LAPORAN UAS
MENGEMBANGKAN APLIKASI BERBASIS OOP

Mata Kuliah:

Pemrograman Berbasis Objek

Oleh:

Rafi Azis Rachman	(23091397185)
Rizky Amanda Sugiharto	(23091397195)
Samuel Alfredo Silla	(23091397209)

PROGRAM STUDI MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

GitHub : https://github.com/samuelalfredosilla/Flappy_Turtle

Flappy Turtle adalah sebuah game berbasis 2D yang mengadopsi konsep object-oriented programming (OOP). Dalam game ini, pemain mengontrol seekor kura-kura untuk melewati rintangan berupa pipa dan api, dengan penerapan empat pilar utama OOP. Konsep inheritance (pewarisan) diterapkan melalui kelas Turtle (kura-kura) yang mewarisi sifat dan metode dari kelas GameObject sebagai superclass. Objek lain seperti pipa, api dan Background juga merupakan turunan dari GameObject, sehingga dapat berbagi properti dasar seperti posisi, kecepatan, dan metode untuk menggambar di layar. Polimorfisme diterapkan dengan metode yang dioverride, seperti update(), yang memiliki logika berbeda pada setiap kelas turunan, misalnya gerakan terbang pada Turtle dan gerakan horizontal pada Pipes. Abstraksi diwujudkan melalui kelas abstrak GameObject, yang mendefinisikan properti dan metode dasar tanpa implementasi spesifik, memungkinkan kelas turunan seperti Turtle dan Pipes untuk memberikan detail implementasi masing-masing. Sementara itu, enkapsulasi diterapkan dengan menjadikan properti seperti posisi, kecepatan, dan skor bersifat privat, hanya dapat diakses melalui getter dan setter untuk menjaga integritas data. Dengan menerapkan keempat prinsip OOP ini, Flappy Turtle tidak hanya memberikan pengalaman bermain yang seru, tetapi juga menjadi contoh pembelajaran yang menarik dalam pengembangan game berbasis OOP.

STRUKTUR DATA YANG DIPAKAI GAME FLAPPY TURTLE

Pada kode di atas, beberapa **struktur data** digunakan untuk mengelola objek-objek dalam permainan. Salah satu struktur data utama yang digunakan adalah **list (daftar)**, yang berfungsi untuk menyimpan koleksi objek dalam permainan, seperti daftar pipa (self.pipa), api (self.api), dan koin (self.koin). Daftar ini memungkinkan penambahan dan penghapusan objek secara dinamis, misalnya saat pipa, api, atau koin bergerak keluar dari layar dan dihapus, serta objek baru ditambahkan. Selain itu, objek-objek dalam permainan menggunakan **pygame.Rect**, yang merupakan objek berbentuk persegi panjang, untuk mendeteksi tabrakan antar objek. Setiap objek, seperti Turtle, pipa, koin, dan api, memiliki **pygame.Rect** untuk menggambarkan area yang mencakup objek tersebut, sehingga dapat dilakukan pengecekan tabrakan menggunakan metode collliderect(). Dengan menggunakan list dan pygame.Rect, kode ini dapat dengan mudah mengelola objek-objek dalam permainan serta mendeteksi interaksi di antara mereka.

