

LAPORAN UAS PEMROGRAMAN BERORIENTASI OBJEK

“Skyline Defender”



Disusun oleh :

Andhika Abdilah Prasetyo (23091397046)

Danu Prasetya (23091397061)

Ariel Pramudya Risky H. (23091397069)

Dosen Pengampu :

I Gde Agung Sri Sidhimantra, S.Kom., M.Kom.

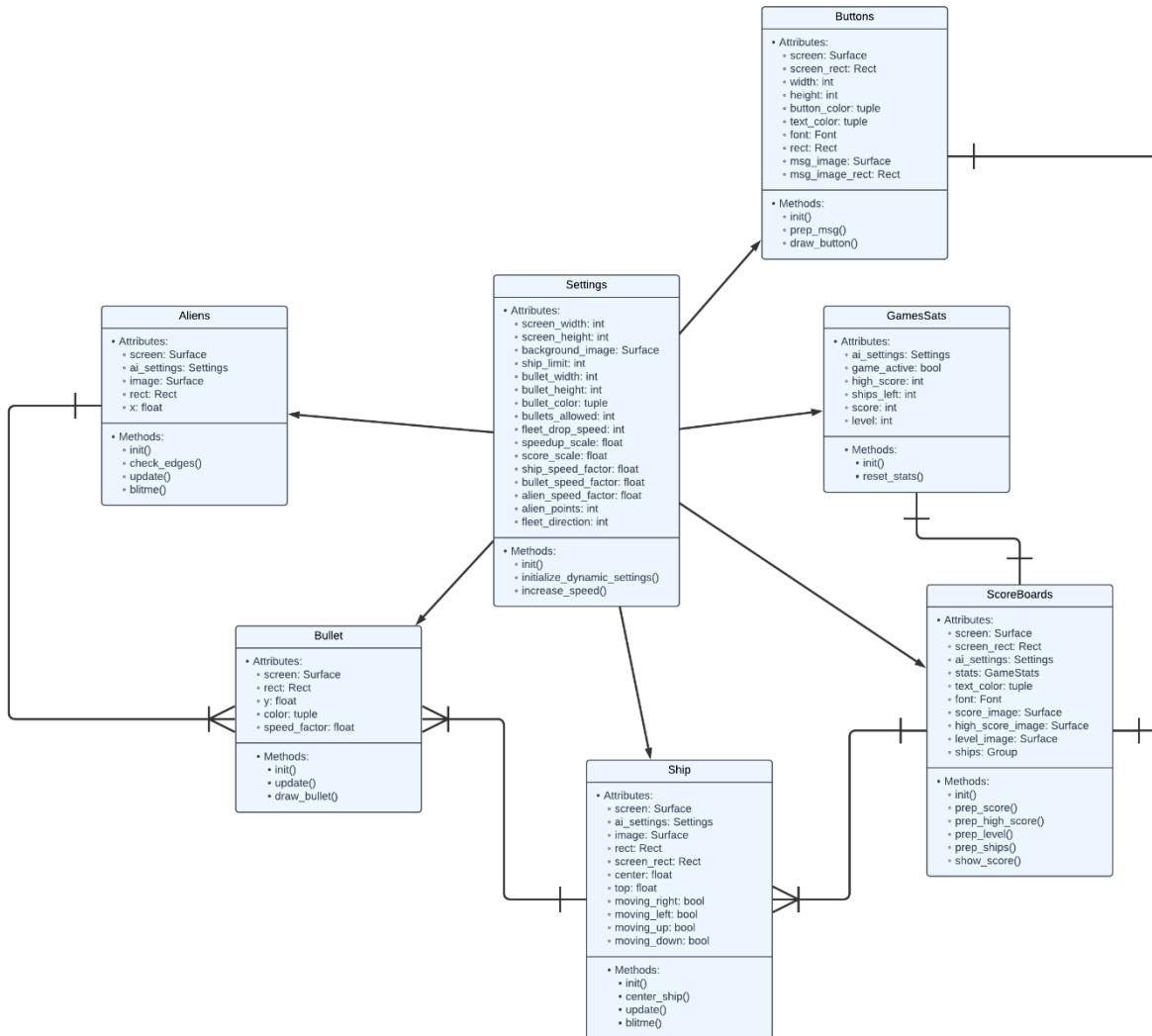
Binti Kholidah, S.Kom., M.Tr.Kom.

Dimas Novian Aditia Syahputra, S.Tr.T., M.Tr.T.

Moch Deny Pratama, S.Tr.Kom., M.Kom.

**PRODI D4 MANAJEMEN INFORMATIKA
FAKULTAS VOKASI
UNIVERSITAS NEGERI SURABAYA
2024**

CLASS DIAGRAM



Kelas:

1. Aliens

- Atribut:**

- screen: Referensi ke layar (surface) tempat alien ditampilkan.
- ai_settings: Objek pengaturan permainan.
- image: Gambar alien.
- rect: Posisi dan dimensi alien.
- x: Posisi horizontal alien.

- Metode:**

- `__init__()`: Konstruktor untuk inisialisasi atribut alien.
- `check_edges()`: Mengecek apakah alien menyentuh tepi layar.

- update(): Mengupdate posisi alien.
- blitme(): Menampilkan alien di layar.

2. Settings

- **Atribut:**
 - Berbagai pengaturan permainan, seperti dimensi layar, kecepatan objek, warna peluru, dan arah pergerakan.
- **Metode:**
 - __init__(): Inisialisasi atribut pengaturan awal.
 - initialize_dynamic_settings(): Mengatur ulang pengaturan dinamis saat permainan dimulai ulang.
 - increase_speed(): Meningkatkan kecepatan objek untuk menambah tantangan permainan.

3. Bullet

- **Atribut:**
 - screen: Referensi ke layar tempat peluru ditampilkan.
 - rect: Posisi dan dimensi peluru.
 - x dan y: Koordinat posisi peluru.
 - color: Warna peluru.
 - speed_factor: Kecepatan peluru.
- **Metode:**
 - __init__(): Konstruktor untuk inisialisasi peluru.
 - update(): Mengupdate posisi peluru (biasanya bergerak ke atas).
 - draw_bullet(): Menampilkan peluru di layar.

4. Ship

- **Atribut:**
 - screen: Referensi ke layar tempat kapal ditampilkan.
 - ai_settings: Objek pengaturan permainan.
 - image: Gambar kapal.
 - rect: Posisi dan dimensi kapal.
 - moving_right, moving_left, moving_up, moving_down: Status apakah kapal bergerak ke arah tertentu.
- **Metode:**

- `__init__()`: Konstruktor untuk inisialisasi kapal.
- `center_ship()`: Mengembalikan kapal ke posisi awal (tengah bawah layar).
- `update()`: Mengupdate posisi kapal berdasarkan input pengguna.
- `blitme()`: Menampilkan kapal di layar.

5. Buttons

- **Atribut:**
 - Elemen-elemen tombol seperti ukuran, warna, font, dan pesan.
- **Metode:**
 - `__init__()`: Inisialisasi tombol.
 - `prep_msg()`: Menyiapkan pesan untuk ditampilkan di tombol.
 - `draw_button()`: Menampilkan tombol di layar.

6. GameStats

- **Atribut:**
 - `ai_settings`: Referensi ke pengaturan permainan.
 - `game_active`: Status apakah permainan sedang berlangsung.
 - `high_score`: Skor tertinggi.
 - `ships_left`: Jumlah nyawa yang tersisa.
 - `score`: Skor saat ini.
 - `level`: Level permainan saat ini.
- **Metode:**
 - `__init__()`: Inisialisasi atribut statistik permainan.
 - `reset_stats()`: Mengatur ulang statistik permainan saat dimulai ulang.

7. Scoreboards

- **Atribut:**
 - Elemen-elemen untuk menampilkan skor, level, dan jumlah kapal tersisa.
- **Metode:**
 - `prep_score()`: Menyiapkan tampilan skor.
 - `prep_high_score()`: Menyiapkan tampilan skor tertinggi.
 - `prep_level()`: Menyiapkan tampilan level.
 - `prep_ships()`: Menyiapkan tampilan jumlah kapal tersisa.

- `show_score()`: Menampilkan skor dan informasi lainnya di layar.

Hubungan antar kelas:

1. Settings dan Kelas Lain

Relasi: Agregasi

- Kelas **Settings** digunakan oleh hampir semua kelas lainnya (seperti Ship, Aliens, Bullet, GameStats, dan Scoreboards) untuk menyediakan konfigurasi dan parameter permainan.
- **Detail relasi:**
 - Kelas **Ship**, **Aliens**, dan **Bullet** membutuhkan atribut dalam **Settings** untuk menentukan kecepatan, warna, ukuran, dan properti lainnya.
 - **GameStats** memanfaatkan **Settings** untuk mengetahui batas nyawa kapal (`ship_limit`).
 - **Scoreboards** menggunakan **Settings** untuk mengatur format dan gaya tampilan.

2. Aliens dan Settings

Relasi: Dependency

- Kelas **Aliens** bergantung pada **Settings** untuk mendapatkan kecepatan alien (`alien_speed_factor`), arah gerak alien (`fleet_direction`), dan pengaturan lainnya.
- Relasi ini memungkinkan alien untuk diperbarui (melalui metode `update()`) dan mengecek apakah alien menyentuh tepi layar (dengan `check_edges()`).

3. Bullet dan Settings

Relasi: Dependency

- Kelas **Bullet** menggunakan **Settings** untuk menentukan kecepatan peluru (`bullet_speed_factor`), warna (`bullet_color`), dan ukuran (`bullet_width` dan `bullet_height`).
- Relasi ini memastikan peluru dapat dirender dan bergerak sesuai dengan konfigurasi permainan.

4. Ship dan Settings

Relasi: Dependency

- **Ship** menggunakan **Settings** untuk menentukan batas jumlah nyawa kapal (`ship_limit`) dan kecepatan pergerakan (`ship_speed_factor`).
- Kapal ini juga membutuhkan pengaturan layar (`screen_width` dan `screen_height`) untuk menjaga posisi tetap dalam batas layar.

5. GameStats dan Settings

Relasi: Aggregasi

- Kelas **GameStats** membutuhkan objek **Settings** untuk mengakses pengaturan statis seperti jumlah nyawa kapal (`ship_limit`) atau skala kecepatan peningkatan level (`speedup_scale`).

6. Scoreboards dan GameStats

Relasi: Aggregasi

- **Scoreboards** mengambil data dari **GameStats** untuk menampilkan skor saat ini, skor tertinggi, level, dan jumlah kapal yang tersisa.
- Melalui relasi ini, **Scoreboards** dapat menggunakan metode seperti `prep_score()` untuk memperbarui informasi di layar.

7. Buttons

Relasi: Independen

- Kelas **Buttons** tidak memiliki relasi langsung dengan kelas lain. Tombol biasanya digunakan untuk memulai permainan (misalnya, tombol "Play"). Relasi bisa terjadi secara tidak langsung melalui interaksi pengguna dengan permainan.

8. Relasi Aliens dan Bullet

Relasi: Interaksi melalui logika permainan

- Tidak terlihat secara langsung dalam diagram, tetapi berdasarkan pola umum desain permainan:
 - Saat peluru (objek dari **Bullet**) mengenai alien (objek dari **Aliens**), logika permainan akan menghapus alien dan peluru, serta memperbarui skor dalam **GameStats**.

9. Relasi Ship dan Bullet

Relasi: Dependency (logika permainan)

- Kapal (**Ship**) menembakkan peluru (dari **Bullet**) berdasarkan input pengguna.
- Kapal bertanggung jawab membuat instance peluru dan menambahkannya ke dalam daftar peluru aktif.

10. Relasi GameStats dan Scoreboards

Relasi: Aggregasi

- **GameStats** menyimpan data terkait skor, level, dan jumlah nyawa. Data ini digunakan oleh **Scoreboards** untuk menampilkan informasi kepada pemain.

11. Ship dan Scoreboards

Relasi: Dependency

- Ketika kapal kehilangan nyawa, **Scoreboards** diperbarui untuk mencerminkan jumlah kapal yang tersisa di layar.

KONSEP OOP

Encapsulation

Encapsulation adalah salah satu prinsip dasar dalam pemrograman berorientasi objek (OOP) yang mengacu pada pengemasan data dan metode yang beroperasi pada data tersebut ke dalam satu unit, yaitu kelas. Ini membantu dalam menyembunyikan detail implementasi dan hanya mengekspos antarmuka yang diperlukan untuk berinteraksi dengan objek tersebut. Berikut adalah penjelasan tentang bagaimana encapsulation diterapkan dalam kode game Skyline Defender:

1. Kelas dan Objek

- **Definisi Kelas:** Dalam kode, terdapat beberapa kelas seperti **Ship**, **Alien**, **Bullet**, **Button**, **GameStats**, dan **Scoreboard**. Setiap kelas ini mendefinisikan atribut dan metode yang spesifik untuk objek yang mereka wakili.
- **Pembuatan Objek:** Objek dari kelas-kelas ini dibuat di dalam fungsi **run_game()**, seperti **ship = Ship(ai_settings, screen)** dan **bullets = Group()**. Ini menunjukkan bahwa setiap objek memiliki data dan perilaku yang terpisah.

