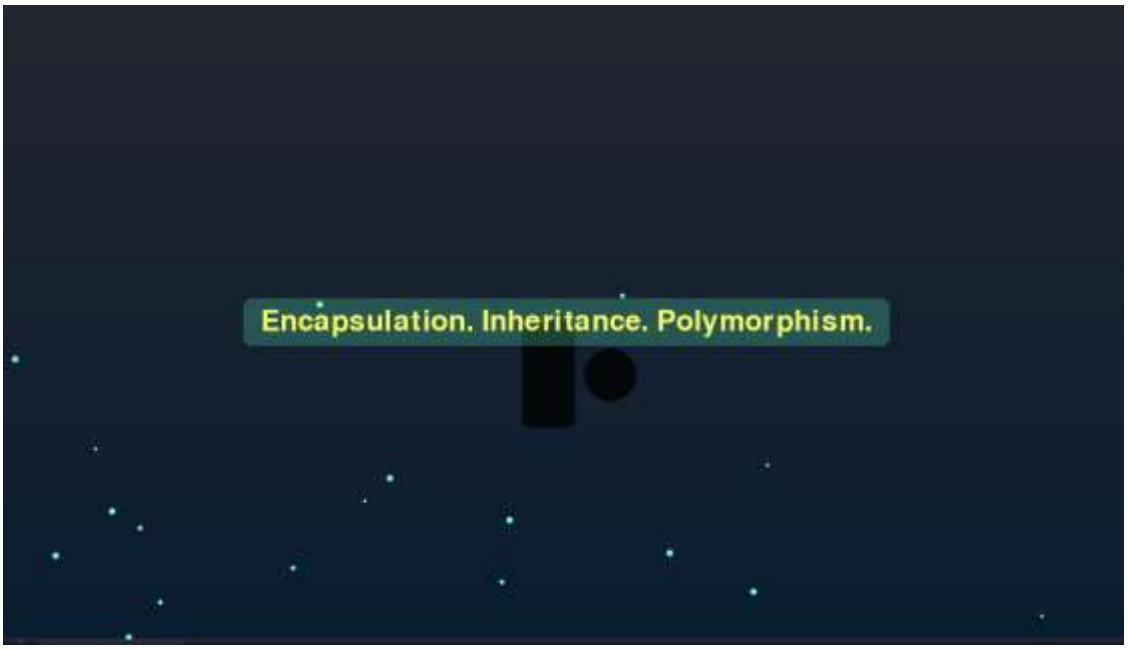
Screenshot HUD menampilkan antarmuka permainan yang selalu muncul saat gameplay berlangsung.

