Mon Co Assessment - Documentation

Binura de Zoysa - 30th May 2025

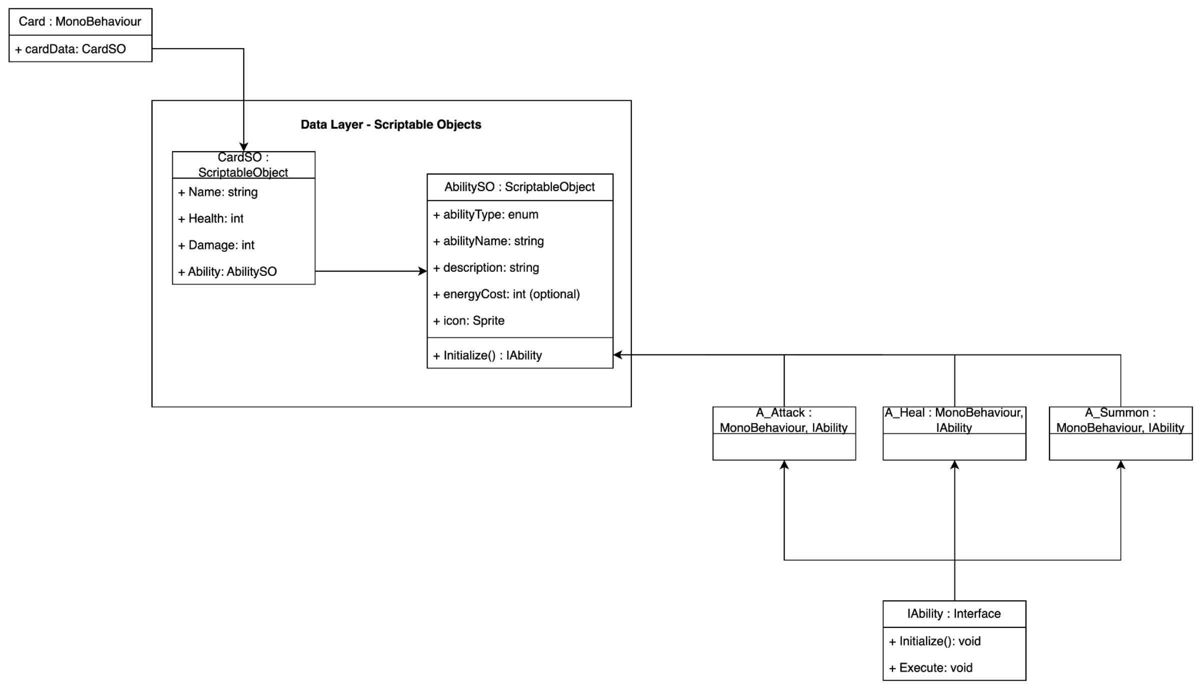
Overview

This documentation provides technical guidelines and explanations of the thought process behind implementing 1) Card Ability, 2) Card Visual, 3) UI features requested as a part of the technical assessment for Mon Co's Senior Game Developer position. As the guidelines suggested to spend not more than 4 hours on the assessment, **the priority was set to define a modular and scalable architecture that is easy for both developers and designers to use and populate content.**

The system was designed with a few limitations in mind:

- Each card can only contain one ability - however, the architecture can support multiple abilities per card with minimal changes.
- A single ability can appear in more than one card.
- Cards and abilities will be populated by designers with minimal scripting knowledge, therefore, Scriptable Objects were used.
- A single non-card enemy was added to demonstrate the use of abilities. Changing to a card vs card system can be achieved with minimal changes.
- No effort was put into visual and game feel to maximize time spent on better systems.

Architecture


Card : MonoBehaviour
+ cardData: CardSO
Data Layer - Scriptable Objects
CardSO : ScriptableObject
+ Name: string
+ Health: int
+ Damage: int
+ Ability: AbilitySO
AbilitySO : ScriptableObject
+ abilityType: enum
+ abilityName: string
+ description: string
+ energyCost: int (optional)
+ icon: Sprite
+ Initialize() : IAbility
A_Attack : MonoBehaviour, IAbility
A_Heal : MonoBehaviour, IAbility
A_Summon : MonoBehaviour, IAbility
IAbility : Interface
+ Initialize(): void
+ Execute: void

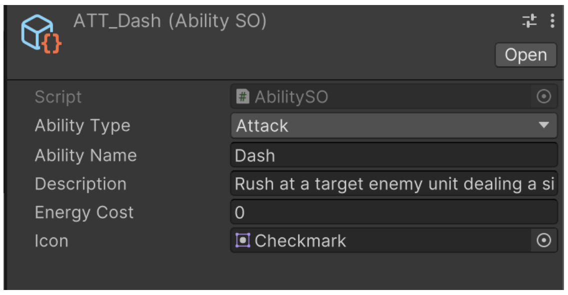
Card and Ability architecture UML Diagram

