# DESIGN DOCUMENT
# PATTERN EXPLANATION

Bomberman Multiplayer Game


**Ad Soyad:** Serdar Can
**Öğrenci No:** 220401096

**Proje:** Bomberman Multiplayer Game
**Teknoloji:** Python + Pygame + Flask + Socket.IO + PostgreSQL
**Tarih:** 25 Aralık 2025

# İçindekiler

# Genel Bakış

## Proje Yapısı

```
projem/
 backend/           # Sunucu (Flask + Socket.IO)
    models/        # Veri modelleri
    repository/    # Veri erişim katmanı
    services/      # İş mantığı
    handlers/      # Socket.IO event handlers
 bomberman/         # İstemci (Pygame)
     model/         # Oyun modelleri
     view/          # Görsel katman
     controller/    # Oyun kontrolcüsü
     service/       # İş mantığı servisleri
     repository/    # Veri erişim katmanı
     network/       # Ağ iletişimi
```

## Tasarım Deseni Özeti

| **Kategori** | **Desen** | **Dosya Konumu** |
|---|---|---|
| Creational | Factory Method | `bomberman/view/characters.py` |
| Structural | Adapter | `bomberman/model/player_decorator.py` |
| Structural | Decorator | `bomberman/model/player_decorator.py` |
| Behavioral | Observer | `bomberman/service/game_event_service.py` |
| Behavioral | Strategy | `bomberman/model/enemy.py` |
| Data Access | Repository | `bomberman/repository/`, `backend/repository/` |
| Architectural | MVC | `bomberman/model/`, `view/`, `controller/` |

Tablo 1: Tasarım Desenleri Özeti

**TOPLAM:** 7 farklı tasarım deseni (Gereksinim: 5)

# Kullanılan Tasarım Desenleri

1. Factory Method Pattern (Creational) ✓
2. Adapter Pattern (Structural) ✓
3. Decorator Pattern (Structural) ✓
4. Observer Pattern (Behavioral) ✓
5. Strategy Pattern (Behavioral) ✓

6. Repository Pattern (Data Access) ✓
7. MVC Pattern (Architectural) ✓

# Detaylı Pattern Açıklamaları

## Factory Method Pattern (Creational) ✓

### 3.1.1 Tanım

Factory Method, nesne oluşturma sorumluluğunu alt sınıflara devreden bir yaratımsal desendir. Nesne oluşturma mantığını merkezi bir yerde toplar.

### 3.1.2 Projede Nerede Kullanıldı?

**Dosya:** `bomberman/view/characters.py`

**Ana Bileşenler:**

- CharacterFactory (Satır 26-43)
- MonsterFactory (Satır 66-102)
- EffectFactory (Satır 37-43)

### 3.1.3 Kod Örneği

**Dosya:** `bomberman/view/characters.py`

**CharacterFactory (Satır 26-43):**

```
@staticmethod
def roster() -> Sequence[Character]:
    """T m karakterleri d nd r r"""
    return [
        Character(
            id="bomberman",
            name="Bomberman",
            description="Klasik bomba ustas ",
            accent_color=(70, 130, 255),
            avatar_color=(255, 200, 100),
            tagline="Bomba patlatma uzman ",
            image_name="bomberman.png"
        ),
        # ... di er karakterler
    ]

@staticmethod
def find_by_id(character_id: str) -> Character | None:
    """ID'ye g re karakter bulur"""
    return next((c for c in CharacterFactory.roster()
                 if c.id == character_id), None)
```

**MonsterFactory (Satır 66-102):**

```
@staticmethod
def roster() -> Sequence[Monster]:
    """T m  d   man tiplerini d  nd  r  r"""
    return [
        Monster(id="m1", name="Static Enemy", ...),
        Monster(id="m2", name="Chasing Enemy", ...),
        Monster(id="m3", name="Smart Enemy", ...),
    ]

@staticmethod
def create(enemy_type: str, position: Tuple[int, int]):
    """Factory Method: D   man tipi ve pozisyona g re d   man
       instance'   olu turur"""
    from model.enemy import StaticEnemy, ChasingEnemy,
       SmartEnemy, EnemyType

    if enemy_type == "STATIC" or enemy_type == EnemyType.STATIC:
        return StaticEnemy(position)
    elif enemy_type == "CHASING" or enemy_type ==
       EnemyType.CHASING:
        return ChasingEnemy(position)
    elif enemy_type == "SMART" or enemy_type == EnemyType.SMART:
        return SmartEnemy(position)
    else:
        raise ValueError(f"Unknown enemy type: {enemy_type}")
```

### 3.1.4 Kullanım Örneği

**Dosya:** `bomberman/controller/game_controller.py` (Satır 153)

```
enemy = MonsterFactory.create(enemy_type, position)
```

### 3.1.5 Neden Kullanıldı?

- Karakter ve düşman nesnelerinin merkezi bir yerden yönetilmesi
- Yeni karakter/düşman eklenmesi kolaylaşır
- Client kod karmaşık nesne yaratma detaylarından ayrıştırılır

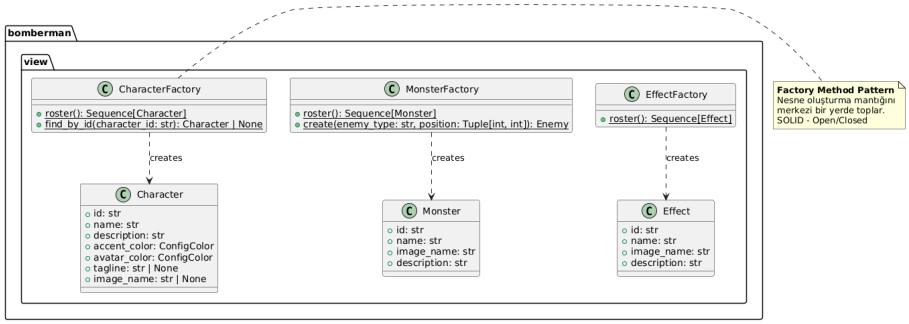
### 3.1.6 Faydaları

✓ **Single Responsibility:** Factory sadece nesne yaratmaktan sorumlu

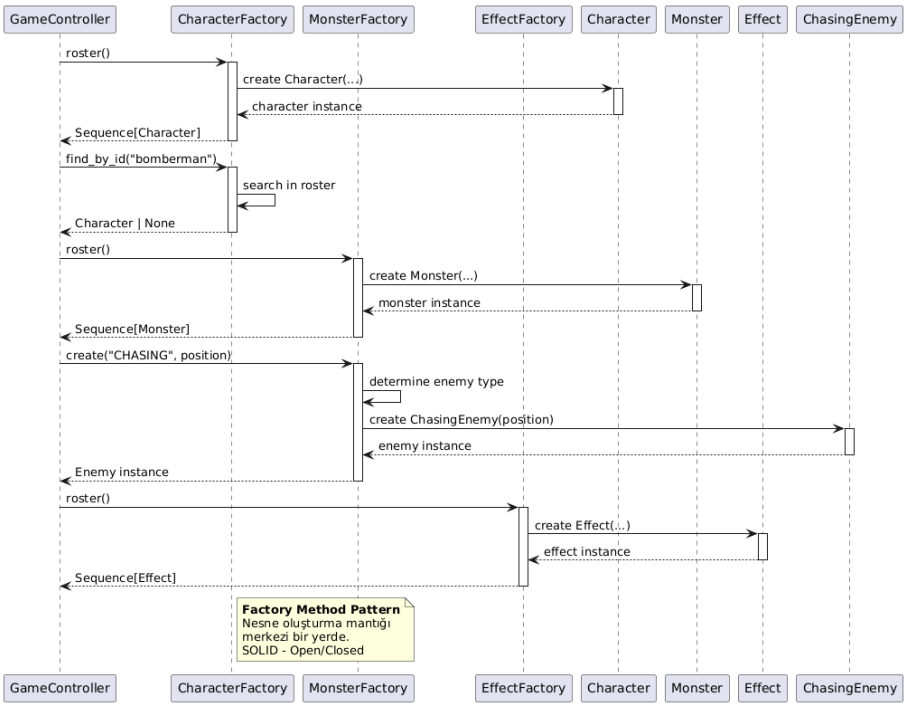
✓ **Open/Closed:** Yeni karakter eklemek için mevcut kod değiştirilmez

✓ **Centralized Management:** Tüm karakterler tek yerden yönetilir

### 3.1.7 UML Diyagramı




Şekil 1: Factory Method Pattern - Class Diagram




Şekil 2: Factory Method Pattern - Sequence Diagram

# Adapter Pattern (Structural) ✓

### 3.2.1 Tanım

Adapter, uyumsuz interface'leri birlikte çalışacak şekilde adapte eden yapısal bir desendir.

