Exploring Static Fields and Events: A Journey into the Observer Pattern

INTRODUCTION

If you have played The Legend of Zelda: Ocarina of Time (1998), you will recognize this scene.

In this game, there is a scene where Link (the boy in green clothes) places three Spiritual Stones (green, red, and blue gems) on a pedestal inside the Temple of Time. Upon placing the three stones, a sequence of events occurs: a door in front of him opens, revealing an entrance to the Chamber of Time; the Triforce (shiny triangle on top of the door entrance) lights up; and epic music plays.

GOAL

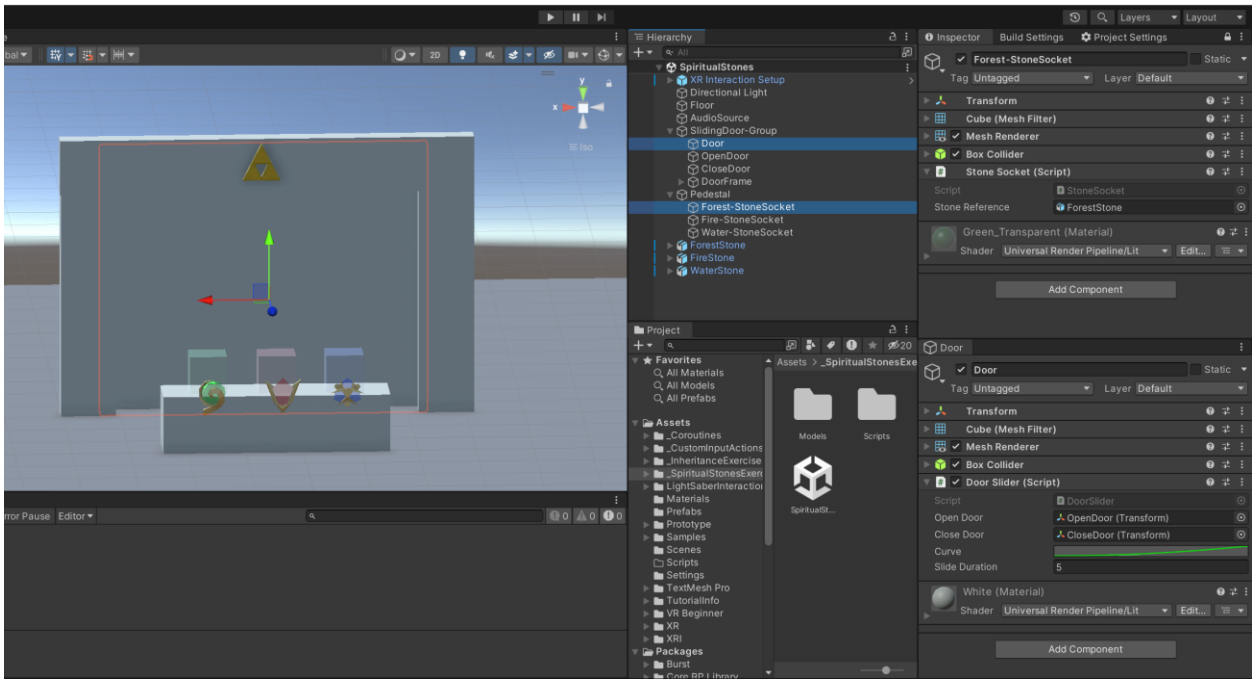
Our goal is to recreate, in a very simple manner, this memorable game mechanic using concepts we have learned. For instance, trigger colliders, static fields, scripting communication using variable references, and later, events.

Exploring Static Fields and Events: A Journey into the Observer Pattern

ACTIVITY

- **Step 1**. Download the *Spiritual Stones Scene* from Canvas: Module 4. Software Design Patterns

- **Step 2**. Examine how the scene is constructed.
  We have two scripts: *DoorSlider* and *StoneSocket*. The *DoorSlider* script is already implemented for you. Its primary responsibility is to open or close the door when the corresponding public functions are called.
  StoneSocket, on the other hand, requires your completion. This script is responsible for keeping track of the number of stones placed on the pedestal. *StoneSockets* serve as components of the colored cubes (triggers), each holding a reference to one of the three Stone game objects in the scene (FireStone, WaterStone, ForestStone).



