Rollback

Non-hardcoded Dispose

Hard problems in production

- Leak of external resources: memory, connections
- Unhandled side effects

How were they solved before?

Manually without any conventions

Benefits of the `IDisposable`

- Standard way to
  - Free up resources
  - Undo side effects
- C# syntax support: `using` keyword

Has this made things better?

Yes, but not fundamentally

Downsides of the `IDisposable`

Developer still needs to:

- Keep **references** that are used only in Dispose
- Check if these references were **assigned**
- **Maintain order** of expressions inside it
- Manually call Dispose of all **internal objects**
- Consider **all possible states** of the object (i.e. exceptions during lifetime)
- Handle if Dispose is called **multiple times**

Can this be automated?

Rollback in a nutshell

- When allocating resources,
  Rollback allows to defer the actions for releasing them

- Rolls back the system to a predetermined states,
  right up to its very beginning
  - App restart: fast & clean, you don’t need to unload the scene
  - Reset singletons
  - Degrade gracefully if exception occured
  - Cascade disposals. Make your game modular

How to use it

```
Rollback popupRollback = rollback.Open()
```

Get or create a `Rollback`

```
ShowPopup(popupRollback);
```

Pass it in every dependent feature, down the execution flow

```
// wait for popup to close;
popupRollback.Dispose()
```

Dispose it when the feature ends

```
void ShowPopup(IRollback popupRollback)
{
  var instance = Instantiate(prefab);
  popupRollback.Defer(() => Destroy(instance));
}
```

Defer “undo” actions in place

Dispose approach

```
class Service : IDisposable {
  AssetBundle bundle;
  ICloudService cloudService;
  Option<PopupWindow> popupInstance;

  Service (AssetBundle bundle, ICloudService cloudService) {
     bundle.Load();
     this.bundle = bundle;

     cloudService.OnLoginResult += this.OnLoginResult;
     this.cloudService = cloudService;
  }

  void ShowPopup (PopupWindow prefab) {
     this.popupInstance = Instantiate(prefab);
  }

  void Dispose () {
     bundle.Unload();

     this.cloudService.OnLoginResult -= this.OnLoginResult;

     foreach(var popup in this.popupInstance)
        Destroy(popup);
  }
}
```

Rollback approach

```
class Service {
 Rollback rollback;

  Service (AssetBundle bundle, ICloudService cloudService, Rollback rollback) {
     bundle.Load();
     rollback.Defer(() => bundle.Unload());

     cloudService.OnLoginResult += this.OnLoginResult;
     rollback.Defer(() => cloudService.OnLoginResult -= this.OnLoginResult);

     this.rollback = rollback;
  }

  void ShowPopup (PopupWindow prefab)  {
     PopupWindow popup = Instantiate(prefab);
     this.rollback.Defer(() => Destroy(popup));
  }
}
```

Dispose approach

```
class Service : IDisposable {
  AssetBundle bundle;
  ICloudService cloudService;
  Option<PopupWindow> popupInstance;

  Service (AssetBundle bundle, ICloudService cloudService) {
     bundle.Load();
     this.bundle = bundle;

     cloudService.OnLoginResult += this.OnLoginResult;
     this.cloudService = cloudService;
  }

  void ShowPopup (PopupWindow prefab) {
     this.popupInstance = Instantiate(prefab);
  }

  void Dispose () {
     bundle.Unload();

     this.cloudService.OnLoginResult -= this.OnLoginResult;

     foreach(var popup in this.popupInstance)
        Destroy(popup);
  }
}
```

Rollback approach

```
class Service {
  Rollback rollback;

  Service (AssetBundle bundle, ICloudService cloudService, Rollback rollback) {
     bundle.Load();
     rollback.Defer(() => bundle.Unload());

     cloudService.OnLoginResult += this.OnLoginResult;
     rollback.Defer(() => cloudService.OnLoginResult -= this.OnLoginResult);

     this.rollback = rollback;
  }

  void ShowPopup (PopupWindow prefab)  {
     PopupWindow popup = Instantiate(prefab);
     this.rollback.Defer(() => Destroy(popup));
  }
}
```