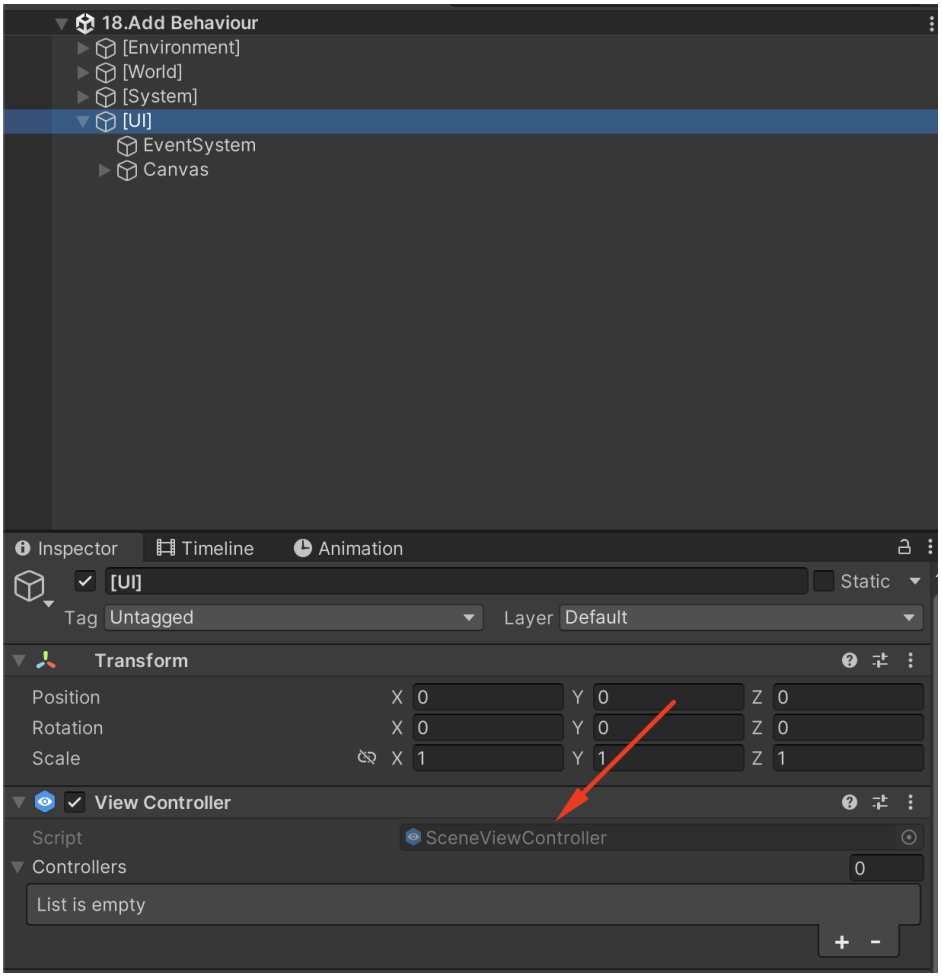
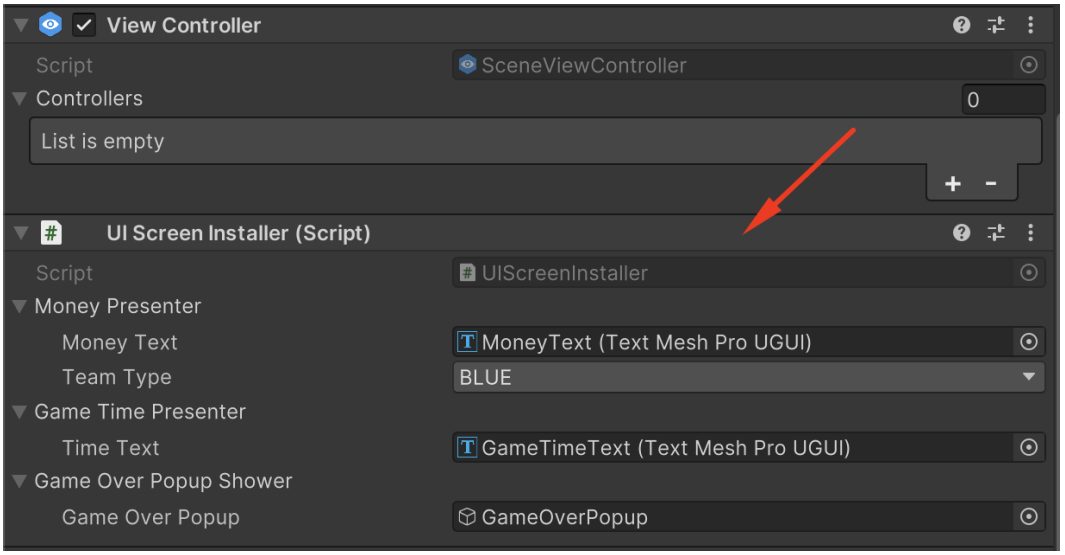
The first step is to add the ViewController component, which will control the various behaviors that will render the UI.