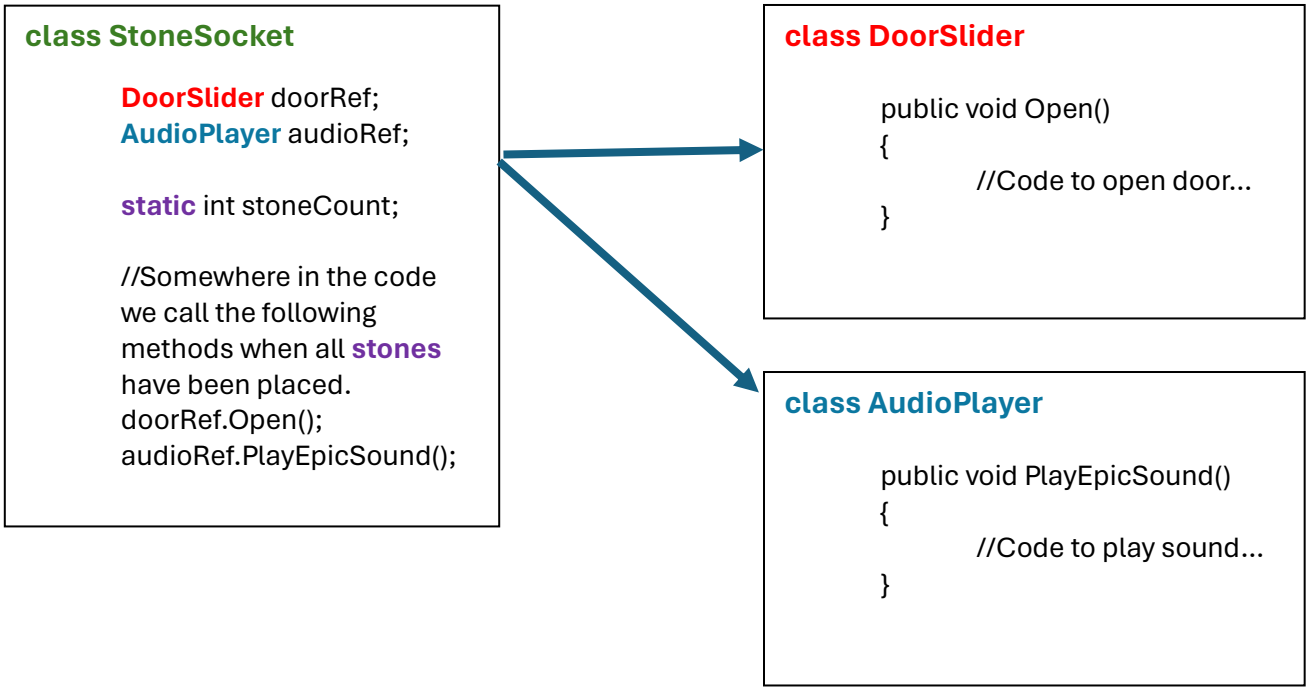
- **Step 3.** Complete the *StoneSocket* script in a way that opens the door when all three stones are placed. Consider using a **static field.** Experiment with the different architectures described below for linking the two scripts and think about their advantages and disadvantages.

