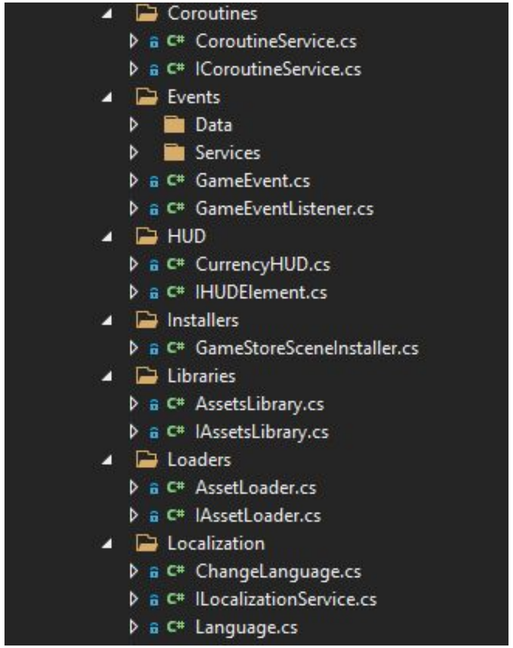
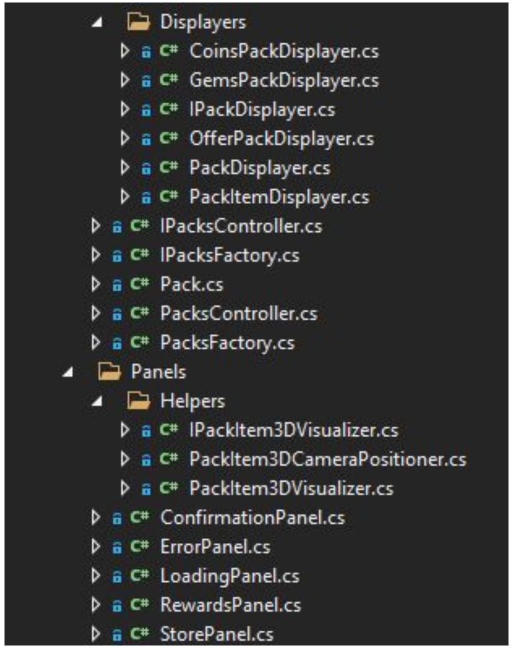
BASIC MOBILE GAME STORE

PROJECT OVERVIEW

- Respecting the single responsibility principle



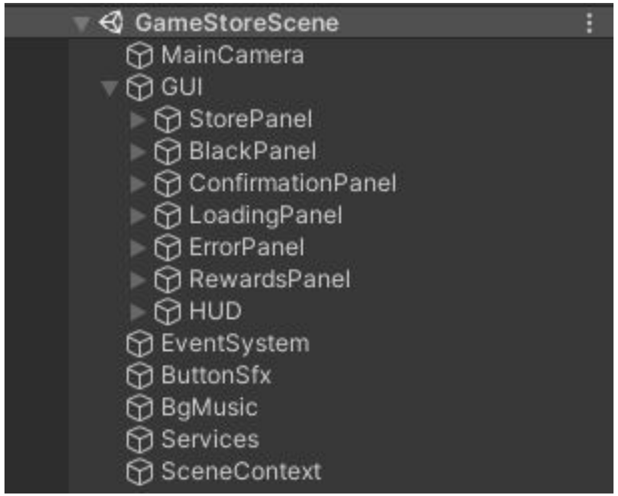
```
Coroutines
    CoroutineService.cs
    ICoroutineService.cs
Events
    Data
    Services
    GameEvent.cs
    GameEventListener.cs
HUD
    CurrencyHUD.cs
    IHUDElement.cs
Installers
    GameStoreSceneInstaller.cs
Libraries
    AssetsLibrary.cs
    IAssetsLibrary.cs
Loaders
    AssetLoader.cs
    IAssetLoader.cs
Localization
    ChangeLanguage.cs
    ILocalizationService.cs
    Language.cs
```



```
    Displayers
        CoinsPackDisplayer.cs
        GemsPackDisplayer.cs
        IPackDisplayer.cs
        OfferPackDisplayer.cs
        PackDisplayer.cs
        PackItemDisplayer.cs
    IPacksController.cs
    IPacksFactory.cs
    Pack.cs
    PacksController.cs
    PacksFactory.cs
Panels
    Helpers
        IPackItem3DVisualizer.cs
        PackItem3DCameraPositioner.cs
        PackItem3DVisualizer.cs
    ConfirmationPanel.cs
    ErrorPanel.cs
    LoadingPanel.cs
    RewardsPanel.cs
    StorePanel.cs
```