In order for these behaviors to get into the ViewController, you need to write an installer that will register them there. To do this, we will make the UIScreen Installer component.



The installer will look like this:

```
public sealed class UIScreenInstaller : SceneViewControllerInstaller
{
    [SerializeField]
    private MoneyPresenter moneyPresenter;

    [SerializeField]
    private GameTimePresenter gameTimePresenter;

    [SerializeField]
    private GameOverPopupShower gameOverPopupShower;

    protected override IEnumerable<IViewController> GetControllers()
    {
        yield return this.moneyPresenter;
        yield return this.gameTimePresenter;
        yield return this.gameOverPopupShower;
    }
}
```

Next, we implement the behaviors in order

1. Create a controller that draws the number of coins of the current player


```
[Serializable]
 public sealed class MoneyPresenter : IViewInit, IViewEnable, IViewDisable
 {
[SerializeField]
     private TMP_Text moneyText;

     [SerializeField]
     private TeamType teamType;

     private IReactiveValue<int> _money;

     public void Init()
     {
         Dictionary<TeamType,IContext> playerMap =
GameContext.Instance.GetPlayerMap();
         IContext playerContext = playerMap[this.teamType];
         _money = playerContext.GetMoney();
     }

     public void Enable()
     {
         _money.Observe(this.OnMoneyChanged);
     }

     public void Disable()
     {
         _money.Unsubscribe(this.OnMoneyChanged);
     }
```

```
    private void OnMoneyChanged(int money)
    {
        this.moneyText.text = $"Money: {money}";
    }
}
```

Here I would like to draw attention to the IViewInit, I View Enable & IViewDisable interfaces. Each of the interfaces triggers a Unity event:

- Init() — called at the start of the ViewController
- Enable() — called when the ViewController is enabled
- Disable() — called when the ViewController is disabled
- Dispose() — called when the ViewController is destroyed
- Update()/FixedUpdate()/LateUpdate() — called when ViewController updated