### 3.2.2 Projede Nerede Kullanıldı?

**Dosya:** `bomberman/model/player_decorator.py`
**Ana Bileşenler:**

- PlayerInterface (Satır 19-40) - Target Interface
- BombermanAdapter (Satır 73-96) - Adapter
- Bomberman (`bomberman/model/bomberman.py`, Satır 9-40) - Adaptee

### 3.2.3 Kod Örneği

**Dosya:** `bomberman/model/player_decorator.py`

**PlayerInterface (Satır 19-40):**

```
class PlayerInterface(ABC):
    """Target interface for Adapter and Decorator patterns"""

    @abstractmethod
    def get_speed(self) -> float:
        """Oyuncu h z n  d nd r r"""
        pass

    @abstractmethod
    def get_bomb_count(self) -> int:
        """Bomba say s n  d nd r r"""
        pass

    @abstractmethod
    def get_bomb_power(self) -> int:
        """Bomba g c n  d nd r r"""
        pass

    @abstractmethod
    def get_health(self) -> int:
        """Sa l k puan n  d nd r r"""
        pass
```

**BombermanAdapter (Satır 73-96):**

```
class BombermanAdapter(PlayerInterface):
    """
    Adapter Pattern: Bomberman model'ini PlayerInterface'e
       adapte eder.
    Bu sayede Bomberman'  decorator pattern ile kullanabiliriz.
    """

    def __init__(self, bomberman: 'Bomberman') -> None:
        self._bomberman = bomberman

    def get_speed(self) -> float:
        return self._bomberman.speed

    def get_bomb_count(self) -> int:
        return self._bomberman.bomb_count
```

```
15
16      def get_bomb_power(self) -> int:
17          return self._bomberman.bomb_power
18
19      def get_health(self) -> int:
20          return self._bomberman.health
```

#### 3.2.4 Kullanım Örneği

**Dosya:** `bomberman/service/powerup_service.py` (Satır 72-106)

```
def apply_powerup(self, player: PlayerInterface, powerup_type:
    PowerupType) -> PlayerInterface:
1       """Power-up uygula ve decorator d nd r"""
2       # E er player Bomberman ise,  nce  adapter ile sarmala
3       if isinstance(player, Bomberman):
4           player = BombermanAdapter(player)
5
6       # Power-up tipine g re decorator olu tur
7       if powerup_type == PowerupType.SPEED:
8           return SpeedBoostDecorator(player)
9       elif powerup_type == PowerupType.BOMB_COUNT:
10          return BombCountBoostDecorator(player)
11      # ...
12
```

#### 3.2.5 Neden Kullanıldı?

- Bomberman sınıfı orijinal olarak PlayerInterface implement etmiyordu
- Decorator Pattern kullanabilmek için ortak bir interface gerekiyordu
- Mevcut Bomberman kodunu değiştirmeden adapte ettik

#### 3.2.6 Faydaları

✓ **Interface Compatibility:** Uyumsuz interface'leri birleştirir

✓ **Non-Invasive:** Mevcut Bomberman koduna dokunmadan çalışır

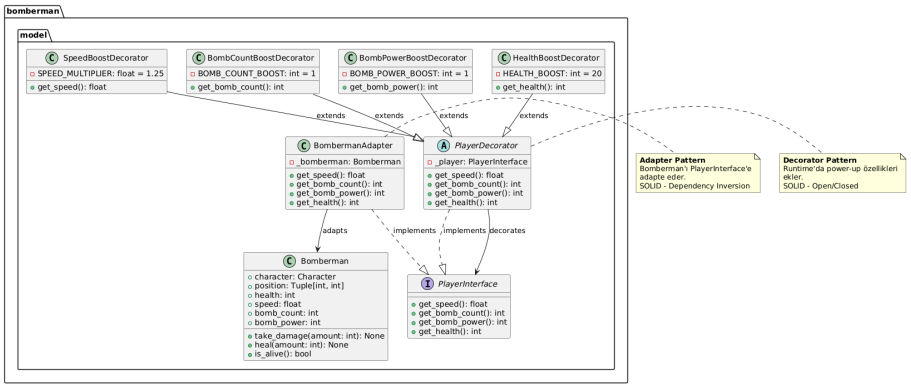
✓ **Enables Decorator:** Decorator Pattern için gerekli interface sağlar

#### 3.2.7 UML Diyagramı

## Decorator Pattern (Structural) ✓

#### 3.3.1 Tanım

Decorator, nesnelere runtime'da dinamik olarak yeni davranışlar ekleyen yapısal bir desendir.




Şekil 3: Adapter & Decorator Pattern - Class Diagram

### 3.3.2 Projede Nerede Kullanıldı?

**Dosya:** `bomberman/model/player_decorator.py`

**Ana Bileşenler:**

- PlayerDecorator (Satır 43-70) - Base Decorator
- SpeedBoostDecorator (Satır 99-109)
- BombCountBoostDecorator (Satır 112-122)
- BombPowerBoostDecorator (Satır 125-135)
- HealthBoostDecorator (Satır 138-148)

### 3.3.3 Kod Örneği

**PlayerDecorator (Satır 43-70):**

```
class PlayerDecorator(PlayerInterface):
    """Base decorator class - Decorator Pattern"""

    def __init__(self, player: PlayerInterface) -> None:
        self._player = player

    def get_speed(self) -> float:
        return self._player.get_speed()

    def get_bomb_count(self) -> int:
        return self._player.get_bomb_count()

    def get_bomb_power(self) -> int:
        return self._player.get_bomb_power()

    def get_health(self) -> int:
        return self._player.get_health()
```

**SpeedBoostDecorator (Satır 99-109):**

```
class SpeedBoostDecorator(PlayerDecorator):
    """Speed Boost power-up decorator"""
    SPEED_MULTIPLIER = 1.25

    def get_speed(self) -> float:
        return self._player.get_speed() * self.SPEED_MULTIPLIER
```

**BombCountBoostDecorator (Satır 112-122):**

```
class BombCountBoostDecorator(PlayerDecorator):
    """Bomb Count power-up decorator"""
    BOMB_COUNT_BOOST = 1

    def get_bomb_count(self) -> int:
        return self._player.get_bomb_count() +
           self.BOMB_COUNT_BOOST
```

### 3.3.4 Kullanım Örneği

**Dosya:** bomberman/controller/game_controller.py (Satır 375-395)

```
def _check_powerup_collection(self) -> None:
    """Power-up toplama kontrol """
    # Power-up topland  m  kontrol et
    powerup = self._check_powerup_collision()
    if powerup:
        # Decorator olu tur ve uygula
        self._player_decorator =
           self._powerup_service.apply_powerup(
            self._player_decorator or
               BombermanAdapter(self.player),
            powerup.powerup_type
        )

        # Decorated player'dan de erleri al ve base Bomberman'a
           uygula
        self.player.speed = self._player_decorator.get_speed()
        self.player.bomb_count =
           self._player_decorator.get_bomb_count()
        self.player.bomb_power =
           self._player_decorator.get_bomb_power()
        self.player.health = self._player_decorator.get_health()
```

**Decorator Chain Örneği:**

```
# Base player
player = BombermanAdapter(bomberman)

# Power-up'lar  ekle (decorator chain)
player = SpeedBoostDecorator(player)          # H z %25 artar
```

```
player = BombCountBoostDecorator(player)        # Bomba say s
    +1
player = BombPowerBoostDecorator(player)        # Bomba g  c   +1

# Art k player' n  h  z   %25 fazla, bomba say s  ve g  c   +1
speed = player.get_speed()        # base_speed * 1.25
bomb_count = player.get_bomb_count()  # base_count + 1
```

### 3.3.5 Neden Kullanıldı?

- Power-up sistemini esnek ve genişletilebilir yapma
- Runtime'da dinamik olarak özellik ekleme/çıkarma
- Power-up kombinasyonlarını kolayca yönetme

