PEMROGRAMAN BERORIENTASI OBJEK

Tugas Akhir Pembuatan game *Memory Match*

Dosen Pengampu:

1. Binti Kholifah, S.Kom., M.Tr.Kom.
2. Dimas Novian Aditia Syahputra, S.Tr.T., M.Tr.T.
3. I Gde Agung Sri Sidhimantra, S.Kom., M.Kom.
4. Moch Deny Pratama, S.Tr.Kom., M.Kom.

Disusun Oleh:

1. Bathari Nafira Husna (23091397123)
2. Muhammad Ilham Bintang (23091397129)
3. Adeline Aurel Yafi Septania (23091397130)

Program Studi D4 Manajemen Informatika
Fakultas Vokasi
Universitas Negeri Surabaya
2024

DAFTAR ISI

BAB I
PENDAHULUAN

Link github: https://github.com/TanggBintang/PBO_Kelompok2/

1.1 Latar Belakang

Memory Match adalah permainan atau kegiatan yang dirancang untuk melatih dan meningkatkan kemampuan memori seseorang. Konsep dasar dari permainan ini adalah mencocokkan pasangan kartu atau gambar yang tersebar di atas meja. Pemain harus mengingat letak pasangan-pasangan tersebut dan membalik kartu secara bergantian untuk menemukan pasangan yang sesuai.

Latar belakang permainan ini sering kali terkait dengan pengembangan kognitif, di mana tujuan utamanya adalah untuk melatih otak dalam mengingat dan memperkuat koneksi neural. Ini sangat berguna dalam meningkatkan daya ingat jangka pendek dan pengenalan pola. Secara khusus, dalam konteks psikologi, permainan ini digunakan untuk mengeksplorasi cara kerja memori, baik itu memori jangka pendek maupun memori jangka panjang.

Selain itu, Memory Match dapat digunakan dalam pendidikan sebagai alat pembelajaran yang menyenangkan. Permainan ini sering dimodifikasi untuk membantu anak-anak mengenali huruf, angka, atau bahkan kata-kata baru. Dalam hal ini, permainan ini dapat berfungsi sebagai alat yang efektif untuk membantu pengajaran dan pembelajaran, sambil memberikan pengalaman yang menyenangkan dan menantang bagi pemainnya.

1.2 Tujuan

Tujuan dari Game memory match ini adalah :

