CS 383 Team Lead 3
Quality Assurance and Patterns

Tosin Bangudu, Kyle Hash, Liam Mathews

Presentation Outline

Introduction:

- What is testing
- Why is it important
- Types of testing

Unit Testing

- Writing testable code
- Edit vs Play mode tests
- Setting up testing (assemblies)
- Boundary Tests
- Creating Boundary Tests
- Executing tests

What is Testing?

Testing is intended to show that a program does what it is meant to and to catch defects before it’s release.

- Testing is executing a program with artificial data
- This is a part of software validation and the verification process

Testing can reveal the presence of errors but not their absence

- Results of a test can be checked for information on errors and non-functional aspects of the program

Why is Testing Important?

Testing allows us to find situations in which the software operators incorrectly, and it enables us to validate that the software meets its requirements.

Validation:

- Shows that the system is operating as intended from design and implementation
- Operates correctly under a set of test conditions

Types of Testing

There are several different types of software testing:

- System Testing:
  - Use-case testing
- Release Testing
- User Testing:
  - Alpha
  - Beta
  - Acceptance
- Requirement Based Testing
- Performance Testing
  - Stress testing

System Testing

System testing is the testing of a fully integrated and complete piece of software

Use-case testing is a basis for system testing:

- Use cases are used to identify the interaction of the system
- These interaction between system components are then tested

Release Testing

Release testing is a form of system testing

There are some key difference between the two:

- A separate team not involved in development is responsible for release testing
- System testing should be focused on discovering bugs
- Release testing should check whether a system meets its requirements and is ready for validation testing

User Testing

User testing is a type of testing in which the customers/users provide input on system testing.

It is essential to conduct even after release and system testing because the user’s environment can’t be reproduced in a testing environment.

Alpha

- Users work with the dev team to test the software at the developer’s site

Beta

- A release of the software is made available to users to allow them to experiment and find problems with the system

Acceptance

- Customers test a system to decide whether or not it is ready to be accepted from the developers

Requirement Based Testing

Involves examining each requirement of a system and designing a specific test(or tests) for that requirement

Create Loosely Coupled Code

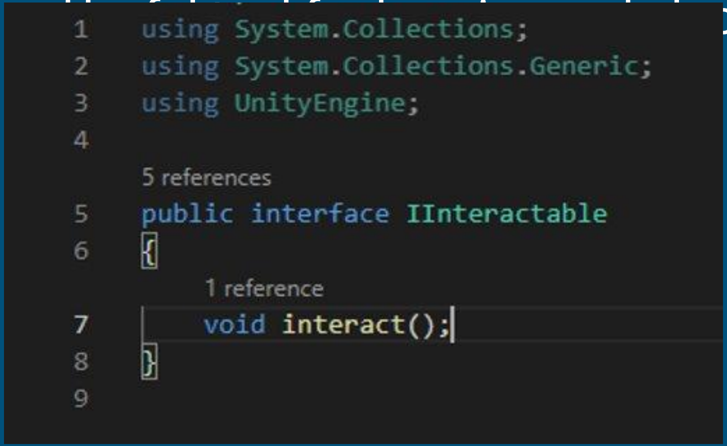
- Code is easier to understand
- Makes program more modular
- Program is easier to change, update, and expand
- More testable code
- Basically, the connections between different parts of your code are not dependent on one another

Function Contents

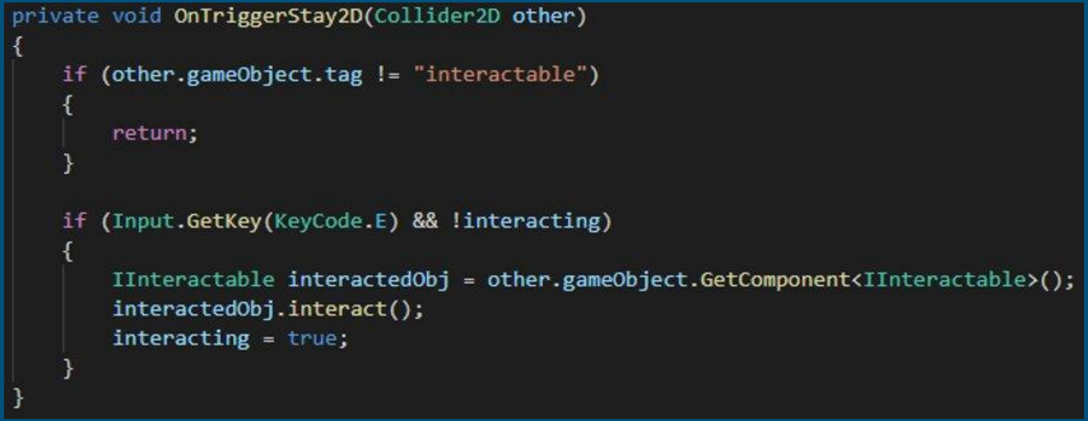
- One function should have only one function
- Functions should be non-deterministic (different results)
- Single Responsibility Principle
  - A function should either product or process information, not both.
- If a function needs extra data, have it passed as a parameter
- Inversion of Control
  - Separate decision making code and action code