SCENE HIERARCHY

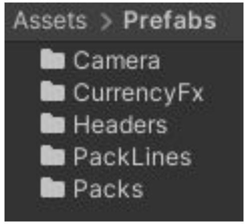
- Clean structure for easy modification

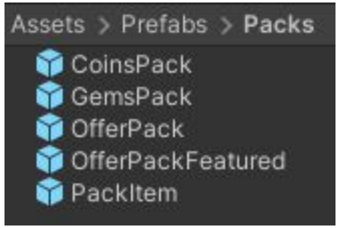


```
▼ GameStoreScene
    MainCamera
  ▼ GUI
    ▶ StorePanel
    ▶ BlackPanel
    ▶ ConfirmationPanel
    ▶ LoadingPanel
    ▶ ErrorPanel
    ▶ RewardsPanel
    ▶ HUD
    EventSystem
    ButtonSfx
    BgMusic
    Services
    SceneContext
```

PREFABS MANAGEMENT

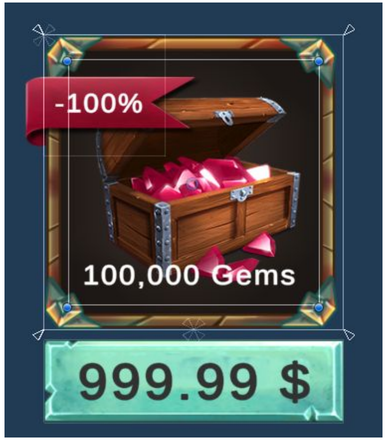
- Prefabs are the key to avoid scene conflicts and for designers and artists to test/make updates without interfering in the programmers work!