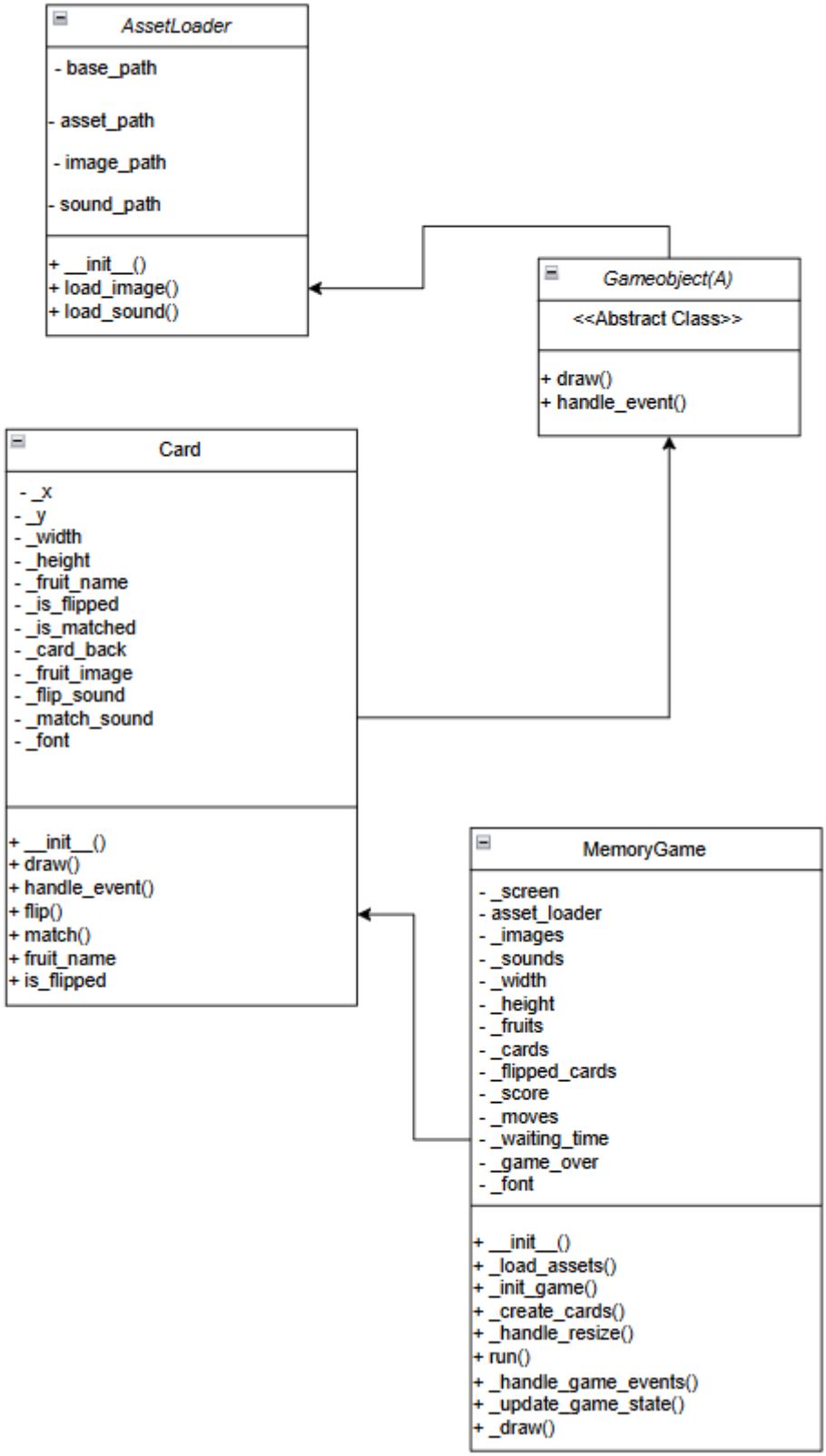
1. Meningkatkan Kemampuan Memori: Game ini dirancang untuk melatih daya ingat pemain, baik memori jangka pendek maupun jangka panjang, dengan cara mengingat posisi kartu atau gambar yang tersebar di meja.
2. Mengasah Keterampilan Pengenalan Pola: Pemain ditantang untuk mengidentifikasi dan mencocokkan pasangan yang sesuai, yang membantu mereka mengembangkan kemampuan untuk mengenali pola dan hubungan antar objek.
3. Mengembangkan Fokus dan Konsentrasi: Karena pemain harus mengingat letak pasangan kartu atau gambar, mereka akan terlatih untuk fokus dan meningkatkan konsentrasi mereka selama permainan berlangsung.

BAB II
HASIL DAN PEMBAHASAN

2.1 Konsep “Pair Match Shuffle”

Pair Match Shuffle adalah sebuah permainan jenis "*memory matching*" atau **permainan mencocokkan pasangan** yang melibatkan pencocokan gambar atau elemen serupa di antara kartu yang tertutup. Tujuan dari permainan ini adalah untuk menemukan pasangan yang cocok dari sekelompok kartu yang awalnya ditempatkan dengan posisi tertutup. Game ini melatih ingatan dan intuisi pemain dengan mencari pasangan kartu dan mengingat-ingat letak kartu yang berpasangan. Game ini diberi visual yang menarik dengan gambar berwarna-warni dan ditambahkan efek suara dan musik background sehingga tidak terasa membosankan.

2.2 Class Diagram



➢ Penjelasan Class Diagram

- **AssetLoader**:
- memuat gambar dan suara untuk game.
- Menyediakan direktori untuk aset, termasuk membuat direktori jika belum ada.

- **GameObject**:
- Abstract class yang memiliki dua metode abstrak draw dan handle_event.
- Class ini menjadi dasar bagi objek game yaitu **Card**.

- **Card**:
- Turunan dari **GameObject**.
- Merepresentasikan kartu individual dalam game.
- Memiliki atribut seperti posisi, ukuran, nama buah, status flipped/matched, gambar, dan suara.
- Metode flip untuk membalik kartu, match untuk mencocokkan, serta properti seperti fruit_name.