BERIKUT ADALAH KONSEP OOP YANG DITERAPKAN DIAGRAM CLASS

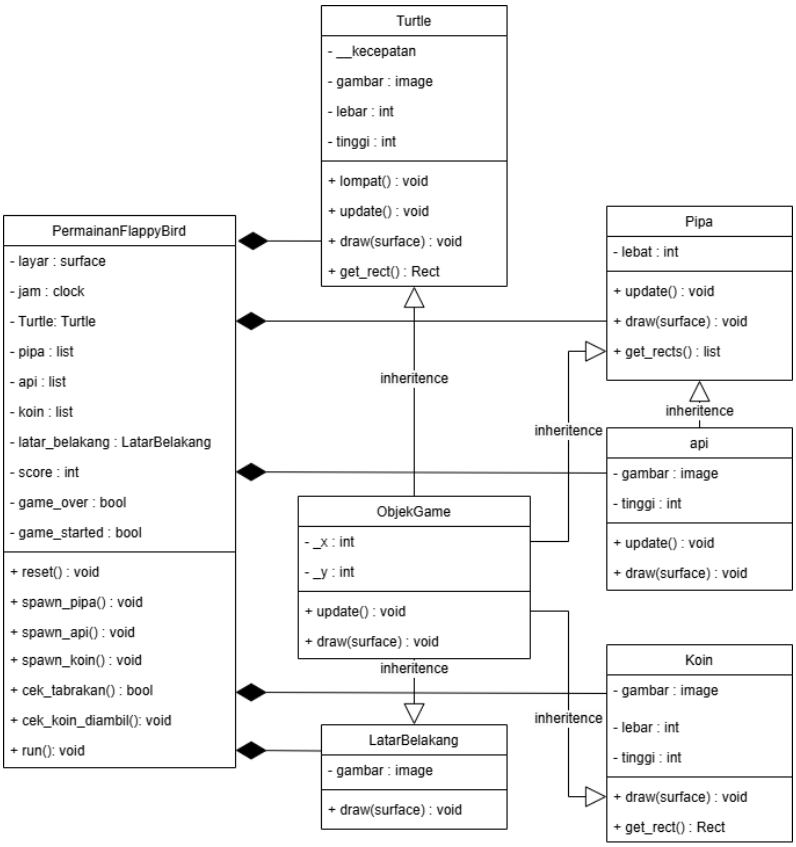


Diagram kelas ini menggambarkan struktur permainan Flappy Turtle secara detail. Permainan ini terdiri dari beberapa kelas utama, yaitu PermainanFlappyBird yang bertindak sebagai kelas utama pengontrol permainan, ObjekGame sebagai kelas dasar untuk semua objek yang bergerak dalam permainan seperti burung, pipa, dan koin, serta kelas-kelas turunan lainnya seperti Turtle, Pipa, Api, Koin, dan LatarBelakang. Kelas PermainanFlappyBird mengelola seluruh elemen permainan, termasuk skor, waktu, dan kondisi permainan (misalnya, apakah permainan sudah selesai). Kelas-kelas turunan dari ObjekGame memiliki atribut dan metode spesifik untuk masing-masing objek, seperti posisi, gambar, dan cara pergerakan. Hubungan antar kelas menunjukkan bagaimana objek-objek ini berinteraksi satu sama lain dalam permainan, misalnya burung berinteraksi dengan pipa untuk mendeteksi tabrakan atau dengan koin untuk mengumpulkan poin.

ALUR APLIKASI CODINGAN GAME FLAPPY TURTLE

```python
import pygame
import random

# Konstanta untuk ukuran layar, gravitasi, kecepatan pipa, kecepatan api, dan interval spawn koin
LEBAR_LAYAR = 1000
TINGGI_LAYAR = 660
GRAVITASI = 0.7
KECEPATAN_PIPA = 4
KECEPATAN_API = 10  # Kecepatan Api lebih cepat dari Pipa
SPAWN_KOIN_INTERVAL = 1500  # Interval waktu untuk spawn koin dalam milidetik
```

Kode di atas merupakan bagian dari skrip permainan yang menggunakan pustaka **Pygame**. Konstanta-konstanta yang didefinisikan mengatur elemen-elemen inti dalam permainan. Variabel LEBAR_LAYAR dan TINGGI_LAYAR menentukan ukuran layar permainan, yaitu lebar 1000 piksel dan tinggi 660 piksel. Konstanta GRAVITASI digunakan untuk mensimulasikan gaya gravitasi dalam permainan, dengan nilai 0.7 yang memengaruhi kecepatan jatuh objek. Selanjutnya, KECEPATAN_PIPA mengatur kecepatan gerakan pipa yang bergerak ke arah pemain, yaitu sebesar 4 piksel per frame, sedangkan KECEPATAN_API mengatur kecepatan elemen api dengan nilai 10, lebih cepat dibandingkan kecepatan pipa. Akhirnya, SPAWN_KOIN_INTERVAL adalah interval waktu dalam milidetik (1500 ms atau 1.5 detik) untuk menampilkan koin secara periodik di layar permainan. Semua konstanta ini dirancang untuk memberikan dinamika gameplay yang menarik dan menantang.

```python
# Kelas dasar untuk objek game
class ObjekGame:
    def __init__(self, x, y):
        self._x = x  # Atribut protected untuk posisi X
        self._y = y  # Atribut protected untuk posisi Y

    def update(self):
        pass  # Metode untuk pembaruan (override di kelas turunan)

    def draw(self, surface):
        pass  # Metode untuk menggambar objek (override di kelas turunan)
```

Kode di atas mendefinisikan kelas dasar bernama **ObjekGame**, yang berfungsi sebagai blueprint untuk objek-objek dalam permainan. Kelas ini memiliki konstruktor __init__ yang digunakan untuk menginisialisasi posisi awal objek di layar melalui atribut *protected* _x dan _y. Atribut ini diberi garis bawah sebagai tanda bahwa mereka dimaksudkan untuk digunakan secara internal oleh kelas tersebut atau kelas turunannya. Selain itu, terdapat dua metode utama, yaitu update dan draw, yang dibiarkan kosong untuk di-*override* oleh kelas turunan. Metode update digunakan untuk memperbarui status atau posisi objek selama permainan, sedangkan metode draw bertanggung jawab untuk menggambar objek pada layar menggunakan pustaka Pygame, dengan parameter surface sebagai permukaan tempat objek akan digambar.

Kode ini menerapkan beberapa konsep utama dalam **Object-Oriented Programming (OOP)**. Konsep **encapsulation** diterapkan melalui atribut *protected* _x dan _y, yang melindungi data agar tidak diakses langsung dari luar. **Inheritance** terlihat dari desain kelas