The system architecture uses six different layers:

- MonoBehaviour base for runtime functions. Eg: Display card information, execute abilities, etc.
- Scriptable Object base for data. Eg: Card data, Ability data and attributes, Player Deck data, etc.
- Interfaces for modular and scalable functions. Eg: Abilities
- Enums for categorizing. Eg: Ability types
- Static classes for triggering events to communicate data across modules and submodules of the system. It reduces the necessity for singletons. Eg: Gameplay Events
- Singletons are used for managing global components for efficient use. Eg: a manager for object pooling that refers to multiple pools.

AbilitySO.cs

AbilitySO is a Scriptable Object class that allows developers and designers to easily create new Abilities that inherit core elements from different ability types. So far, there are Attack, Heal, and Summon ability types added.


ATT_Dash (Ability SO)

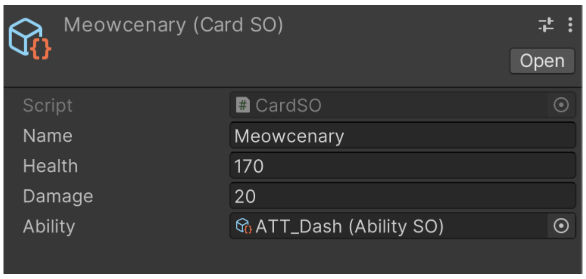
Open

Script	AbilitySO
Ability Type	Attack
Ability Name	Dash
Description	Rush at a target enemy unit dealing a si
Energy Cost	0
Icon	Checkmark

Due to the CardSO class already defining a “Damage” property, AbilitySO does not override the value. However, it can be implemented to allow abilities to independently hold damage data similar to Pokemon TCG Pocket.

CardSO.cs

CardSO is a Scriptable Object class that was pre-provided in the project. It was extended to include AbilitySO to communicate abilities included in the card, and an Initialize() function was added to discover and populate required components for modular abilities at runtime. It was decided to populate at runtime to ensure a maximum developer and designer friendly workflow at scale.


Meowcenary (Card SO)

Script	CardSO
Name	Meowcenary
Health	170
Damage	20
Ability	ATT_Dash (Ability SO)

PlayerDeckSO.cs

PlayerDeck is a Scriptable Object class that can hold data of cards owned by the player. It uses a List of CardSO that can be modified at runtime using methods such as AddCard, RemoveCard, and GetPlayerCards.

Card.cs

Card is a MonoBehaviour script that controls runtime functionalities of each Card in play. It communicates with the data layer and other MonoBehaviours such as the UI through static events defined in GameplayEvents.cs. Some of its core functions include managing health, executing abilities, and updating display information (eg: name, health, description, etc.)

GameplayEvents.cs

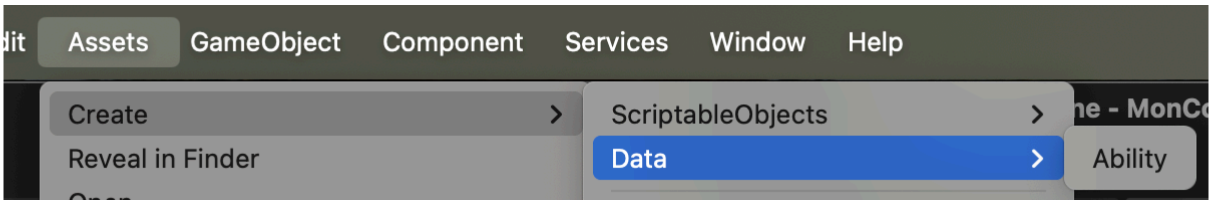
GameplayEvents is a static class holding session based events. It decentralizes the architecture and allows communication across many components on a session basis.

Object Pooling

Considering a pooling system at early stages of architecture can help a TCG style game that requires many active units of the same GameObject when it comes to late stage performance. ObjectPool.cs allows for modular creation of GameObject pools and PoolManager.cs allows easy access to object pools as a singleton.