- **MemoryGame**:
- Class utama yang mengontrol jalannya game.
- Memiliki atribut untuk layar, loader aset, kartu, status game (score, moves, dll.).
- Menyediakan logika utama game yaitu memulai, menangani input, memperbarui status, dan menggambar elemen-elemen game.

2.3 Kode Program

```python
import pygame
import random
import os
import sys
from abc import ABC, abstractmethod
from typing import List, Tuple

# Inisialisasi pygame
pygame.init()
pygame.mixer.init()

# Konstanta
WINDOW_WIDTH = 800
WINDOW_HEIGHT = 600
CARD_WIDTH = 100
CARD_HEIGHT = 150
MARGIN = 20
CARDS_PER_ROW = 4

# Warna
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
BLUE = (200, 230, 255)
```

```python
class AssetLoader:
    def __init__(self):
        # Tentukan path untuk assets
        self.base_path = os.path.dirname(os.path.abspath(__file__))
        self.asset_path = os.path.join(self.base_path, 'assets')
        self.image_path = os.path.join(self.asset_path, 'images')
        self.sound_path = os.path.join(self.asset_path, 'sounds')

        # Buat direktori jika belum ada
        os.makedirs(self.image_path, exist_ok=True)
        os.makedirs(self.sound_path, exist_ok=True)

        print(f"Asset paths initialized:")
        print(f"Base path: {self.base_path}")
        print(f"Image path: {self.image_path}")
        print(f"Sound path: {self.sound_path}")

    def load_image(self, filename: str) -> pygame.Surface:
        try:
            filepath = os.path.join(self.image_path, filename)
            if os.path.exists(filepath):
                print(f"Loading image: {filepath}")
                return pygame.image.load(filepath).convert_alpha()
            else:
                print(f"Image file not found: {filepath}")
                return None
        except Exception as e:
            print(f"Error loading image {filename}: {str(e)}")
            return None
```

```python
    def load_sound(self, filename: str) -> pygame.mixer.Sound:
        try:
            filepath = os.path.join(self.sound_path, filename)
            if os.path.exists(filepath):
                print(f"Loading sound: {filepath}")
                return pygame.mixer.Sound(filepath)
            else:
                print(f"Sound file not found: {filepath}")
                return None
        except Exception as e:
            print(f"Error loading sound {filename}: {str(e)}")
            return None

class GameObject(ABC):
    @abstractmethod
    def draw(self, screen: pygame.Surface):
        pass

    @abstractmethod
    def handle_event(self, event: pygame.event.Event):
        pass

class Card(GameObject):
    def __init__(self, x: int, y: int, fruit_name: str, images: dict, sounds: dict):
        self._x = x
        self._y = y
        self._width = CARD_WIDTH
        self._height = CARD_HEIGHT
```

```
        self._fruit_name = fruit_name
        self._is_flipped = False
        self._is_matched = False

        # Load images
        self._card_back = images.get('card_back')
        self._fruit_image = images.get(f'{fruit_name.lower()}')

        # Load sounds
        self._flip_sound = sounds.get('flip')
        self._match_sound = sounds.get('match')

        # Font
        self._font = pygame.font.Font(None, 36)

        # Print debug info
        print(f"Card created: {fruit_name}")
        print(f"Card back image: {'Loaded' if self._card_back else 'Not loaded'}")
        print(f"Fruit image: {'Loaded' if self._fruit_image else 'Not loaded'}")

    def draw(self, screen: pygame.Surface):
        if self._is_matched:
            return

        card_rect = pygame.Rect(self._x, self._y, self._width, self._height)

        if not self._is_flipped and self._card_back:
```

```
            # Scale and draw card back
            scaled_back = pygame.transform.scale(self._card_back, (self._width, self._height))
            screen.blit(scaled_back, card_rect)
        elif self._is_flipped and self._fruit_image:
            # Draw white background
            pygame.draw.rect(screen, WHITE, card_rect)

            # Scale fruit image to fit inside card with padding
            padding = 10
            scaled_size = (self._width - 2*padding, self._height - 2*padding)
            scaled_fruit = pygame.transform.scale(self._fruit_image, scaled_size)

            # Center the fruit image on the card
            fruit_rect = scaled_fruit.get_rect()
            fruit_rect.center = card_rect.center
            screen.blit(scaled_fruit, fruit_rect)
        else:
            # Fallback drawing
            pygame.draw.rect(screen, WHITE, card_rect)
            if self._is_flipped:
                text = self._font.render(self._fruit_name, True, BLACK)
                text_rect = text.get_rect(center=card_rect.center)
                screen.blit(text, text_rect)

        # Draw border
        pygame.draw.rect(screen, BLACK, card_rect, 2)
```

```
    def handle_event(self, event: pygame.event.Event) -> bool:
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            if self._is_clicked(pygame.mouse.get_pos()):
                return True
        return False

    def _is_clicked(self, pos: Tuple[int, int]) -> bool:
        x, y = pos
        return (self._x <= x <= self._x + self._width and
                self._y <= y <= self._y + self._height and
                not self._is_matched)

    def flip(self):
        if not self._is_matched:
            self._is_flipped = not self._is_flipped
            if self._flip_sound:
                self._flip_sound.play()

    def match(self):
        self._is_matched = True
        if self._match_sound:
            self._match_sound.play()

    @property
    def fruit_name(self):
        return self._fruit_name
```

```
    @property
    def is_flipped(self):
        return self._is_flipped

class MemoryGame:
    def __init__(self):
        self._screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT), pygame.RESIZABLE)
        pygame.display.set_caption("Fruit Memory Game")

        # Initialize asset loader
        self.asset_loader = AssetLoader()

        # Load assets
        self._load_assets()

        self._init_game()

    def _load_assets(self):
        # Load images
        self._images = {
            'background': self.asset_loader.load_image('background.png'),
            'card_back': self.asset_loader.load_image('card_back.png')
        }

        # Load fruit images
        fruits = ["apple", "banana", "orange", "mango", "grape", "pear", "lemon", "peach"]
        for fruit in fruits:
            self._images[fruit] = self.asset_loader.load_image(f'{fruit}.png')
```

```
        # Load sounds
        self._sounds = {
            'flip': self.asset_loader.load_sound('flip.wav'),
            'match': self.asset_loader.load_sound('match.wav'),
            'win': self.asset_loader.load_sound('win.wav')
        }

        # Load and start background music
        bg_music_path = os.path.join(self.asset_loader.sound_path, 'background_music.mp3')
        if os.path.exists(bg_music_path):
            try:
                pygame.mixer.music.load(bg_music_path)
                pygame.mixer.music.play(-1)
                pygame.mixer.music.set_volume(0.5)
                print("Background music loaded and playing")
            except Exception as e:
                print(f"Error loading background music: {str(e)}")

    def _init_game(self):
        self._width = WINDOW_WIDTH
        self._height = WINDOW_HEIGHT
        self._fruits = ["Apple", "Banana", "Orange", "Mango", "Grape", "Pear", "Lemon", "Peach"] * 2
        random.shuffle(self._fruits)

        self._cards = []
        self._create_cards()
```

```
        self._flipped_cards = []
        self._score = 0
        self._moves = 0
        self._waiting_time = 0
        self._game_over = False

        self._font = pygame.font.Font(None, 36)

    def _create_cards(self):
        self._cards.clear()
        for i, fruit in enumerate(self._fruits):
            row = i // CARDS_PER_ROW
            col = i % CARDS_PER_ROW
            x = col * (CARD_WIDTH + MARGIN) + MARGIN + (self._width - (CARD_WIDTH + MARGIN) * CARDS_PER_ROW) // 2
            y = row * (CARD_HEIGHT + MARGIN) + MARGIN + 100
            self._cards.append(Card(x, y, fruit, self._images, self._sounds))

    def _handle_resize(self, event):
        self._width = event.w
        self._height = event.h
        self._screen = pygame.display.set_mode((self._width, self._height), pygame.RESIZABLE)
        self._create_cards()

    def run(self):
        clock = pygame.time.Clock()
        running = True
```

```
        while running:
            current_time = pygame.time.get_ticks()

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                elif event.type == pygame.VIDEORESIZE:
                    self._handle_resize(event)
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_F11:
                        # Toggle fullscreen
                        pygame.display.toggle_fullscreen()