ini yang dirancang untuk diwarisi oleh kelas turunan, sehingga atribut dan metode dasar dapat digunakan kembali. **Polymorphism** diwujudkan melalui metode update dan draw, yang memungkinkan kelas turunan mengimplementasikan perilaku berbeda meskipun antarmuka metode tetap sama. Selain itu, konsep **abstraction** diterapkan dengan menyediakan kerangka kerja umum tanpa implementasi spesifik, sehingga memaksa kelas turunan untuk mengisi logika yang sesuai. Dengan desain seperti ini, kode menjadi modular, mudah diperluas, dan mendukung pengembangan berbagai jenis objek permainan tanpa harus mengulang kode dasar.

```python
# Kelas untuk Turtle (karakter utama)
class Turtle(ObjekGame):
    def __init__(self, x, y, gambar_path, lebar=None, tinggi=None):
        super().__init__(x, y)
        self.__kecepatan = 0  # Atribut private untuk kecepatan Turtle
        self.gambar = pygame.image.load(gambar_path).convert_alpha()  # Memuat gambar Turtle

        # Mengatur ukuran gambar jika diberikan
        if lebar and tinggi:
            self.gambar = pygame.transform.scale(self.gambar, (lebar, tinggi))

        self.lebar = self.gambar.get_width()  # Lebar Turtle
        self.tinggi = self.gambar.get_height()  # Tinggi Turtle

    def lompat(self):
        self.__kecepatan = -10  # Memberikan kecepatan ke atas saat Turtle melompat

    def update(self):
        self.__kecepatan += GRAVITASI  # Menambahkan gravitasi ke kecepatan Turtle
        self._y += self.__kecepatan  # Memperbarui posisi Turtle berdasarkan kecepatan

    def draw(self, surface):
        surface.blit(self.gambar, (self._x, int(self._y)))  # Menggambar Turtle pada layar

    def get_rect(self):
        return pygame.Rect(self._x, self._y, self.lebar, self.tinggi)  # Mendapatkan bounding box Turtle
```

Kode di atas mendefinisikan kelas **Turtle** yang merupakan turunan dari kelas **ObjekGame**. Kelas ini digunakan untuk merepresentasikan karakter utama dalam permainan dengan atribut dan perilaku spesifik. Konstruktor __init__ memanfaatkan fungsi super() untuk menginisialisasi atribut dasar dari kelas induk, kemudian menambahkan atribut tambahan seperti __kecepatan yang bersifat *private* untuk mengatur kecepatan gerak karakter. Selain itu, gambar karakter dimuat melalui parameter gambar_path menggunakan fungsi pygame.image.load, dan ukurannya dapat disesuaikan dengan parameter opsional lebar dan tinggi menggunakan pygame.transform.scale. Lebar dan tinggi gambar juga disimpan sebagai atribut untuk mempermudah penghitungan.

Kelas ini menyediakan metode lompat untuk mengatur kecepatan vertikal karakter menjadi negatif, memungkinkan karakter melompat ke atas. Metode update digunakan untuk memperbarui posisi karakter berdasarkan gravitasi yang menambah kecepatan vertikalnya secara bertahap. Posisi vertikal Turtle diperbarui dengan menambahkan kecepatan tersebut ke koordinat y. Metode draw bertanggung jawab untuk menggambar karakter pada layar menggunakan fungsi blit, sedangkan metode get_rect mengembalikan objek pygame.Rect yang merepresentasikan *bounding box* karakter, berguna untuk deteksi tabrakan.

Kode ini menerapkan beberapa konsep penting dalam **Object-Oriented Programming (OOP)**. **Inheritance** diterapkan dengan menjadikan Turtle sebagai turunan dari ObjekGame, memungkinkan penggunaan kembali atribut dan metode dasar. **Encapsulation** digunakan melalui atribut *private* __kecepatan untuk melindungi data dari akses langsung luar kelas, sedangkan atribut *protected* _x dan _y diwarisi dari kelas induk. **Polymorphism** terlihat pada metode update dan draw yang di-*override* untuk memberikan

perilaku spesifik pada kelas Turtle. Desain ini juga menunjukkan prinsip **abstraction** dengan membangun logika spesifik di kelas turunan berdasarkan kerangka kerja yang disediakan oleh kelas dasar, menciptakan kode yang modular, terorganisir, dan mudah diperluas.

```python
# Kelas untuk pipa (rintangan)
class Pipa(ObjekGame):
    def __init__(self, x, tinggi):
        super().__init__(x, tinggi)
        self.lebar = 50  # Lebar pipa

    def update(self):
        self._x -= KECEPATAN_PIPA  # Memindahkan pipa ke kiri

    def draw(self, surface):
        # Menggambar pipa atas dan bawah
        pygame.draw.rect(surface, (0, 255, 0), (self._x, 0, self.lebar, self._y))  # Pipa atas
        pygame.draw.rect(surface, (0, 255, 0), (self._x, self._y + 150, self.lebar, TINGGI_LAYAR - self._y - 150))  # Pipa bawah

    def get_rects(self):
        # Mendapatkan bounding box untuk pipa atas dan bawah
        return [
            pygame.Rect(self._x, 0, self.lebar, self._y),  # Pipa atas
            pygame.Rect(self._x, self._y + 150, self.lebar, TINGGI_LAYAR - self._y - 150)  # Pipa bawah
        ]
```

