**Add Abilities**

**Preparation**

To add a new ability you would need to create at least three new classes - one for ability data, ability level and ability behavior. Depending on the type of the new ability we recomend using one of these two folders:



**Ability Type Enum**

Open the AbilityType.cs script located at Path: 

Add a new value to the AbilityType enum that will represent the new ability.

**Ability Level class**

Create an ability level class. It should have [System.Serializable] attribute and inherit from AbilityLevel. Add fields to this class that will change with the progression of the ability.

Copy

[System.Serializable]

public class NewAbilityLevel : AbilityLevel

{

[SerializeField] int someUpgradableValue = 5;

public int SomeUpgradableValue => someUpgradableValue ;

}

**Ability Data class**

Create an ability data class. It should inherit from GenericAbilityData<NewAbilityLevel>. All necessary functionality is inside this parent class. If you have a need for an additional fields that does not change with the progression of the ability, add them here. Also, create Awake and OnValidate methods and assign the type field inside them to prevent accidentally changing the type of created asset.

Copy

[CreateAssetMenu(fileName = "New Ability Data")]

public class NewAbilityData : GenericAbilityData<NewAbilityLevel>

{

private void Awake()

{

type = AbilityType.NewAbilityType;

}

private void OnValidate()

{

type = AbilityType.NewAbilityTypedgame;

}

}

**Ability Behavior class**

Create an Ability Behavior Class. Implement the custom behavior of the new ability here. The game object with this class assigned will be instantiated when the player selects this ability for the first time. After evolution or the end of the game this game object will be destroyed.

Copy

public class NewAbilityBehavior : AbilityBehavior<HealEndgameAbilityData, HealEndgameAbilityLevel>

{

// Gets called when the ability is selected for the first time

public override void Init(AbilityData data, int levelId)

{

base.Init(data, levelId);

}

// Gets called when the ability is selected for the first time

protected override void SetData(HealEndgameAbilityData data)

{

base.SetData(data);

}

// Gets called when the ability is selected for the first time and after every upgrade

protected override void SetAbilityLevel(int levelId)

{

base.SetAbilityLevel(levelId);

}

// Gets called every time the ability get's upgraded.

// Dose not get called when the ability is selected for the first time

protected override void OnAbilityUpgraded(int levelId)

{

base.OnAbilityUpgraded(levelId);

}

// Calls when the ability is removed due to evolution or due to the end of the game

// Clear pools, projectiles, spawned game objects here

public override void Clear()

{

base.Clear();

}

}

**Prefab**

Create a new empty prefab We recomend using this folder 

Assign created ability behavior script to it's root object.

**Scriptable Object**

Create scriptable object of the ability data class, using context menu (Select the name from the [CreateAssetMenu] attribute). We recomend using folder.

Fill it's fields. Use tooltips if you get confused. Assign the prefab from the previous step to the "prefab" field.

Add the scriptable object to the Abilities Database, located here:.

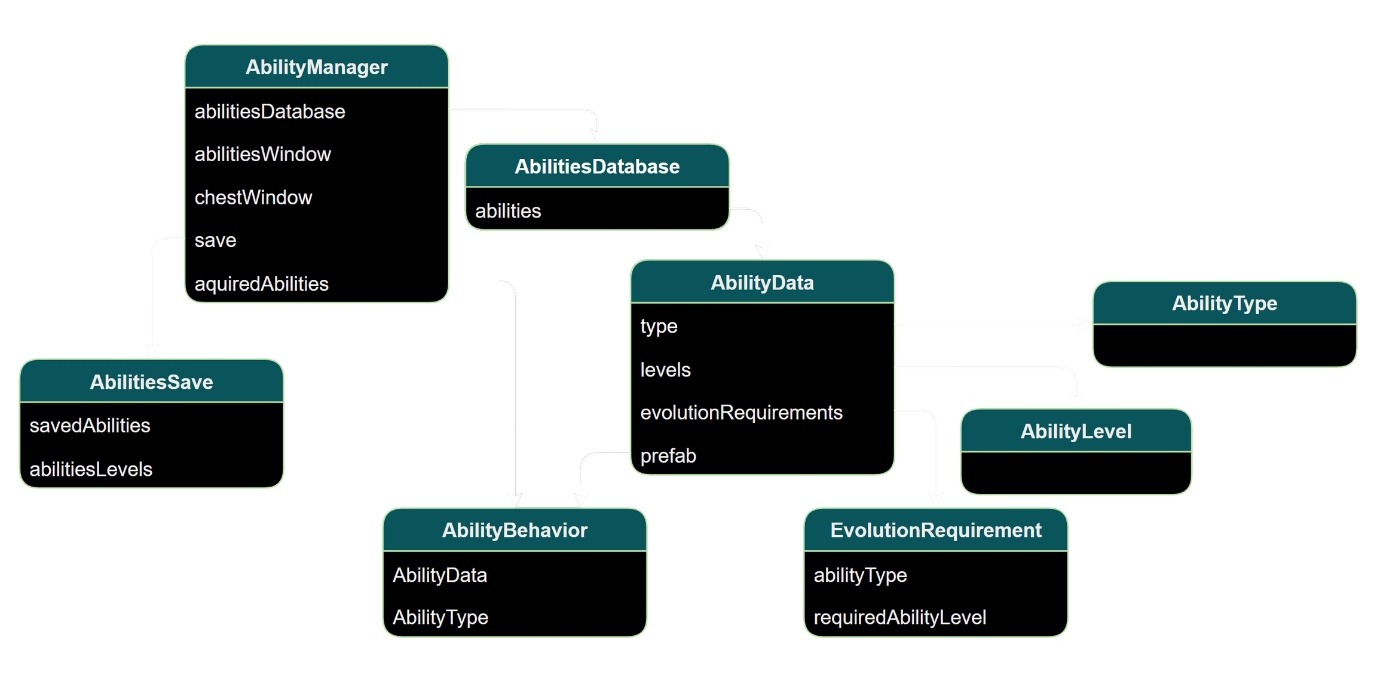
You're good to go!