                elif not self._game_over:
                    self._handle_game_events(event)

            self._update_game_state(current_time)
            self._draw()

            pygame.display.flip()
            clock.tick(60)

        pygame.quit()
        sys.exit()

    def _handle_game_events(self, event):
        if len(self._flipped_cards) >= 2:
            return
```

```
        for card in self._cards:
            if card.handle_event(event):
                if not card.is_flipped and card not in self._flipped_cards:
                    card.flip()
                    self._flipped_cards.append(card)

                    if len(self._flipped_cards) == 2:
                        self._moves += 1
                        self._waiting_time = pygame.time.get_ticks()

    def _update_game_state(self, current_time):
        if len(self._flipped_cards) == 2 and current_time - self._waiting_time >= 1000:
            if self._flipped_cards[0].fruit_name == self._flipped_cards[1].fruit_name:
                self._flipped_cards[0].match()
                self._flipped_cards[1].match()
                self._score += 1

                if self._score == len(self._fruits) // 2:
                    self._game_over = True
                    if self._sounds.get('win'):
                        self._sounds['win'].play()
            else:
                self._flipped_cards[0].flip()
                self._flipped_cards[1].flip()

            self._flipped_cards = []
```

```
    def _draw(self):
        # Draw background
        if self._images.get('background'):
            # Scale background to fit screen while maintaining aspect ratio
            bg_image = self._images['background']
            bg_aspect = bg_image.get_width() / bg_image.get_height()
            screen_aspect = self._width / self._height

            if screen_aspect > bg_aspect:
                scaled_width = self._width
                scaled_height = int(scaled_width / bg_aspect)
            else:
                scaled_height = self._height
                scaled_width = int(scaled_height * bg_aspect)

            scaled_bg = pygame.transform.scale(bg_image, (scaled_width, scaled_height))

            # Center the background
            x_offset = (self._width - scaled_width) // 2
            y_offset = (self._height - scaled_height) // 2

            self._screen.blit(scaled_bg, (x_offset, y_offset))
        else:
            self._screen.fill(BLUE)

        # Draw semi-transparent overlay to make cards more visible
        overlay = pygame.Surface((self._width, self._height))
        overlay.fill((255, 255, 255))
```

```
        overlay.set_alpha(128)
        self._screen.blit(overlay, (0, 0))

        # Draw title with shadow
        title_text = "Fruit Memory Game"
        shadow = self._font.render(title_text, True, BLACK)
        title = self._font.render(title_text, True, WHITE)
        title_pos = (self._width//2 - title.get_width()//2, 20)
        self._screen.blit(shadow, (title_pos[0]+2, title_pos[1]+2))
        self._screen.blit(title, title_pos)

        # Draw score panel
        score_panel = pygame.Surface((200, 80))
        score_panel.fill(WHITE)
        score_panel.set_alpha(180)
        self._screen.blit(score_panel, (10, 10))

        score_text = self._font.render(f"Matches: {self._score}", True, BLACK)
        moves_text = self._font.render(f"Moves: {self._moves}", True, BLACK)
        self._screen.blit(score_text, (20, 20))
        self._screen.blit(moves_text, (20, 60))

        # Draw cards
        for card in self._cards:
            card.draw(self._screen)

        # Draw game over overlay
```

```
        if self._game_over:
            overlay = pygame.Surface((self._width, self._height))
            overlay.fill(BLACK)
            overlay.set_alpha(128)
            self._screen.blit(overlay, (0, 0))

            game_over = self._font.render("Congratulations! Game Over!", True, WHITE)
            score_final = self._font.render(f"Final Score: {self._moves} moves", True, WHITE)

            self._screen.blit(game_over,
                              (self._width//2 - game_over.get_width()//2,
                               self._height//2 - game_over.get_height()))
            self._screen.blit(score_final,
                              (self._width//2 - score_final.get_width()//2,
                               self._height//2 + score_final.get_height()))

if __name__ == "__main__":
    game = MemoryGame()
    game.run()
```

