# Kế Hoạch Phát Triển Game Deckbuilding Roguelike Trên Unity 2D

## Phân Tích Design Pattern Phù Hợp

Sau khi phân tích yêu cầu của game deckbuilding roguelike với hệ thống ngũ hành, nạp âm, và lá bài phụ, tôi đề xuất sử dụng kết hợp giữa **ECS (Entity-Component-System)** làm kiến trúc chính với một số nguyên tắc từ **MVC** cho phần UI.

### Tại Sao Chọn ECS?

1. **Tính Mô-đun Cao**:
   - Dễ dàng tạo ra và mở rộng các loại thẻ bài mới
   - Có thể thêm hiệu ứng và thuộc tính mới mà không ảnh hưởng đến hệ thống hiện có
2. **Hiệu Năng Tốt**:
   - Xử lý hiệu quả nhiều thực thể (thẻ bài, hiệu ứng) cùng lúc
   - Tối ưu hóa cho các tính toán phức tạp trong hệ thống chiến đấu
3. **Dễ Bảo Trì**:
   - Tách biệt dữ liệu (Components) và logic (Systems)
   - Dễ dàng gỡ lỗi và kiểm tra từng hệ thống riêng biệt
4. **Phù Hợp Với Gameplay**:
   - ECS rất phù hợp với các game có nhiều thực thể tương tác với nhau
   - Xử lý tốt các trạng thái và hiệu ứng phức tạp (tương sinh, tương khắc, trạng thái kích hoạt)

### Áp Dụng ECS Vào Game

#### Entities (Thực Thể)

- Thẻ bài (Card)
- Người chơi (Player)
- Đối thủ (Enemy)
- Trận đấu (Battle)
- Hiệu ứng (Effect)
- Môi trường (Environment)

#### Components (Thành Phần)

- CardStats (chỉ số của thẻ bài)
- ElementType (loại nguyên tố)
- NapAm (nạp âm)
- Health (máu)
- Mana (năng lượng)
- StatusEffect (hiệu ứng trạng thái)
- ActivationCondition (điều kiện kích hoạt)
- CardVisual (hình ảnh của thẻ)

**Systems (Hệ Thống)**

- CardSystem (quản lý thẻ bài)
- BattleSystem (xử lý chiến đấu)
- ScoreCalculationSystem (tính điểm)
- ElementInteractionSystem (tương tác ngũ hành)
- SupportCardSystem (xử lý thẻ phụ)
- SeasonSystem (ảnh hưởng mùa màng)
- DeckBuildingSystem (xây dựng bộ bài)
- ProgressionSystem (tiến trình và nâng cấp)

## Cấu Trúc Project Unity

```
Assets/
├── Scripts/
│   ├── Core/
│   │   ├── ECS/
│   │   │   ├── Components/
│   │   │   ├── Systems/
│   │   │   └── Entities/
│   │   ├── Managers/
│   │   │   ├── GameManager.cs
│   │   │   ├── AudioManager.cs
│   │   │   ├── UIManager.cs
│   │   │   └── InputManager.cs
│   │   └── Utils/
│   ├── Gameplay/
│   │   ├── Cards/
│   │   ├── Battle/
│   │   ├── Deck/
│   │   ├── Map/
│   │   └── Progression/
│   ├── UI/
│   │   ├── Screens/
│   │   ├── HUD/
│   │   ├── Animation/
│   │   └── Effects/
│   └── Data/
│       ├── ScriptableObjects/
│       │   ├── CardData/
│       │   ├── EnemyData/
│       │   └── EventData/
│       └── Serialization/
├── Resources/
│   ├── Cards/
│   ├── Characters/
│   ├── Environments/
│   └── Effects/
├── Prefabs/
│   ├── Cards/
│   ├── UI/
│   └── Effects/
├── Scenes/
│   ├── MainMenu.unity
│   ├── Map.unity
│   ├── Battle.unity
```

```
│   └── DeckBuilder.unity
└── Art/
    ├── Sprites/
    ├── Animations/
    ├── Particles/
    └── UI/
```

## Áp Dụng Design Pattern Cho Các Tính Năng

### 1. Singleton Pattern

**Áp dụng cho**: Các Manager classes

- **GameManager**: Quản lý trạng thái game tổng thể
- **AudioManager**: Quản lý âm thanh
- **SaveManager**: Quản lý lưu trữ dữ liệu

