SFAbilitySystem Documentation

Core Architecture

1. **AbilityBase.cs**

Abstract base class for all ability types.

```
1. [Serializable]
2. public abstract class AbilityBase
3. {
4.     // Base class for all abilities
5. }
```

2. **AbilityContainer.cs**

Wrapper class for managing ability tiers.

Field/Method	Description
AbilityBase[] abilityTiers	Array of ability instances for different tiers
int Count	Returns the number of ability tiers

3. AbilityManager.cs

Central hub for card and ability management.

Key Features:

- Maintains current card collection
- Provides card retrieval methods
- Handles card addition/removal
- Manages callback subscriptions

Public API:

```
 1. // Card Retrieval
 2. bool TryGetCard<CD, T>(out CD card, out T ability, out int level)
 3. bool TryGetCards<CD, Ability>(ref CD[] cards, ref Ability[] abilities)
 4.
 5.
 6. // Card Management
 7. void AddCard(CardData card, int level = -1)
 8. void RemoveCard(CardData card)
 9. void UpdateCards(List<CardEntry> cardsData, bool notifyCallback = true)
10.
11. // Callbacks
12. void AddCardsPoolUpdatedCallback(ICardsPoolUpdated callback)
13. void RemoveCardsPoolUpdatedCallback(ICardsPoolUpdated callback)
```

4. **CardData.cs**

ScriptableObject base for card definitions.

Property/Field	Description
AbilityBase[] abilities	All ability tiers for this card
string abilityName	Display name
Sprite abilityIcon	Visual representation
string abilityDescription	Dynamic description with template variables
CardData cardToUnlock	Required card to unlock this one

Key Methods:

```
1. T GetAbility<T>(int tier = 0) // Gets ability of specific type/tier
2. string GetDescription(int level) // Formats description with current values
```

5. CardDatabase.cs

Central registry of all cards.

Features:

- Card lookup by hash
- Random card selection
- Duplicate cleaning
- Packing/Unpacking card IDs

Utility Methods:

```
1. short GetHashByAbility(AbilityBase ability)
2. AbilityBase GetAbilityByHash(short hash)
3. CardEntry[] GetRandomCards(CardEntry[] availableCards, int count = 5)
```

6. **CardEntry.cs**

Data container for card instances.

```
1. [Serializable]
2. public class CardEntry
3. {
4.     public CardData card;
5.     public int level;
6. }
```

7. ICardsPoolUpdated.cs

Callback interface for card pool changes.

```
1. public interface ICardsPoolUpdated
2. {
3.     void OnCardsPoolUpdated(AbilityManager abilityManager);
4. }
```

Demo Implementation

Ability Types

1. ActiveAbilityBase.cs

Base class for active abilities.

```
1. public abstract class ActiveAbilityBase : AbilityBase
2. {
3.     public float cooldown = 5f;
4.     public float castTime = 0.5f;
5. }
```

2. **FireballAbility.cs**

```
1. public class FireballAbility : ActiveAbilityBase
2. {
3.     public float damage = 50f;
4.     public float projectileSpeed = 30f;
5. }
6.
```

3. **LaserAbility.cs**

```
1. public class LaserAbility : ActiveAbilityBase
2. {
3.     public float damage = 10f;
4.     public float duration = 5f;
5.     public float range = 50f;
6.     public float damageTickInterval = 0.2f;
7. }
8.
```

4. **Passive Abilities**

```
1. public class FastLegs : AbilityBase { public float speedMultiplier = 1.2f; }
2. public class MoreHP : AbilityBase { public float hpBoost = 0.1f; }
```

Active Ability System

1. **ActiveCardData.cs**

Specialized CardData for active abilities.

```
1. public class ActiveCardData : CardData
2. {
3.     public MonoBehaviour abilityLogicPrefab => _abilityLogicPrefab;
4.     [SerializeField] private ActiveAbilityLogicBase _abilityLogicPrefab;
5. }
```

2. **ActiveAbilitiesController.cs**

MonoBehavior managing active ability UI and input.

Key Responsibilities:

- Hotkey binding (1-3)
- Cooldown visualization
- Ability instantiation
- Dependency injection

Implementation Notes:

- Implements ICardsPoolUpdated for automatic ability updates
- Manages ActiveCardEntry instances
- Handles particle effects and UI updates

3. ActiveAbilityLogicBase.cs

Base class for ability logic.

```
1. public abstract class ActiveAbilityLogicBase : MonoBehaviour
2. {
3.     public float CurrentCooldown { get; protected set; }
4.     public abstract void Initialize(AbilityManager abilityManager, ActiveAbilityBase
abilityBase);
5.     public abstract void PerformAction();
6. }
```

4. ActiveLogicBase<T>.cs

Generic implementation with cooldown management.

```
1. public abstract class ActiveLogicBase<T> : ActiveAbilityLogicBase
```

```
    where T : ActiveAbilityBase
{
    // Handles:
    // - Cooldown tracking
    // - Casting state
    // - Automatic initialization
}
```

5. Concrete Implementations

FireballLogic.cs

- Spawns projectile prefab
- Applies damage on hit
- Requires Camera dependency for spawn point

LaserLogic.cs

- Continuous beam attack
- Damage ticks at intervals
- Visual effects system
- Camera dependency for aiming

Usage Flow

1. **Create Abilities**:
    - Derive from AbilityBase or ActiveAbilityBase

2. **Create Cards**:
    - Make CardData (RMB/Create/SFAbilitySystem/Card)
    - Assign abilities and logic prefabs
    - Configure stats in inspector
3. **Setup Database**:
    - Use CreateCardDatabase menu item (Tools/SFAbilitySystem/Create Card Database)
    - Populate CardDatabase with all cards
4. **Runtime**:
    - AbilityManager tracks active cards
    - ActiveAbilitiesController handles player input
    - Logic classes manage specific behaviors

Sources:

https://game-icons.net/