### 3.3.6 Faydaları

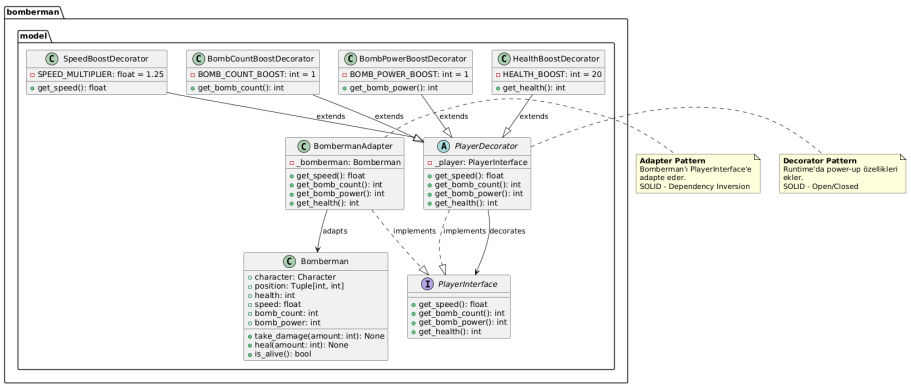
✓ **Runtime Flexibility:** Oyun sırasında özellikler eklenebilir

✓ **Composability:** Decorator'lar zincirlenerek kombinasyonlar oluşturulur

✓ **Open/Closed:** Yeni power-up eklemek için mevcut kod değişmez

✓ **Single Responsibility:** Her decorator tek bir power-up'tan sorumlu

### 3.3.7 UML Diyagramı

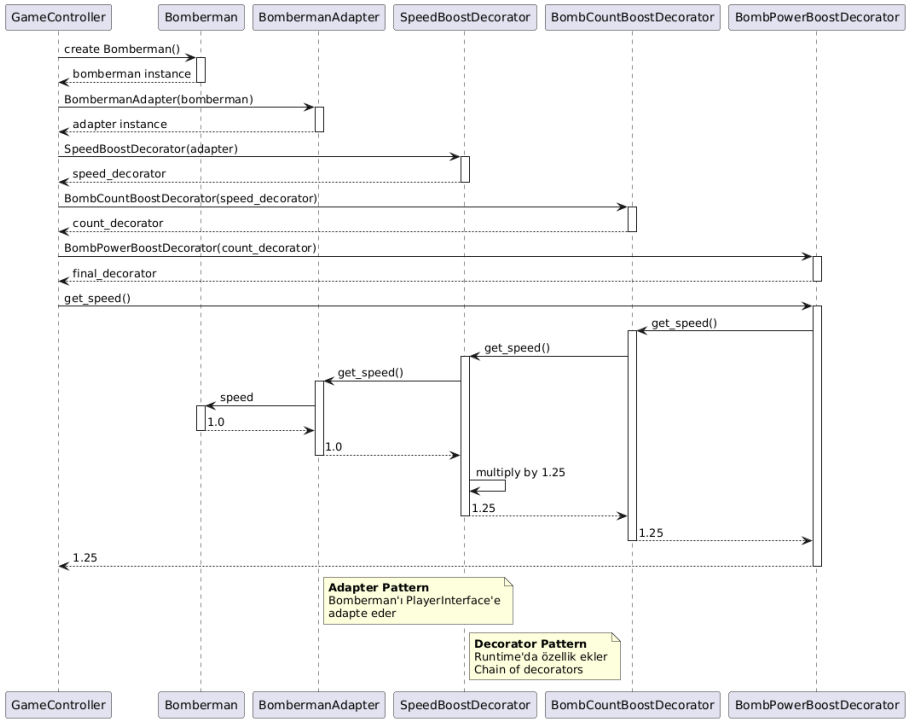



Şekil 4: Adapter & Decorator Pattern - Class Diagram

## Observer Pattern (Behavioral) ✓

### 3.4.1 Tanım

Observer, bir nesnede (Subject) olan değişiklikleri diğer nesnelere (Observer) otomatik olarak bildiren davranışsal bir desendir.




Şekil 5: Decorator Pattern - Sequence Diagram

### 3.4.2 Projede Nerede Kullanıldı?

**Dosyalar:**

- `bomberman/service/game_event_service.py` (Subject)
- `bomberman/service/game_observers.py` (Concrete Observers)

**Ana Bileşenler:**

- GameEventService (Satır 41-82) - Subject
- GameObserver (Satır 32-38) - Observer Interface
- SoundObserver (Satır 18-46) - Concrete Observer
- ScoreObserver (Satır 49-91) - Concrete Observer
- LoggerObserver (Satır 94-99) - Concrete Observer

### 3.4.3 Kod Örneği

**GameEventType (Satır 13-22):**

```
class GameEventType(Enum):
    BOMB_PLACED = "bomb_placed"
    BOMB_EXPLODED = "bomb_exploded"
    ENEMY_KILLED = "enemy_killed"
    PLAYER_DAMAGED = "player_damaged"
```

```
    PLAYER_DIED = "player_died"
    POWERUP_COLLECTED = "powerup_collected"
    WALL_DESTROYED = "wall_destroyed"
    LEVEL_COMPLETED = "level_completed"
```

**GameEventService (Satır 41-82):**

```
class GameEventService:
    """Subject - Observer Pattern"""

    def __init__(self) -> None:
        self._observers: list[GameObserver] = []

    def attach(self, observer: GameObserver) -> None:
        """Observer ekle"""
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer: GameObserver) -> None:
        """Observer   kar   """
        if observer in self._observers:
            self._observers.remove(observer)

    def notify(self, event: GameEvent) -> None:
        """T m observer'lar  bilgilendir"""
        for observer in self._observers:
            observer.on_event(event)

    def emit(self, event_type: GameEventType, **data) -> None:
        """Event yay nla (shortcut method)"""
        event = GameEvent(event_type, data)
        self.notify(event)
```

**SoundObserver (Satır 18-46):**

```
class SoundObserver(GameObserver):
    """Ses efektlerini y neten observer"""

    def __init__(self, sound_service: SoundService) -> None:
        self._sound_service = sound_service

    def on_event(self, event: GameEvent) -> None:
        if event.event_type == GameEventType.BOMB_EXPLODED:
            self._sound_service.play_sound("explosion.wav")
        elif event.event_type == GameEventType.ENEMY_KILLED:
            self._sound_service.play_sound("enemy_death.wav")
        elif event.event_type == GameEventType.PLAYER_DIED:
            self._sound_service.play_sound("player_death.wav")
        # ...
```

**ScoreObserver (Satır 49-91):**

```python
class ScoreObserver(GameObserver):
    """Skor takibi yapan observer"""

    def __init__(self) -> None:
        self.score = 0
        self.walls_destroyed = 0
        self.enemies_killed = 0

    def on_event(self, event: GameEvent) -> None:
        if event.event_type == GameEventType.ENEMY_KILLED:
            self.enemies_killed += 1
            self.score += 100
        elif event.event_type == GameEventType.WALL_DESTROYED:
            self.walls_destroyed += 1
            self.score += 10
        elif event.event_type == GameEventType.POWERUP_COLLECTED:
            self.score += 5
        elif event.event_type == GameEventType.LEVEL_COMPLETED:
            self.score += 500
        # ...
```

### 3.4.4 Kullanım Örneği

**Dosya:** `bomberman/controller/game_controller.py` (Satır 48-53)

```python
def __init__(self, ...):
    # Observer'lar  olu tur ve attach et
    self._event_service = GameEventService()
    self._event_service.attach(SoundObserver(self._sound_service))
    self._event_service.attach(ScoreObserver())
    self._event_service.attach(LoggerObserver())
```

**Event Yayınlama Örnekleri:**

- Duvar yıkıldığında (Satır 279-282, 470-473): `emit(GameEventType.WALL_DESTROYED, position=(x, y))`
- Power-up toplandığında (Satır 396-400): `emit(GameEventType.POWERUP_COLLECTED, ...)`
- Bomba patladığında (Satır 490-493): `emit(GameEventType.BOMB_EXPLODED, ...)`
- Düşman öldürüldüğünde (Satır 627-630, 657-660): `emit(GameEventType.ENEMY_KILLED, ...)`
- Oyuncu öldüğünde (Satır 651): `emit(GameEventType.PLAYER_DIED)`

### 3.4.5 Bildirim Akışı

1. GameController bir event oluştuğunda `emit()` çağırır
2. GameEventService.emit() çağrılır (Satır 71-74) - GameEvent oluşturulur, notify() çağrılır
3. GameEventService.notify() çağrılır (Satır 66-69) - Tüm observer'lar döngüye alınır
4. Observer'lar event'i işler: SoundObserver ses çalar, ScoreObserver skoru günceller, LoggerObserver log yazar