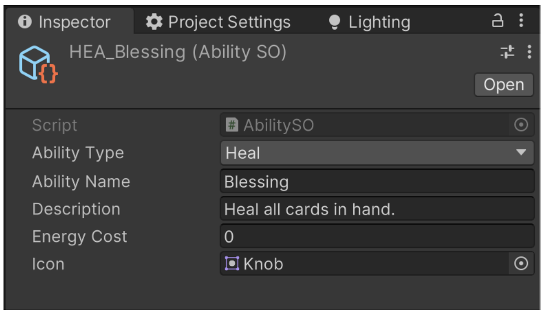
Developer/Designer Guidelines

Creating a new Ability



Navigate to Assets > Create > Data > Ability to create a new Ability Scriptable Object.

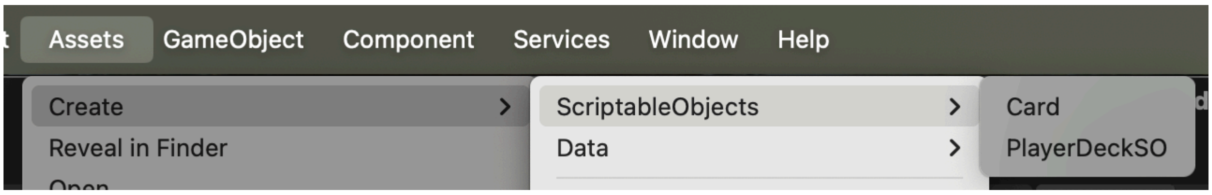
Select the Ability Type and populate the rest of the information.



HEA_Blessing (Ability SO)

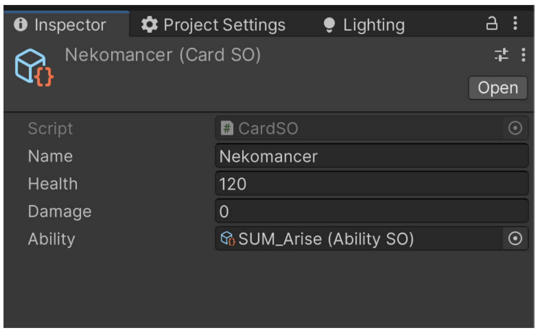
Script	AbilitySO
Ability Type	Heal
Ability Name	Blessing
Description	Heal all cards in hand.
Energy Cost	0
Icon	Knob

Creating a new Card



Navigate to Assets > Create > ScriptableObject > Card to create a new Ability Scriptable Object.

Assign an Ability from the created Abilities and populate the rest of the information.



Nekomancer (Card SO)

Script	CardSO
Name	Nekomancer
Health	120
Damage	0
Ability	SUM_Arise (Ability SO)

Challenges

- Creating a modular and scalable architecture for thousands of abilities and cards required thinking beyond a developer’s use of the system. Designing systems that are editor friendly while utilizing the given CardSO script was challenging.

  The solution was to create Abilities as modular scriptable objects that inherit standard properties and methods through an Interface. However, both interface and Scriptable Objects do not provide much control over runtime gameplay functionalities (eg: lifecycle methods, coroutines, cannot be attached to GameObjects, etc.). In order to maintain use of use and scalability, the ability type scripts inherited from both MonoBehaviour and IAbility interface, allowing it to be attached to GameObjects at runtime. The relevant ability type script was added as a component when a new card was instantiated on to the player’s hand. This system allows for unlimited creation of abilities and ability types to seamlessly work at scale.

- Deciding between allowing only one ability per card vs multiple abilities per card was a challenge as it was not defined in the assessment guidelines. The decision to go with a single ability per card was based on CardSO containing a property for “Damage” and the guidelines specified only to extend on it.

- Visual presentation of abilities was challenging given the 4 hour timeframe. While creating an enemy card may have been more relevant, the architecture needed for it to be robust would have required more planning. A simpler system was to use a dummy enemy to receive damage.

Future Implementations

- Based on the game’s design, the ability type scripts can be extended further to allow more visual and functional control over each ability. At the moment, only an execute function is added to demonstrate how it would work functionally.

- Adding properties and methods to support UI/UX, animations, effects, and overall game feel should be considered into the architecture.

- Creating editor tools to view, simulate, and design abilities and cards can save hundreds of hours of production time at scale.

- Implement more event systems and managers for overall processing of runtime functionalities and data.

Timeline

Task	**Duration**
Research on TCG and planning tasks	30 minutes
Designing system architecture (SO, Interfaces, Events, etc.)	60 minutes
Systems Implementation	90 minutes
UI and Abilities Gameplay (MonoBehaviours) Implementation	40 minutes
Documentation	20 minutes
	240 minutes / 4 hours

Use of AI

- ChatGPT was used to research TCG architecture and popular abilities. It was not used for scripting.
- GitHub Co-Pilot (Visual Studio Code) was used for ease of coding.