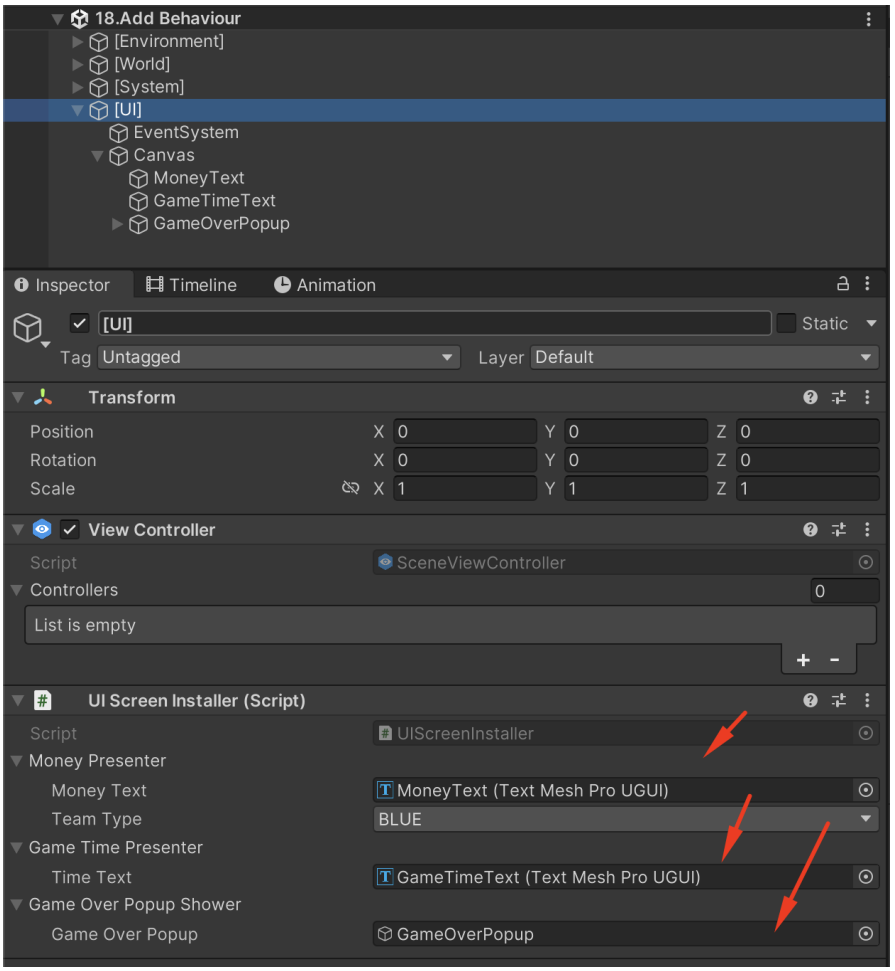
2. Create a controller that renders the game time


```
[Serializable]
public sealed class GameTimePresenter : IViewInit, IViewEnable, IViewDisable
{
    [SerializeField]
    private TMP_Text timeText;

    private Countdown _gameTimer;

    public void Init()
    {
        _gameTimer = GameContext.Instance.GetGameCountdown();
    }

    public void Enable()
    {
        _gameTimer.OnCurrentTimeChanged += this.OnTimeChanged;
        this.OnTimeChanged(_gameTimer.CurrentTime);
    }

    public void Disable()
    {
        _gameTimer.OnCurrentTimeChanged -= this.OnTimeChanged;
    }

    private void OnTimeChanged(float time)
    {
        this.timeText.text = $"Remaining time: {time:F0}";
    }
}
```