### 3.4.6 Neden Kullanıldı?

- Oyun eventlerini merkezi bir yerden yönetme
- Ses, skor, log gibi sistemleri birbirinden bağımsız tutma
- Yeni observer eklemek kolay (örn: achievements, statistics)

### 3.4.7 Faydaları

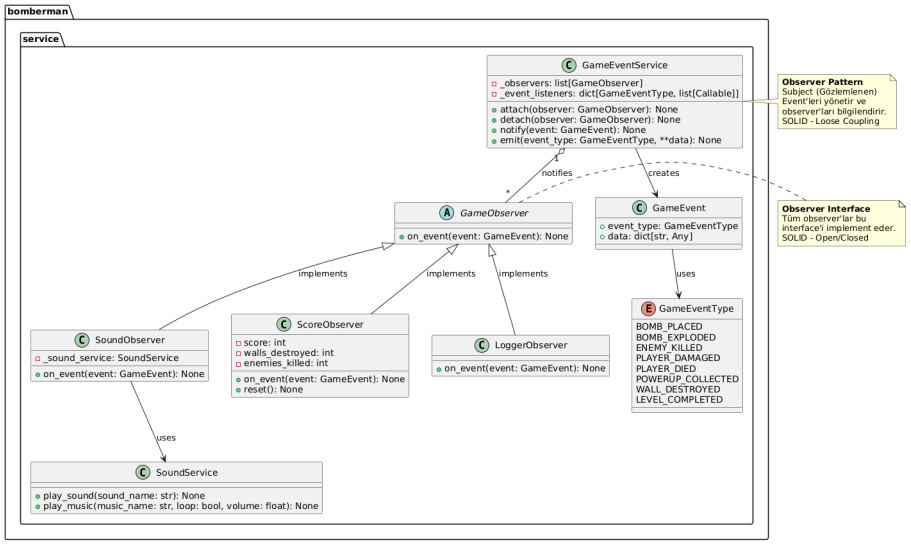
✓ **Loose Coupling:** Subject ve Observer birbirinden bağımsız

✓ **Scalability:** Yeni observer eklemek kolay

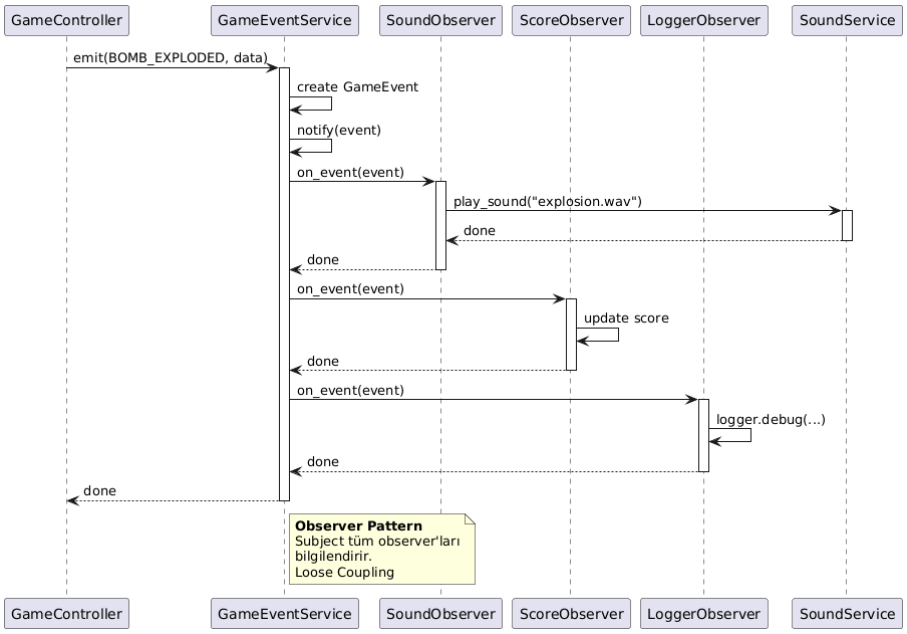
✓ **Separation of Concerns:** Her observer kendi işinden sorumlu

✓ **Dynamic Subscription:** Runtime'da observer eklenip çıkarılabilir

### 3.4.8 UML Diyagramı




Şekil 6: Observer Pattern - Class Diagram




Şekil 7: Observer Pattern - Sequence Diagram

## Strategy Pattern (Behavioral) ✓

### 3.5.1 Tanım

Strategy, bir algoritma ailesini tanımlayan ve birbirinin yerine kullanılabilir hale getiren davranışsal bir desendir.

### 3.5.2 Projede Nerede Kullanıldı?

**Dosya:** `bomberman/model/enemy.py`

**Ana Bileşenler:**

- Enemy (Satır 30-77) - Strategy Interface (Abstract Base Class)
- StaticEnemy (Satır 80-131) - Concrete Strategy 1
- ChasingEnemy (Satır 133-208) - Concrete Strategy 2
- SmartEnemy (Satır 210-278) - Concrete Strategy 3

### 3.5.3 Kod Örneği

**Enemy (Satır 30-77) - Strategy Interface:**

```
1 class Enemy(ABC):
2     """Base Enemy s n f   - Strategy Pattern"""
3
4     def __init__(self, position: Tuple[int, int], enemy_type:
         EnemyType):
5         self.position = position
```

```
        self.health = self.max_health
        self.enemy_type = enemy_type
        self.move_interval = 0.0

    @property
    @abstractmethod
    def max_health(self) -> int:
        pass

    @abstractmethod
    def update(
        self,
        player_pos: Tuple[int, int] | None,
        tile_provider: Callable[[int, int], TileType],
    ) -> None:
        """Her alt s n f kendi hareket stratejisini uygular"""
        pass
```

**StaticEnemy (Satır 80-131) - Concrete Strategy 1:**

```
class StaticEnemy(Enemy):
    """
    Statik D man: Do du u yerden sadece 1 birim uzakl a
       hareket eder.
    Rastgele y nlerde s n rl hareket.
    """
    MAX_HEALTH = 20

    def __init__(self, position: Tuple[int, int]):
        super().__init__(position, EnemyType.STATIC)
        self._spawn_position = position

    def update(self, player_pos, tile_provider) -> None:
        """Do du u yerden max 1 birim uzakta hareket et"""
        for dx, dy in [(0,1), (0,-1), (1,0), (-1,0)]:
            new_x = self.position[0] + dx
            new_y = self.position[1] + dy

            distance = abs(new_x - self._spawn_position[0]) + \
                       abs(new_y - self._spawn_position[1])

            if distance <= 1 and can_move_to(new_x, new_y,
               tile_provider):
                self.position = (new_x, new_y)
                break
```

**ChasingEnemy (Satır 133-208) - Concrete Strategy 2:**

```
class ChasingEnemy(Enemy):
    """
```

```
    Takip Eden D   man: Kendi sat r/s tunu boyunca oyuncuya
       yakla   r.
    Bomberman'a do ru hareket eder.
    """
    MAX_HEALTH = 30

    def update(self, player_pos, tile_provider) -> None:
        """Sat r/s tun boyunca oyuncuya yakla """
        if not player_pos:
            return

        # Yatay veya dikey eksende oyuncuya yakla
        if self._move_horizontal:
            # Yatay hareket (ayn  sat rda)
            if player_pos[0] > self.position[0]:
                new_pos = (self.position[0] + 1,
                   self.position[1])
            else:
                new_pos = (self.position[0] - 1,
                   self.position[1])
        else:
            # Dikey hareket (ayn  s tunda)
            if player_pos[1] > self.position[1]:
                new_pos = (self.position[0], self.position[1] +
                   1)
            else:
                new_pos = (self.position[0], self.position[1] -
                   1)

        #    arpma    kontrol  ve hareket
        if can_move_to(new_pos[0], new_pos[1], tile_provider):
            self.position = new_pos
        else:
            # Y n de i tir
            self._move_horizontal = not self._move_horizontal
```

**SmartEnemy (Satır 210-278) - Concrete Strategy 3:**

```
class SmartEnemy(Enemy):
    """
    Ak ll  D   man: A* algoritmas  ile en k sa yolu bulur.
    Bomberman'  k  eye s k  t rmaya    alr  .
    """
    MAX_HEALTH = 40

    def update(self, player_pos, tile_provider) -> None:
        """A* pathfinding ile en k sa yolu bul"""
        if not player_pos:
            return

```

```
        # A* pathfinding algoritmas  ile en k sa yolu bul
        path = self._find_path(self.position, player_pos,
            tile_provider)

        if len(path) > 1:
            next_pos = path[1]  # path[0] = current position
            self.position = next_pos
```