2. Atribut Privat dan Publik

- **Atribut Privat:** Meskipun tidak ada atribut yang secara eksplisit ditandai sebagai privat (dengan awalan underscore), prinsip encapsulation tetap diterapkan dengan cara bahwa atribut yang penting untuk pengoperasian kelas tidak diakses langsung dari luar kelas. Misalnya, atribut seperti **self.screen**, **self.ai_settings**, dan **self.rect** dalam kelas **Ship** dan **Alien** tidak dapat diakses langsung dari luar kelas.
- **Metode Publik:** Metode seperti **update()**, **blitme()**, dan **draw_button()** adalah metode publik yang dapat diakses dari luar kelas. Ini memungkinkan interaksi dengan objek tanpa mengungkapkan detail implementasi internal.

3. Pengelolaan Status dan Logika Permainan

- **Kelas GameStats:** Kelas ini mengelola statistik permainan seperti skor dan jumlah kapal yang tersisa. Dengan menyimpan data ini dalam satu kelas, kita dapat mengontrol akses dan modifikasi data tersebut melalui metode yang disediakan, seperti **reset_stats()**.
- **Kelas Scoreboard:** Kelas ini bertanggung jawab untuk menampilkan informasi skor dan level. Dengan memisahkan logika tampilan dari logika permainan, kita menjaga agar kode tetap terorganisir dan mudah dikelola.

4. Interaksi Antar Kelas

- **Komposisi:** Kelas-kelas seperti **GameStats**, **Scoreboard**, dan **Button** berinteraksi dengan kelas lain (seperti **Ship** dan **Bullet**) melalui metode yang ditentukan.

Misalnya, `check_events()` dalam `game_functions.py` memanggil metode dari objek `ship` dan `bullets`, menunjukkan bagaimana objek dapat berinteraksi tanpa mengungkapkan detail internal mereka.

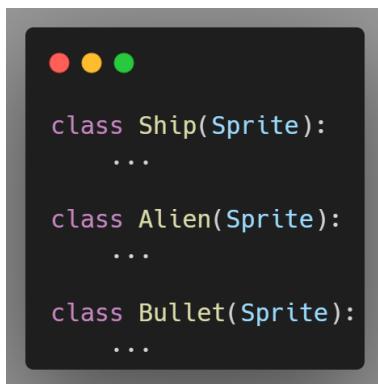
- **Penggunaan Parameter:** Metode dalam kelas sering kali menerima parameter yang diperlukan untuk berfungsi, seperti `ai_settings`, `screen`, dan objek lain. Ini memungkinkan fleksibilitas dan penggunaan kembali kode.

Inheritance

Inheritance (pewarisan) adalah salah satu prinsip dasar dalam pemrograman berorientasi objek (OOP) yang memungkinkan sebuah kelas (kelas anak) untuk mewarisi atribut dan metode dari kelas lain (kelas induk). Ini membantu dalam mengurangi duplikasi kode dan memfasilitasi penggunaan kembali kode. Berikut adalah penjelasan tentang bagaimana inheritance diterapkan dalam kode game Skyline Defender:

1. Penggunaan Kelas Induk dan Kelas Anak

- **Kelas Sprite:** Dalam kode, banyak kelas seperti `Ship`, `Alien`, dan `Bullet` mewarisi dari kelas `Sprite` yang disediakan oleh modul `pygame`. Dengan mewarisi dari `Sprite`, kelas-kelas ini mendapatkan semua atribut dan metode yang ada dalam kelas `Sprite`, seperti pengelolaan posisi dan gambar objek.



```
class Ship(Sprite):
    ...
class Alien(Sprite):
    ...
class Bullet(Sprite):
    ...
```

2. Memanfaatkan Metode dan Atribut dari Kelas Induk

- **Atribut dan Metode:** Dengan mewarisi dari `Sprite`, kelas-kelas seperti `Ship`, `Alien`, dan `Bullet` dapat menggunakan metode seperti `get_rect()` untuk mendapatkan ukuran dan posisi objek. Ini mengurangi kebutuhan untuk menulis ulang kode yang sama untuk setiap kelas.



```
self.rect = self.image.get_rect() # Menggunakan metode dari kelas Sprite
```

3. Penggunaan super() untuk Memanggil Kelas Induk

- **Inisialisasi Kelas Induk:** Dalam konstruktur (`__init__`) dari kelas anak, `super()` digunakan untuk memanggil konstruktur kelas induk. Ini memastikan

bahwa semua inisialisasi yang diperlukan dari kelas induk dilakukan sebelum menambahkan atribut spesifik kelas anak.

```
super(Alien, self).__init__() # Memanggil konstruktor kelas Sprite
```

4. Kelas yang Berfungsi Sebagai Template

- **Kelas Alien:** Kelas **Alien** berfungsi sebagai template untuk semua objek alien dalam permainan. Dengan menggunakan inheritance, kita dapat membuat variasi dari kelas ini jika diperlukan, misalnya, dengan membuat kelas **FastAlien** yang mewarisi dari **Alien** dan memiliki kecepatan yang lebih tinggi.

```
class FastAlien(Alien):  
    def __init__(self, ai_settings, screen):  
        super(FastAlien, self).__init__(ai_settings, screen)  
        self.ai_settings.alien_speed_factor *= 2 # Meningkatkan kecepatan
```

5. Memudahkan Ekstensi dan Pemeliharaan

- **Ekstensi Kode:** Dengan inheritance, pengembang dapat dengan mudah menambahkan fitur baru atau variasi objek tanpa mengubah kode yang sudah ada. Misalnya, jika ingin menambahkan jenis peluru baru, cukup membuat kelas baru yang mewarisi dari **Bullet**.

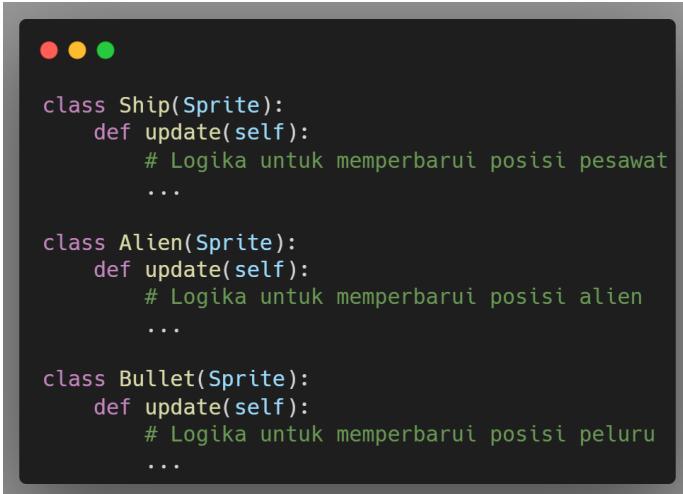
```
class ExplosiveBullet(Bullet):  
    def __init__(self, ai_settings, screen, ship):  
        super(ExplosiveBullet, self).__init__(ai_settings, screen, ship)  
        self.explosion_radius = 50 # Menambahkan atribut baru
```

Polymorphism

Polymorphism adalah salah satu prinsip dasar dalam pemrograman berorientasi objek (OOP) yang memungkinkan objek dari kelas yang berbeda untuk diperlakukan sebagai objek dari kelas yang sama melalui antarmuka yang sama. Ini memungkinkan metode yang sama untuk berfungsi dengan cara yang berbeda tergantung pada objek yang memanggilnya. Berikut adalah penjelasan tentang bagaimana polymorphism diterapkan dalam kode game Skyline Defender:

1. Metode dengan Nama yang Sama

- **Metode update():** Dalam kode, beberapa kelas seperti **Ship**, **Alien**, dan **Bullet** memiliki metode dengan nama yang sama, yaitu **update()**. Meskipun nama metode tersebut sama, implementasinya berbeda untuk setiap kelas.



```

● ● ●

class Ship(Sprite):
    def update(self):
        # Logika untuk memperbarui posisi pesawat
        ...

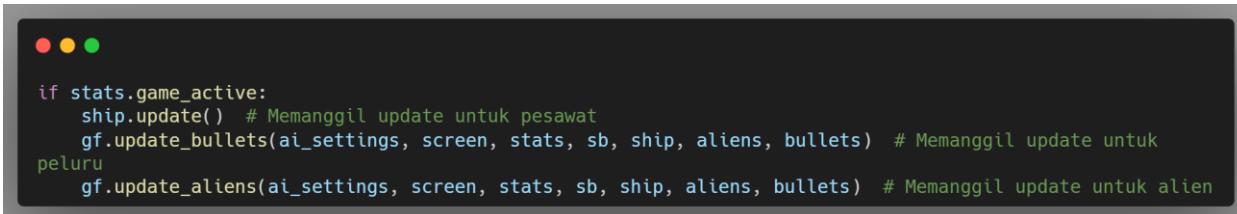
class Alien(Sprite):
    def update(self):
        # Logika untuk memperbarui posisi alien
        ...

class Bullet(Sprite):
    def update(self):
        # Logika untuk memperbarui posisi peluru
        ...

```

2. Memanggil Metode Secara Umum

- **Penggunaan dalam Loop Game:** Dalam fungsi **run_game()**, ketika memanggil **ship.update()**, **aliens.update()**, dan **bullets.update()**, kita tidak perlu mengetahui detail implementasi dari masing-masing kelas. Kita hanya memanggil metode **update()** dan setiap objek akan menjalankan logika yang sesuai dengan kelasnya.



```

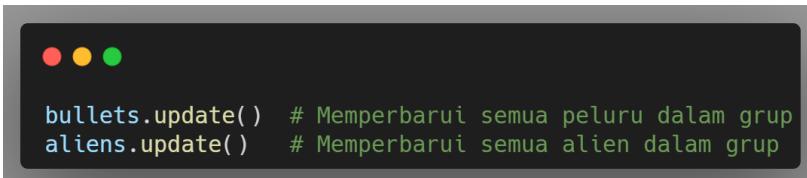
● ● ●

if stats.game_active:
    ship.update() # Memanggil update untuk pesawat
    gf.update_bullets(ai_settings, screen, stats, sb, ship, aliens, bullets) # Memanggil update untuk peluru
    gf.update.aliens(ai_settings, screen, stats, sb, ship, aliens, bullets) # Memanggil update untuk alien

```

3. Menggunakan Koleksi Objek

- **Pengelolaan Grup:** Dalam game, kita menggunakan grup untuk mengelola sekumpulan objek, seperti **bullets** dan **aliens**. Dengan menggunakan metode **update()** pada grup, kita dapat memperbarui semua objek dalam grup tersebut tanpa harus memanggil metode untuk setiap objek secara individual.



```

● ● ●

bullets.update() # Memperbarui semua peluru dalam grup
aliens.update() # Memperbarui semua alien dalam grup

```

4. Interaksi dengan Objek Berbeda

- **Tabrakan Peluru dan Alien:** Dalam fungsi **check_bullet_alien_collisions()**, kita memeriksa tabrakan antara peluru dan alien. Meskipun peluru dan alien adalah objek dari kelas yang berbeda, kita dapat menggunakan metode yang sama untuk memeriksa tabrakan dan memperbarui skor.

```
collisions = pygame.sprite.groupcollide(bullets, aliens, True, True)
```

5. Memudahkan Ekstensi Kode

- **Menambahkan Jenis Baru:** Dengan polymorphism, kita dapat dengan mudah menambahkan jenis objek baru yang memiliki metode yang sama. Misalnya, jika kita ingin menambahkan jenis alien baru dengan perilaku yang berbeda, kita cukup membuat kelas baru yang mewarisi dari **Alien** dan mengimplementasikan metode **update()** sesuai kebutuhan.

```
class FastAlien(Alien):  
    def update(self):  
        # Logika untuk memperbarui posisi alien cepat  
        ...
```

Abstraction

Abstraction (abstraksi) adalah salah satu prinsip dasar dalam pemrograman berorientasi objek (OOP) yang menyembunyikan detail implementasi dan hanya menampilkan fitur-fitur penting dari suatu objek. Ini membantu dalam mengurangi kompleksitas dan meningkatkan fokus pada interaksi antar objek. Berikut adalah penjelasan tentang bagaimana abstraction diterapkan dalam kode game Skyline Defender:

1. Penggunaan Kelas untuk Menyembunyikan Detail

- **Kelas yang Terpisah:** Dalam kode, terdapat beberapa kelas seperti **Ship**, **Alien**, **Bullet**, **Button**, dan **GameStats**. Setiap kelas ini menyembunyikan detail implementasi dari objek yang mereka wakili. Misalnya, kelas **Ship** mengelola semua logika yang berkaitan dengan pesawat, termasuk posisi, gambar, dan pergerakan, tanpa perlu mengungkapkan detail tersebut kepada pengguna kelas.

```
class Ship(Sprite):  
    def __init__(self, ai_settings, screen):  
        # Inisialisasi pesawat  
        ...  
  
    def update(self):  
        # Memperbarui posisi pesawat  
        ...  
  
    def blitme(self):  
        # Menampilkan pesawat di layar  
        ...
```

2. Antarmuka yang Jelas

- **Metode Publik:** Kelas-kelas tersebut menyediakan metode publik yang dapat digunakan untuk berinteraksi dengan objek. Misalnya, metode **update()** dan **blitme()** dalam kelas **Ship** memungkinkan pengguna untuk memperbarui posisi dan menampilkan pesawat tanpa perlu mengetahui bagaimana semua itu diimplementasikan.

```
● ● ●
ship.update() # Memperbarui posisi pesawat
ship.blitme() # Menampilkan pesawat di layar
```

3. Mengelola Logika Permainan

- **Kelas GameStats:** Kelas ini menyimpan dan mengelola statistik permainan seperti skor dan jumlah kapal yang tersisa. Dengan menyembunyikan detail tentang bagaimana statistik ini dikelola, pengguna kelas hanya perlu memanggil metode seperti **reset_stats()** untuk mengatur ulang statistik tanpa perlu memahami logika internalnya.

```
● ● ●
class GameStats:
    def reset_stats(self):
        # Mengatur ulang statistik permainan
        ...
```

4. Menggunakan Parameter untuk Fleksibilitas

- **Parameter dalam Metode:** Banyak metode dalam kelas menerima parameter yang memungkinkan fleksibilitas dalam penggunaannya. Misalnya, metode **draw_bullet()** dalam kelas **Bullet** menerima parameter untuk menentukan warna dan ukuran peluru, tetapi detail tentang bagaimana peluru digambar disembunyikan dari pengguna.

```
● ● ●
def draw_bullet(self):
    # Menampilkan peluru di layar
    pygame.draw.rect(self.screen, self.color, self.rect)
```

5. Memudahkan Ekstensi dan Pemeliharaan

- **Kelas yang Dapat Diperluas:** Dengan menggunakan abstraksi, pengembang dapat dengan mudah menambahkan fitur baru atau jenis objek baru tanpa mengubah kode yang ada. Misalnya, jika ingin menambahkan jenis alien baru, cukup membuat kelas baru yang mewarisi dari **Alien** dan mengimplementasikan metode yang diperlukan.



```
● ● ●

class FastAlien(Alien):
    def update(self):
        # Logika untuk memperbarui posisi alien cepat
        ...
```

PENJELASAN FITUR

Fitur-Fitur dalam Game Skyline Defender

Game Skyline Defender memiliki berbagai fitur yang dirancang untuk memberikan pengalaman bermain yang menarik dan interaktif. Berikut adalah penjelasan tentang fitur-fitur utama yang terdapat dalam game:

1. Pengaturan Game

- **Pengaturan Layar:** Game ini memiliki pengaturan untuk ukuran layar, yang ditentukan dalam kelas **Settings**. Ukuran layar diatur menjadi 1200x800 piksel, dan juga memiliki gambar latar belakang yang ditampilkan selama permainan.
- **Pengaturan Kapal dan Peluru:** Pengaturan untuk jumlah kapal yang dapat dimiliki pemain, ukuran peluru, dan warna peluru juga dikelola dalam kelas **Settings**.

2. Karakter dan Objek

- **Pesawat Pemain (Ship):** Pemain mengendalikan pesawat yang dapat bergerak ke atas, bawah, kiri, dan kanan. Pesawat ini memiliki gambar yang dimuat dari file dan dapat ditampilkan di layar.
- **Alien:** Alien adalah musuh dalam permainan yang bergerak secara horizontal dan dapat mencapai tepi layar. Jika alien mencapai tepi, mereka akan mengubah arah dan turun ke bawah.
- **Peluru:** Pemain dapat menembakkan peluru untuk menghancurkan alien. Peluru bergerak ke atas layar dan memiliki batas jumlah yang dapat ditembakkan sekaligus.

3. Interaksi dan Kontrol

- **Kontrol Pemain:** Pemain dapat menggerakkan pesawat menggunakan tombol panah (kiri, kanan, atas, bawah) dan menembakkan peluru dengan menekan tombol spasi. Pemain juga dapat keluar dari permainan dengan menekan tombol escape.
- **Tombol Play:** Terdapat tombol "Play" yang memungkinkan pemain untuk memulai permainan baru. Ketika tombol ini diklik, permainan akan direset dan siap untuk dimainkan.

4. Statistik dan Skor

- **Statistik Permainan:** Kelas **GameStats** mengelola statistik permainan, termasuk skor saat ini, skor tertinggi, level permainan, dan jumlah kapal yang tersisa. Statistik ini direset ketika permainan dimulai.

- **Tampilan Skor:** Kelas **Scoreboard** bertanggung jawab untuk menampilkan informasi skor, level, dan jumlah kapal yang tersisa di layar. Ini memberikan umpan balik visual kepada pemain tentang kemajuan mereka.

5. Level dan Kesulitan

- **Tingkat Kesulitan:** Game ini memiliki mekanisme untuk meningkatkan kesulitan seiring dengan kemajuan pemain. Ketika semua alien dihancurkan, kecepatan permainan akan meningkat, dan level akan bertambah.

6. Efek Suara

- **Suara Efek:** Game ini menggunakan efek suara untuk meningkatkan pengalaman bermain. Suara peluru dan ledakan dimainkan saat peluru ditembakkan dan saat alien dihancurkan.

7. Tabrakan dan Respons

- **Tabrakan:** Game ini memeriksa tabrakan antara peluru dan alien. Jika peluru mengenai alien, alien akan dihancurkan, dan pemain akan mendapatkan poin. Jika alien mencapai bagian bawah layar, pemain akan kehilangan satu kapal.

8. Pengelolaan Alien

- **Armada Alien:** Game ini menciptakan armada alien yang terdiri dari beberapa baris dan kolom. Armada ini dapat diperbarui dan direset ketika semua alien dihancurkan.

ALUR APLIKASI

Alur Aplikasi dalam Game Skyline Defender

Alur aplikasi dalam game Skyline Defender mengikuti serangkaian langkah yang terstruktur, mulai dari inisialisasi hingga interaksi pemain dan pengelolaan permainan. Berikut adalah penjelasan tentang alur aplikasi yang terdapat dalam game:

1. Inisialisasi Game

- **Inisialisasi Pygame:** Game dimulai dengan memanggil **pygame.init()**, yang menginisialisasi semua modul Pygame yang diperlukan untuk menjalankan game.
- **Pengaturan Game:** Objek **Settings** dibuat untuk mengatur parameter dasar game, seperti ukuran layar, jumlah kapal, dan pengaturan peluru.
- **Membuat Layar:** Layar game dibuat dengan ukuran yang ditentukan dalam pengaturan, dan judul jendela diatur menjadi "Skyline Defender".
- **Membuat Elemen Utama:** Elemen-elemen utama seperti tombol "Play", statistik permainan (**GameStats**), papan skor (**Scoreboard**), pesawat pemain (**Ship**), dan grup untuk peluru dan alien diinisialisasi.

2. Membuat Armada Alien

- **Pembuatan Armada:** Setelah inisialisasi, armada alien dibuat dengan memanggil fungsi `create_fleet()`, yang menghitung jumlah alien yang dapat dimuat dalam baris dan kolom berdasarkan ukuran layar dan ukuran alien.

3. Loop Utama Game

- **Pengelolaan Input Pemain:** Dalam loop utama, game terus memeriksa input dari pemain dengan memanggil fungsi `check_events()`. Ini mencakup penekanan tombol untuk menggerakkan pesawat, menembakkan peluru, dan mengklik tombol "Play".
- **Memproses Logika Permainan:** Jika permainan aktif (`stats.game_active`), logika permainan diproses:
 - **Update Pesawat:** Metode `update()` pada objek `ship` dipanggil untuk memperbarui posisi pesawat berdasarkan input pemain.
 - **Update Peluru:** Fungsi `update_bullets()` dipanggil untuk memperbarui posisi semua peluru dan memeriksa tabrakan dengan alien.
 - **Update Alien:** Fungsi `update.aliens()` dipanggil untuk memperbarui posisi alien dan memeriksa tabrakan dengan pesawat.

4. Memperbarui Tampilan Layar

- **Menggambar Ulang Layar:** Setelah memproses logika permainan, fungsi `update_screen()` dipanggil untuk menggambar ulang semua elemen di layar, termasuk latar belakang, pesawat, peluru, alien, dan informasi skor.

5. Menangani Tabrakan

- **Tabrakan Peluru dan Alien:** Ketika peluru mengenai alien, fungsi `check_bullet_alien_collisions()` menangani tabrakan tersebut, menghancurkan alien yang terkena, menambah skor, dan memeriksa apakah ada alien yang tersisa.
- **Tabrakan Alien dan Pesawat:** Jika alien bertabrakan dengan pesawat, fungsi `ship_hit()` dipanggil untuk mengurangi jumlah kapal yang tersisa dan mengatur ulang permainan jika kapal habis.

6. Meningkatkan Kesulitan

- **Level dan Kecepatan:** Ketika semua alien dihancurkan, kecepatan permainan meningkat, level bertambah, dan armada alien baru dibuat. Ini memberikan tantangan yang lebih besar kepada pemain.

7. Menampilkan Informasi Skor

- **Papan Skor:** Selama permainan, informasi skor, level, dan jumlah kapal yang tersisa ditampilkan di layar menggunakan kelas **Scoreboard**. Ini memberikan umpan balik visual kepada pemain tentang kemajuan mereka.

8. Mengakhiri Permainan

- **Keluar dari Game:** Jika pemain memilih untuk keluar dengan menekan tombol escape, game akan berhenti dengan memanggil `sys.exit()`.

CARA MENGGUNAKAN APLIKASI

Cara Menggunakan Game Skyline Defender

Game Skyline Defender dirancang untuk memberikan pengalaman bermain yang interaktif dan menyenangkan. Berikut adalah langkah-langkah untuk menggunakan dan memainkan game:

1. Memulai Game

- **Menjalankan Game:**
 - Pastikan Anda memiliki semua file yang diperlukan (termasuk gambar, suara, dan skrip Python) di dalam folder yang sama.
 - Jalankan file utama game, yaitu **Skyline Defender.py**, menggunakan Python. Pastikan Anda telah menginstal Pygame dan semua dependensi yang diperlukan.

2. Antarmuka Awal

- **Tampilan Awal:** Setelah game dimulai, Anda akan melihat layar dengan latar belakang yang telah ditentukan dan tombol "Play" di tengah layar.
- **Tombol Play:**
 - Klik tombol "Play" untuk memulai permainan. Ini akan mengatur ulang semua statistik permainan dan mempersiapkan armada alien.

3. Kontrol Permainan

- **Menggerakkan Pesawat:**
 - Gunakan tombol panah pada keyboard untuk menggerakkan pesawat:
 - **Tombol Kiri:** Menggerakkan pesawat ke kiri.
 - **Tombol Kanan:** Menggerakkan pesawat ke kanan.
 - **Tombol Atas:** Menggerakkan pesawat ke atas.
 - **Tombol Bawah:** Menggerakkan pesawat ke bawah.
- **Menembakkan Peluru:**
 - Tekan tombol **Spasi** untuk menembakkan peluru ke arah alien. Pastikan untuk tidak melebihi batas jumlah peluru yang dapat ditembakkan sekaligus.

4. Tujuan Permainan

- **Menghancurkan Alien:**
 - Tujuan utama permainan adalah menghancurkan semua alien yang muncul di layar dengan menembakkan peluru. Setiap alien yang dihancurkan akan menambah skor Anda.

- **Menghindari Tabrakan:**
 - Hindari tabrakan dengan alien. Jika alien mencapai bagian bawah layar atau bertabrakan dengan pesawat Anda, Anda akan kehilangan satu kapal.

5. Statistik Permainan

- **Menampilkan Skor:**
 - Selama permainan, skor Anda, level, dan jumlah kapal yang tersisa akan ditampilkan di sudut layar. Ini memberikan umpan balik tentang kemajuan Anda.
- **Meningkatkan Level:**
 - Setelah semua alien dihancurkan, level permainan akan meningkat, dan kecepatan gerakan alien akan bertambah, membuat permainan semakin menantang.