### 2. Factory Pattern

**Áp dụng cho**: Tạo thẻ bài và hiệu ứng

- **CardFactory**: Tạo ra các loại thẻ bài khác nhau từ dữ liệu
- **EffectFactory**: Tạo ra các hiệu ứng từ định nghĩa

### 3. Observer Pattern

**Áp dụng cho**: Hệ thống sự kiện và kích hoạt

- **EventManager**: Quản lý các sự kiện trong game
- **CardEventSystem**: Theo dõi và phản ứng với các sự kiện liên quan đến thẻ bài

### 4. Strategy Pattern

**Áp dụng cho**: AI đối thủ và tính toán điểm

- **EnemyAI**: Các chiến lược khác nhau cho đối thủ
- **ScoreCalculator**: Các phương pháp tính điểm khác nhau

### 5. Command Pattern

**Áp dụng cho**: Hệ thống hành động trong trận đấu

- **ActionManager**: Quản lý các hành động, cho phép undo/redo

### 6. State Pattern

**Áp dụng cho**: Quản lý trạng thái game và thẻ bài

- **GameStateMachine**: Quản lý các trạng thái game (MainMenu, Battle, Map, etc.)
- **CardStateMachine**: Quản lý trạng thái của thẻ bài (InHand, InPlay, Discarded, etc.)

### 7. Decorator Pattern

**Áp dụng cho**: Nâng cấp và áp dụng hiệu ứng cho thẻ bài

- **CardDecorator**: Thêm hiệu ứng và nâng cấp cho thẻ bài

## Kế Hoạch Xây Dựng Base Code

### Giai Đoạn 1: Thiết Lập Framework (4 tuần)

- **Tuần 1-2**: Thiết lập kiến trúc ECS cơ bản
  - Xây dựng core components và systems
  - Thiết lập các manager classes
  - Tạo cấu trúc dữ liệu cho thẻ bài
- **Tuần 3-4**: Xây dựng hệ thống UI cơ bản
  - Thiết kế giao diện trong trận đấu
  - Tạo các màn hình chính (menu, bản đồ, xây deck)
  - Tích hợp hệ thống sự kiện

### Giai Đoạn 2: Core Gameplay (6 tuần)

- **Tuần 5-6**: Hệ thống thẻ bài và deck
  - Xây dựng CardFactory
  - Tạo ScriptableObjects cho dữ liệu thẻ bài
  - Thiết lập hệ thống quản lý deck
- **Tuần 7-8**: Hệ thống chiến đấu
  - Xây dựng BattleSystem
  - Tạo ScoreCalculationSystem
  - Triển khai ElementInteractionSystem (tương sinh tương khắc)
- **Tuần 9-10**: Hệ thống thẻ phụ và điều kiện kích hoạt
  - Xây dựng SupportCardSystem
  - Tạo các loại điều kiện kích hoạt

- Triển khai cơ chế kích hoạt tự động

### Giai Đoạn 3: Tính Năng Mở Rộng (4 tuần)

- **Tuần 11-12**: Hệ thống môi trường và mùa màng
  - Xây dựng SeasonSystem
  - Tạo EnvironmentSystem
  - Triển khai các hiệu ứng môi trường
- **Tuần 13-14**: Hệ thống tiến trình và nâng cấp
  - Xây dựng ProgressionSystem
  - Tạo hệ thống nâng cấp thẻ bài
  - Triển khai hệ thống nuôi Thập Nhị Chi

### Giai Đoạn 4: Tối Ưu Hóa Và Testing (2 tuần)

- **Tuần 15-16**: Test và cân bằng
  - Tối ưu hóa hiệu năng
  - Cân bằng gameplay
  - Sửa lỗi và tối ưu hóa trải nghiệm

## Chi Tiết Triển Khai ECS

### Entity

Định nghĩa thực thể bằng ID duy nhất và tập hợp các Component:

csharp

```csharp
public class Entity
{
    public int Id { get; private set; }
    private Dictionary<Type, Component> components = new Dictionary<Type, Component>();

    public void AddComponent(Component component)
    {
        Type type = component.GetType();
        if (components.ContainsKey(type))
            components[type] = component;
        else
            components.Add(type, component);
    }

    public T GetComponent<T>() where T : Component
    {
        Type type = typeof(T);
        if (components.ContainsKey(type))
            return (T)components[type];
        return null;
    }

    public bool HasComponent<T>() where T : Component
    {
        return components.ContainsKey(typeof(T));
    }
}
```