### 3.5.4 Kullanım Örneği

**Dosya:** `bomberman/controller/game_controller.py` (Satır 191)

```
# Game loop i inde
for enemy in self._enemies:
    enemy.update(self.player.position, self._tile_provider)
    # Her d   man kendi stratejisini uygular:
    # - StaticEnemy: S  n  r l   hareket
    # - ChasingEnemy: Sat r/s tun boyunca takip
    # - SmartEnemy: A* ile en k sa yol
```

**Düşman Oluşturma (Satır 153):**

```
enemy = MonsterFactory.create(enemy_type, position)
# Factory Method Pattern ile strateji se ilir
```

### 3.5.5 Düşman Stratejileri Karşılaştırması

| Strateji | Sağlık | Hız | Davranış | Zorluk |
|---|---|---|---|---|
| StaticEnemy | 20 HP | Yavaş (1.6s) | Sadece spawn noktası etrafında | Kolay |
| ChasingEnemy | 30 HP | Orta (0.8s) | Satır/sütun boyunca takip | Orta |
| SmartEnemy | 40 HP | Hızlı (0.4s) | A* ile en kısa yol | Zor |

Tablo 2: Düşman Stratejileri Karşılaştırması

### 3.5.6 Neden Kullanıldı?

- Farklı düşman davranışlarını birbirinden bağımsız yapma
- Yeni düşman tipi eklemek kolay
- Her düşman kendi hareket algoritmasına sahip

### 3.5.7 Faydaları

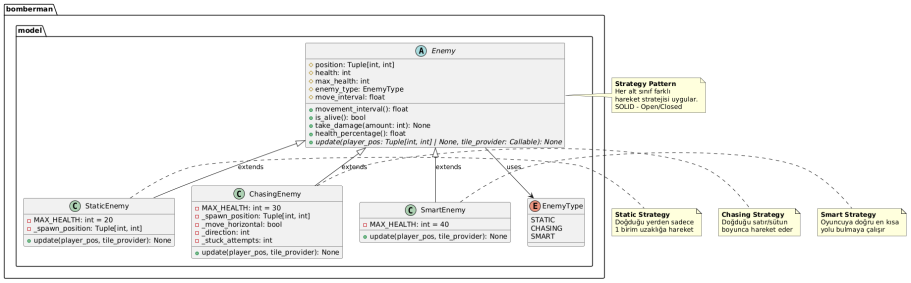
✓ **Interchangeable:** Düşmanlar birbirinin yerine kullanılabilir

✓ **Open/Closed:** Yeni strateji eklemek için mevcut kod değişmez

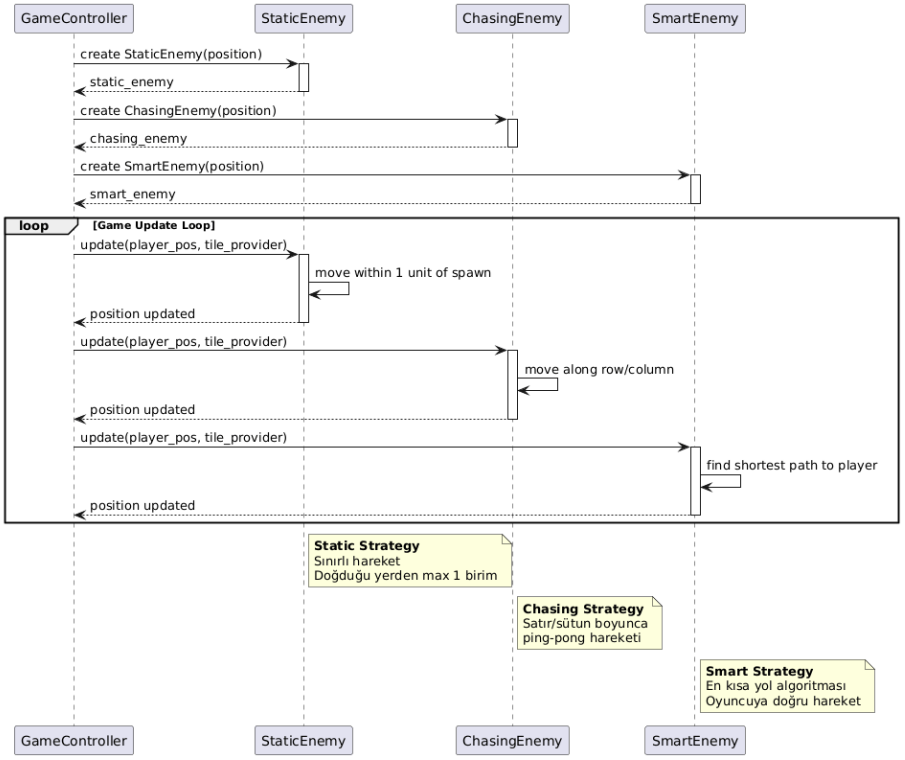
✓ **Single Responsibility:** Her strateji kendi algoritmasından sorumlu

✓ **Testability:** Her strateji ayrı ayrı test edilebilir

### 3.5.8 UML Diyagramı




Şekil 8: Strategy Pattern - Class Diagram




Şekil 9: Strategy Pattern - Sequence Diagram

## Repository Pattern (Data Access) ✓

### 3.6.1 Tanım

Repository, veri erişim mantığını iş mantığından ayıran ve veri kaynağını soyutlayan bir desendir.

### 3.6.2 Projede Nerede Kullanıldı?

**Frontend (Client-side):**

- `bomberman/repository/level_repository_json.py`
- `bomberman/repository/level_repository_postgresql.py`

**Backend (Server-side):**

- `backend/repository/room_repository.py`

### 3.6.3 Frontend Repository (Client-side)

**Dosya:** `bomberman/repository/level_repository_json.py` (Satır 14-203)

```
class LevelRepositoryJSON:
    """JSON dosyas ndan level verilerini y netir"""

    def __init__(self, json_path: str | None = None) -> None:
        self._json_path = Path(json_path or "data/levels.json")
        self._cache: dict[str, LevelDefinition] | None = None

    def find_by_id(self, level_id: str) ->
       Optional[LevelDefinition]:
        """ID'ye g re level bulur"""
        definitions = self._load_all()
        return definitions.get(level_id)

    def find_all(self) -> Iterable[LevelDefinition]:
        """T m levellar  getirir"""
        definitions = self._load_all()
        for key in sorted(definitions.keys()):
            yield definitions[key]

    def save(self, definition: LevelDefinition) -> None:
        """Level kaydeder"""
        definitions = self._load_all()
        definitions[definition.id] = definition
        self._save_all(definitions)

    def delete(self, level_id: str) -> bool:
        """Level siler"""
        definitions = self._load_all()
        if level_id in definitions:
```

```
            del definitions[level_id]
            self._save_all(definitions)
            return True
        return False
```

**Dosya:** bomberman/repository/level_repository_postgresql.py (Satır 25-220)

```
class LevelRepositoryPostgreSQL:
    """PostgreSQL'den level verilerini y netir"""

    def __init__(self, connection_string: str) -> None:
        self._connection_string = connection_string

    def find_by_id(self, level_id: str) ->
       Optional[LevelDefinition]:
        """ID'ye g re level bulur"""
        conn = self._get_connection()
        try:
            cursor = conn.cursor(cursor_factory=RealDictCursor)
            cursor.execute("SELECT * FROM levels WHERE id = %s",
               (level_id,))
            row = cursor.fetchone()
            if row:
                return self._map_row_to_definition(row, ...)
            return None
        finally:
            conn.close()

    def find_all(self) -> Iterable[LevelDefinition]:
        """T m levellar  getirir"""
        conn = self._get_connection()
        try:
            cursor.execute("SELECT * FROM levels ORDER BY id")
            for row in cursor.fetchall():
                yield self._map_row_to_definition(row, ...)
        finally:
            conn.close()
```