6. Mengakhiri Permainan

- **Kehabisan Kapal:**
 - Jika Anda kehabisan kapal, permainan akan berakhir. Anda dapat melihat skor akhir Anda dan memilih untuk memulai permainan baru dengan mengklik tombol "Play" lagi.
- **Keluar dari Game:**
 - Untuk keluar dari permainan, tekan tombol **Escape** pada keyboard. Ini akan menutup jendela permainan.

TANTANGAN SELAMA PENGEMBANGAN

Tantangan dalam Pengembangan Game Skyline Defender

Pengembangan game, termasuk Skyline Defender, sering kali melibatkan berbagai tantangan yang dapat mempengaruhi proses dan hasil akhir. Berikut adalah beberapa tantangan yang mungkin dihadapi selama pengembangan game:

1. Desain dan Perencanaan

- **Mendefinisikan Konsep Game:** Menentukan konsep dan mekanisme permainan yang menarik bisa menjadi tantangan. Memastikan bahwa gameplay seimbang dan menyenangkan memerlukan perencanaan yang matang.
- **Membuat Alur Permainan:** Merancang alur permainan yang logis dan menarik, termasuk bagaimana pemain berinteraksi dengan objek dan musuh, bisa menjadi sulit.

2. Pengelolaan Sumber Daya

- **Penggunaan Gambar dan Suara:** Memastikan bahwa semua asset grafis (seperti gambar pesawat dan alien) dan suara (seperti efek suara peluru dan ledakan) tersedia dan dioptimalkan untuk digunakan dalam game.

3. Implementasi Logika Permainan

- **Mekanisme Tabrakan:** Mengimplementasikan logika untuk mendeteksi tabrakan antara peluru dan alien, serta antara alien dan pesawat, bisa menjadi rumit. Memastikan bahwa tabrakan terdeteksi dengan akurat dan responsif sangat penting untuk pengalaman bermain yang baik.
- **Pengaturan Kesulitan:** Menyeimbangkan tingkat kesulitan permainan agar tetap menantang tetapi tidak membuat frustrasi bagi pemain. Ini termasuk mengatur kecepatan gerakan alien dan jumlah peluru yang dapat ditembakkan.

4. Pengujian dan Debugging

- **Menemukan dan Memperbaiki Bug:** Selama pengembangan, bug dapat muncul yang mempengaruhi gameplay, seperti masalah dengan pergerakan objek, tabrakan, atau tampilan skor. Mengidentifikasi dan memperbaiki bug ini memerlukan waktu dan usaha.

5. Antarmuka Pengguna

- **Desain Antarmuka:** Menciptakan antarmuka pengguna yang intuitif dan menarik, termasuk tampilan skor dan tombol, bisa menjadi tantangan. Pengguna harus dapat dengan mudah memahami cara bermain dan mengakses fitur.
- **Responsif terhadap Input:** Memastikan bahwa kontrol permainan responsif terhadap input pemain, seperti pergerakan pesawat dan penembakan peluru.

6. Pengelolaan Status Permainan

- **Statistik dan Skor:** Mengelola statistik permainan, seperti skor dan jumlah kapal yang tersisa, serta memastikan bahwa informasi ini ditampilkan dengan benar di layar.
- **Reset Permainan:** Mengimplementasikan logika untuk mereset permainan dengan benar setelah pemain kalah atau memilih untuk memulai permainan baru.

7. Peningkatan Fitur

- **Menambahkan Fitur Baru:** Jika ingin menambahkan fitur baru, seperti jenis alien baru atau power-up, ini dapat memerlukan perubahan signifikan pada kode yang ada dan pengujian tambahan untuk memastikan semuanya berfungsi dengan baik.

DOKUMENTASI

Skyline Defender.py

1. Import Module

```
● ● ●  
import pygame  
from pygame.sprite import Group  
import sys  
from settings import Settings  
from game_stats import GameStats  
from scoreboard import Scoreboard  
from button import Button  
from ship import Ship  
import game_functions as gf
```

- **pygame:** Digunakan untuk membuat game 2D dengan menyediakan berbagai fungsi untuk menangani grafis, suara, input, dan lainnya.
- **Group:** Mengimpor kelas **Group** dari **pygame.sprite** untuk mengelola kumpulan objek (misalnya, peluru dan alien). Ini memudahkan pengelolaan dan pembaruan objek dalam game.
- **sys:** Digunakan untuk keluar dari program atau mengakses informasi sistem lainnya.
- **Settings:** Mengimpor pengaturan umum game dari file lain untuk memudahkan pengaturan konfigurasi seperti ukuran layar, kecepatan permainan, dan lainnya.
- **GameStats:** Untuk melacak statistik permainan, seperti skor dan status apakah game sedang aktif.
- **Scoreboard:** Untuk menampilkan skor pada layar game.
- **Button:** Kelas untuk membuat tombol interaktif seperti tombol "Play".
- **Ship:** Kelas untuk membuat objek pesawat yang dikendalikan oleh pemain.
- **game_functions:** Modul yang berisi fungsi-fungsi yang menangani logika utama dalam game, seperti peristiwa (events), pembaruan objek, dan tampilan layar.

2. Fungsi run_game

```
● ● ●  
def run_game():
```

- **run_game** adalah fungsi utama yang menjalankan seluruh game. Semua logika game, pembaruan, dan pemrosesan event berlangsung dalam fungsi ini.

3. Inisialisasi Game

```
● ● ●  
pygame.init()  
ai_settings = Settings()  
screen = pygame.display.set_mode(  
    (ai_settings.screen_width, ai_settings.screen_height))  
pygame.display.set_caption("Skyline Defender")
```

- **pygame.init()**: Inisialisasi semua modul pygame yang diperlukan.
- **Settings()**: Membuat objek **Settings** yang menyimpan pengaturan game, seperti lebar dan tinggi layar.

- **pygame.display.set_mode(...)**: Menentukan ukuran layar game sesuai dengan pengaturan dalam **ai_settings**.
- **pygame.display.set_caption("Skyline Defender")**: Mengatur judul jendela aplikasi menjadi "Skyline Defender".

4. Membuat Elemen Game

```
● ● ●

play_button = Button(ai_settings, screen, "Play")
stats = GameStats(ai_settings)
sb = Scoreboard(ai_settings, screen, stats)
ship = Ship(ai_settings, screen)
bullets = Group()
aliens = Group()
```

- **Button()**: Membuat tombol **Play** yang akan muncul di layar, memungkinkan pemain untuk memulai game.
- **GameStats()**: Membuat objek untuk melacak statistik permainan, seperti apakah permainan sedang aktif dan skor.
- **Scoreboard()**: Membuat objek **Scoreboard** untuk menampilkan skor permainan di layar.
- **Ship()**: Membuat objek **Ship** (pesawat pemain) yang akan dikendalikan oleh pemain.
- **Group()**: Membuat grup kosong untuk **bullets** dan **aliens**. Keduanya akan menampung objek peluru dan alien yang ada di game.

5. Membuat Formasi Alien

```
● ● ●

gf.create_fleet(ai_settings, screen, ship, aliens)
```

- **gf.create_fleet()**: Fungsi dalam **game_functions** yang membuat formasi alien di layar untuk pertama kalinya.

6. Loop Utama Game

```
● ● ●

while True:
    # Mengelola input pemain
    gf.check_events(
        ai_settings, screen, stats, sb, play_button, ship, aliens, bullets
    )
```

- **while True**: Ini adalah loop utama yang terus berjalan selama game tidak dihentikan.
- **gf.check_events()**: Fungsi yang menangani semua event, seperti input dari keyboard atau klik mouse (misalnya, memulai game dengan menekan tombol Play).

7. Memproses Logika Permainan

```
● ● ●  
if stats.game_active:  
    ship.update()  
    gf.update_bullets(ai_settings, screen, stats, sb, ship, aliens, bullets)  
    gf.update_aliens(ai_settings, screen, stats, sb, ship, aliens, bullets)
```

- **if stats.game_active:**: Memastikan bahwa logika permainan hanya diproses jika game sedang aktif (misalnya, jika pemain sudah memulai permainan).
- **ship.update()**: Memperbarui posisi pesawat pemain berdasarkan input pengguna.
- **gf.update_bullets()**: Memperbarui posisi dan status peluru, menghapus peluru yang sudah melewati layar atau menabrak alien.
- **gf.update_aliens()**: Memperbarui posisi alien dan menangani interaksi dengan peluru atau pesawat pemain.

8. Memperbarui Tampilan Layar

```
● ● ●  
gf.update_screen(  
    ai_settings, screen, stats, sb, ship, aliens, bullets, play_button  
)
```

- **gf.update_screen()**: Memperbarui tampilan layar, menggambar semua elemen game (pesawat, alien, peluru, skor, tombol, dll.) agar pemain melihat keadaan game yang terbaru.

9. Menjalankan Game

```
● ● ●  
run_game()
```

- **run_game()**: Memanggil fungsi utama untuk memulai dan menjalankan game.

Ship.py

1. Import Modul

```
● ● ●  
import pygame  
from pygame.sprite import Sprite
```

- **pygame**: Modul utama yang digunakan untuk membuat game dengan menyediakan berbagai fungsi untuk grafis, suara, input, dan lainnya.
- **Sprite**: Kelas dari pygame yang digunakan untuk mengelola objek bergerak dalam game. Kelas **Ship** mewarisi kelas **Sprite** untuk memanfaatkan fitur pengelolaan objek sprite.

2. Kelas Ship



```
class Ship(Sprite):
```

- **class Ship(Sprite)::** Mendefinisikan kelas **Ship** yang merupakan turunan dari **Sprite**, sehingga **Ship** dapat diubah, digerakkan, dan ditampilkan seperti objek sprite lainnya dalam game.

3. Inisialisasi Kelas Ship



```
def __init__(self, ai_settings, screen):
    super(Ship, self).__init__()
    self.screen = screen
    self.ai_settings = ai_settings
```

- **__init__**: Merupakan konstruktor kelas **Ship** yang akan dipanggil ketika objek **Ship** dibuat.
- **super(Ship, self).__init__()**: Memanggil konstruktor dari kelas induk **Sprite** untuk menginisialisasi fungsi-fungsi sprite yang akan digunakan oleh objek ini.
- **self.screen**: Menyimpan objek **screen** yang digunakan untuk menggambar pesawat pada layar.
- **self.ai_settings**: Menyimpan objek **ai_settings** yang berisi pengaturan permainan (seperti kecepatan pesawat).

4. Memuat Gambar dan Menentukan Ukuran



```
# Memuat gambar pesawat dan mendapatkan ukurannya.
self.image = pygame.image.load("pesawattempur.png")
self.rect = self.image.get_rect()
self.screen_rect = screen.get_rect()
```

- **pygame.image.load("pesawattempur.png")**: Memuat gambar pesawat dari file "pesawattempur.png" dan menyimpannya dalam atribut **image**.
- **self.rect = self.image.get_rect()**: Mengambil objek **rect** (rectangle) dari gambar yang digunakan untuk menentukan posisi dan ukuran pesawat di layar.
- **self.screen_rect = screen.get_rect()**: Mengambil **rect** dari objek **screen**, yang digunakan untuk menentukan batas-batas layar.

5. Menempatkan Pesawat di Posisi Awal

```
● ● ●  
# Menempatkan pesawat di bagian bawah-tengah layar.  
self.rect.centerx = self.screen_rect.centerx  
self.rect.bottom = self.screen_rect.bottom
```

- **self.rect.centerx = self.screen_rect.centerx**: Menempatkan pesawat di tengah layar secara horizontal.
- **self.rect.bottom = self.screen_rect.bottom**: Menempatkan pesawat di bagian bawah layar.

6. Menyimpan Posisi Decimal untuk Pergerakan Halus

```
● ● ●  
# Menyimpan posisi decimal untuk pergerakan yang halus.  
self.center = float(self.rect.centerx)  
self.top = float(self.rect.top)
```

- **self.center = float(self.rect.centerx)**: Menyimpan posisi horizontal (centerx) pesawat dalam bentuk angka desimal untuk memungkinkan pergerakan yang lebih halus.
- **self.top = float(self.rect.top)**: Menyimpan posisi vertikal (top) pesawat dalam bentuk angka desimal.

7. Mengatur Flag untuk Pergerakan

```
● ● ●  
# Mengatur flag untuk pergerakan.  
self.moving_right = False  
self.moving_left = False  
self.moving_up = False  
self.moving_down = False
```

- **self.moving_right, self.moving_left, self.moving_up, self.moving_down**: Keempat flag ini digunakan untuk menentukan arah pergerakan pesawat. Jika nilai flag bernilai **True**, pesawat akan bergerak dalam arah yang sesuai.