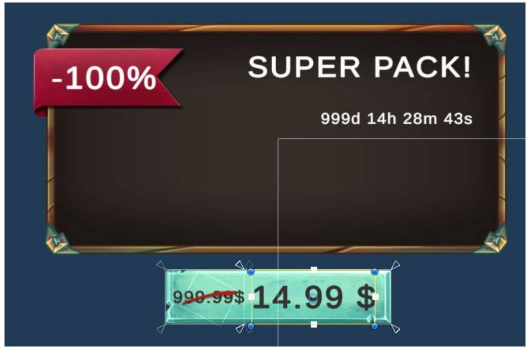

Assets > Prefabs
Camera
CurrencyFx
Headers
PackLines
Packs


Assets > Prefabs > Packs
CoinsPack
GemsPack
OfferPack
OfferPackFeatured
PackItem

STORE BUILDING PROCESS

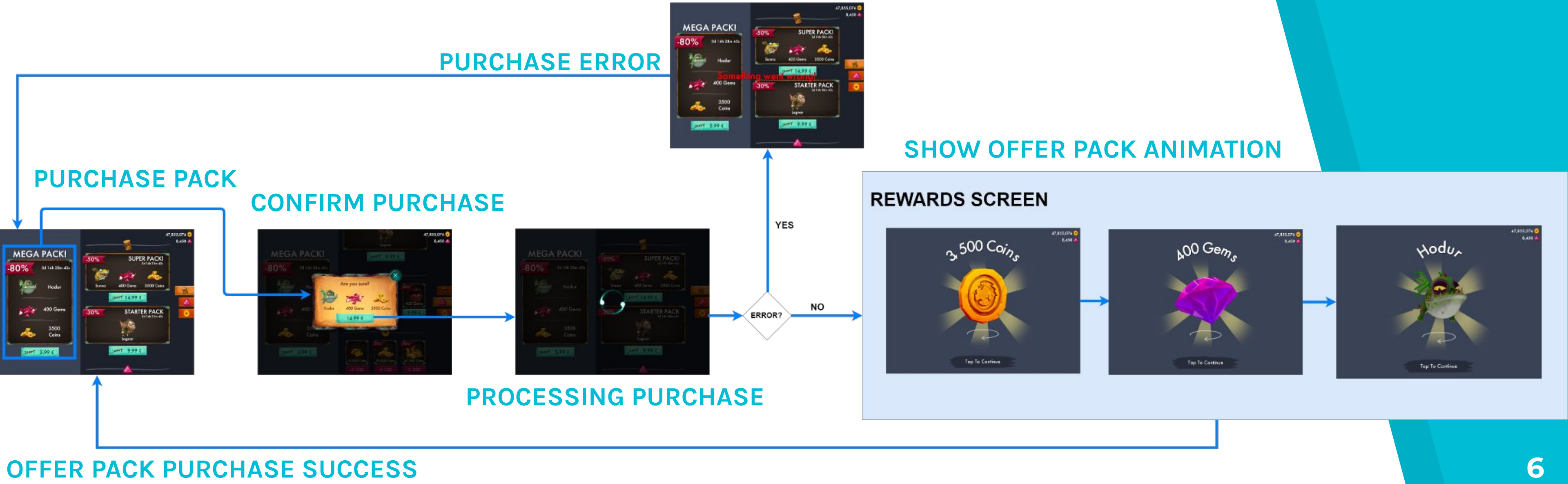
- For starters, prepare the assets adjusting anchors and pivots for compatibility between multiple mobile devices resolutions