Interfaces

osely coupled

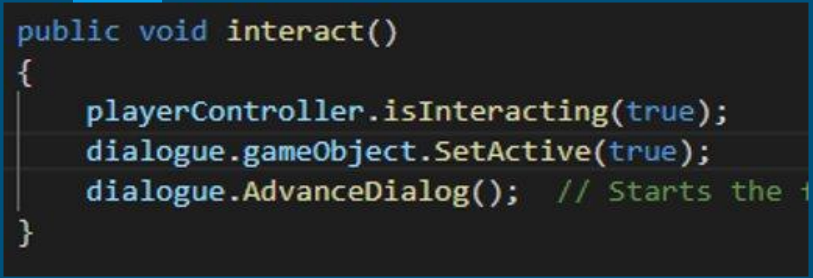


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

5 references
public interface IInteractable
{
    1 reference
    void interact();
}
```

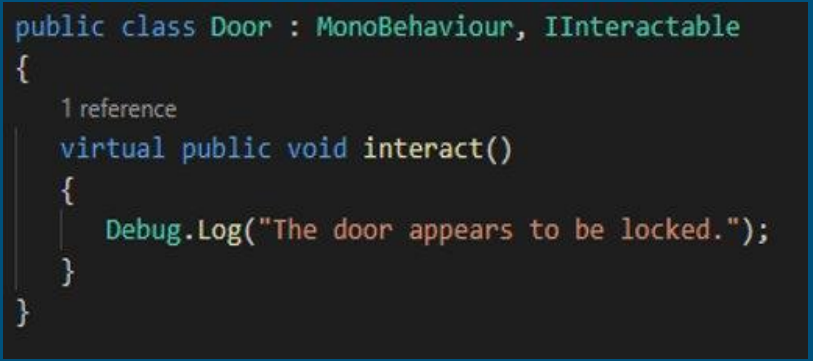


```
private void OnTriggerStay2D(Collider2D other)
{
    if (other.gameObject.tag != "interactable")
    {
        return;
    }

    if (Input.GetKey(KeyCode.E) && !interacting)
    {
        IInteractable interactedObj = other.gameObject.GetComponent<IInteractable>();
        interactedObj.interact();
        interacting = true;
    }
}
```

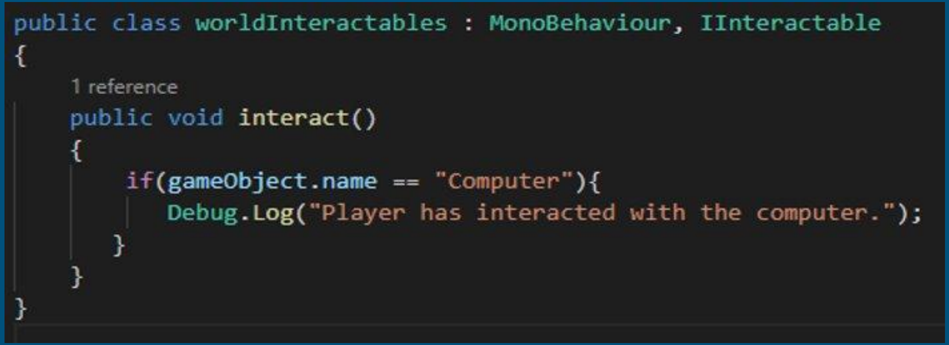
More Interfaces



```
public void interact()
{
    playerController.isInteracting(true);
    dialogue.gameObject.SetActive(true);
    dialogue.AdvanceDialog();  // Starts the t
}
```



```
public class Door : MonoBehaviour, IInteractable
{
    1 reference
    virtual public void interact()
    {
        Debug.Log("The door appears to be locked.");
    }
}
```



```
public class worldInteractables : MonoBehaviour, IInteractable
{
    1 reference
    public void interact()
    {
        if(gameObject.name == "Computer"){
            Debug.Log("Player has interacted with the computer.");
        }
    }
}
```

Edit Mode Vs. Play Mode Tests

Edit mode:

- Tests code that doesn’t require a running scene to test
- More useful for calculation
- Much faster to run

Play Mode:

- Tests code that needs to be executed in a running scene
- Tests are ran as coroutines
- More useful for testing things like movement

What are Boundary Tests?

- A boundary test is any sort of test that checks whether a value is within some specified range
- These tests can check for if a value is within, on, or outside of a boundary

Creating Unit Boundary Tests

3 Main areas of a Unit Test:

- Arrange
- Act
- Assert

- A boundary test is a specific type of unit test
- Tests to verify that edge cases are properly handled
- Verify that unexpected behavior doesn’t occur if given unexpected values

Test cases:

- Test just inside the boundary
- Test on the boundary
- Test just outside the boundary

Team Deliverables (End of Pres)

- Thursday (10/6)
  - One Test Case to Demo /2
  - Initial Test Plan to Demo /10
- Oral Exam
  - Full Test Plan /40
- Patterns
  - Must implement 2 by Oral Exam /60