## Component

Định nghĩa các thành phần dữ liệu:

csharp

```csharp
public abstract class Component
{
    public Entity Entity { get; set; }
}

public class CardComponent : Component
{
    public string Name { get; set; }
    public CardType Type { get; set; }
    public Rarity Rarity { get; set; }
    public Sprite Artwork { get; set; }
    public string Description { get; set; }
}

public class ElementComponent : Component
{
    public ElementType Element { get; set; }
    public NapAmType NapAm { get; set; }
    public int Power { get; set; }
}

public class SupportCardComponent : Component
{
    public ActivationType ActivationType { get; set; }
    public ActivationCondition Condition { get; set; }
    public Effect Effect { get; set; }
    public int CooldownTime { get; set; }
    public bool IsActive { get; set; }
}
```

## System

Định nghĩa các hệ thống xử lý logic:

csharp

```
public abstract class System
{
    protected EntityManager entityManager;

    public System(EntityManager entityManager)
    {
        this.entityManager = entityManager;
    }

    public abstract void Update(float deltaTime);
}

public class BattleSystem : System
{
    public BattleSystem(EntityManager entityManager) : base(entityManager) { }

    public override void Update(float deltaTime)
    {
        // Xử Lý Logic chiến đấu
    }

    public void CalculateDamage(Entity attacker, Entity defender)
    {
        // Tính toán sát thương dựa trên ngũ hành và nạp âm
    }

    public void ApplyEffects(Entity source, Entity target, Effect effect)
    {
        // Áp dụng hiệu ứng
    }
}

public class SupportCardSystem : System
{
    public SupportCardSystem(EntityManager entityManager) : base(entityManager) { }

    public override void Update(float deltaTime)
    {
        // Kiểm tra và kích hoạt các thẻ phụ
        var supportCards = entityManager.GetEntitiesWithComponents<SupportCardComponent>();

        foreach (var entity in supportCards)
        {
```

```
            var supportCard = entity.GetComponent<SupportCardComponent>();

            if (CheckActivationCondition(entity, supportCard.Condition))
            {
                ActivateSupportCard(entity);
            }
        }
    }

    private bool CheckActivationCondition(Entity entity, ActivationCondition condition)
    {
        // Kiểm tra điều kiện kích hoạt
        return true; // Placeholder
    }

    private void ActivateSupportCard(Entity entity)
    {
        // Kích hoạt hiệu ứng của thẻ phụ
    }
}
```

## Lộ Trình Phát Triển Tổng Thể

### MVP (Minimum Viable Product) - 16 tuần

- Thiết lập framework ECS
- Hệ thống thẻ bài cơ bản
- Cơ chế chiến đấu và tính điểm
- Hệ thống ngũ hành và nạp âm
- UI cơ bản

### Phiên Bản Alpha - thêm 8 tuần

- Hệ thống thẻ phụ đầy đủ
- Hệ thống mùa màng và môi trường
- Đa dạng loại thẻ bài
- Bản đồ và tiến trình roguelike

### Phiên Bản Beta - thêm 8 tuần

- Hệ thống tiến trình và nâng cấp

- Nuôi và phát triển Thập Nhị Chi
- Cân bằng gameplay
- Polish và tối ưu hóa

### Phiên Bản Hoàn Chỉnh - thêm 8 tuần

- Content đầy đủ (thẻ bài, đối thủ, sự kiện)
- Hoàn thiện UI/UX
- Âm thanh và hiệu ứng hình ảnh
- Testing và sửa lỗi cuối cùng

## Kết Luận

Kiến trúc ECS là lựa chọn phù hợp nhất cho dự án game deckbuilding roguelike này vì:

1. Tính mô-đun cao, dễ dàng mở rộng và thêm nội dung mới
2. Xử lý hiệu quả các hệ thống phức tạp (ngũ hành, nạp âm, thẻ phụ)
3. Tách biệt rõ ràng dữ liệu và logic, giúp dễ bảo trì và gỡ lỗi
4. Phù hợp với việc quản lý nhiều thực thể và trạng thái khác nhau

Thời gian dự kiến để xây dựng base code là khoảng 16 tuần (4 tháng), sau đó là các giai đoạn phát triển nội dung và tối ưu hóa. Tổng thời gian phát triển từ concept đến phiên bản hoàn chỉnh là khoảng 40 tuần (10 tháng).