### 3.6.4 Backend Repository (Server-side)

**Dosya:** backend/repository/room_repository.py (Satır 17-464)

```
class RoomRepository:
    """PostgreSQL'de oda y netimi repository"""

    def __init__(self) -> None:
        self.connection_string = get_database_url()

    def create_room(self, room: GameRoom) -> bool:
        """Yeni oda olu tur"""
```

```
        with self._get_connection() as conn:
            cur = conn.cursor()
            cur.execute("""
                INSERT INTO rooms (room_id, room_code, level_id,
                                   level_width, level_height,
                                      started)
                VALUES (%s, %s, %s, %s, %s, %s)
            """, (room.room_id, room.room_code, room.level_id,
                  room.level_width, room.level_height,
                     room.started))

            # Oyuncular  ekle
            for player in room.players:
                cur.execute("""
                    INSERT INTO room_players (room_id,
                       player_id, username,
                                              socket_id,
                                                 position_x,
                                                 position_y,
                                              health, ready)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """, (...))

            conn.commit()
            return True

    def get_room_by_code(self, room_code: str) ->
       Optional[GameRoom]:
        """Oda koduna g re oda bul"""
        with self._get_connection() as conn:
            cursor = conn.cursor(cursor_factory=RealDictCursor)
            cursor.execute("SELECT * FROM rooms WHERE room_code
               = %s", (room_code,))
            row = cursor.fetchone()
            if row:
                return self._map_row_to_room(row, conn)
            return None
```

### 3.6.5 Veri Kaynakları

- **JSON:** `data/levels.json` (local development)
- **PostgreSQL:** Neon.tech (production)
- **Tables:** `levels`, `rooms`, `room_players`

### 3.6.6 Kullanım Örneği

**Dosya:** `bomberman/service/level_service.py`

```python
# Repository se imi (JSON veya PostgreSQL)
repository = LevelRepositoryJSON()  # veya
    LevelRepositoryPostgreSQL(...)

# Level y kleme
level = repository.find_by_id("level_1")

# T m levellar listeleme
for level in repository.find_all():
    print(level.id)
```

### 3.6.7 Neden Kullanıldı?

- Veri erişim mantığını iş mantığından ayırma
- Veri kaynağını değiştirmek kolay (JSON ↔ PostgreSQL)
- Test yazmak kolaylaşır (mock repository kullanılabilir)
- Database sorguları tek yerde merkezi olarak yönetilir

### 3.6.8 Faydaları

✓ **Separation of Concerns:** Veri erişim ve iş mantığı ayrı

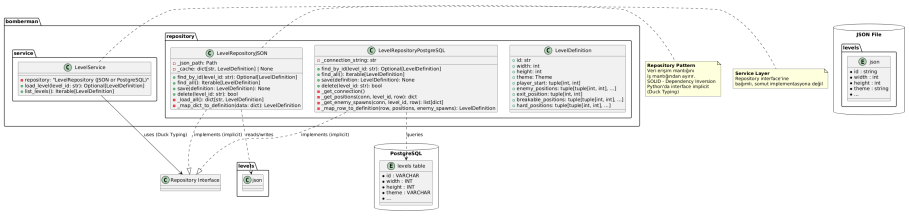
✓ **Testability:** Mock repository ile kolayca test edilir

✓ **Flexibility:** Veri kaynağı kolayca değiştirilebilir

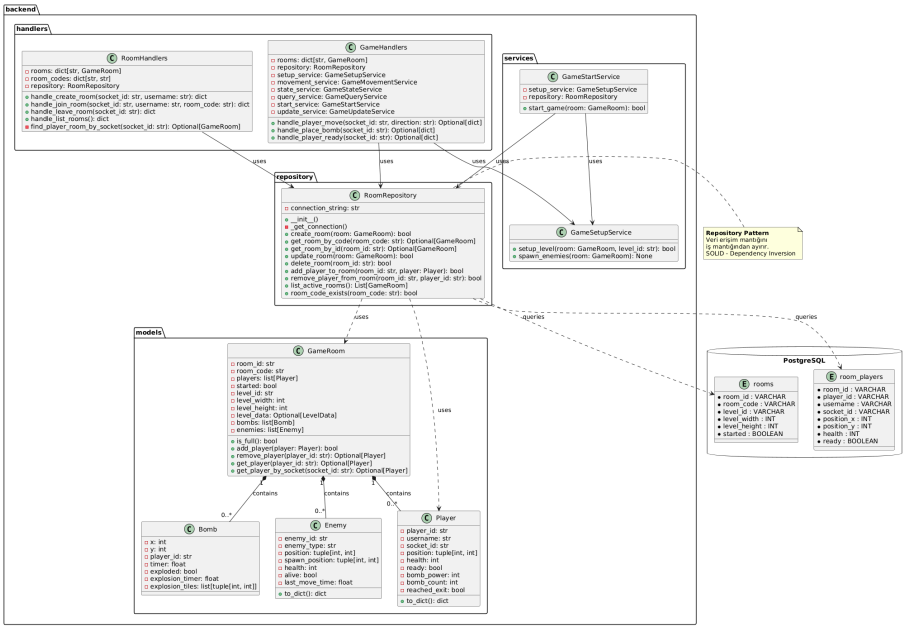
✓ **Centralized Logic:** Database sorguları tek yerde

✓ **Dependency Inversion:** Service'ler repository interface'ine bağlı

### 3.6.9 UML Diyagramı



Şekil 10: Repository Pattern - Class Diagram (Frontend)




Şekil 11: Repository Pattern - Class Diagram (Backend)

# MVC Pattern (Architectural) ✓

### 3.7.1 Tanım

MVC (Model-View-Controller), uygulamayı üç katmana ayıran mimari bir desendir:

- **Model:** Veri ve iş mantığı
- **View:** Kullanıcı arayüzü
- **Controller:** Model ve View arasındaki etkileşim

### 3.7.2 Projede Nerede Kullanıldı?

- **Model:** `bomberman/model/`
- **View:** `bomberman/view/`
- **Controller:** `bomberman/controller/`

### 3.7.3 Model (Veri Katmanı)

**Dizin:** `bomberman/model/`
**Dosya:** `bomberman/model/bomberman.py` (Satır 9-40)

```
class Bomberman:
    """Oyuncu modeli"""
    def __init__(self, character: Character, position:
        Tuple[int, int]):
         self.character = character
         self.position = position
```

```
        self.health = 100
        self.speed = 1.0
        self.bomb_count = 1
        self.bomb_power = 1

    def take_damage(self, amount: int) -> None:
        self.health -= amount

    def heal(self, amount: int) -> None:
        self.health = min(100, self.health + amount)

    def is_alive(self) -> bool:
        return self.health > 0
```

**Dosya:** bomberman/model/enemy.py (Satır 30-77)

```
class Enemy(ABC):
    """D   man modeli (Strategy Pattern)"""
    def __init__(self, position: Tuple[int, int], enemy_type:
       EnemyType):
        self.position = position
        self.health = self.max_health
        self.enemy_type = enemy_type

    @abstractmethod
    def update(self, player_pos, tile_provider) -> None:
        pass
```

### 3.7.4 View (Görsel Katman)

**Dizin:** bomberman/view/

**Dosya:** bomberman/view/game_scene.py (Satır 22-758)

```
class GameScene:
    """Ana oyun  g  r  n  m   - View Layer"""

    def __init__(self, controller: GameController):
        self._controller = controller

    def handle_events(self, events: list[pygame.event.Event]) ->
       None:
        """Kullan c  input'u al ve Controller'a ilet"""
        for event in events:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    self._controller.move_player("up")
                elif event.key == pygame.K_SPACE:
                    self._controller.place_bomb()

    def render(self, surface: pygame.Surface) -> None:
```

```
        """Oyun durumunu ekrana  iz """
        # Controller'dan state al
        game_state = self._controller.view_state()

        # Render i lemleri
        self._draw_map(game_state.level)
        self._draw_player(game_state.player)
        self._draw_enemies(game_state.enemies)
        self._draw_bombs(game_state.bombs)
        self._draw_powerups(game_state.powerups)
```

### 3.7.5 Controller (Kontrol Katmanı)

**Dizin:** bomberman/controller/

**Dosya:** bomberman/controller/game_controller.py (Satır 28-680)

```
class GameController:
    """Oyun kontrolc s  - Model ve View aras ndaki  k pr  """

    def __init__(self, ...):
        # Model'ler
        self.player: Bomberman
        self._enemies: list[Enemy] = []
        self._bombs: list[Bomb] = []

        # Service'ler
        self._level_service: LevelService
        self._collision_service: CollisionService
        self._explosion_service: ExplosionService
        self._event_service: GameEventService
        self._powerup_service: PowerupService

    def move_player(self, direction: str) -> None:
        """Kullan c  input'u     Model g ncelle"""
        if direction == "up":
            new_pos = (self.player.position[0],
               self.player.position[1] - 1)
        # ...

        #     arpma     kontrol
        if self._collision_service.can_move(new_pos,
           self._level):
            self.player.position = new_pos  # Model g ncellendi

    def place_bomb(self) -> None:
        """Bomba yerle tir     Model g ncelle"""
        if len(self._bombs) < self.player.bomb_count:
            bomb = Bomb(self.player.position,
               self.player.bomb_power)
```

```
            self._bombs.append(bomb)  # Model g ncellendi
            self._event_service.emit(GameEventType.BOMB_PLACED)

    def update(self, dt: float) -> None:
        """Game loop - Model'i g ncelle"""
        # D   manlar   g ncelle
        for enemy in self._enemies:
            enemy.update(self.player.position,
               self._tile_provider)

        # Bombalar  g ncelle
        for bomb in self._bombs:
            bomb.timer -= dt
            if bomb.timer <= 0:
                self._handle_explosion(bomb)

    def view_state(self) -> GameViewState:
        """View i in read-only state d nd r"""
        return GameViewState(
            player=self.player,
            enemies=self._enemies,
            bombs=self._bombs,
            powerups=self._powerups,
            level=self._level,
            # ...
        )
```