8. Fungsi center_ship

```
● ● ●  
def center_ship(self):  
  
    self.center = self.screen_rect.centerx  
    self.rect.bottom = self.screen_rect.bottom
```

- **center_ship**: Fungsi untuk mengatur ulang posisi pesawat ke tengah layar bagian bawah, yang berguna saat permainan dimulai ulang atau pesawat hancur dan perlu kembali ke posisi semula.

9. Fungsi update

```
● ● ●

def update(self):

    if self.moving_right and self.rect.right < self.screen_rect.right:
        self.center += self.ai_settings.ship_speed_factor
    if self.moving_left and self.rect.left > 0:
        self.center -= self.ai_settings.ship_speed_factor
    if self.moving_up and self.rect.top > 0:
        self.top -= self.ai_settings.ship_speed_factor
    if self.moving_down and self.rect.bottom < self.screen_rect.bottom:
        self.top += self.ai_settings.ship_speed_factor

    # Memperbarui posisi berdasarkan koordinat.
    self.rect.centerx = self.center
    self.rect.top = self.top
```

- **update**: Fungsi untuk memperbarui posisi pesawat berdasarkan status pergerakan (apakah pemain menekan tombol untuk bergerak atau tidak).
 - **self.moving_right and self.rect.right < self.screen_rect.right**: Jika flag pergerakan kanan aktif dan pesawat belum mencapai batas kanan layar, pesawat akan bergerak ke kanan.
 - **self.moving_left and self.rect.left > 0**: Jika flag pergerakan kiri aktif dan pesawat belum mencapai batas kiri layar, pesawat akan bergerak ke kiri.
 - **self.moving_up and self.rect.top > 0**: Jika flag pergerakan atas aktif dan pesawat belum mencapai batas atas layar, pesawat akan bergerak ke atas.
 - **self.moving_down and self.rect.bottom < self.screen_rect.bottom**: Jika flag pergerakan bawah aktif dan pesawat belum mencapai batas bawah layar, pesawat akan bergerak ke bawah.
- Setelah memproses pergerakan, **self.rect.centerx** dan **self.rect.top** diperbarui untuk memindahkan posisi pesawat.

10. Fungsi blitme

```
● ● ●

def blitme(self):

    self.screen.blit(self.image, self.rect)
```

- **blitme**: Fungsi untuk menggambar gambar pesawat ke layar pada posisi yang telah diperbarui menggunakan **rect**.

Alien.py

1. Import Modul

```
import pygame
from pygame.sprite import Sprite
```

- **pygame**: Modul utama untuk membuat game 2D, yang menyediakan berbagai fungsi untuk menangani grafis, suara, input, dan lainnya.
- **Sprite**: Kelas dari **pygame.sprite** yang digunakan untuk mempermudah pengelolaan objek game (seperti alien). Kelas **Alien** mewarisi **Sprite**, yang memungkinkan pengelolaan objek alien menggunakan fitur yang disediakan oleh **Sprite**.

2. Kelas Alien

```
class Alien(Sprite):
```

- **class Alien(Sprite)::**: Mendefinisikan kelas **Alien** yang merupakan turunan dari kelas **Sprite**. Ini memungkinkan objek alien untuk diubah, digerakkan, dan ditampilkan di layar seperti objek sprite lainnya dalam game.

3. Inisialisasi Kelas Alien

```
def __init__(self, ai_settings, screen):
    super(Alien, self).__init__()
    self.screen = screen
    self.ai_settings = ai_settings
```

- **__init__**: Merupakan konstruktor untuk kelas **Alien** yang dipanggil saat objek alien dibuat.
- **super(Alien, self).__init__()**: Memanggil konstruktor dari kelas induk **Sprite** untuk menginisialisasi fungsi-fungsi sprite yang akan digunakan oleh objek ini.
- **self.screen**: Menyimpan objek **screen** yang digunakan untuk menggambar alien pada layar.
- **self.ai_settings**: Menyimpan objek **ai_settings** yang berisi pengaturan permainan, seperti kecepatan pergerakan alien dan arah formasi alien.

4. Memuat Gambar Alien dan Menetapkan Posisi

```
● ● ●

# Memuat gambar alien dan menetapkan posisinya.
self.image = pygame.image.load("alien.png")
self.rect = self.image.get_rect()
self.rect.x = self.rect.width
self.rect.y = self.rect.height
```

- **pygame.image.load("alien.png")**: Memuat gambar alien dari file "alien.png" dan menyimpannya dalam atribut **image**.
- **self.rect = self.image.get_rect()**: Mengambil objek **rect** (rectangle) dari gambar, yang digunakan untuk menentukan posisi dan ukuran alien di layar.
- **self.rect.x = self.rect.width**: Menetapkan posisi horizontal alien di layar, dimulai dari lebar gambar alien (agar alien muncul di sebelah kiri layar).
- **self.rect.y = self.rect.height**: Menetapkan posisi vertikal alien, dimulai dari tinggi gambar alien (agar alien muncul sedikit dari atas layar).

5. Menyimpan Posisi Horizontal Alien

```
● ● ●

# Menyimpan posisi horizontal alien.
self.x = float(self.rect.x)
```

- **self.x = float(self.rect.x)**: Menyimpan posisi horizontal alien dalam bentuk desimal (menggunakan **float**) untuk memungkinkan pergerakan yang lebih halus. Ini akan digunakan untuk memperbarui posisi alien.

6. Fungsi `check_edges`

```
● ● ●

def check_edges(self):

    screen_rect = self.screen.get_rect()
    if self.rect.right >= screen_rect.right or self.rect.left <= 0:
        return True
```

- **check_edges**: Fungsi untuk memeriksa apakah alien sudah mencapai tepi layar (kanan atau kiri).
 - **screen_rect = self.screen.get_rect()**: Mengambil **rect** dari objek **screen**, yang mewakili batas-batas layar.
 - **self.rect.right >= screen_rect.right or self.rect.left <= 0**: Memeriksa apakah posisi alien sudah mencapai tepi kanan (**right**) atau tepi kiri (**left**) layar. Jika iya, fungsi ini akan mengembalikan nilai **True**, yang menandakan alien sudah mencapai tepi layar.

7. Fungsi `update`

```
def update(self):  
    self.x += self.ai_settings.alien_speed_factor * self.ai_settings.fleet_direction  
    self.rect.x = self.x
```

- **update:** Fungsi untuk menggerakkan alien berdasarkan pengaturan kecepatan dan arah.
 - **self.x += self.ai_settings.alien_speed_factor * self.ai_settings.fleet_direction:** Memperbarui posisi horizontal alien berdasarkan kecepatan yang ditentukan dalam **ai_settings.alien_speed_factor** mengatur seberapa cepat alien bergerak, dan **fleet_direction** mengatur apakah alien bergerak ke kanan (**1**) atau ke kiri (**-1**).
 - **self.rect.x = self.x:** Memperbarui posisi alien di layar berdasarkan nilai **x** yang baru.

8. Fungsi blitme

```
def blitme(self):  
    self.screen.blit(self.image, self.rect)
```

- **blitme:** Fungsi untuk menggambar alien pada layar di posisi yang telah diperbarui menggunakan **rect**. Fungsi ini akan dipanggil untuk menampilkan gambar alien di layar setiap kali posisi alien diperbarui.

Bullet.py

1. Import Modul

```
import pygame  
from pygame.sprite import Sprite
```

- **pygame:** Modul utama yang digunakan untuk membuat game 2D, menyediakan fungsi-fungsi untuk grafis, suara, input, dan lainnya.
- **Sprite:** Kelas dari **pygame.sprite** yang digunakan untuk mempermudah pengelolaan objek dalam game, seperti peluru. Kelas **Bullet** mewarisi **Sprite**, sehingga objek peluru akan dikelola dengan cara yang lebih mudah.

2. Kelas Bullet



```
class Bullet(Sprite):
```

- **class Bullet(Sprite):**: Mendefinisikan kelas **Bullet** yang merupakan turunan dari **Sprite**, memungkinkan objek peluru untuk diubah, digerakkan, dan ditampilkan di layar sebagai objek sprite.

3. Inisialisasi Kelas Bullet



```
def __init__(self, ai_settings, screen, ship):  
    super(Bullet, self).__init__()  
    self.screen = screen
```

- **__init__**: Konstruktor kelas **Bullet** yang dipanggil ketika objek peluru dibuat.
- **super(Bullet, self).__init__()**: Memanggil konstruktor dari kelas induk **Sprite** untuk menginisialisasi fitur-fitur sprite yang akan digunakan oleh objek peluru.
- **self.screen**: Menyimpan objek **screen** yang digunakan untuk menggambar peluru pada layar.

4. Mengatur Ukuran dan Posisi Peluru



```
# Mengatur ukuran dan posisi awal peluru.  
self.rect = pygame.Rect(  
    0, 0, ai_settings.bullet_width, ai_settings.bullet_height  
)  
self.rect.centerx = ship.rect.centerx  
self.rect.top = ship.rect.top
```

- **self.rect = pygame.Rect(0, 0, ai_settings.bullet_width, ai_settings.bullet_height)**: Membuat objek **rect** untuk peluru dengan ukuran yang ditentukan dalam **ai_settings** (lebar dan tinggi peluru).
- **self.rect.centerx = ship.rect.centerx**: Menetapkan posisi horizontal peluru agar sejajar dengan pesawat saat ini (mengambil **centerx** dari pesawat).
- **self.rect.top = ship.rect.top**: Menetapkan posisi vertikal peluru agar berada di bagian atas pesawat (mengambil **top** dari pesawat).

5. Menyimpan Posisi Vertikal Peluru



```
# Menyimpan posisi vertikal peluru.  
self.y = float(self.rect.y)
```

- **self.y = float(self.rect.y)**: Menyimpan posisi vertikal peluru dalam bentuk desimal (menggunakan **float**), yang memungkinkan pergerakan peluru yang lebih halus.

6. Mengatur Warna dan Kecepatan Peluru

```
# Mengatur warna dan kecepatan peluru.
self.color = ai_settings.bullet_color
self.speed_factor = ai_settings.bullet_speed_factor
```

- **self.color = ai_settings.bullet_color**: Menyimpan warna peluru yang ditentukan dalam **ai_settings**.
- **self.speed_factor = ai_settings.bullet_speed_factor**: Menyimpan kecepatan pergerakan peluru yang ditentukan dalam **ai_settings**.

7. Fungsi update

```
def update(self):
    self.y -= self.speed_factor
    self.rect.y = self.y
```

- **update**: Fungsi untuk memperbarui posisi peluru di layar setiap frame.
 - **self.y -= self.speed_factor**: Mengurangi posisi vertikal peluru untuk membuatnya bergerak ke atas layar dengan kecepatan yang telah ditentukan (**speed_factor**).
 - **self.rect.y = self.y**: Memperbarui posisi peluru di layar sesuai dengan nilai y yang baru.

8. Fungsi draw_bullet

```
def draw_bullet(self):
    pygame.draw.rect(self.screen, self.color, self.rect)
```

- **draw_bullet**: Fungsi untuk menggambar peluru pada layar.
 - **pygame.draw.rect(self.screen, self.color, self.rect)**: Menggambar sebuah persegi panjang (yang mewakili peluru) pada layar menggunakan warna yang telah ditentukan (**self.color**) dan posisi serta ukuran yang ditentukan dalam **self.rect**.

Button.py

1. Import Module

```
● ● ●
```

```
import pygame.font
```

- **pygame.font**: Modul yang menangani font dan teks di **pygame**. Digunakan untuk membuat dan merender teks yang ditampilkan di layar.

2. Deklarasi Kelas Button

```
● ● ●
```

```
class Button:
```

- **class Button**:: Mendeklarasikan kelas **Button** yang digunakan untuk membuat dan mengelola tombol.

3. Inisialisasi Kelas Button

```
● ● ●
```

```
def __init__(self, ai_settings, screen, msg):  
    self.screen = screen  
    self.screen_rect = screen.get_rect()
```

- **__init__**: Konstruktor untuk mengatur atribut awal tombol.
- **ai_settings**: Parameter yang membawa pengaturan umum game, seperti ukuran layar.
- **screen**: Objek layar di mana tombol akan ditampilkan.
- **msg**: Teks yang akan ditampilkan di tombol.
- **self.screen**: Menyimpan referensi ke layar game.
- **self.screen_rect**: Mengambil ukuran dan posisi layar sebagai referensi untuk memposisikan tombol.

4. Mengatur Ukuran dan Warna Tombol

```
● ● ●
```

```
# Mengatur dimensi dan properti tombol.  
self.width, self.height = 200, 50  
self.button_color = (0, 255, 0) # Warna tombol (hijau).  
self.text_color = (255, 255, 255) # Warna teks (putih).  
self.font = pygame.font.SysFont(None, 48) # Font dan ukuran teks.
```

- **self.width, self.height = 200, 50**: Mengatur ukuran tombol dengan lebar **200 piksel** dan tinggi **50 piksel**.
- **self.button_color = (0, 255, 0)**: Mengatur warna tombol menjadi hijau (RGB: Merah=0, Hijau=255, Biru=0).