Kode di atas mendefinisikan kelas **Pipa**, yang merupakan turunan dari kelas **ObjekGame** dan berfungsi sebagai rintangan dalam permainan. Konstruktor __init__ memanfaatkan fungsi super() untuk menginisialisasi posisi awal pipa pada sumbu x dan ketinggian tertentu pada sumbu y. Kelas ini juga memiliki atribut tambahan lebar yang ditetapkan secara langsung dengan nilai 50, mewakili lebar pipa. Metode update digunakan untuk menggerakkan pipa ke kiri dengan mengurangi nilai koordinat _x sebesar kecepatan yang ditentukan oleh konstanta KECEPATAN_PIPA. Metode draw bertanggung jawab untuk menggambar pipa atas dan bawah di layar menggunakan fungsi pygame.draw.rect, di mana pipa atas digambar dari koordinat atas layar hingga nilai _y, dan pipa bawah digambar mulai dari jarak 150 piksel di bawah _y hingga bagian bawah layar. Metode get_rects mengembalikan daftar objek pygame.Rect yang merepresentasikan *bounding box* dari pipa atas dan bawah, berguna untuk mendeteksi tabrakan dengan objek lain dalam permainan.

Kode ini mengimplementasikan beberapa konsep **Object-Oriented Programming (OOP)**. **Inheritance** diterapkan dengan menjadikan Pipa sebagai turunan dari ObjekGame, sehingga mewarisi atribut dan metode dasar seperti _x dan _y. **Polymorphism** terlihat pada metode update dan draw yang di-*override* untuk memberikan perilaku unik, yaitu pergerakan horizontal dan penggambaran pipa. **Encapsulation** diterapkan melalui penggunaan atribut *protected* _x dan _y, yang melindungi data posisi dari akses langsung luar kelas. Selain itu, kode ini menunjukkan prinsip **abstraction**, di mana logika khusus untuk rintangan (pipa) diimplementasikan dalam kelas Pipa, sementara kerangka dasar disediakan oleh kelas induknya. Desain ini membuat kode lebih modular dan fleksibel untuk menambahkan elemen permainan lainnya.

```python
# Kelas untuk koin (item yang dapat diambil)
class Koin(ObjekGame):
    def __init__(self, x, y, gambar_path, lebar=None, tinggi=None):
        super().__init__(x, y)
        self.gambar = pygame.image.load(gambar_path).convert_alpha()  # Memuat gambar koin

        # Mengatur ukuran gambar jika diberikan
        if lebar and tinggi:
            self.gambar = pygame.transform.scale(self.gambar, (lebar, tinggi))

        self.lebar = self.gambar.get_width()  # Lebar koin
        self.tinggi = self.gambar.get_height()  # Tinggi koin

    def draw(self, surface):
        surface.blit(self.gambar, (self._x, self._y))  # Menggambar koin pada layar

    def get_rect(self):
        return pygame.Rect(self._x, self._y, self.lebar, self.tinggi)  # Mendapatkan bounding box koin
```

Kode di atas mendefinisikan kelas **Koin**, yang merupakan turunan dari kelas **ObjekGame**, dan digunakan untuk merepresentasikan elemen koin dalam permainan. Konstruktor __init__ memanfaatkan fungsi super() untuk menginisialisasi posisi awal koin pada sumbu x dan y, serta memuat gambar koin dari file yang diberikan melalui parameter gambar_path menggunakan fungsi pygame.image.load. Jika parameter opsional lebar dan tinggi disediakan, gambar koin akan diubah ukurannya menggunakan pygame.transform.scale. Lebar dan tinggi koin kemudian disimpan dalam atribut lebar dan tinggi menggunakan metode get_width dan get_height dari gambar. Metode draw bertanggung jawab untuk menggambar koin di layar pada posisi yang sesuai, dengan menggunakan fungsi blit. Selain itu, metode get_rect mengembalikan objek pygame.Rect yang mewakili *bounding box* koin, berguna untuk deteksi tabrakan.

Kode ini menerapkan beberapa konsep **Object-Oriented Programming (OOP)**. **Inheritance** diterapkan dengan menjadikan Koin sebagai turunan dari ObjekGame, sehingga mewarisi atribut dan metode dasar seperti _x, _y, dan draw. **Encapsulation** digunakan dengan atribut *protected* _x dan _y, yang melindungi data posisi koin dari akses langsung luar kelas. **Polymorphism** diwujudkan dengan meng-*override* metode draw untuk menggambarkan koin secara khusus menggunakan gambar. Desain ini juga mencerminkan prinsip **abstraction**, di mana logika spesifik untuk elemen koin, seperti pengaturan gambar dan deteksi tabrakan, diimplementasikan dalam kelas ini, sementara kerangka dasar disediakan oleh kelas induk. Dengan pendekatan ini, kode menjadi modular, fleksibel, dan mudah diperluas untuk elemen permainan lainnya.

```python
# Kelas untuk api (rintangan tambahan dengan kecepatan lebih tinggi)
class Api(Pipa):
    def __init__(self, x, tinggi, gambar_path, lebar=None, tinggi_api=None):
        super().__init__(x, tinggi)
        self.gambar = pygame.image.load(gambar_path).convert_alpha()  # Memuat gambar api

        # Mengatur ukuran gambar jika diberikan
        if lebar and tinggi_api:
            self.gambar = pygame.transform.scale(self.gambar, (lebar, tinggi_api))

        self.lebar = self.gambar.get_width()  # Lebar gambar api
        self.tinggi = self.gambar.get_height()  # Tinggi gambar api

    def update(self):
        self._x -= KECEPATAN_API  # Memindahkan api ke kiri dengan kecepatan lebih tinggi

    def draw(self, surface):
        surface.blit(self.gambar, (self._x, self._y))  # Menggambar api pada layar
```

