2-EconomyManager

In the Economy manager Scene we'll use delegates as parameters to pass functionalities to the EcocnomyManager.cs so it will make the decision to apply it or not.

I used Economy Manager as an example because it's very common functionality in games but you can apply it for anything.

So to be clear things out, the economy manager work is to hold of all currencies, so other scripts (we'll call them Askers) can check if there's enough resources to do their Action or not, the Economy Manager will calculate how much Gold is needed to pay for the missing currencies and ask the player if he would like to pay, if yes do the action that the Asker asked for, if not tell the Asker so it may do something in that case.
Oh and there's 5 currencies, 4 normal and 1 premium.

So why should you use delegates as parameters? Let's take a look on other ways first.

-Making all currencies Public static
That will give all scripts an easy access to the currencies value but that mean that every Asker will have to deal with the logic of calculating if there's eough resources to do their actions, that mean a lot of repeated code.

```
public class EconomyManager : MonoBehaviour {

    public static int Fuel = 300;
    public static int Iron = 400;
    public static int Powder = 350;
    public static int Wood = 500;
    public static int Gold = 30;
```

-Making functions for checking values
This one is slightly better but still suffers from the same problem, we have
5 currencies!

```
public class RocketsManager : MonoBehaviour {

    void StartSpawnProcess(){

        if (EconomyManager.GetWalletValue () >= RocketPrice) {

            EconomyManager.MinusPrice (RocketPrice);

            SpawnRocket ();

        }

    }
```

-Holding a reference to the Asker
The good news that you'll only write the logic for calculating the needed resources once, but what if you have 10 types of Askers?

```
[SerializeField] UnitSpawner spawner;

public static void CanIDoTheAction(){

    if ( DoesHaveEnoughResources() ){
        PayTheBill ();
        spawner.Do ();
    } else {
        spawner.Drop ();
    }
}
```

The easiest and most scalable way (I think) is to make the Asker store their Action in a delegate and send it with the bill, this way the EconomyManager won't need to know nor care who asked or what is the action.

```
public void SpawningProcess(){

    Action SpawnAction = new Action (Spawn);

    Bill bill = DataStorage.GetBillFor (CostType.SpownUnit);

    EconomyManager.PayFor ( bill , SpawnAction);

}

void Spawn(){

    GameObject obj = MasterPool.Get (UnitType);

    obj.transform.position = SpawnPoint.position;

}
```

The same thing apply between EconomyManager.cs and MissingResourcesPanel.cs if the player don't have the resources to pay for something just calculate the missing resources and needed gold and send them along with the actions, no need for refereces or getting back to the Asker.

This way regardless of how many currencies or Askers or what is the Action you can check and calcualate without having to go back and change the code that the Ackers are dependent on.

The main point is to keep your logic moving linearly, not going back and forward.