### 3.7.6 MVC İletişim Akışı

1. **User Input** → View
2. **View** → Controller: `GameScene.handle_events()` → `GameController.move_player("up")`
3. **Controller** → Model: `GameController.move_player()` → `self.player.position = new_pos`
4. **Controller** → Service: `GameController` → `CollisionService.can_move()`
5. **Controller** → View: `GameController.view_state()` → `GameViewState`
6. **View** → Render: `GameScene.render()` → Ekrana çiz

### 3.7.7 Örnek Akış: Oyuncu Hareketi

1. **View:** Kullanıcı input alır
   - Dosya: `bomberman/view/game_scene.py` (Satır 60-99)
   - `if keys[pygame.K_UP]: self._controller.move_player("up")`
2. **Controller:** Model'i günceller

- Dosya: `bomberman/controller/game_controller.py` (Satır 200-250)
- `self.player.position = new_pos` *(Model güncellendi)*

3. **Controller:** View için state sağlar
   - Dosya: `bomberman/controller/game_controller.py` (Satır 403-450)
   - `game_state = controller.view_state()`
4. **View:** State'i render eder
   - Dosya: `bomberman/view/game_scene.py` (Satır 418-758)
   - `view.render(game_state)`

#### 3.7.8 Örnek Akış: Düşman Güncellemesi (Enemy Class İçin Tam Akış)

1. **Model:** Enemy sınıfı tanımlanır
   - Dosya: `bomberman/model/enemy.py`
   - Enemy (Satır 30-77) - Abstract base class
   - StaticEnemy (Satır 80-131), ChasingEnemy (Satır 133-208), SmartEnemy (Satır 210-278)
2. **Controller:** Düşmanları oluşturur
   - Dosya: `bomberman/controller/game_controller.py` (Satır 153)
   - `enemy = MonsterFactory.create(enemy_type, position)`
   - `self._enemies.append(enemy)`
3. **Controller:** Game loop içinde düşmanları günceller
   - Dosya: `bomberman/controller/game_controller.py` (Satır 191)
   - `for enemy in self._enemies: enemy.update(...)`
   - Her düşman kendi stratejisini uygular (Strategy Pattern)
4. **Controller:** View için state hazırlar
   - Dosya: `bomberman/controller/game_controller.py` (Satır 403-450)
   - `return GameViewState(enemies=self._enemies, ...)` *(Model'den View'a)*
5. **View:** Düşmanları render eder
   - Dosya: `bomberman/view/game_scene.py` (Satır 600-650)
   - `def _draw_enemies(self, enemies): ...`

### 3.7.9 Neden Kullanıldı?

- İş mantığı (Controller) ve görsel (View) ayrılması
- Model bağımsız, test edilebilir
- View değiştirilse bile Controller/Model aynı kalır

### 3.7.10 Faydaları

✓ **Separation of Concerns:** Her katman kendi sorumluluğuna odaklanır

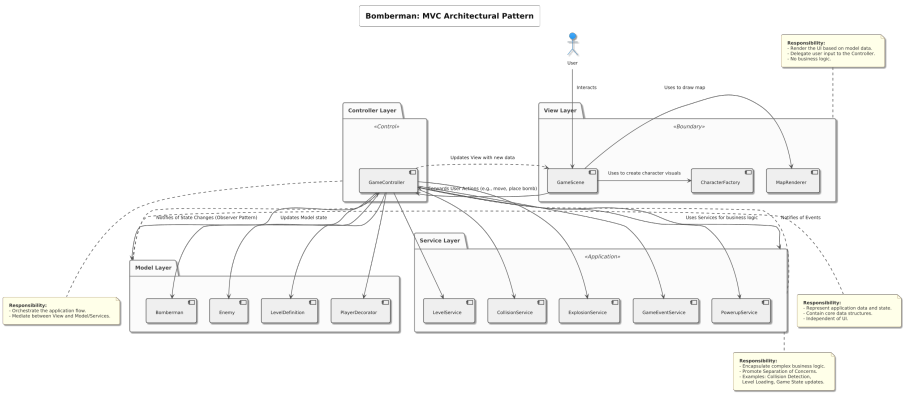
✓ **Testability:** Model ve Controller ayrı ayrı test edilir

✓ **Maintainability:** Kod değişiklikleri lokalize edilir

✓ **Reusability:** Model başka view'larla kullanılabilir

✓ **Parallel Development:** Ekipler farklı katmanlarda çalışabilir

### 3.7.11 UML Diyagramı




Şekil 12: MVC Architectural Pattern - Component Diagram

# UML Diyagramları

## Frontend UML Diyagramları (bomberman/uml/)

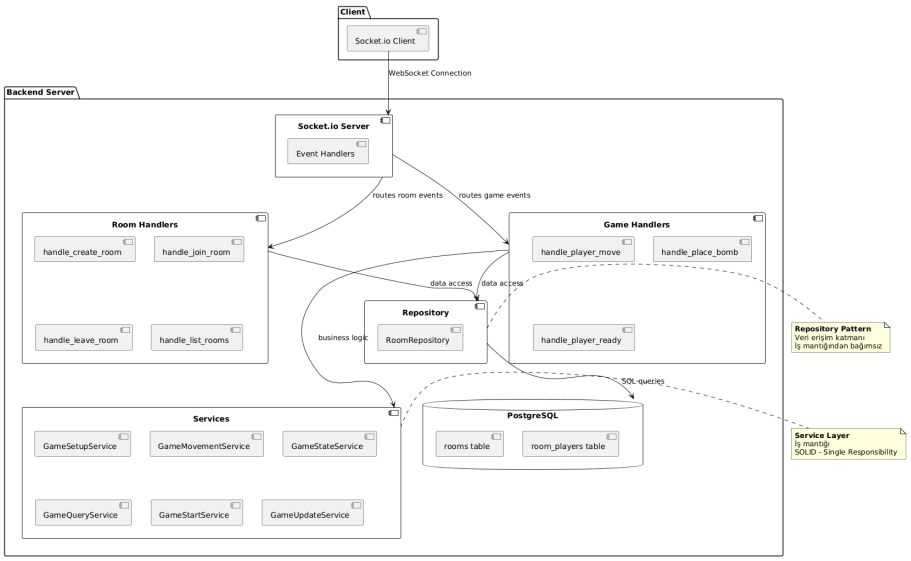
1. `1_factory_method_pattern.puml` - Factory Method class diagram
2. `2_adapter_decorator_pattern.puml` - Adapter & Decorator class diagram
3. `3_observer_pattern.puml` - Observer class diagram
4. `4_strategy_pattern.puml` - Strategy class diagram
5. `5_repository_pattern.puml` - Repository class diagram

6. `6_mvc_architectural_pattern.puml` - MVC architectural diagram
7. `7_observer_sequence.puml` - Observer sequence diagram
8. `8_decorator_sequence.puml` - Decorator sequence diagram
9. `9_strategy_sequence.puml` - Strategy sequence diagram
10. `10_factory_sequence.puml` - Factory sequence diagram

## Backend UML Diyagramları (backend/uml/)

1. `1_repository_pattern_class.puml` - Repository class diagram
2. `2_component_diagram.puml` - Component diagram
3. `3_room_creation_sequence.puml` - Room creation sequence
4. `4_game_start_sequence.puml` - Game start sequence
5. `5_service_layer_class.puml` - Service layer class diagram