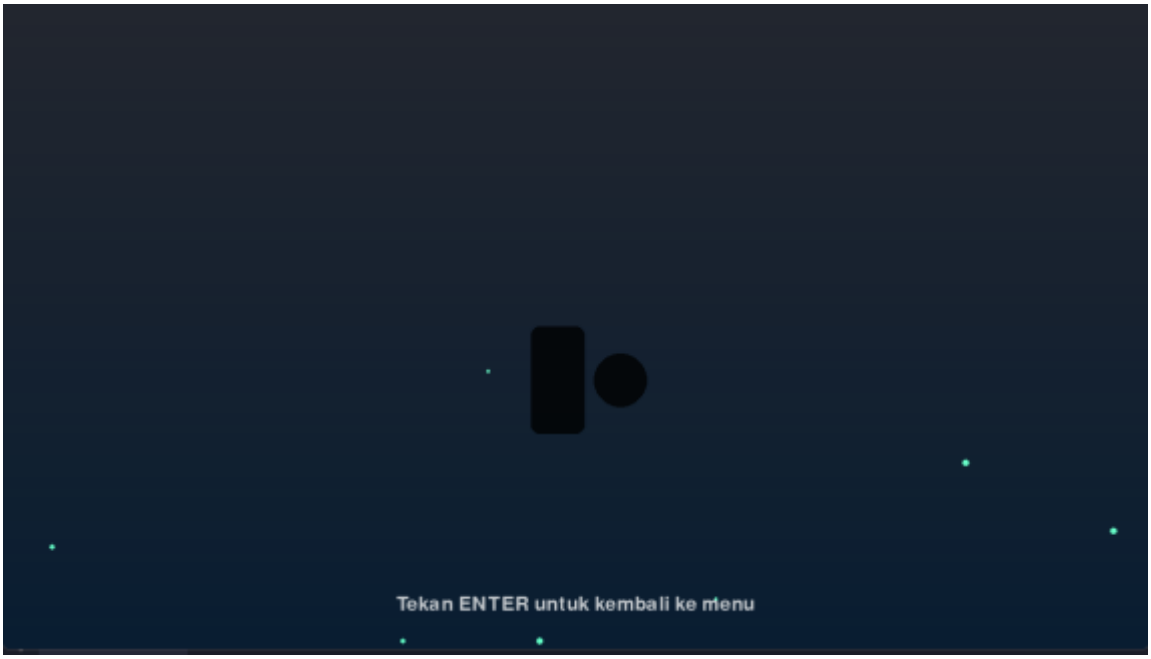
5.8 Tampilan Saat Menang

Screenshot ini menampilkan tampilan akhir ketika pemain berhasil menyelesaikan permainan.

5.9 Refleksi Ending

Screenshot ini menampilkan tampilan akhir untuk menunjukkan penutup permainan yang bersifat reflektif dan edukatif.


Encapsulation. Inheritance. Polymorphism.


Tekan ENTER untuk kembali ke menu

BAB VI

KESIMPULAN & SARAN

6.1 Kesimpulan

Berdasarkan hasil perancangan dan implementasi yang telah dilakukan, dapat disimpulkan bahwa aplikasi game Elion – The Last Lightkeeper berhasil dikembangkan sesuai dengan tujuan awal proyek. Game ini memanfaatkan konsep Object-Oriented Programming (OOP) secara penuh melalui penggunaan encapsulation, inheritance, dan polymorphism pada kelas-kelas utama seperti Player, Enemy, GlimpEnemy, UmbraEnemy, Portal, Gem, dan sistem pendukung lainnya.

Struktur program yang modular dan terorganisasi memungkinkan setiap komponen game dapat bekerja secara independen namun tetap saling terintegrasi. Hal ini mempermudah proses pengembangan fitur, pengaturan animasi, kontrol karakter, logika musuh, hingga transisi state game seperti menu, gameplay, game over, dan win screen.

Game dapat dijalankan dengan baik, menampilkan antarmuka yang jelas, mekanisme permainan yang berfungsi sesuai perancangan, serta seluruh fitur inti yang bekerja stabil. Dengan demikian, proyek ini telah menunjukkan keberhasilan dalam menerapkan konsep dasar pemrograman berorientasi objek ke dalam sebuah aplikasi game 2D berbasis Pygame.

6.2 Saran Pengembangan

Meskipun game sudah berjalan dengan baik, masih terdapat beberapa aspek yang dapat dikembangkan lebih lanjut untuk meningkatkan kualitas gameplay maupun sistem aplikasinya. Saran pengembangan ke depan antara lain sebagai berikut:

1. Penambahan Level Tambahan

   Game dapat diperluas menjadi beberapa level dengan tingkat kesulitan bertahap, termasuk area baru, variasi musuh, atau mekanisme puzzle yang lebih kompleks.
2. Sistem Pertarungan Lebih Lengkap

   Saat ini kemampuan serangan masih sederhana (hanya pada level lanjutan). Di masa depan dapat ditambahkan:
   - skill charge
   - kombo serangan
   - sistem stamina

3. Penambahan Audio, Musik Latar, dan Efek Suara

   Untuk meningkatkan suasana permainan, game dapat dilengkapi BGM, ambience background, dan SFX yang lebih bervariasi.
4. Meningkatkan Animasi dan Efek Visual

   Sprite dapat diperluas menjadi animasi berjalan lengkap, animasi idle, animasi hit, serta efek partikel yang lebih dinamis.
5. Sistem Save–Load Game

   Pemain dapat menyimpan progres dan melanjutkan permainan di lain waktu.
6. Kompatibilitas ke Platform Lain

   Game dapat dikembangkan untuk Android atau Web menggunakan library tambahan seperti Kivy atau pyscript.
7. Optimasi Performa dan Pengaturan Grafis

   Menambahkan opsi pengaturan kualitas grafis, resolusi game window, serta optimasi loop render untuk perangkat berspesifikasi rendah.
8. Dengan adanya pengembangan lebih lanjut, game ini berpotensi menjadi proyek yang lebih besar, menarik, dan layak dipresentasikan sebagai karya kreatif dalam bidang pemrograman dan game development.