2.4 Penerapan OOP dalam kode program

1. **Encapsulation**

Encapsulation adalah konsep untuk menyembunyikan detail implementasi dan hanya menyediakan akses yang diperlukan melalui method atau property tertentu. Dalam script ini terdapat dalam class Card

```
class Card(GameObject):
    def __init__(self, x: int, y: int, fruit_name: str, images: dict, sounds: dict):
        self._x = x
        self._y = y
        self._width = CARD_WIDTH
        self._height = CARD_HEIGHT
        self._fruit_name = fruit_name
        self._is_flipped = False
        self._is_matched = False

        # Load images
        self._card_back = images.get('card_back')
        self._fruit_image = images.get(f'{fruit_name.lower()}')

        # Load sounds
        self._flip_sound = sounds.get('flip')
        self._match_sound = sounds.get('match')

        # Font
        self._font = pygame.font.Font(None, 36)
```

Dalam class ini terdapat atribut privat yaitu self._x, self._y, self._is_flipped, dan lainnya, Prefix _ menandakan atribut bersifat privat dan hanya dapat diakses dalam kelas itu sendiri. Akses terhadap atribut privat diatur melalui @property

```
@property
def fruit_name(self):
    return self._fruit_name

@property
def is_flipped(self):
    return self._is_flipped
```

2. **Inheritance**

Inheritance adalah konsep yang memungkinkan kelas baru (child) mewarisi properti dan metode dari kelas lain (parent). Pada kode ini, kelas Card mewarisi dari kelas GameObject.

```
class GameObject(ABC):
    @abstractmethod
    def draw(self, screen: pygame.Surface):
        pass

    @abstractmethod
    def handle_event(self, event: pygame.event.Event):
        pass

class Card(GameObject):
    def __init__(self, x: int, y: int, fruit_name: str, images: dict, sounds: dict):
        self._x = x
        self._y = y
        self._width = CARD_WIDTH
        self._height = CARD_HEIGHT
        self._fruit_name = fruit_name
        self._is_flipped = False
        self._is_matched = False

        # Load images
        self._card_back = images.get('card_back')
        self._fruit_image = images.get(f'{fruit_name.lower()}')

        # Load sounds
        self._flip_sound = sounds.get('flip')
        self._match_sound = sounds.get('match')

        # Font
        self. font = pygame.font.Font(None, 36)
```

GameObject adalah kelas abstrak yang mendefinisikan metode abstrak draw dan handle_event, dan kelas Card mengimplementasikan metode abstrak tersebut.

3. **Polymorphism**

Polymorphism memungkinkan objek dari kelas yang berbeda untuk diperlakukan sama jika mereka berasal dari kelas induk yang sama. Dengan kata lain, metode dari kelas turunan akan dipanggil sesuai implementasinya, meskipun objeknya diakses sebagai kelas induk.

Dalam kode ini terdapat pada objek turunan GameObject yaitu Card dengan metode draw dan handle_event:

Pada metode draw:

```
def draw(self, screen: pygame.Surface):
    if self._is_matched:
        return

    card_rect = pygame.Rect(self._x, self._y, self._width, self._height)

    if not self._is_flipped and self._card_back:
        # Scale and draw card back
        scaled_back = pygame.transform.scale(self._card_back, (self._width, self._height))
        screen.blit(scaled_back, card_rect)
    elif self._is_flipped and self._fruit_image:
        # Draw white background
        pygame.draw.rect(screen, WHITE, card_rect)

        # Scale fruit image to fit inside card with padding
        padding = 10
        scaled_size = (self._width - 2*padding, self._height - 2*padding)
        scaled_fruit = pygame.transform.scale(self._fruit_image, scaled_size)

        # Center the fruit image on the card
        fruit_rect = scaled_fruit.get_rect()
        fruit_rect.center = card_rect.center
        screen.blit(scaled_fruit, fruit_rect)
    else:
        # Fallback drawing
        pygame.draw.rect(screen, WHITE, card_rect)
        if self._is_flipped:
            text = self._font.render(self._fruit_name, True, BLACK)
```

```
            text_rect = text.get_rect(center=card_rect.center)
            screen.blit(text, text_rect)

        # Draw border
        pygame.draw.rect(screen, BLACK, card_rect, 2)
```

Pada metode handle_event:

```
    def handle_event(self, event: pygame.event.Event) -> bool:
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            if self._is_clicked(pygame.mouse.get_pos()):
                return True
        return False

    def _is_clicked(self, pos: Tuple[int, int]) -> bool:
        x, y = pos
        return (self._x <= x <= self._x + self._width and
                self._y <= y <= self._y + self._height and
                not self._is_matched)

    def flip(self):
        if not self._is_matched:
            self._is_flipped = not self._is_flipped
            if self._flip_sound:
                self._flip_sound.play()

    def match(self):
        self._is_matched = True
        if self._match_sound:
            self._match_sound.play()

    @property
    def fruit_name(self):
        return self._fruit_name

    @property
```

```python
@property
def is_flipped(self):
    return self._is_flipped
```

Semua kelas yang mewarisi GameObject dapat memiliki implementasi yang berbeda untuk metode draw dan handle_event.