Kode di atas mendefinisikan kelas **Api**, yang merupakan turunan dari kelas **Pipa**, digunakan untuk merepresentasikan elemen api sebagai rintangan yang bergerak lebih cepat dalam permainan. Konstruktor __init__ memanfaatkan fungsi super() untuk menginisialisasi posisi awal dan atribut dasar dari kelas Pipa. Selain itu, gambar api dimuat melalui parameter gambar_path menggunakan fungsi pygame.image.load. Jika parameter opsional lebar dan tinggi_api diberikan, gambar api akan diubah ukurannya menggunakan pygame.transform.scale. Atribut lebar dan tinggi kemudian disimpan berdasarkan dimensi gambar yang telah dimuat. Metode update di-*override* untuk memperbarui posisi horizontal api, yang bergerak ke kiri dengan kecepatan lebih tinggi dibandingkan pipa, menggunakan konstanta KECEPATAN_API. Metode draw bertugas menggambar gambar api di layar pada posisi yang sesuai dengan menggunakan fungsi blit.

Kode ini menerapkan beberapa konsep **Object-Oriented Programming (OOP)**. **Inheritance** diterapkan dengan menjadikan Api sebagai turunan dari Pipa, yang pada gilirannya adalah turunan dari ObjekGame. Dengan demikian, Api mewarisi atribut seperti _x dan _y, serta metode dasar seperti update dan draw, tetapi juga dapat menambahkan atau memodifikasi perilaku sesuai kebutuhan. **Polymorphism** terlihat pada metode update dan draw, yang di-*override* untuk memberikan logika spesifik bagi kelas Api, seperti pergerakan lebih cepat dan penggambaran dengan gambar. **Encapsulation** diterapkan melalui atribut *protected* _x dan _y, yang membatasi akses langsung ke atribut posisi dari luar kelas. Kelas ini juga menunjukkan **abstraction**, dengan fokus pada logika khusus untuk elemen api sementara kerangka dasar disediakan oleh kelas induk. Desain ini menciptakan kode yang modular dan fleksibel, memudahkan integrasi elemen-elemen permainan yang berbeda.

```python
# Kelas untuk latar belakang permainan
class LatarBelakang(ObjekGame):
    def __init__(self, gambar_path):
        super().__init__(0, 0)
        self.gambar = pygame.image.load(gambar_path).convert()  # Memuat gambar latar belakang

    def draw(self, surface):
        surface.blit(self.gambar, (0, 0))  # Menggambar latar belakang pada layar
```

Kode di atas mendefinisikan sebuah kelas LatarBelakang yang merupakan subclass dari kelas ObjekGame. Kelas ini bertujuan untuk menggambar latar belakang pada permainan. Pada konstruktor (__init__), kelas LatarBelakang menerima parameter gambar_path yang berfungsi untuk memuat gambar latar belakang menggunakan pustaka pygame. Fungsi super().__init__(0, 0) memanggil konstruktor dari kelas induk (ObjekGame) untuk menginisialisasi posisi objek (dalam hal ini, posisi (0, 0)), meskipun posisi objek ini tidak digunakan dalam kelas LatarBelakang. Dalam metode draw, gambar latar belakang yang telah dimuat digambar ke layar menggunakan metode blit dari pustaka pygame, yang memindahkan gambar ke koordinat (0, 0) pada permukaan (surface).

Dari sudut pandang **konsep OOP (Object-Oriented Programming)**, kode ini mengilustrasikan beberapa prinsip dasar OOP, seperti **pewarisan** (inheritance) dan **polimorfisme** (polymorphism). Kelas LatarBelakang mewarisi sifat dan metode dari kelas ObjekGame, yang memungkinkan LatarBelakang untuk menggunakan atau mengubah perilaku yang sudah ada di kelas induknya. Misalnya, jika kelas ObjekGame memiliki atribut

