AstroEscape Test Cases

Main Menu

ID	Step Description	Expected Result
1	Run main menu	Main menu scene should not be loaded as a playable level
2	Run main menu	All scenes beginning with 'Level' should be loaded as a playable level
3	Run main menu	'Play' button should be present and functional
4	Click 'Play' button	Level selection screen should appear
5	Click 'Options' button	Options menu should appear
6	Click 'Help' button	Help menu should appear

Level Selection

ID	Step Description	Expected Result
1	Run level selection screen	Display level buttons corresponding to available levels
2	Click level button	Corresponding level should load
3	Click 'Back' button	Return to the main menu
4	Click 'Stats' button	Player statistics should be displayed

Levels

Player Mechanics

ID	Step Description	Expected Result
1	Load level -> Press 'up' key	Player should move up
2	Load level -> Press 'down' key	Player should move down
3	Load level -> Press 'left' key	Player should move left
4	Load level -> Press 'right' key	Player should move right

Maze Navigation & Question Areas

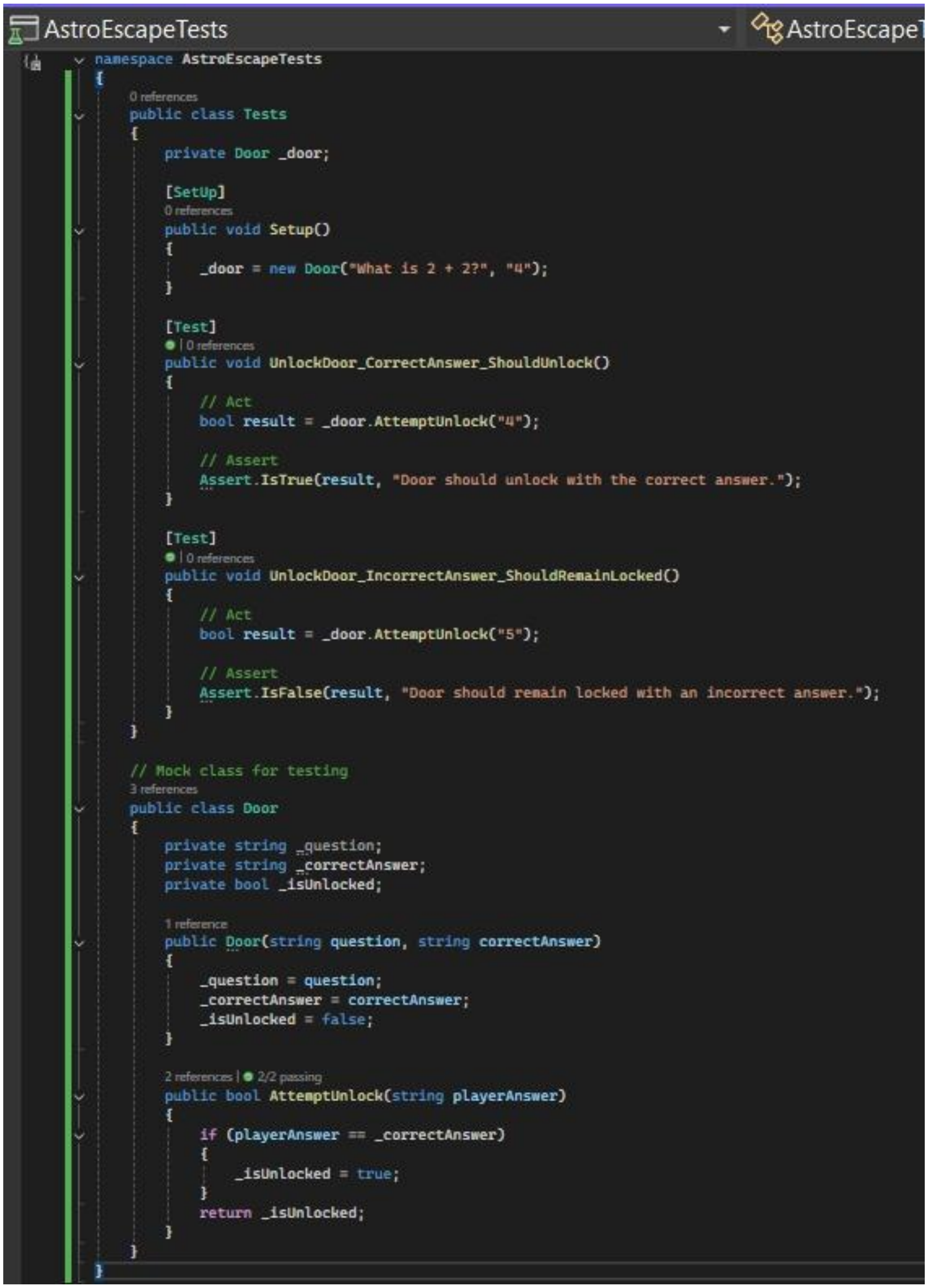
ID	Step Description	Expected Result
1	Load level	Question and text prompts should be set to invisible
2	Load level	Program throws an error if no answer is loaded
3	Move to question area	Question and text prompts should appear
4	Move out of question area	Question and text prompts should disappear
7	Attempt to move while answering	Player should be unable to move
8	Press 'Enter' key with invalid input	Answer is rejected
9	Input incorrect answer -> Press 'Enter'	Answer is rejected
10	Input correct answer -> Press 'Enter'	Answer is accepted

