2.独立的 Update

在课程简介的时候，老师讲了一种比较麻烦的情况。就是在 MonoBehaviour 的 Update 中掺杂了大量互相无关的逻辑，导致代码非常不容易阅读。

这种情况比较常见，代码如下:

```
private void Update()
{
    if (A)
    {
        ...
    }

    if (B)
    {
        ...
        if (D)
        {
            ...
        }
        else {}
    }

    switch (C)
    {
        ...
    }

    if (Input.GetMouseButtonUp(0))
    {
        ...
    }
}
```

代码冗长，而且干扰视线。

UniRx 可以改善这个问题。

```
void Start()
{
    // A 逻辑,实现了 xx
    Observable.EveryUpdate()
            .Subscribe(_ =>
            {
                if (A)
                {
                    ...
                }
            }).AddTo(this);


    // B 逻辑,实现了 xx
    Observable.EveryUpdate()
            .Subscribe(_ =>
            {
                if (B)
                {
                    ...
                    if (D)
                    {
                        ...
                    }
                else {}
                }
            }).AddTo(this);

    // C 逻辑, 实现了 xx
    Observable.EveryUpdate()
            .Subscribe(_ =>
            {
                switch (C)
                {
                    ...
                }
```

```
            }).AddTo(this);

    // 鼠标点击检测逻辑
    Observable.EveryUpdate()
            .Subscribe(_ => {
            {
                if (Input.GetMouseButtonUp(0))
                {
                    ...
                }
            }).AddTo(this);
}
```

虽然在代码长度上没有任何改善，但是最起码，这些 Update 逻辑互相之间独立了。

状态跳转、延时等等这些经常在 Update 里实现的逻辑，都可以使用以上这种方式独立。

我们使用 UniRx 对代码进行了一点改善，老师在接触 UniRx 之后，就再也没有使用过 MonoBehaviour 提供的 Update 方法了。

不过这种 UniRx 的使用还比较初级，本节课所介绍的方式，随着对 UniRx 的深入，也会渐渐淘汰，因为后边有更好的实现方式。

今天的内容就这些。