4. **Abstraction**

Abstraction menyembunyikan detail implementasi dan hanya menampilkan fungsionalitas penting. Pada kode ini, kelas abstrak GameObject mendefinisikan kerangka kerja untuk objek permainan tanpa memberikan detail spesifik.

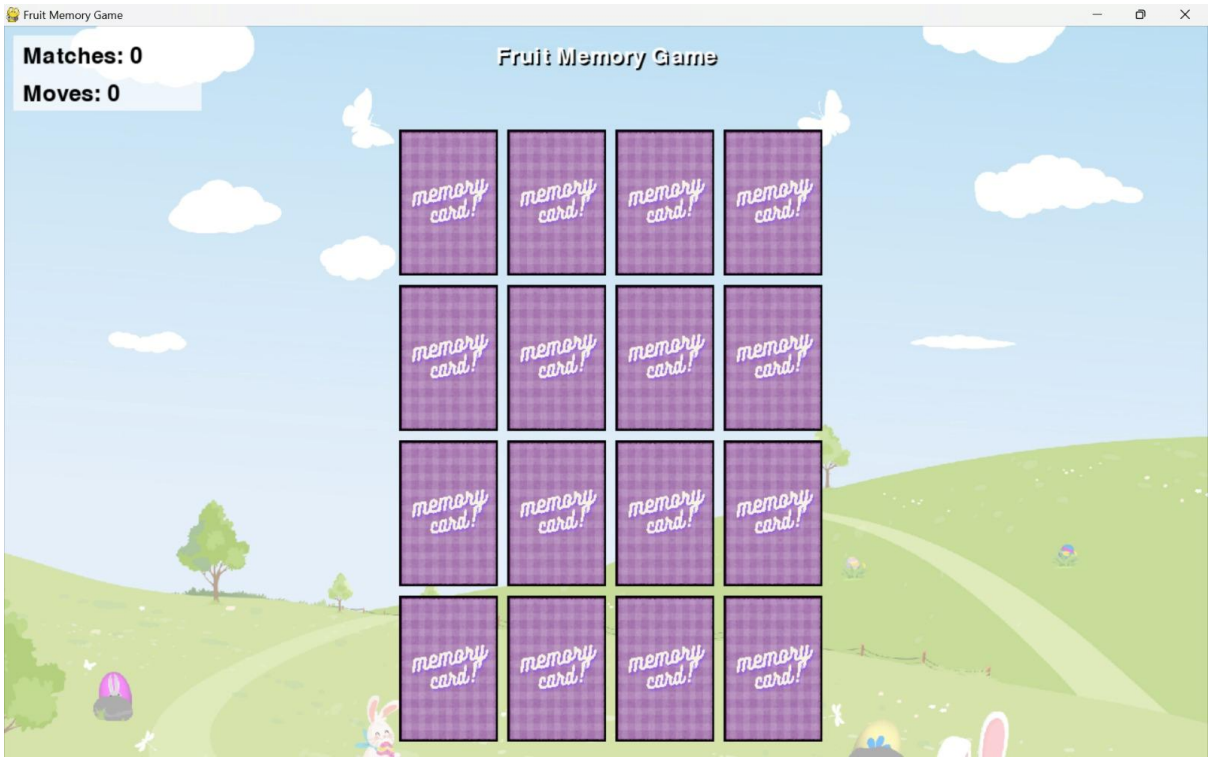
```python
class GameObject(ABC):
    @abstractmethod
    def draw(self, screen: pygame.Surface):
        pass

    @abstractmethod
    def handle_event(self, event: pygame.event.Event):
        pass
```

Kelas GameObject menyembunyikan detail implementasi, sehingga setiap kelas turunan (seperti Card) harus memberikan implementasinya sendiri.