atau metode lain, kelas LatarBelakang dapat mengaksesnya tanpa perlu mendefinisikannya ulang. Polimorfisme terlihat dalam penggunaan metode draw, yang bisa memiliki implementasi berbeda tergantung jenis objek yang memanggilnya. Dengan pendekatan ini, kode menjadi lebih modular, mudah diperluas, dan lebih efisien dalam mengelola objek dan fungsi yang saling berhubungan.

```python
# Kelas utama untuk permainan
class PermainanFlappyBird:
    def __init__(self):
        pygame.init()  # Inisialisasi pygame
        self.layar = pygame.display.set_mode((LEBAR_LAYAR, TINGGI_LAYAR))  # Membuat layar permainan
        self.jam = pygame.time.Clock()  # Untuk mengatur frame rate
        # Membuat objek permainan
        self.Turtle = Turtle(100, 300, "turtle.png", lebar=50, tinggi=50)
        self.pipa = []  # Daftar pipa
        self.api = []  # Daftar api
        self.koin = []  # Daftar koin
        self.latar_belakang = LatarBelakang("sky3.jpg")  # Gambar latar belakang
        self.score = 0  # Skor awal
        self.spawn_pipa()  # Spawn pipa pertama
        self.spawn_api()  # Spawn api pertama
        self.last_koin_spawn_time = pygame.time.get_ticks()  # Waktu terakhir koin spawn
        self.game_over = False  # Status game over
        self.game_started = False  # Status permainan dimulai

    def reset(self):
        # Reset status permainan ke awal
        self.__init__()

    def spawn_pipa(self):
        # Menambahkan pipa baru ke dalam daftar
        tinggi = random.randint(100, 400)
        self.pipa.append(Pipa(LEBAR_LAYAR, tinggi))

    def spawn_api(self):
        # Menambahkan api baru ke dalam daftar
        tinggi = random.randint(0, TINGGI_LAYAR - 20)
        self.api.append(Api(LEBAR_LAYAR, tinggi, "fire.png", lebar=40, tinggi_api=40))

    def spawn_koin(self):
        # Menambahkan koin baru ke dalam daftar
        x = LEBAR_LAYAR
        y = random.randint(50, TINGGI_LAYAR - 50)
        self.koin.append(Koin(x, y, "koin.png", lebar=40, tinggi=40))

    def cek_tabrakan(self):
        # Mengecek apakah Turtle bertabrakan dengan pipa atau api
        for pipa in self.pipa:
            for rect in pipa.get_rects():
                if self.Turtle.get_rect().colliderect(rect):
                    return True
        for api in self.api:
            if self.Turtle.get_rect().colliderect(pygame.Rect(api._x, api._y, api.lebar, api.tinggi)):
                return True
        return False

    def cek_koin_diambil(self):
        # Mengecek apakah Turtle mengambil koin
        for koin in self.koin:
            if self.Turtle.get_rect().colliderect(koin.get_rect()):
                self.koin.remove(koin)  # Menghapus koin dari daftar
                self.score += 1  # Menambahkan skor

    def run(self):
        berjalan = True  # Variabel untuk menentukan apakah permainan berjalan
        while berjalan:  # Loop utama permainan
            # Mengolah event dari pengguna
            for event in pygame.event.get():
                if event.type == pygame.QUIT:  # Jika pengguna menutup jendela
                    berjalan = False
                if event.type == pygame.KEYDOWN:  # Jika tombol ditekan
                    if event.key == pygame.K_SPACE:  # Jika tombol SPACE ditekan
                        if not self.game_started:  # Jika permainan belum dimulai
                            self.game_started = True  # Memulai permainan
                        elif self.game_over:  # Jika permainan telah berakhir
                            self.reset()  # Mengatur ulang permainan
                        else:  # Jika permainan sedang berlangsung
                            self.Turtle.lompat()  # Membuat Turtle melompat

            if self.game_started:  # Jika permainan telah dimulai
                if not self.game_over:  # Jika permainan belum berakhir
                    self.Turtle.update()  # Mengupdate posisi Turtle
                    # Mengupdate posisi pipa
                    for pipa in self.pipa:
                        pipa.update()
                        if pipa._x < -pipa.lebar:  # Jika pipa keluar dari layar
                            self.pipa.remove(pipa)  # Menghapus pipa
                            self.spawn_pipa()  # Menambah pipa baru
                    # Mengupdate posisi api
                    for api in self.api:
                        api.update()
                        if api._x < -api.lebar:  # Jika api keluar dari layar
                            self.api.remove(api)  # Menghapus api
                            self.spawn_api()  # Menambah api baru
                    # Mengatur spawn koin berdasarkan waktu
                    current_time = pygame.time.get_ticks()  # Waktu sekarang
                    if current_time - self.last_koin_spawn_time > SPAWN_KOIN_INTERVAL:
                        self.spawn_koin()  # Menambah koin baru
                        self.last_koin_spawn_time = current_time
                    # Mengupdate posisi koin
                    for koin in self.koin:
                        koin._x -= KECEPATAN_PIPA  # Menggerakkan koin ke kiri
                        if koin._x < -koin.lebar:  # Jika koin keluar dari layar
                            self.koin.remove(koin)  # Menghapus koin
                    self.cek_koin_diambil()  # Mengecek apakah Turtle mengambil koin
                    # Mengecek apakah Turtle jatuh ke bawah layar
                    if self.Turtle._y + self.Turtle.tinggi >= TINGGI_LAYAR:
                        self.Turtle._y = 0  # Mengatur ulang posisi Turtle
                    if self.cek_tabrakan():  # Mengecek apakah terjadi tabrakan
                        self.game_over = True  # Mengakhiri permainan
                # Menggambar latar belakang dan semua objek permainan
                self.layar.fill((135, 206, 235))  # Membersihkan layar
                self.latar_belakang.draw(self.layar)  # Menggambar latar belakang
                self.Turtle.draw(self.layar)  # Menggambar Turtle
                for pipa in self.pipa:
                    pipa.draw(self.layar)  # Menggambar semua pipa
                for api in self.api:
                    api.draw(self.layar)  # Menggambar semua api
                for koin in self.koin:
                    koin.draw(self.layar)  # Menggambar semua koin
                # Menampilkan skor pada layar
                font = pygame.font.Font(None, 36)
                text = font.render(f"Score: {self.score}", True, (255, 255, 255))
                self.layar.blit(text, (10, 10))  # Menampilkan skor di pojok kiri atas
                if self.game_over:  # Jika permainan berakhir
                    font = pygame.font.Font(None, 74)
                    text = font.render("Game Over", True, (255, 0, 0))
                    self.layar.blit(text, (LEBAR_LAYAR // 2.86, TINGGI_LAYAR // 2.5))  # Menampilkan teks "Game Over"
                    font_small = pygame.font.Font(None, 36)
                    text_restart = font_small.render("Press Space to Restart", True, (255, 255, 255))
                    self.layar.blit(text_restart, (LEBAR_LAYAR // 2.8, TINGGI_LAYAR // 2))  # Menampilkan teks restart
            else:
                # Jika permainan belum dimulai, menampilkan pesan "Press Space to Start"
                font = pygame.font.Font(None, 74)
                text = font.render("Press Space to Start", True, (255, 255, 255))
                self.layar.blit(text, (LEBAR_LAYAR // 4, TINGGI_LAYAR // 2))
            pygame.display.flip()  # Memperbarui layar
            self.jam.tick(60)  # Menjaga kecepatan frame tetap stabil
        pygame.quit()  # Mengakhiri permainan dan keluar dari loop utama
```