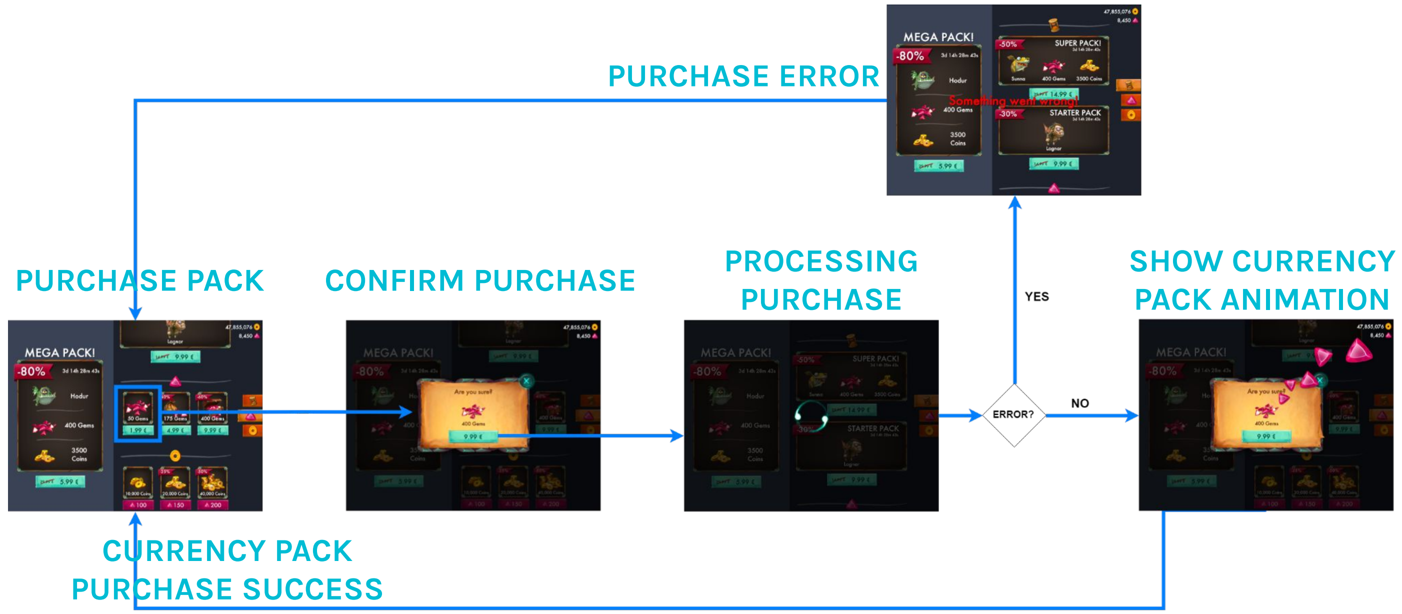
STORE BUILDING PROCESS

- Then, make an events system to go from one step to another of the flow and code the API



STORE BUILDING PROCESS

- Then, make an events system to go from one step to another of the flow and code the API



WHY USE EVENTS? BECAUSE...

- They make your code decoupled
  (“Yay, I hate dependencies!”)
- It lets you adapt to change and customize the flow (which we know will happen)
- You can send an event from anywhere in the code, it’s magic! (e.g. Send “Refresh Health Bar” event on player hit.)
- It’s easy to implement

MY ROUTINE WORK PROCESS

UNDERSTAND REQUIREMENTS

↓

MAKE IT WORK

↓

CLEAN AND OPTIMIZE IT

PROJECT FEATURES

- Store Buy and Claim
- Animations
- Visual FX
- Sound FX
- Language Change Option
- Events System
- Dependency Injection
- Unit Tests

DEPENDENCY INJECTION

Why I’m not a big fan of **singletons**?

- Because you are limited to one instance
  (e.g. Imagine that in January the requirements are to have 1 avatar replayer, but then in May they change to 3 avatar replayers per scene, you are screwed my friend...)
- Because they are public to everyone
  (This can lead to misuse, specially for junior devs.)
- Because they are immutable
  (e.g. I have a singleton that does X, but on some cases I want to do Y, so I have to constantly change it.)
- Refactoring a singleton is painful
  (If you need to manually change every “.Instance” in your code base for another class “.Instance”, it could take years to complete the refactor...)

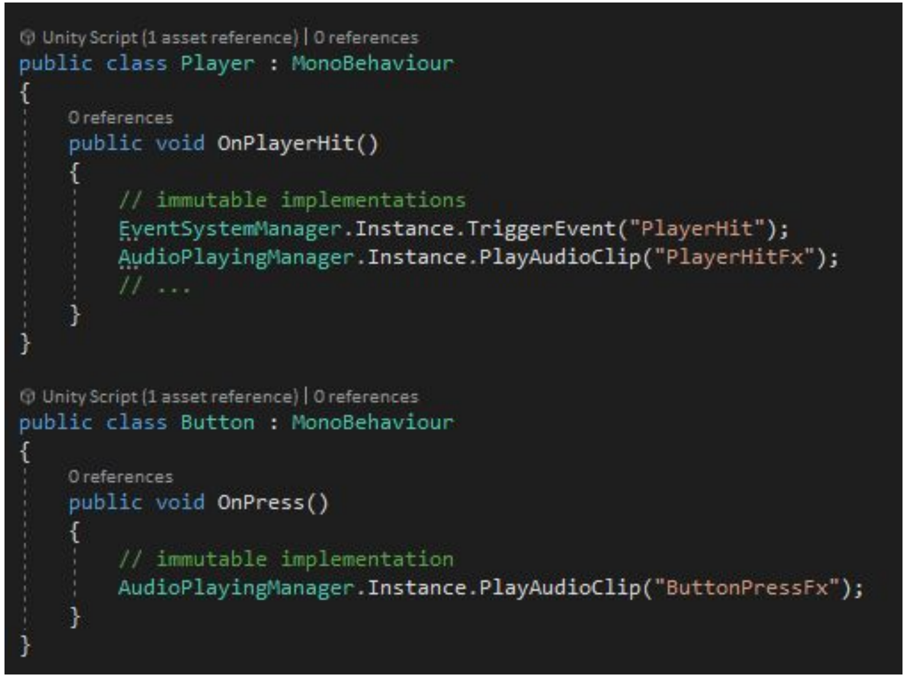
DEPENDENCY INJECTION

What do I prefer to do instead?