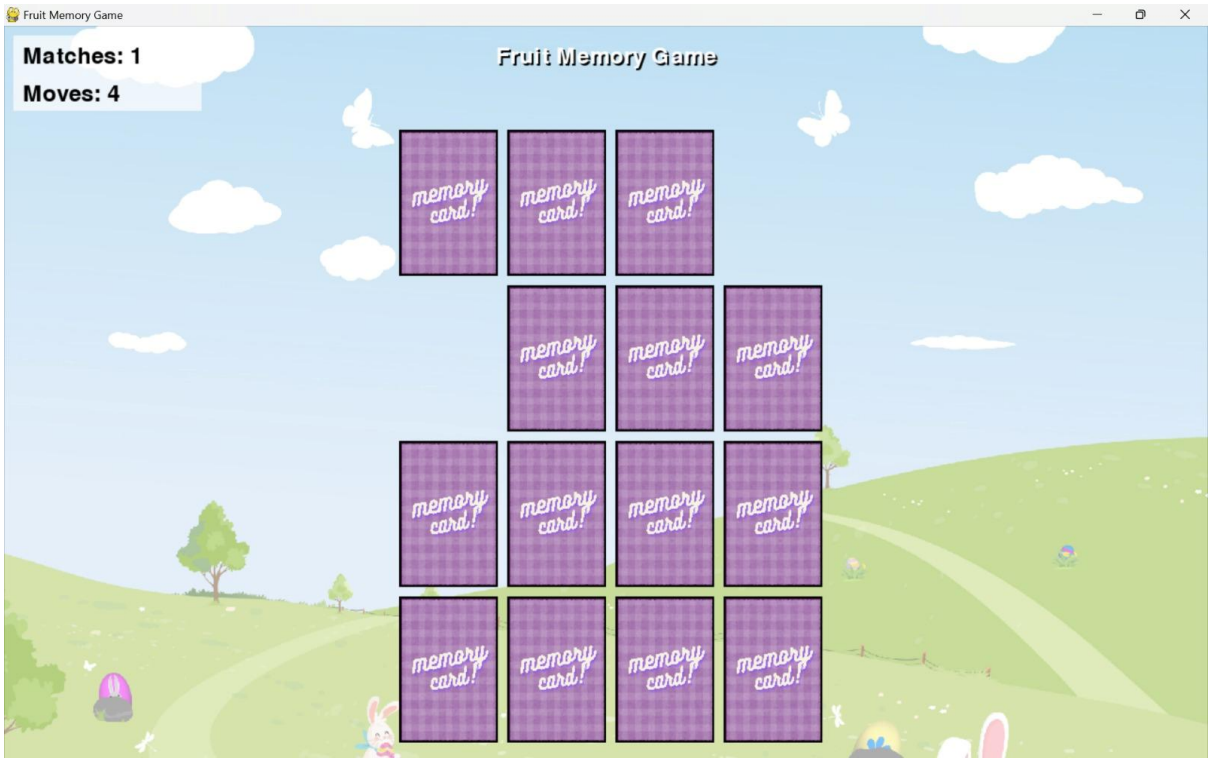
2.5 Tampilan Hasil

1. tampilan awal kartu sebelum dibalik



2. Tampilan saat kartu dibalik

3. Tampilan saat kartu berhasil dicocokkan dan dimasukkan dalam skor



4. Tampilan Menang

BAB III

PENUTUP

Pembuatan game "Fruit Memory Game" menggunakan Python dan Pygame menunjukkan bagaimana konsep dasar pemrograman dapat diterapkan untuk menghasilkan aplikasi interaktif. Game ini mengimplementasikan berbagai aspek pengembangan perangkat lunak, seperti manajemen aset melalui struktur direktori terorganisir untuk menyimpan gambar dan suara dengan modul AssetLoader, penerapan Object-Oriented Programming (OOP) menggunakan kelas-kelas yang terdapat pada kode program, serta interaksi pengguna yang mendukung klik mouse untuk membalik kartu dan resize jendela untuk menyesuaikan ukuran layar. Game ini juga memanfaatkan media seperti gambar dan suara untuk menciptakan pengalaman visual dan audio yang menarik sehingga pengalaman bermain memory card match ini terasa menyenangkan. Secara keseluruhan, game ini menjadi contoh nyata bagaimana pemrograman dapat digunakan untuk menciptakan aplikasi yang interaktif dan menyenangkan. Dengan melibatkan berbagai elemen pengembangan perangkat lunak, proyek ini memberikan pengalaman praktis yang bermanfaat, sekaligus menjadi dasar untuk membangun game atau aplikasi lain yang lebih kompleks di masa depan.