Kode di atas mendefinisikan kelas PermainanFlappyBird yang menyusun logika permainan Flappy Bird dengan menggunakan pustaka pygame. Pada konstruktor (__init__), objek permainan diinisialisasi, termasuk layar permainan, objek Turtle, dan daftar objek seperti pipa, api, dan koin. Setiap objek permainan ini memiliki posisi dan gambar yang spesifik. Metode seperti spawn_pipa, spawn_api, dan spawn_koin digunakan untuk menambahkan objek baru ke dalam permainan secara acak. Metode cek_tabrakan dan cek_koin_diambil digunakan untuk memeriksa apakah Turtle bertabrakan dengan objek lain atau mengambil koin. Loop utama permainan ada di dalam metode run, di mana setiap frame permainan diperbarui, termasuk event handling, pembaruan posisi objek, pengecekan kondisi permainan (seperti tabrakan dan pengambilan koin), serta menggambar semua objek ke layar. Jika permainan berakhir, pesan "Game Over" akan ditampilkan dan pengguna dapat memulai ulang permainan dengan menekan tombol space.

Dari perspektif **Object-Oriented Programming (OOP)**, kode ini menerapkan prinsip-prinsip seperti **modularitas**, **abstraksi**, **pewarisan**, dan **encapsulation**. Kelas PermainanFlappyBird menjadi kelas utama yang mengelola semua aspek permainan, sedangkan objek-objek seperti Turtle, Pipa, Api, dan Koin diharapkan merupakan subclass dari kelas dasar ObjekGame atau kelas serupa, yang masing-masing memiliki tanggung jawab dan logika yang terpisah. Penggunaan **pewarisan** memungkinkan pengelompokan logika yang lebih baik, seperti metode draw atau update yang mungkin digunakan di berbagai kelas. **Encapsulation** terlihat dalam cara objek-objek seperti Turtle atau Pipa menyembunyikan implementasi internal dan hanya mengekspos antarmuka yang diperlukan untuk permainan, misalnya melalui metode draw atau update. Dengan pendekatan ini, kode lebih mudah untuk dikelola, diperluas, dan dimodifikasi tanpa mempengaruhi bagian lainnya, karena setiap objek memiliki tanggung jawab yang terpisah dan spesifik.

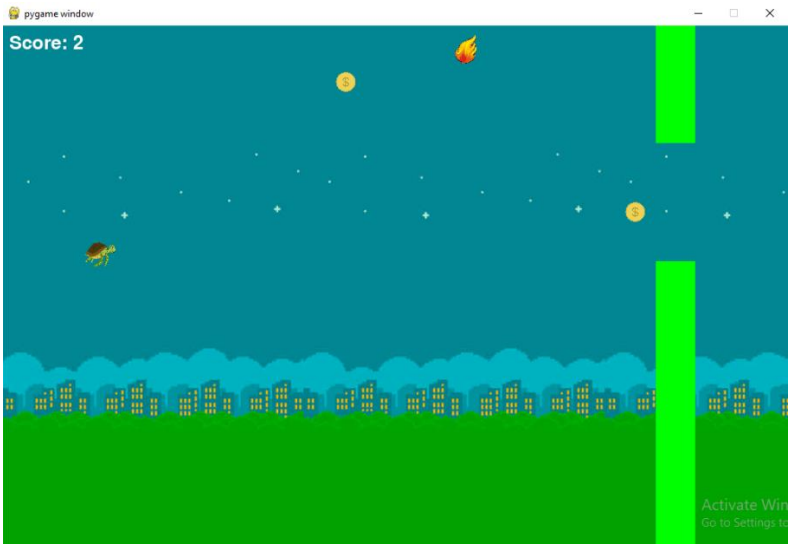
Kode di atas menggunakan konstruksi Python if __name__ == "__main__": yang memastikan bahwa bagian kode tersebut hanya dijalankan jika file tersebut dieksekusi sebagai program utama, bukan saat diimpor sebagai modul. Di dalam blok tersebut, pertama-tama sebuah instance dari kelas PermainanFlappyBird dibuat dengan nama permainan. Ini berarti objek permainan baru dibuat dan siap untuk berinteraksi dengan kelas dan metode yang ada di dalamnya. Setelah itu, metode run() dari objek permainan dipanggil untuk menjalankan permainan. Metode run() mengandung logika utama permainan, termasuk pengelolaan event, pembaruan objek, pengecekan kondisi permainan, dan rendering grafik ke layar.