- Program to an interface and use a dependency injection framework
  (either code your own or use an existing one)

DEPENDENCY INJECTION

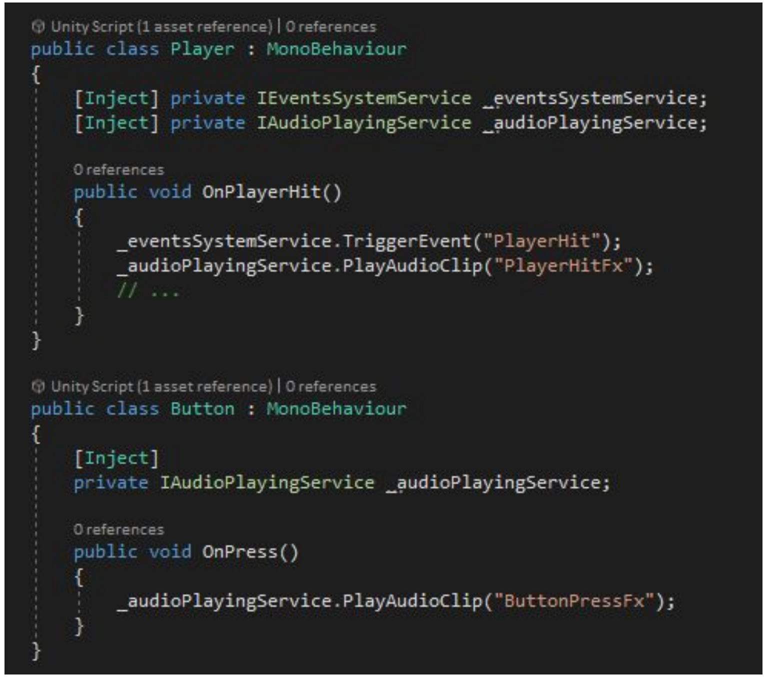
❌ NOT RECOMMENDED WAY



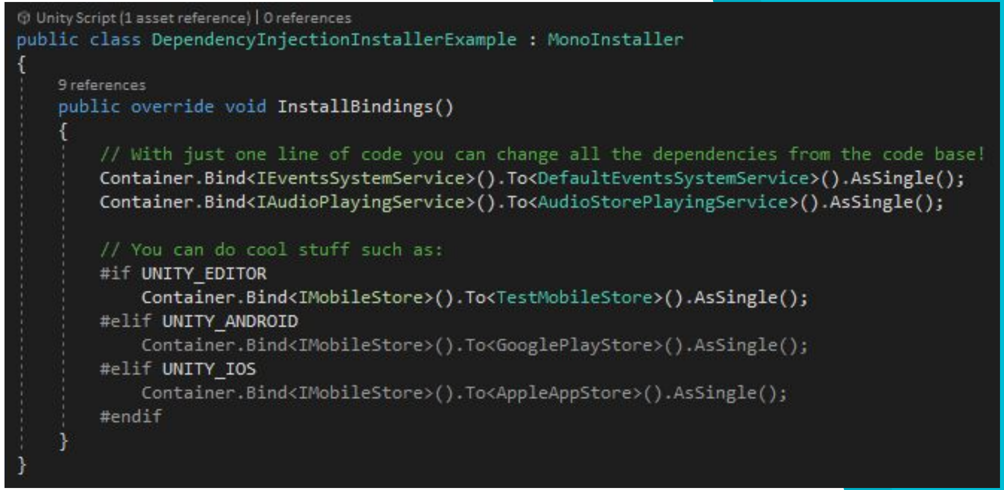
```
Unity Script (1 asset reference) | 0 references
public class Player : MonoBehaviour
{
    0 references
    public void OnPlayerHit()
    {
        // immutable implementations
        EventSystemManager.Instance.TriggerEvent("PlayerHit");
        AudioPlayingManager.Instance.PlayAudioClip("PlayerHitFx");
        // ...
    }
}

Unity Script (1 asset reference) | 0 references
public class Button : MonoBehaviour
{
    0 references
    public void OnPress()
    {
        // immutable implementation
        AudioPlayingManager.Instance.PlayAudioClip("ButtonPressFx");
    }
}
```