Exploring Static Fields and Events: A Journey into the Observer Pattern

ARCHITECTURES

Architecture 1-Direct reference

- **Stone Socket** *references/depends on* **DoorSlider** and **AudioPlayer**

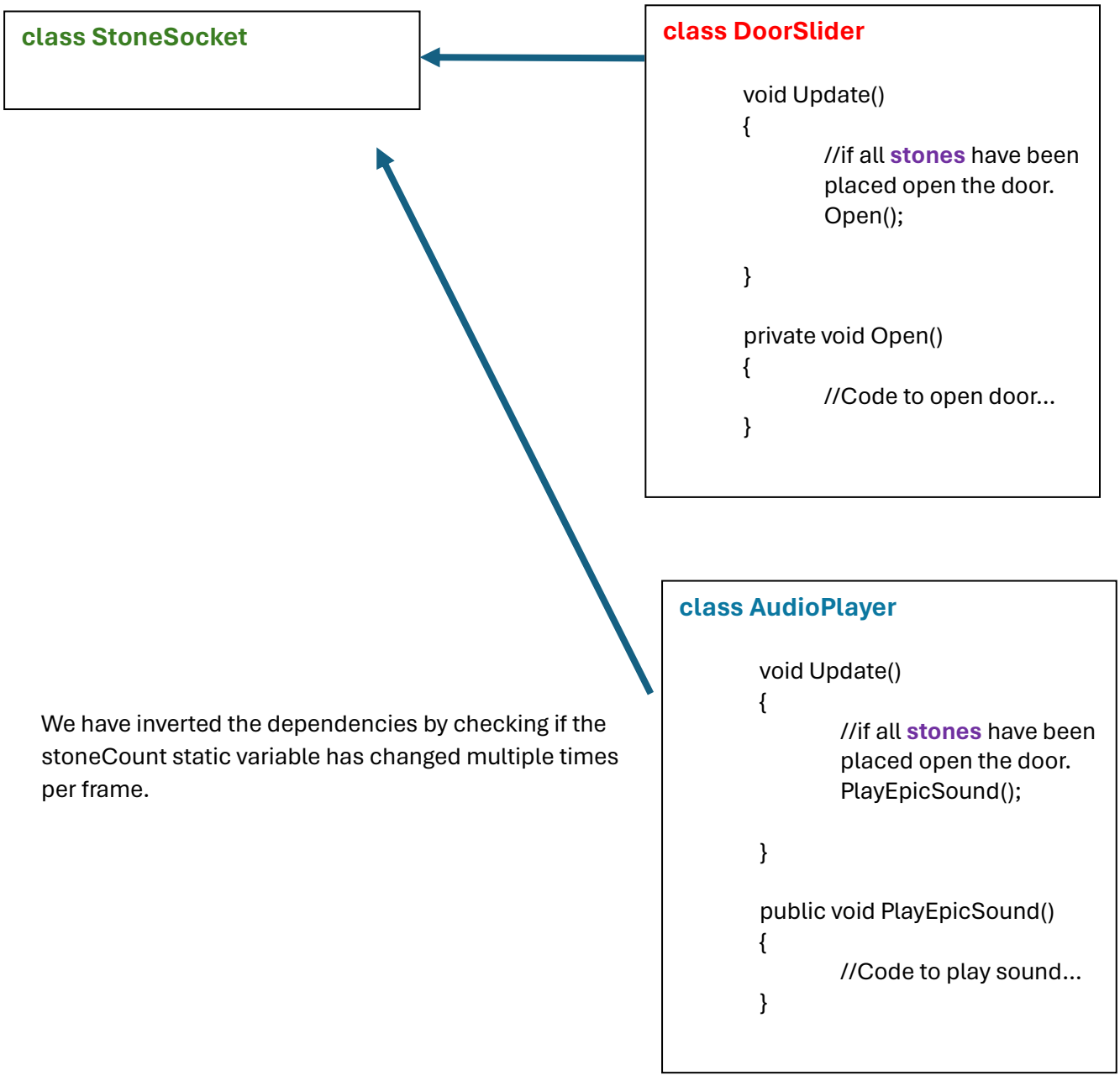


If we want to add a Triforce reaction, we must add a third dependency to the StoneSocket class, making it hard to maintain.