- **self.text_color = (255, 255, 255)**: Mengatur warna teks menjadi putih (RGB: 255, 255, 255).
- **self.font = pygame.font.SysFont(None, 48)**:
 - Mengatur jenis dan ukuran font.
 - **None**: Menggunakan font default sistem.
 - **48**: Ukuran teks dalam piksel.

5. Membuat Objek Tombol

```
● ● ●
# Membuat objek tombol dan memusatkaninya di layar.
self.rect = pygame.Rect(0, 0, self.width, self.height)
self.rect.center = self.screen_rect.center
```

- **self.rect = pygame.Rect(0, 0, self.width, self.height)**:
 - Membuat tombol sebagai **rect** (persegi panjang) dengan ukuran yang sudah ditentukan.
 - Koordinat awalnya diatur ke **(0,0)**.
- **self.rect.center = self.screen_rect.center**:
 - Memposisikan tombol di tengah layar berdasarkan dimensi layar yang disimpan dalam **self.screen_rect**.

6. Menyiapkan Pesan Teks Tombol

```
● ● ●
# Mempersiapkan pesan tombol.
self.prep_msg(msg)
```

- **self.prep_msg(msg)**: Memanggil metode **prep_msg** untuk membuat gambar teks dari pesan yang diberikan sebagai parameter **msg**.

7. Fungsi prep_msg

```
● ● ●
def prep_msg(self, msg):
    self.msg_image = self.font.render(msg, True, self.text_color, self.button_color)
    self.msg_image_rect = self.msg_image.get_rect()
    self.msg_image_rect.center = self.rect.center
```

- **prep_msg**: Fungsi untuk mengatur teks yang akan ditampilkan di tombol.
- **msg**: Teks yang akan ditampilkan pada tombol (contoh: "Play").
- **self.font.render()**:
 - Mengubah teks menjadi gambar yang bisa dirender di layar.

- **msg**: Teks yang dirender.
- **True**: Mengaktifkan anti-aliasing untuk tampilan teks yang lebih halus.
- **self.text_color**: Warna teks.
- **self.button_color**: Warna latar belakang teks.
- **self.msg_image.get_rect()**: Mendapatkan ukuran dan posisi gambar teks.
- **self.msg_image_rect.center = self.rect.center**: Memposisikan teks agar berada di tengah tombol.

8. Fungsi draw_button

```
● ● ●
def draw_button(self):
    self.screen.fill(self.button_color, self.rect)
    self.screen.blit(self.msg_image, self.msg_image_rect)
```

- **draw_button**: Fungsi untuk menggambar tombol dan teksnya di layar.
- **self.screen.fill(self.button_color, self.rect)**:
 - Mengisi tombol dengan warna hijau (**button_color**) di dalam area persegi panjang yang didefinisikan oleh **self.rect**.
- **self.screen.blit(self.msg_image, self.msg_image_rect)**:
 - Menampilkan gambar teks (**msg_image**) di posisi yang ditentukan (**msg_image_rect**) di dalam tombol.

Scoreboard.py

1. Import Modul dan Dependensi

```
● ● ●
import pygame.font
from pygame.sprite import Group
from ship import Ship
```

- **pygame.font**: Modul untuk mengatur font dan teks dalam game.
- **Group**: Mengelola sekumpulan objek sprite (digunakan untuk menyimpan gambar pesawat yang tersisa).
- **Ship**: Mengimpor kelas **Ship** untuk menampilkan ikon pesawat yang merepresentasikan nyawa pemain.

2. Deklarasi Kelas



```
class Scoreboard:
```

- **class Scoreboard**: Mendeklarasikan kelas **Scoreboard** yang berfungsi menampilkan skor, level, dan sisa nyawa pemain.

3. Inisialisasi Atribut

```
def __init__(self, ai_settings, screen, stats):  
    self.screen = screen  
    self.screen_rect = screen.get_rect()  
    self.ai_settings = ai_settings  
    self.stats = stats
```

- **__init__**: Konstruktor untuk mengatur atribut awal dari papan skor.
- **ai_settings**: Mengatur konfigurasi permainan (seperti kecepatan atau batas layar).
- **screen**: Objek layar tempat semua elemen akan ditampilkan.
- **stats**: Menyimpan statistik permainan, seperti skor, level, dan nyawa tersisa.
- **self.screen_rect**: Mengambil dimensi layar untuk membantu memposisikan elemen UI.

4. Pengaturan Font dan Teks Skor

```
# Pengaturan font untuk teks skor.  
self.text_color = (255, 255, 255)  
self.font = pygame.font.SysFont(None, 48)
```

- **self.text_color**: Mengatur warna teks menjadi putih (RGB: 255, 255, 255).
- **pygame.font.SysFont(None, 48)**:
 - Menggunakan font default sistem.
 - Mengatur ukuran font ke **48 piksel**.

5. Persiapan Tampilan Awal

```
self.prep_score()  
self.prep_high_score()  
self.prep_level()  
self.prep_ships()
```

- Memanggil metode:
 - **prep_score()**: Menampilkan skor saat ini.

- **prep_high_score()**: Menampilkan skor tertinggi.
- **prep_level()**: Menampilkan level pemain.
- **prep_ships()**: Menampilkan jumlah nyawa yang tersisa.

6. Menampilkan Skor Saat Ini

```
● ● ●

def prep_score(self):

    rounded_score = int(round(self.stats.score, -1))
    score_str = "{:,}".format(rounded_score)
    self.score_image = self.font.render(
        score_str, True, self.text_color
    )

    # Menampilkan skor di pojok kanan atas.
    self.score_rect = self.score_image.get_rect()
    self.score_rect.right = self.screen_rect.right - 20
    self.score_rect.top = 20
```

- **self.stats.score**: Mengambil skor saat ini dari statistik.
- **round(self.stats.score, -1)**: Membulatkan skor ke kelipatan 10.
- **"{:,}" .format()**: Memformat skor dengan tanda koma (contoh: 1,000).
- **self.font.render()**: Mengubah teks skor menjadi gambar yang bisa dirender di layar.
- **self.score_rect**: Mengatur posisi skor di pojok kanan atas layar.

7. Menampilkan Skor Tertinggi

```
● ● ●

def prep_high_score(self):

    high_score = int(round(self.stats.high_score, -1))
    high_score_str = "{:,}" .format(high_score)
    self.high_score_image = self.font.render(
        high_score_str, True, self.text_color
    )

    # Menampilkan skor tertinggi di tengah atas layar.
    self.high_score_rect = self.high_score_image.get_rect()
    self.high_score_rect.centerx = self.screen_rect.centerx
    self.high_score_rect.top = self.score_rect.top
```

- Sama seperti **prep_score()**, tetapi khusus untuk skor tertinggi.
- Skor ditampilkan di **tengah atas layar**.

8. Menampilkan Level

```
def prep_level(self):

    self.level_image = self.font.render(
        str(self.stats.level), True, self.text_color
    )

    # Menampilkan level di bawah skor.
    self.level_rect = self.level_image.get_rect()
    self.level_rect.right = self.score_rect.right
    self.level_rect.top = self.score_rect.bottom + 10
```

- **self.stats.level:** Mengambil level pemain saat ini.
- Menampilkan level tepat di bawah skor.

9. Menampilkan Nyawa yang Tersisa

```
def prep_ships(self):

    self.ships = Group()
    for ship_number in range(self.stats.ships_left):
        ship = Ship(self.ai_settings, self.screen)
        ship.rect.x = 10 + ship_number * ship.rect.width
        ship.rect.y = 10
        self.ships.add(ship)
```

- **self.ships:** Mengelola ikon pesawat yang mewakili nyawa pemain.
- **Loop for:**
 - Mengulang sebanyak **nyawa yang tersisa**.
 - Membuat objek **Ship** untuk setiap nyawa dan memposisikannya di kiri atas layar.

10. Menampilkan Semua Elemen di Layar

```
def show_score(self):

    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)
    self.ships.draw(self.screen)
```

- **blit:** Menampilkan gambar di layar.
- Menggambar skor, skor tertinggi, level, dan ikon pesawat yang tersisa.

Settings.py

1. Import Modul



```
import pygame
```

- **pygame**: Mengimpor modul untuk menangani grafis dan interaksi pengguna di game.

2. Deklarasi Kelas



```
class Settings:
```

- **class Settings**: Membuat kelas **Settings** untuk menyimpan semua pengaturan game dalam satu tempat, baik yang statis maupun dinamis.

3. Konstruktor Pengaturan Statis



```
def __init__(self):
```

- **__init__**: Konstruktor yang dipanggil saat objek **Settings** dibuat, untuk mengatur nilai default permainan.

4. Pengaturan Layar



```
self.screen_width = 1200  
self.screen_height = 800  
self.background_image = pygame.image.load("background.jpeg")
```

- **self.screen_width = 1200**: Lebar layar diatur menjadi **1200 piksel**.
- **self.screen_height = 800**: Tinggi layar diatur menjadi **800 piksel**.
- **pygame.image.load()**: Memuat gambar latar belakang (background.jpeg) untuk digunakan sebagai latar permainan.

5. Pengaturan Kapal



```
self.ship_limit = 3
```

- **self.ship_limit = 3**: Pemain diberi **3 nyawa** (kapal) sebelum permainan berakhir.

6. Pengaturan Peluru

```
● ● ●  
self.bullet_width = 3  
self.bullet_height = 15  
self.bullet_color = (200, 0, 0)  
self.bullets_allowed = 5
```

- **self.bullet_width = 3:** Lebar peluru diatur **3 piksel**.
- **self.bullet_height = 15:** Tinggi peluru diatur **15 piksel**.
- **self.bullet_color = (200, 0, 0):** Warna peluru merah (RGB: 200, 0, 0).
- **self.bullets_allowed = 5:** Maksimum **5 peluru** yang bisa ditembakkan sekaligus berada di layar.

7. Pengaturan Alien

```
● ● ●  
self.fleet_drop_speed = 10
```

- **self.fleet_drop_speed = 10:** Mengatur jarak jatuh alien saat mencapai tepi layar (turun **10 piksel**).

8. Skala Percepatan Permainan

```
● ● ●  
self.speedup_scale = 1.1  
self.score_scale = 1.5
```

- **self.speedup_scale = 1.1:** Kecepatan permainan meningkat sebesar **10% (1.1)** setiap kali pemain naik level.
- **self.score_scale = 1.5:** Poin yang diperoleh dari alien meningkat **50% (1.5)** setiap level.

9. Pengaturan Dinamis

```
● ● ●  
self.initialize_dynamic_settings()
```

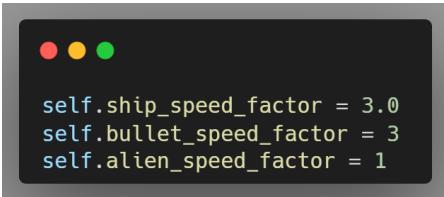
- Memanggil metode **initialize_dynamic_settings()** untuk mengatur nilai yang dapat berubah selama permainan berlangsung.

10. Metode Pengaturan Dinamis

```
● ● ●  
def initialize_dynamic_settings(self):
```

- **initialize_dynamic_settings():** Mengatur nilai yang bisa berubah selama permainan, seperti kecepatan dan poin alien.

11.Pengaturan Kecepatan Dinamis



```
self.ship_speed_factor = 3.0
self.bullet_speed_factor = 3
self.alien_speed_factor = 1
```

- **self.ship_speed_factor = 3.0**: Kecepatan kapal diatur **3.0 unit per frame**.
- **self.bullet_speed_factor = 3**: Kecepatan peluru diatur **3 unit per frame**.
- **self.alien_speed_factor = 1**: Kecepatan alien diatur **1 unit per frame**.

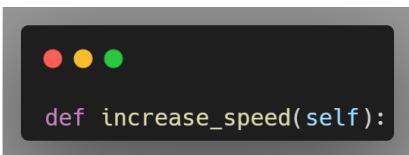
12.Pengaturan Poin dan Arah Alien



```
self.alien_points = 50
self.fleet_direction = 1
```

- **self.alien_points = 50**: Setiap alien yang dihancurkan bernilai **50 poin**.
- **self.fleet_direction = 1**: Arah gerakan alien (1 untuk **kanan**, -1 untuk **kiri**).