Dalam konteks **Object-Oriented Programming (OOP)**, kode ini menunjukkan penggunaan **instansiasi objek** dan **panggilan metode**. Ketika kita membuat objek permainan, kita menginstansiasi kelas PermainanFlappyBird, yang berarti kita menciptakan salinan objek dengan atribut dan metode yang didefinisikan dalam kelas tersebut. Setelah

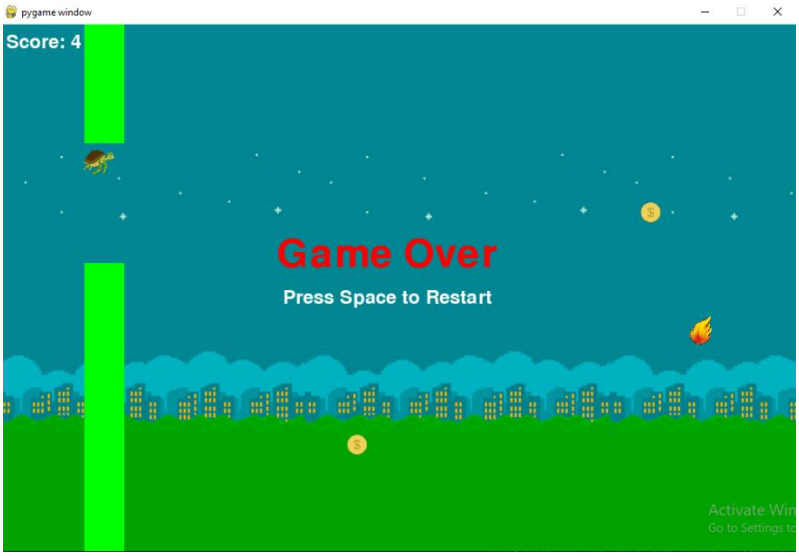
objek dibuat, kita memanggil metode run() yang merupakan aksi spesifik dari objek tersebut untuk memulai eksekusi permainan. Ini adalah contoh **encapsulation** dalam OOP, di mana logika permainan (seperti loop utama, pengelolaan status permainan, dan rendering grafik) disembunyikan dalam metode run() dan hanya perlu dipanggil untuk menjalankan permainan. Kode ini juga memperlihatkan prinsip **modularitas**, karena logika permainan dipecah menjadi kelas dan metode yang terpisah, memudahkan pengelolaan dan pengembangan lebih lanjut.

PENJELASAN FITUR DAN CARA PRNGGUNAAN APLIKASI.

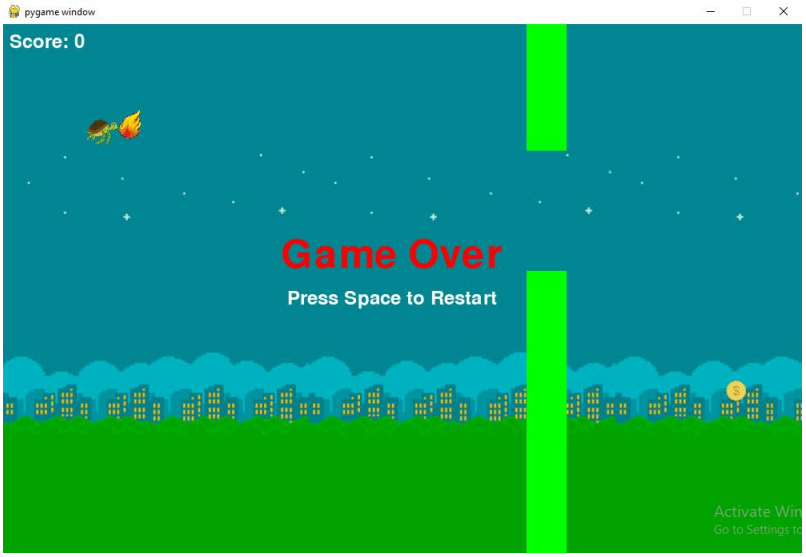
Gambar pertama diatas merupakan tampilan awal ketika game dijalankan



Setelah player menekan tompol spasi game akan berjalan seperti gambar diatas, game digerakan dengan menggunakan spasi, ketika player mengeklik tombol spasi turtle akan melompat, ketika turtle menyentuh 1 koin, player akan mendapatkan score 1.



Ketika player turtle(Kura – kura) menyentuh pipa game akan berakhir (Game Over)



Ketika turtle menyentuh api, game juga akan berakhir(Game Over), disitu dijelaskan bahwa player bisa menekan spasi untuk Kembali bermain, jika player ingin mengakhiri permainan,maka player tinggal klik tombol close(x) yang berada di pojok kanan atas pada layar permainan.