**Testing ability**

To quickly test the new ability, locate "Testing Preset" asset, add the ability to the abilities list, then assing this "Testing Preset" to the corresponding field of the "Stage Controller" script. It is attached to the "Game Management" gameObject in the "Game" scene.

Don't forget to remove the preset when you're done

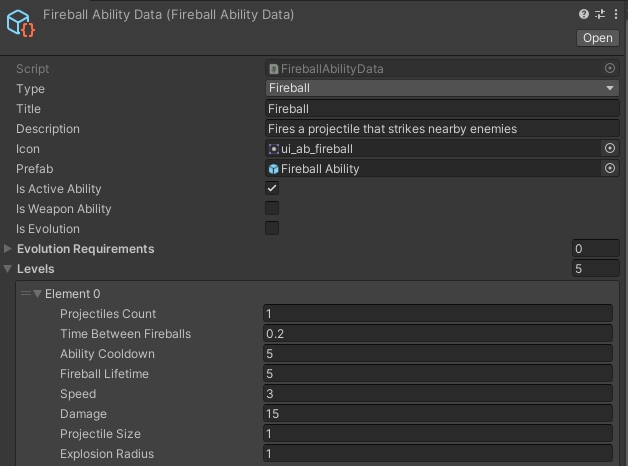
**Abilities System Structure**



Simplified diagram of Abilities System

Each ability is described by a two classes - Ability Data and Ability Behavior.

**Ability Data**

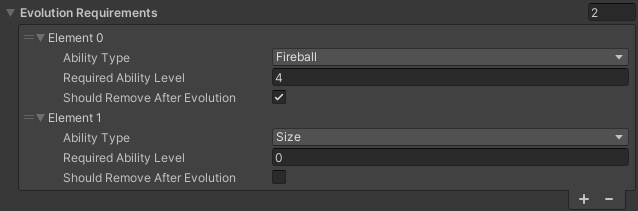


Ability Data example

Ability Data - is an abstract class that contains all the necessary information about the ability (such as it's type, title, description, prefab, levels, etc.). Each ability has it's unique class that inherits from the AbilityData. Ability Data class is a Scriptable Object, which means that it contains presistent information that does not get erased between game sessions.

Each ability has levels - the upgradable part of an ability. They usually contain such information as damage, projectiles count and speed, etc.

If an ability is an evolution, it has a evolution requirement - a list of abilities with their levels.

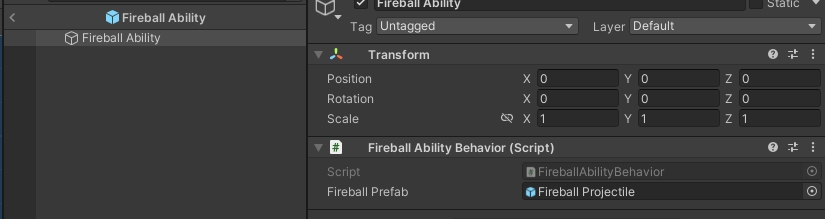


Evolution Requirement List example

All Ability Data scriptable objects are stored in the AbilitiesDatabase scriptable object.

**Ability Behavior**

Ability Behavior is a gameplay representation of an ability. It has it's own prefab and implements the behavior of the ability. For example, FireballAbilityBehavior spawns fireball projectiles according to the FireballAbilityData parameters. It initiates the pool of fireball projectiles in it's Awake method, and starts AbilityCoroutine when it gets it's ability level. This coroutine is responsible for the spawning and initializing data of the fireball projectiles.



Fireball Ability Prefab

All Ability Behaviors are managed by an Ability Manager - a singleton object that are present in the Game scene hierarchy. It is responsible for spawning, upgrading, and removing ability behaviors, selecting ability types for the Abilities Window and Chest Window.

**Ability Types**

There are two main types of player abilities in the game - active and passive.

**Active abilities** are the ones that damage enemies. They usually have projectiles or fields that damage enemies on collision. An active ability has it's corresponding evolution - a more powerful version of the said ability. Every evolution also requires a specific passive ability.

**Passive abilities** - are the ones that improve player stats, such as damage, HP, speed, etc.

The template features 36 abilities (12 active, 12 passive, 12 evolution)