Exploring Static Fields and Events: A Journey into the Observer Pattern

Architecture 2- Inverted reference

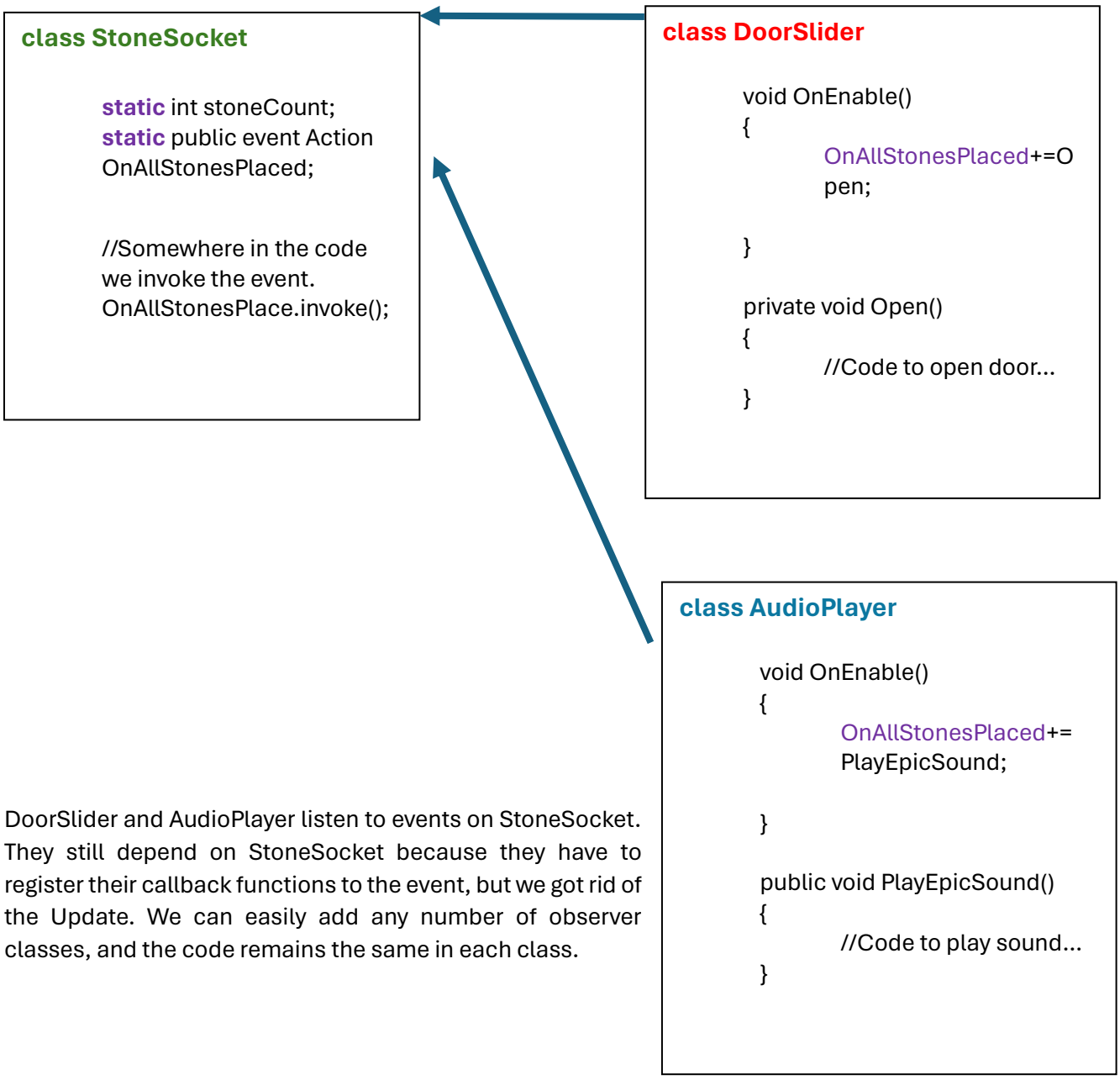
- **DoorSlider** *references/depends on* **StoneSocket**
- **AudioPlayer** *references/depends on* **StoneSocket**



We have inverted the dependencies by checking if the stoneCount static variable has changed multiple times per frame.

Exploring Static Fields and Events: A Journey into the Observer Pattern

Architecture 3-Events



DoorSlider and AudioPlayer listen to events on StoneSocket. They still depend on StoneSocket because they have to register their callback functions to the event, but we got rid of the Update. We can easily add any number of observer classes, and the code remains the same in each class.