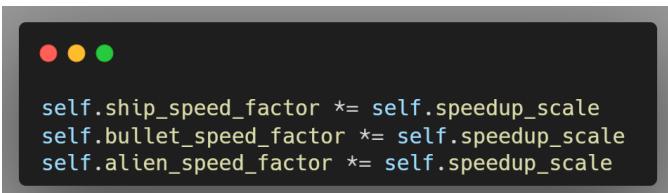
13.Metode Meningkatkan Kecepatan



```
def increase_speed(self):
```

- **increase_speed()**: Digunakan untuk mempercepat permainan dan meningkatkan poin setiap kali level naik.

14.Mempercepat Elemen Dinamis



```
self.ship_speed_factor *= self.speedup_scale
self.bullet_speed_factor *= self.speedup_scale
self.alien_speed_factor *= self.speedup_scale
```

- **self.ship_speed_factor *= 1.1**: Kecepatan kapal meningkat sebesar **10%**.
- **self.bullet_speed_factor *= 1.1**: Kecepatan peluru meningkat **10%**.
- **self.alien_speed_factor *= 1.1**: Kecepatan alien meningkat **10%**.

15.Meningkatkan Poin Alien



```
self.alien_points = int(self.alien_points * self.score_scale)
```

- **self.alien_points *= 1.5**: Poin alien meningkat **50%** setiap level.
- **int()**: Mengonversi nilai menjadi bilangan bulat.

Game_Stats.py

1. Deklarasi Kelas

```
● ● ●  
class GameStats:
```

- **class GameStats:** Mendeklarasikan kelas **GameStats** yang bertanggung jawab untuk menyimpan dan mengelola semua statistik permainan, seperti skor, level, dan jumlah nyawa yang tersisa.

2. Konstruktor (Inisialisasi)

```
● ● ●  
def __init__(self, ai_settings):
```

- **__init__**: Metode konstruktor yang akan dijalankan saat objek **GameStats** dibuat.
- **ai_settings**: Parameter yang menerima objek pengaturan dari kelas **Settings** untuk mengakses nilai-nilai pengaturan game.

3. Menyimpan Objek Pengaturan

```
● ● ●  
self.ai_settings = ai_settings
```

- **self.ai_settings = ai_settings**: Menyimpan referensi ke pengaturan permainan dalam atribut **self.ai_settings** agar dapat diakses dalam metode lain di kelas ini.

4. Mengatur Statistik Awal

```
● ● ●  
self.reset_stats()
```

- Memanggil metode **reset_stats()** untuk mengatur ulang statistik permainan (nyawa, skor, dan level) setiap kali permainan dimulai atau di-reset.

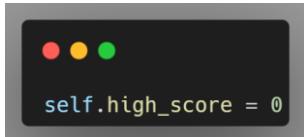
5. Status Permainan Aktif

```
● ● ●  
self.game_active = False
```

- **self.game_active = False**: Mengatur status permainan ke **tidak aktif** saat pertama kali dimulai.

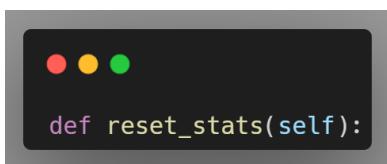
Artinya, permainan akan dimulai hanya setelah pemain menekan tombol **Start** atau **Play**.

6. Skor Tertinggi



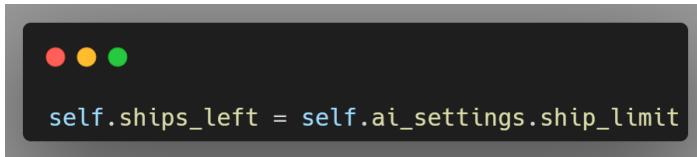
- `self.high_score = 0`: Mengatur skor tertinggi menjadi **0** saat permainan dimulai.
- Skor tertinggi ini **tidak di-reset** meskipun pemain memulai ulang permainan, sehingga akan selalu menyimpan skor terbaik pemain.

7. Metode Reset Statistik



- `reset_stats()`: Metode untuk mengatur ulang statistik yang **dinamis** dan dapat berubah, seperti jumlah nyawa, skor, dan level permainan.

8. Jumlah Nyawa Tersisa



- `self.ships_left`: Menyimpan jumlah nyawa (kapal) yang tersisa untuk pemain.
- `self.ai_settings.ship_limit`: Mengambil nilai maksimum nyawa dari pengaturan **Settings** (default: **3 kapal**).

9. Skor Pemain



- `self.score = 0`: Mengatur skor awal pemain menjadi **0** setiap kali permainan dimulai ulang atau di-reset.

10. Level Permainan



- `self.level = 1`: Mengatur level awal permainan ke **1**.
- Level ini akan meningkat saat pemain menyelesaikan kelompok alien tertentu.

Game_functions.py

1. Import Modul dan Depedensi

```
● ● ●  
import sys  
from time import sleep  
import sound as se  
import pygame  
from bullet import Bullet  
from alien import Alien
```

- **sys**: Digunakan untuk mengontrol sistem, seperti keluar dari program.
- **sleep**: Digunakan untuk memberikan jeda waktu dalam program.
- **sound as se**: Modul untuk mengatur suara dalam game.
- **pygame**: Modul utama untuk membuat game.
- **Bullet dan Alien**: Kelas yang diimpor dari modul **bullet** dan **alien** untuk membuat objek peluru dan alien.

2. Fungsi check_keydown_events

```
● ● ●  
def check_keydown_events(event, ai_settings, screen, ship, bullets):  
    """  
    Merespons penekanan tombol.  
    """  
    if event.key == pygame.K_RIGHT:  
        ship.moving_right = True  
    elif event.key == pygame.K_LEFT:  
        ship.moving_left = True  
    elif event.key == pygame.K_UP:  
        ship.moving_up = True  
    elif event.key == pygame.K_DOWN:  
        ship.moving_down = True  
    elif event.key == pygame.K_SPACE:  
        fire_bullet(ai_settings, screen, ship, bullets)  
    elif event.key == pygame.K_ESCAPE:  
        sys.exit()
```

- Fungsi ini merespons penekanan tombol keyboard.
- **event.key**: Menentukan tombol mana yang ditekan.
- **ship.moving_right, ship.moving_left, ship.moving_up, ship.moving_down**: Mengatur pergerakan kapal.
- **fire_bullet**: Menembakkan peluru jika tombol spasi ditekan.
- **sys.exit()**: Keluar dari program jika tombol escape ditekan.

3. Fungsi check_keyup_events

```
def check_keyup_events(event, ship):
    """
    Merespons pelepasan tombol.
    """
    if event.key == pygame.K_RIGHT:
        ship.moving_right = False
    elif event.key == pygame.K_LEFT:
        ship.moving_left = False
    elif event.key == pygame.K_UP:
        ship.moving_up = False
    elif event.key == pygame.K_DOWN:
        ship.moving_down = False
```

- Fungsi ini merespons pelepasan tombol keyboard.
- **event.key**: Menentukan tombol mana yang dilepas.
- **ship.moving_right, ship.moving_left, ship.moving_up, ship.moving_down**: Menghentikan pergerakan kapal.

4. Fungsi check_events

```
def check_events(ai_settings, screen, stats, sb, play_button, ship, aliens, bullets):
    """
    Merespons peristiwa seperti penekanan tombol dan klik mouse.
    """
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            check_keydown_events(event, ai_settings, screen, ship, bullets)
        elif event.type == pygame.KEYUP:
            check_keyup_events(event, ship)
        elif event.type == pygame.MOUSEBUTTONDOWN:
            mouse_x, mouse_y = pygame.mouse.get_pos()
            check_play_button(
                ai_settings,
                screen,
                stats,
                sb,
                play_button,
                ship,
                aliens,
                bullets,
                mouse_x,
                mouse_y
            )
```

- Fungsi ini memeriksa semua jenis peristiwa, seperti penekanan tombol dan klik mouse.
- **pygame.event.get()**: Mengambil semua peristiwa yang terjadi.
- **pygame.QUIT**: Keluar dari program jika pengguna menutup jendela game.
- **pygame.KEYDOWN dan pygame.KEYUP**: Memanggil fungsi **check_keydown_events** dan **check_keyup_events**.

- **pygame.MOUSEBUTTONDOWN**: Memeriksa apakah tombol Play diklik.

5. Fungsi check_play_button

```
● ● ●

def check_play_button(ai_settings, screen, stats, sb, play_button, ship, aliens, bullets, mouse_x, mouse_y):
    """
    Memulai permainan baru ketika pemain mengklik tombol Play.
    """
    button_clicked = play_button.rect.collidepoint(mouse_x, mouse_y)
    if button_clicked and not stats.game_active:
        # Reset pengaturan permainan.
        ai_settings.initialize_dynamic_settings()

        # Sembunyikan kursor mouse.
        pygame.mouse.set_visible(False)

        # Reset statistik permainan.
        stats.reset_stats()
        stats.game_active = True

        # Reset gambar papan skor.
        sb.prep_score()
        sb.prep_high_score()
        sb.prep_level()
        sb.prep_ships()

        # Kosongkan daftar alien dan peluru.
        aliens.empty()
        bullets.empty()

        # Buat armada alien baru dan pusatkan kapal.
        create_fleet(ai_settings, screen, ship, aliens)
        ship.center_ship()
```

- Fungsi ini memulai permainan baru ketika tombol Play diklik.
- **play_button.rect.collidepoint(mouse_x, mouse_y)**: Memeriksa apakah klik mouse berada di area tombol Play.
- **ai_settings.initialize_dynamic_settings()**: Mengatur ulang pengaturan permainan.
- **pygame.mouse.set_visible(False)**: Menyembunyikan kursor mouse.
- **stats.reset_stats()**: Mengatur ulang statistik permainan.
- **create_fleet**: Membuat armada alien baru.
- **ship.center_ship()**: Memposisikan kapal di tengah layar.

6. Fungsi fire_bullet

```
● ● ●

def fire_bullet(ai_settings, screen, ship, bullets):
    """
    Menembakkan peluru jika batas belum tercapai.
    """
    # Periksa apakah jumlah peluru yang ada kurang dari batas.
    if len(bullets) < ai_settings.bullets_allowed:
        new_bullet = Bullet(ai_settings, screen, ship)
        bullets.add(new_bullet)
        se.bullet_sound.play()
```

- Fungsi ini menembakkan peluru jika batas jumlah peluru belum tercapai.

- **len(bullets) < ai_settings.bullets_allowed**: Memeriksa apakah jumlah peluru masih di bawah batas.
- **Bullet(ai_settings, screen, ship)**: Membuat objek peluru baru.
- **bullets.add(new_bullet)**: Menambahkan peluru baru ke grup peluru.
- **se.bullet_sound.play()**: Memainkan suara peluru yang ditembakkan.

7. Fungsi update_screen

```
● ● ●

# Fungsi untuk memperbarui tampilan layar.
def update_screen(ai_settings, screen, stats, sb, ship, aliens, bullets, play_button):
    """
    Memperbarui gambar di layar dan mengganti ke layar baru.
    """
    # Gambar ulang latar belakang.
    screen.blit(ai_settings.background_image, [0, 0])

    # Gambar ulang semua peluru di belakang kapal dan alien.
    for bullet in bullets.sprites():
        bullet.draw_bullet()
    ship.blitme()
    aliens.draw(screen)

    # Gambar informasi skor.
    sb.show_score()

    # Gambar tombol Play jika permainan tidak aktif.
    if not stats.game_active:
        play_button.draw_button()

    # Buat layar terbaru terlihat.
    pygame.display.flip()
```

- Fungsi ini memperbarui tampilan layar dan mengganti ke layar baru.
- **screen.blit(ai_settings.background_image, [0, 0])**: Menggambar latar belakang.
- **bullet.draw_bullet()**: Menggambar semua peluru di layar.
- **sb.show_score()**: Menampilkan skor saat ini.
- **pygame.display.flip()**: Memperbarui layar untuk menampilkan gambar terbaru.

8. Fungsi update_bullets

```
● ● ●

# Fungsi untuk memperbarui posisi peluru dan menghapus peluru yang sudah tidak terlihat.
def update_bullets(ai_settings, screen, stats, sb, ship, aliens, bullets):
    """
    Memperbarui posisi peluru dan menghapus peluru lama.
    """
    # Perbarui posisi setiap peluru.
    bullets.update()

    # Hapus peluru yang keluar dari layar.
    for bullet in bullets.copy():
        if bullet.rect.bottom <= 0:
            bullets.remove(bullet)

    # Periksa tabrakan peluru dengan alien.
    check_bullet_alien_collisions(ai_settings, screen, stats, sb, ship, aliens, bullets)
```

- Fungsi ini memperbarui posisi peluru dan menghapus peluru yang sudah tidak terlihat.
- **bullets.update()**: Memperbarui posisi semua peluru.

- **bullets.remove(bullet)**: Menghapus peluru yang keluar dari layar.
- **check_bullet_alien_collisions**: Memeriksa tabrakan antara peluru dan alien.