DEPENDENCY INJECTION

✔ RECOMMENDED WAY

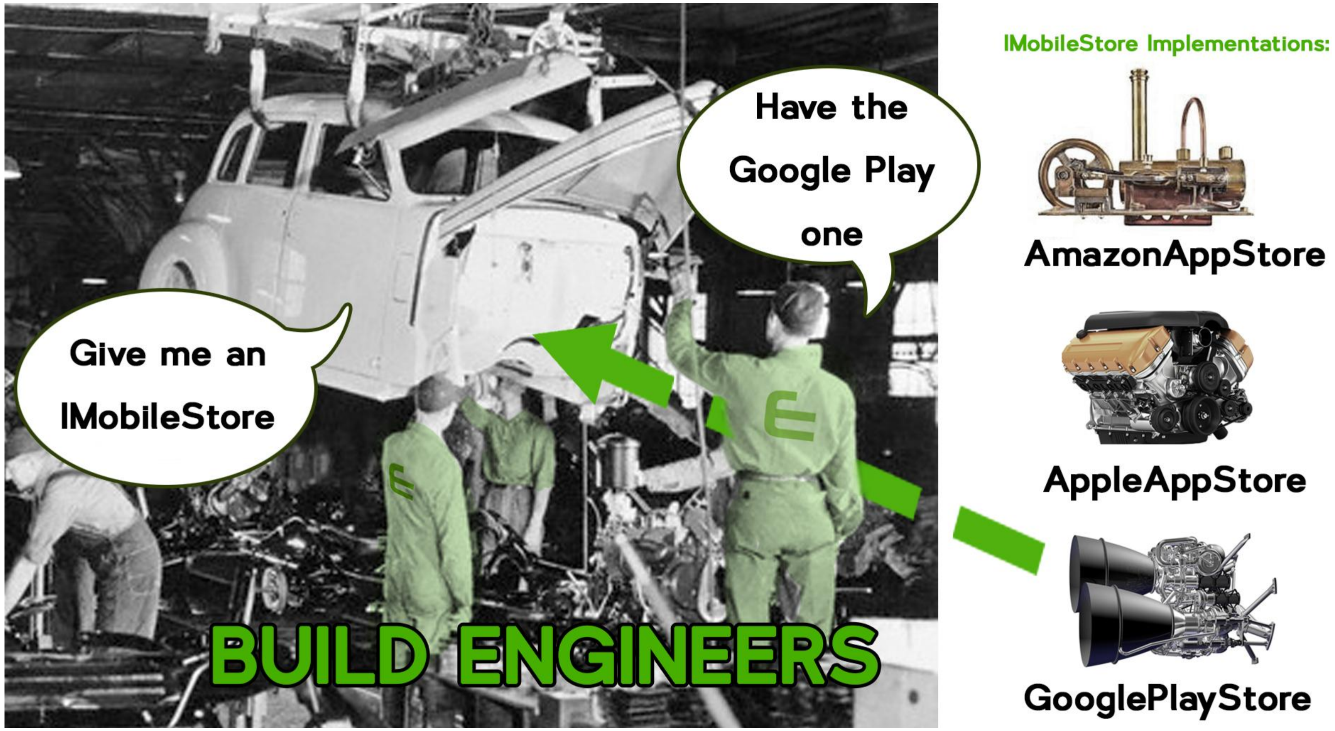


```
Unity Script (1 asset reference) | 0 references
public class Player : MonoBehaviour
{
    [Inject] private IEventsSystemService _eventsSystemService;
    [Inject] private IAudioPlayingService _audioPlayingService;

    0 references
    public void OnPlayerHit()
    {
        _eventsSystemService.TriggerEvent("PlayerHit");
        _audioPlayingService.PlayAudioClip("PlayerHitFx");
        // ...
    }
}

Unity Script (1 asset reference) | 0 references
public class Button : MonoBehaviour
{
    [Inject]
    private IAudioPlayingService _audioPlayingService;

    0 references
    public void OnPress()
    {
        _audioPlayingService.PlayAudioClip("ButtonPressFx");
    }
}
```