6.3 Tantangan dan Kendala Pengembangan

Selama proses pengembangan game **Elion – The Last Lightkeeper**, terdapat beberapa tantangan teknis dan konseptual yang dihadapi. Tantangan-tantangan ini menjadi bagian penting dalam proses pembelajaran, khususnya dalam menerapkan konsep Object-Oriented Programming (OOP) secara nyata dalam sebuah aplikasi game interaktif. Berikut adalah beberapa kendala utama yang dihadapi selama pengembangan:

1) Perancangan Struktur Kelas yang Kompleks dalam Satu File

Salah satu tantangan utama adalah mengelola banyak kelas OOP dalam satu file Python (elion_pygame.py). Game ini terdiri dari berbagai sistem seperti Player, Enemy, Companion, ParticleSystem, Camera, TileMap, UI, hingga Game Controller.

Menjaga agar setiap kelas tetap memiliki tanggung jawab yang jelas dan tidak saling tumpang tindih membutuhkan perencanaan desain yang matang. Tantangan ini diatasi dengan menerapkan prinsip Single Responsibility Principle (SRP), di mana setiap kelas hanya fokus pada satu peran utama, meskipun berada dalam satu file yang sama.

2) **Implementasi Encapsulation pada Game Real-Time**

Dalam game real-time, banyak data seperti posisi pemain, jumlah nyawa, dan status inventory terus berubah setiap frame. Tantangan yang muncul adalah menjaga agar atribut-atribut tersebut tetap terlindungi dan tidak dimodifikasi secara langsung dari luar kelas.

Solusi yang diterapkan adalah penggunaan atribut privat (dengan awalan underscore) seperti _x, _lives, dan _inventory, serta menyediakan method getter dan method khusus seperti take_damage() dan collect_gem() untuk mengubah state internal secara aman. Pendekatan ini memastikan prinsip encapsulation tetap terjaga tanpa mengorbankan performa game.

3) **Pengaturan Perilaku Musuh Menggunakan Polymorphism**

Mengimplementasikan musuh dengan perilaku berbeda namun memiliki antarmuka yang sama menjadi tantangan tersendiri. Musuh patroli dan musuh pengejar harus dapat diproses dalam satu loop tanpa kondisi if-else yang kompleks.

Tantangan ini diselesaikan dengan menerapkan polymorphism, di mana semua musuh mewarisi kelas Enemy dan meng-override method take_action(). Dengan cara ini, game dapat memanggil enemy.take_action() secara seragam, sementara perilaku aktual ditentukan oleh subclass masing-masing.

4) **Sinkronisasi Gameplay, Visual Effect, dan Kamera**

Menggabungkan logika gameplay (collision, damage, win condition) dengan efek visual seperti partikel, glow, dan camera shake menjadi tantangan tersendiri. Kesalahan sinkronisasi dapat menyebabkan efek muncul tidak tepat waktu atau berlebihan.

Masalah ini diatasi dengan memusatkan pemanggilan efek visual pada event tertentu (misalnya saat gem diambil atau pemain terkena serangan), serta mengatur durasi dan intensitas efek menggunakan parameter yang terkontrol di dalam class ParticleSystem dan Camera.

5) **Menjaga Keseimbangan Gameplay (Balancing)**

Menentukan nilai kecepatan pemain, kecepatan musuh, jumlah nyawa, serta jarak deteksi musuh merupakan tantangan dalam menciptakan gameplay yang tidak terlalu

mudah maupun terlalu sulit.

Proses ini dilakukan melalui iterasi dan pengujian berulang, dengan menyesuaikan parameter seperti player_speed, enemy_speed, dan invincible_timer hingga diperoleh pengalaman bermain yang seimbang dan menyenangkan.

6) **Integrasi Unsur Edukasi Tanpa Mengganggu Gameplay**

Menambahkan elemen edukasi OOP ke dalam game tanpa mengurangi kenyamanan bermain juga menjadi tantangan. Jika edukasi disajikan secara eksplisit, gameplay dapat terasa terputus.

Solusinya adalah menjadikan Companion (Moku) sebagai mentor spiritual yang memberikan hint dan informasi OOP secara kontekstual, serta penggunaan Codex Panel yang bersifat opsional sebelum permainan dimulai. Dengan pendekatan ini, edukasi dapat disampaikan secara halus dan tidak memaksa.

7) **Pengujian dan Debugging**

Karena game berjalan dalam loop real-time, bug seperti collision ganda, state yang tidak berpindah, atau objek yang tidak terhapus dengan benar cukup sering terjadi. Debugging dilakukan dengan memanfaatkan print log, visual bounding box sementara, serta pengujian tiap fitur secara bertahap sebelum digabungkan ke sistem utama.

Kesimpulan Tantangan

Tantangan-tantangan tersebut memberikan pengalaman berharga dalam mengembangkan aplikasi berbasis konsep OOP secara teoritis, tetapi juga bagaimana mengaplikasikannya dalam system yang kompleks, dinamis, dan interaktif seperti game.