Collectibles Mechanics

ID	Step Description	Expected Result
1	Player moves over a collectible	Collectible should disappear and be added to player's score
2	Collect all collectibles in a level	Achievement should be unlocked if applicable

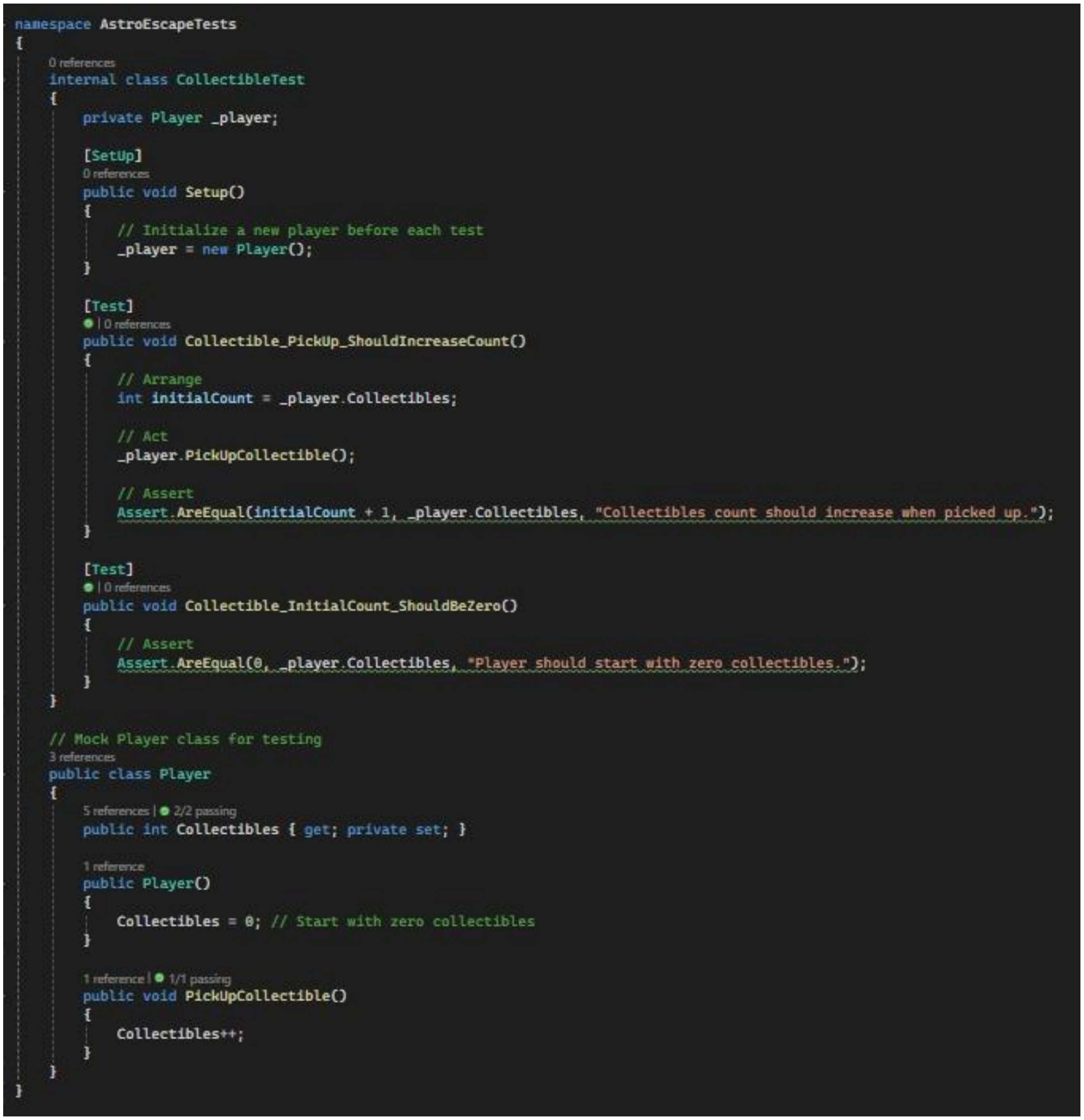
4	Pick up collectible -> Check UI	Score should increase accordingly