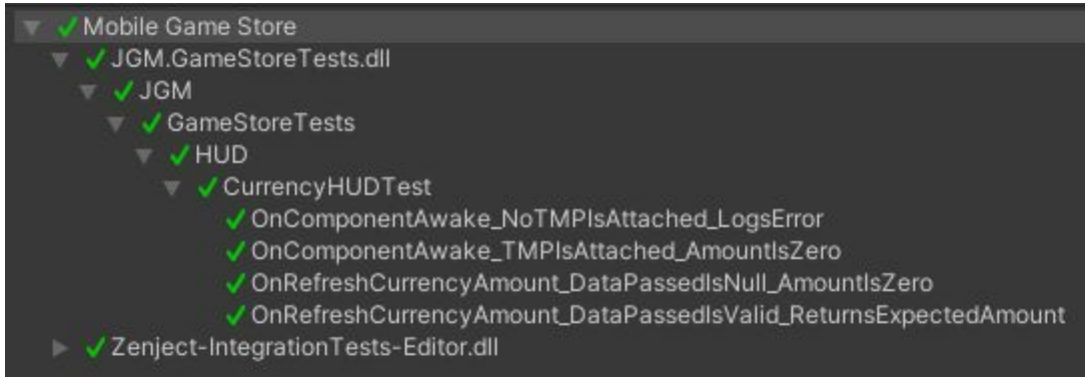
```
Unity Script (1 asset reference) | 0 references
public class DependencyInjectionInstallerExample : MonoInstaller
{
    9 references
    public override void InstallBindings()
    {
        // With just one line of code you can change all the dependencies from the code base!
        Container.Bind<IEventsSystemService>().To<DefaultEventsSystemService>().AsSingle();
        Container.Bind<IAudioPlayingService>().To<AudioStorePlayingService>().AsSingle();

        // You can do cool stuff such as:
        #if UNITY_EDITOR
            Container.Bind<IMobileStore>().To<TestMobileStore>().AsSingle();
        #elif UNITY_ANDROID
            Container.Bind<IMobileStore>().To<GooglePlayStore>().AsSingle();
        #elif UNITY_IOS
            Container.Bind<IMobileStore>().To<AppleAppStore>().AsSingle();
        #endif
    }
}
```

DEPENDENCY INJECTION



UNIT TESTS

- TDD approach is always a good habit



```
▼ ✓ Mobile Game Store
  ▼ ✓ JGM.GameStoreTests.dll
    ▼ ✓ JGM
      ▼ ✓ GameStoreTests
        ▼ ✓ HUD
          ▼ ✓ CurrencyHUDTest
              ✓ OnComponentAwake_NoTMPIsAttached_LogsError
              ✓ OnComponentAwake_TMPIsAttached_AmountIsZero
              ✓ OnRefreshCurrencyAmount_DataPassedIsNull_AmountIsZero
              ✓ OnRefreshCurrencyAmount_DataPassedIsValid_ReturnsExpectedAmount
  ▶ ✓ Zenject-IntegrationTests-Editor.dll
```