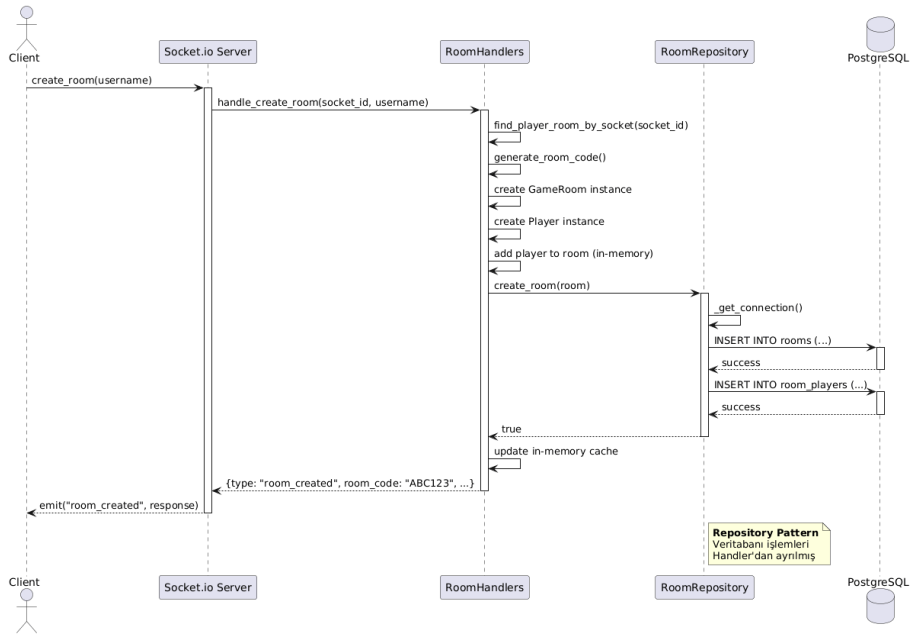
**NOT:** Tüm UML diyagramları PlantUML formatındadır ve render edilmeye hazırdır.




Şekil 13: Backend Component Diagram

# SOLID Prensipleri

Projede her tasarım deseni SOLID prensiplerine uygun şekilde implement edilmiştir:




Şekil 14: Room Creation Sequence Diagram

## Single Responsibility Principle (SRP) ✓

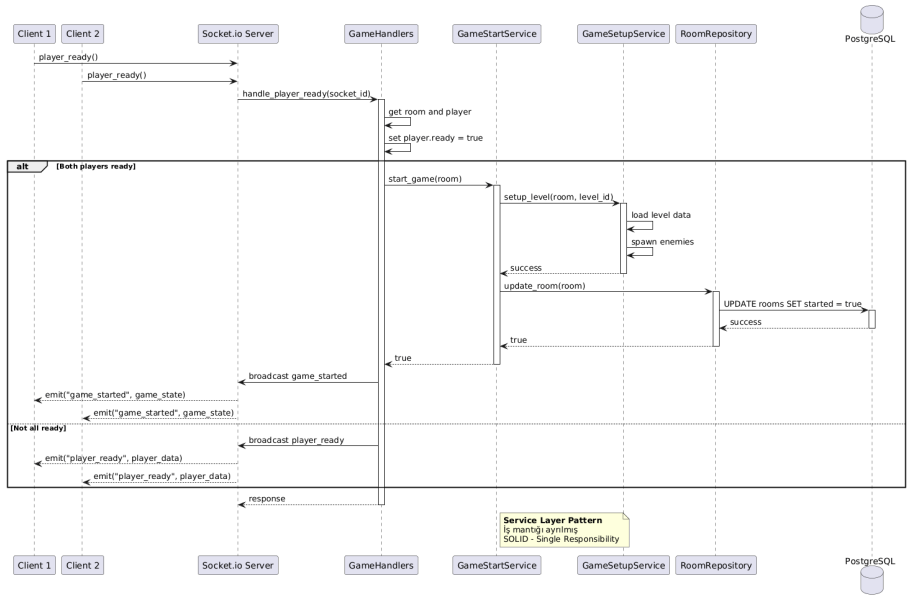
Her sınıf tek bir sorumluluktan sorumlu:

- CharacterFactory: Sadece karakter yaratma
- SoundObserver: Sadece ses efektleri
- ScoreObserver: Sadece skor takibi
- StaticEnemy, ChasingEnemy, SmartEnemy: Her biri kendi stratejisinden sorumlu
- LevelRepositoryJSON: Sadece JSON'dan veri okuma/yazma
- GameController: Sadece oyun akışını yönetme

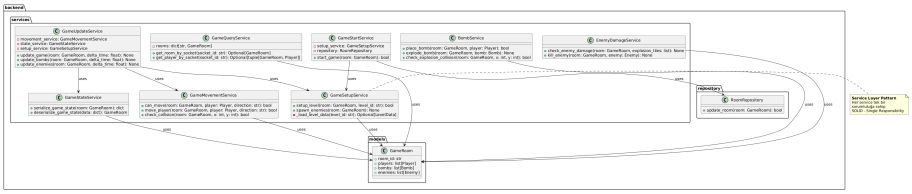
## Open/Closed Principle (OCP) ✓

Genişletmeye açık, değişime kapalı:

- Yeni power-up eklemek: PlayerDecorator sınıfından türet
- Yeni düşman eklemek: Enemy sınıfından türet
- Yeni observer eklemek: GameObserver sınıfından türet
- Yeni repository eklemek: Aynı interface'i implement et
- Mevcut kod değiştirilmez




Şekil 15: Game Start Sequence Diagram



Şekil 16: Service Layer Class Diagram

## Liskov Substitution Principle (LSP) ✓

Alt sınıflar üst sınıfın yerine kullanılabilir:

- Tüm Enemy alt sınıfları Enemy interface'ini implement eder
- Tüm PlayerDecorator alt sınıfları PlayerInterface'i implement eder
- Tüm GameObserver alt sınıfları GameObserver interface'ini implement eder
- Client kod interface'e bağlı, somut implementasyona değil

## Interface Segregation Principle (ISP) ✓

Sınıflar sadece ihtiyaç duydukları metodları implement eder:

- PlayerInterface: Sadece gerekli metodlar (speed, bomb_count, bomb_power, health)
- GameObserver: Sadece `on_event()` metodu
- Repository interface'leri: Sadece CRUD operasyonları
- Her interface minimal ve odaklanmış

### Dependency Inversion Principle (DIP) ✓

Yüksek seviye modüller düşük seviye modüllere bağımlı değil:

- GameController → LevelService (interface'e bağlı)
- GameController → CollisionService (interface'e bağlı)
- Service'ler → Repository interface'lerine bağlı
- Concrete implementasyonlar dependency injection ile verilir

# Sonuç ve Özet

## Proje Gereksinimleri ve Uygulama

| **Tasarım Deseni Kategorisi** | **Gerekli** | **Projede Kullanılan** |
|---|---|---|
| Creational Pattern | 1 | 1 (Factory) |
| Structural Pattern | 1 | 2 (Adapter, Decorator) |
| Behavioral Pattern | 2 | 2 (Observer, Strategy) |
| Repository Pattern | 1 | 3 (JSON, PostgreSQL, Room) |
| Architectural Pattern | 1 | 1 (MVC) |
| **TOPLAM** | **5** | **7** |

Tablo 3: Tasarım Desenleri Uygulama Özeti

## Kod İstatistikleri

**Frontend:**

- Model: 6+ dosya
- View: 8+ dosya
- Controller: 1 dosya (680 satır)
- Service: 12+ dosya
- Repository: 3 dosya

**Backend:**

- Models: 2 dosya
- Repository: 1 dosya (464 satır)
- Services: 10+ dosya
- Handlers: 2 dosya

### Tasarım Deseni Kullanımı

- **Factory Method:** 3 factory sınıfı (Character, Monster, Effect)
- **Decorator:** 4 decorator sınıfı (Speed, BombCount, BombPower, Health)
- **Observer:** 3 observer sınıfı (Sound, Score, Logger)
- **Strategy:** 3 strateji sınıfı (Static, Chasing, Smart enemies)
- **Repository:** 3 repository sınıfı (LevelJSON, LevelPostgreSQL, Room)
- **MVC:** Tam katmanlı mimari (10+ model, 8+ view, 1 controller)

### UML Diyagramları

- **Frontend:** 10 diyagram (Class + Sequence)
- **Backend:** 5 diyagram (Class + Component + Sequence)
- **Toplam:** 15 UML diyagramı

## Belge Bilgileri

| | |
|---|---|
| **Hazırlayan:** | Serdar Can |
| **Öğrenci No:** | 220401096 |
| **Proje:** | Bomberman Game |
| **Tarih:** | 27 Aralık 2025 |