Simple Tests:



AstroEscapeTests | AstroEscape

```
namespace AstroEscapeTests
{
    0 references
    public class Tests
    {
        private Door _door;

        [SetUp]
        0 references
        public void Setup()
        {
            _door = new Door("What is 2 + 2?", "4");
        }

        [Test]
        ● | 0 references
        public void UnlockDoor_CorrectAnswer_ShouldUnlock()
        {
            // Act
            bool result = _door.AttemptUnlock("4");

            // Assert
            Assert.IsTrue(result, "Door should unlock with the correct answer.");
        }

        [Test]
        ● | 0 references
        public void UnlockDoor_IncorrectAnswer_ShouldRemainLocked()
        {
            // Act
            bool result = _door.AttemptUnlock("5");

            // Assert
            Assert.IsFalse(result, "Door should remain locked with an incorrect answer.");
        }
    }

    // Mock class for testing
    3 references
    public class Door
    {
        private string _question;
        private string _correctAnswer;
        private bool _isUnlocked;

        1 reference
        public Door(string question, string correctAnswer)
        {
            _question = question;
            _correctAnswer = correctAnswer;
            _isUnlocked = false;
        }

        2 references | ● 2/2 passing
        public bool AttemptUnlock(string playerAnswer)
        {
            if (playerAnswer == _correctAnswer)
            {
                _isUnlocked = true;
            }
            return _isUnlocked;
        }
    }
}
```



```
namespace AstroEscapeTests
{
    0 references
    internal class CollectibleTest
    {
        private Player _player;

        [SetUp]
        0 references
        public void Setup()
        {
            // Initialize a new player before each test
            _player = new Player();
        }

        [Test]
        ● | 0 references
        public void Collectible_PickUp_ShouldIncreaseCount()
        {
            // Arrange
            int initialCount = _player.Collectibles;

            // Act
            _player.PickUpCollectible();

            // Assert
            Assert.AreEqual(initialCount + 1, _player.Collectibles, "Collectibles count should increase when picked up.");
        }

        [Test]
        ● | 0 references
        public void Collectible_InitialCount_ShouldBeZero()
        {
            // Assert
            Assert.AreEqual(0, _player.Collectibles, "Player should start with zero collectibles.");
        }
    }

    // Mock Player class for testing
    3 references
    public class Player
    {
        5 references | ● 2/2 passing
        public int Collectibles { get; private set; }

        1 reference
        public Player()
        {
            Collectibles = 0; // Start with zero collectibles
        }

        1 reference | ● 1/1 passing
        public void PickUpCollectible()
        {
            Collectibles++;
        }
    }
}
```