9. Fungsi check_high_score



```
# Fungsi untuk memeriksa dan memperbarui skor tertinggi.
def check_high_score(stats, sb):
    """
    Periksa apakah ada skor tertinggi baru.
    """
    if stats.score > stats.high_score:
        stats.high_score = stats.score
        sb.prep_high_score()
```

- Fungsi ini memeriksa apakah ada skor tertinggi baru.
- **stats.score > stats.high_score**: Memeriksa apakah skor saat ini lebih tinggi dari skor tertinggi.
- **sb.prep_high_score()**: Memperbarui tampilan skor tertinggi.

10. Fungsi check_bullet_alien_collisions



```
# Fungsi untuk menangani tabrakan antara peluru dan alien.
def check_bullet_alien_collisions(ai_settings, screen, stats, sb, ship, aliens, bullets):
    """
    Menangani tabrakan antara peluru dan alien.
    """
    # Periksa tabrakan antara peluru dan alien.
    collisions = pygame.sprite.groupcollide(bullets, aliens, True, True)

    if collisions:
        se.explosion_sound.play() # Mainkan efek suara ledakan.
        for aliens in collisions.values():
            # Tambahkan skor berdasarkan jumlah alien yang hancur.
            stats.score += ai_settings.alien_points * len(aliens)
            sb.prep_score()
            check_high_score(stats, sb)

    # Jika semua alien telah dihancurkan, tingkatkan level permainan.
    if len(aliens) == 0:
        bullets.empty() # Hapus semua peluru yang tersisa.
        ai_settings.increase_speed() # Tingkatkan kecepatan permainan.

        # Tingkatkan level.
        stats.level += 1
        sb.prep_level() # Perbarui tampilan level.

        # Buat armada alien baru.
        create_fleet(ai_settings, screen, ship, aliens)
```

- Fungsi ini menangani tabrakan antara peluru dan alien.
- **pygame.sprite.groupcollide(bullets, aliens, True, True)**: Memeriksa tabrakan dan menghapus peluru dan alien yang bertabrakan.
- **stats.score += ai_settings.alien_points * len(aliens)**: Menambahkan skor berdasarkan jumlah alien yang hancur.

- **create_fleet**: Membuat armada alien baru jika semua alien dihancurkan.

11. Fungsi check_fleet_edges

```
● ● ●

# Fungsi untuk memeriksa apakah ada alien yang mencapai tepi layar.
def check_fleet_edges(ai_settings, aliens):
    """
    Menanggapi jika ada alien yang mencapai tepi layar.
    """
    for alien in aliens.sprites():
        if alien.check_edges():
            change_fleet_direction(ai_settings, aliens) # Ubah arah armada alien.
            break
```

- Fungsi ini menanggapi jika ada alien yang mencapai tepi layar.
- **alien.check_edges()**: Memeriksa apakah alien mencapai tepi layar.
- **change_fleet_direction**: Mengubah arah armada alien jika ada yang mencapai tepi.

12. Fungsi change_fleet_direction

```
● ● ●

# Fungsi untuk mengubah arah armada alien.
def change_fleet_direction(ai_settings, aliens):
    """
    Menurunkan posisi armada dan mengubah arah gerakan mereka.
    """
    for alien in aliens.sprites():
        alien.rect.y += ai_settings.fleet_drop_speed # Turunkan posisi alien.
        ai_settings.fleet_direction *= -1 # Balik arah gerakan armada.
```

- Fungsi ini menurunkan posisi armada dan mengubah arah gerakan mereka.
- **alien.rect.y += ai_settings.fleet_drop_speed**: Menurunkan posisi alien.
- **ai_settings.fleet_direction *= -1**: Membalik arah gerakan armada.

13. Fungsi ship_hit

```

● ● ●

# Fungsi untuk menangani ketika kapal terkena alien.
def ship_hit(ai_settings, screen, stats, sb, ship, aliens, bullets):
    """
    Menanggapi situasi ketika kapal terkena oleh alien.
    """
    if stats.ships_left > 0:
        # Kurangi jumlah kapal yang tersisa.
        stats.ships_left -= 1

        # Perbarui tampilan jumlah kapal di papan skor.
        sb.prep_ships()
    else:
        # Jika tidak ada kapal tersisa, hentikan permainan.
        stats.game_active = False
        pygame.mouse.set_visible(True) # Tampilkan kursor mouse.

        # Kosongkan daftar alien dan peluru.
        aliens.empty()
        bullets.empty()

        # Buat armada alien baru dan pusatkan kapal.
        create_fleet(ai_settings, screen, ship, aliens)
        ship.center_ship()

        # Beri jeda sementara sebelum melanjutkan permainan.
        sleep(0.5)

```

- Fungsi ini menangani situasi ketika kapal terkena oleh alien.
- **stats.ships_left -= 1**: Mengurangi jumlah kapal yang tersisa.
- **stats.game_active = False**: Menghentikan permainan jika tidak ada kapal tersisa.
- **create_fleet**: Membuat armada alien baru setelah kapal terkena.

14. Fungsi check_aliens_bottom

```

● ● ●

# Fungsi untuk memeriksa apakah ada alien yang mencapai bagian bawah layar.
def check_aliens_bottom(ai_settings, screen, stats, sb, ship, aliens, bullets):
    """
    Periksa apakah ada alien yang mencapai dasar layar.
    Jika iya, perlakukan ini seperti kapal yang terkena.
    """
    screen_rect = screen.get_rect()
    for alien in aliens.sprites():
        if alien.rect.bottom >= screen_rect.bottom:
            # Perlakukan ini sama seperti kapal yang terkena.
            ship_hit(ai_settings, screen, stats, sb, ship, aliens, bullets)
            break

```

- Fungsi ini memeriksa apakah ada alien yang mencapai dasar layar.
- **alien.rect.bottom >= screen_rect.bottom**: Memeriksa apakah alien telah mencapai bagian bawah layar.
- **ship_hit**: Memanggil fungsi **ship_hit** jika ada alien yang mencapai dasar layar.

15. Fungsi update_aliens

```
● ● ●
```

```
# Fungsi untuk memperbarui posisi alien dan memeriksa tabrakan.
def update.aliens(ai_settings, screen, stats, sb, ship, aliens, bullets):
    """
    Perbarui posisi semua alien dan periksa tabrakan atau kejadian tertentu.
    """
    # Periksa apakah armada alien telah mencapai tepi layar.
    check_fleet_edges(ai_settings, aliens)
    aliens.update() # Perbarui posisi semua alien.

    # Periksa tabrakan antara kapal dan alien.
    if pygame.sprite.spritecollideany(ship, aliens):
        ship_hit(ai_settings, screen, stats, sb, ship, aliens, bullets)

    # Periksa apakah ada alien yang mencapai bagian bawah layar.
    check_aliens_bottom(ai_settings, screen, stats, sb, ship, aliens, bullets)
```

- Fungsi ini memperbarui posisi semua alien dan memeriksa tabrakan.
- **check_fleet_edges**: Memeriksa apakah armada alien telah mencapai tepi layar.
- **aliens.update()**: Memperbarui posisi semua alien.
- **pygame.sprite.spritecollideany(ship, aliens)**: Memeriksa tabrakan antara kapal dan alien.

16. Fungsi get_number_aliens_x

```
● ● ●
```

```
def get_number_aliens_x(ai_settings, alien_width):
    """
    Menentukan jumlah alien yang dapat muat dalam satu baris.
    """
    # Menghitung ruang horizontal yang tersedia untuk alien.
    available_space_x = ai_settings.screen_width - 2 * alien_width
    # Menghitung jumlah alien yang muat dalam ruang tersebut.
    number_aliens_x = int(available_space_x / (2 * alien_width))
    return number_aliens_x
```

- Fungsi ini menentukan jumlah alien yang dapat muat dalam satu baris.
- **available_space_x**: Menghitung ruang horizontal yang tersedia untuk alien.
- **number_aliens_x**: Menghitung jumlah alien yang muat dalam ruang tersebut.

17. Fungsi get_number_rows

```
● ● ●
```

```
def get_number_rows(ai_settings, ship_height, alien_height):
    """
    Menentukan jumlah baris alien yang dapat muat pada layar.
    """
    # Menghitung ruang vertikal yang tersedia untuk alien.
    available_space_y = ai_settings.screen_height - (3 * alien_height) - ship_height
    # Menghitung jumlah baris alien yang muat dalam ruang tersebut.
    number_rows = int(available_space_y / (2 * alien_height))
    return number_rows
```

- Fungsi ini menentukan jumlah baris alien yang dapat muat pada layar.
- **available_space_y**: Menghitung ruang vertikal yang tersedia untuk alien.

- **number_rows**: Menghitung jumlah baris alien yang muat dalam ruang tersebut.

18. Fungsi create_alien

```
● ● ●

def create_alien(ai_settings, screen, aliens, alien_number, row_number):
    """
    Membuat alien dan menempatkannya di posisi yang tepat dalam baris.
    """
    # Membuat objek alien baru.
    alien = Alien(ai_settings, screen)
    alien_width = alien.rect.width # Mendapatkan lebar alien.

    # Menentukan posisi horizontal alien dalam baris.
    alien.x = alien_width + 2 * alien_width * alien_number
    alien.rect.x = alien.x

    # Menentukan posisi vertikal alien dalam baris.
    alien.rect.y = alien.rect.height + 2 * alien.rect.height * row_number
    # Menambahkan alien ke grup aliens.
    aliens.add(alien)
```

- Fungsi ini membuat alien dan menempatkannya di posisi yang tepat dalam baris.
- **alien.x**: Menentukan posisi horizontal alien dalam baris.
- **alien.rect.y**: Menentukan posisi vertikal alien dalam baris.
- **aliens.add(alien)**: Menambahkan alien ke grup aliens.

19. Fungsi create_fleet

```
● ● ●

def create_fleet(ai_settings, screen, ship, aliens):
    """
    Membuat armada alien lengkap (baris dan kolom alien).
    """
    # Membuat alien pertama untuk menghitung berapa banyak alien yang bisa muat dalam satu baris dan satu kolom.
    alien = Alien(ai_settings, screen)

    # Menentukan jumlah alien yang muat dalam satu baris.
    number_aliens_x = get_number_aliens_x(ai_settings, alien.rect.width)

    # Menentukan jumlah baris alien.
    number_rows = get_number_rows(ai_settings, ship.rect.height, alien.rect.height)

    # Membuat armada alien (baris dan kolom).
    for row_number in range(number_rows):
        for alien_number in range(number_aliens_x):
            create_alien(ai_settings, screen, aliens, alien_number, row_number)
```

- Fungsi ini membuat armada alien lengkap (baris dan kolom alien).
- **get_number_aliens_x**: Menentukan jumlah alien yang muat dalam satu baris.
- **get_number_rows**: Menentukan jumlah baris alien.
- **create_alien**: Membuat alien untuk setiap posisi dalam armada.

Sound.py

1. Import Modul Pygame

```
# Import modul pygame untuk pengelolaan suara.  
import pygame
```

- **Fungsi:** Mengimpor modul **pygame** yang digunakan untuk pengelolaan suara, grafik, dan input dalam pengembangan game.
- **Detail:** Modul ini menyediakan berbagai fungsi dan kelas untuk memudahkan pembuatan game, termasuk pengelolaan suara.

2. Inisialisasi Mixer

```
# Inisialisasi mixer untuk menangani pemutaran suara.  
pygame.mixer.init()
```

- **Fungsi:** Menginisialisasi modul **mixer** dari **pygame** yang bertanggung jawab untuk menangani pemutaran suara.
- **Detail:** Sebelum menggunakan fungsi-fungsi yang berkaitan dengan suara, modul **mixer** harus diinisialisasi terlebih dahulu.

3. Memuat File Suara

```
# Memuat file suara untuk berbagai efek dalam permainan.  
bullet_sound = pygame.mixer.Sound('bullet_sound.wav') # Suara
```

- **Fungsi:** Memuat file suara dengan format **.wav** yang akan digunakan sebagai efek suara peluru.
- **Detail:**
 - **pygame.mixer.Sound()** digunakan untuk membuat objek suara dari file yang diberikan.
 - **bullet_sound** adalah variabel yang menyimpan objek suara tersebut.

```
# Memuat file suara untuk berbagai efek dalam permainan.  
explosion_sound = pygame.mixer.Sound('explosion.wav') # Suara ledakan.
```

- **Fungsi:** Memuat file suara dengan format **.wav** yang akan digunakan sebagai efek suara ledakan.
- **Detail:**

- **pygame.mixer.Sound()** digunakan untuk membuat objek suara dari file yang diberikan.
- **explosion_sound** adalah variabel yang menyimpan objek suara tersebut.

Link Github :

https://github.com/AndhikaAbdilahPrasetyo/UAS_PBO_Game-Skyline-Defender