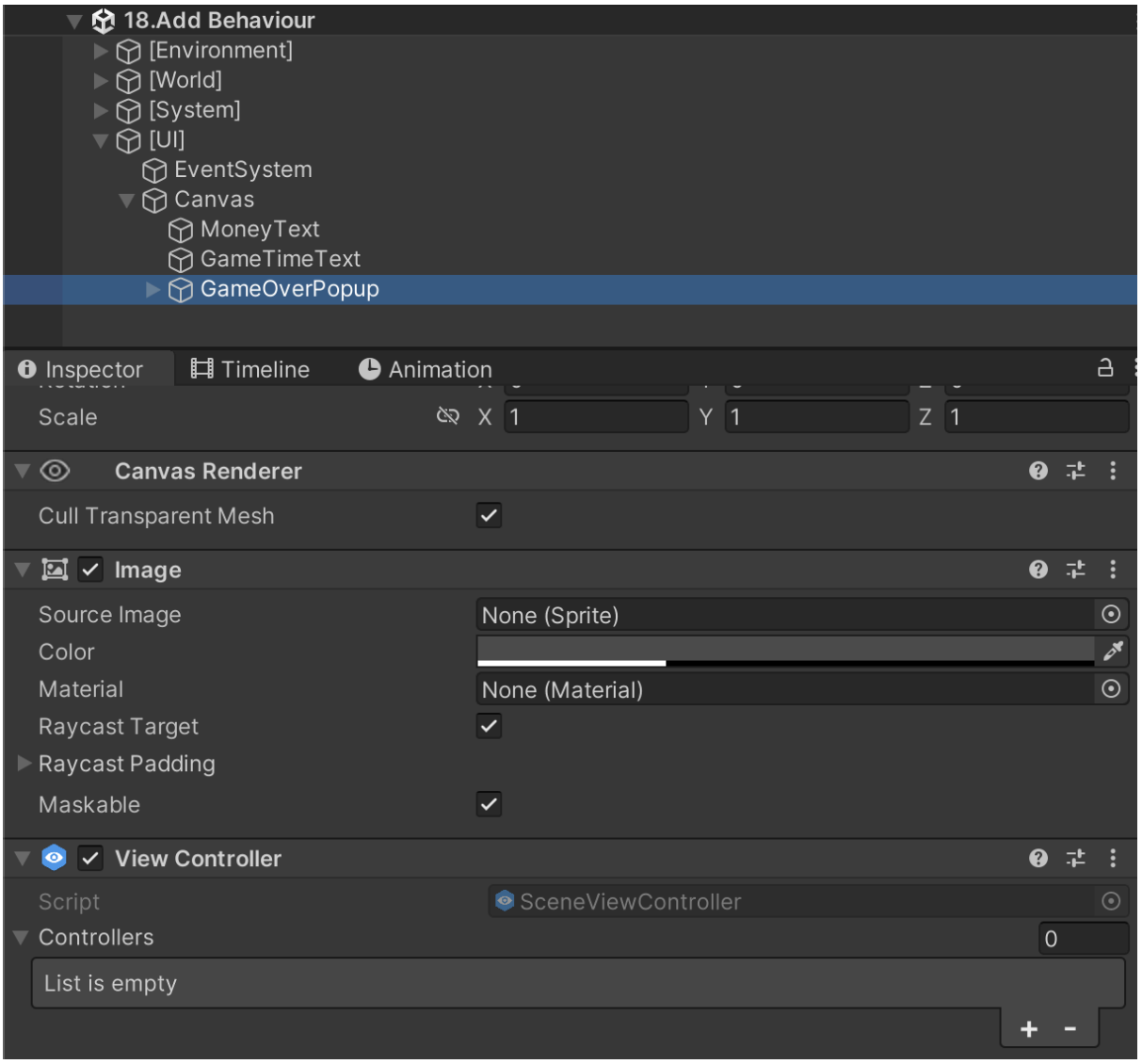
3. Create a controller that opens the end game window when the time is up

```
[Serializable]
public sealed class GameOverPopupShower : IViewInit, IViewEnable, IViewDisable
{
    [SerializeField]
    private GameObject gameOverPopup;

    private Countdown _gameCountdown;

    public void Init()
    {
        _gameCountdown = GameContext.Instance.GetGameCountdown();
        this.gameOverPopup.SetActive(false);
    }

    public void Enable()
    {
        _gameCountdown.OnEnded += this.OnGameCountdownFinished;
    }

    public void Disable()
    {
        _gameCountdown.OnEnded -= this.OnGameCountdownFinished;
    }

    private void OnGameCountdownFinished()
    {
        this.gameOverPopup.SetActive(true);
    }
}
```

4. Compile and fill Serialize Fields



5. Go to GameOverPopup and add ViewController component



5. Write GameOverView class

```
public sealed class GameOverView : MonoBehaviour
{
    public event UnityAction OnRestartClicked
    {
        add { this.restartButton.onClick.AddListener(value); }
        remove { this.restartButton.onClick.RemoveListener(value); }
    }

    [SerializeField]
    private TMP_Text messageText;

    [SerializeField]
    private Button restartButton;

    public void Show()
    {
        this.gameObject.SetActive(true);
    }

    public void SetMessageColor(Color color)
    {
        this.messageText.color = color;
    }

    public void SetMessage(string message)
    {
        this.messageText.text = message;
    }
}
```

6. Write a controller that renders the states of the game completion window

```
[Serializable]
public sealed class GameOverPresenter : IViewEnable, IViewDisable
{
    [SerializeField]
    private GameOverView view;

    public void Enable()
    {
        TeamType teamType = GameContext.Instance.GetWinnerPlayerTeam();
        this.view.SetMessage($"{teamType} PLAYER WINS");
        this.view.SetMessageColor(teamType.GetColor());

        this.view.OnRestartClicked += RestartGameUseCase.RestartGame;
    }

    public void Disable()
    {
        this.view.OnRestartClicked -= RestartGameUseCase.RestartGame;
    }
}
```

7. Write a installer for GameOver Presenter

```
public sealed class GameOverPresenterInstaller : SceneViewControllerInstaller
{
    [SerializeField]
    private GameOverPresenter gameOverPresenter;

    protected override IEnumerable<IViewController> GetControllers()
    {
        yield return this.gameOverPresenter;
    }
}
```

8. Add components to GameOverPopup