To follow along, you can grab the content from GitHub.

https://github.com/mstarks/unite-talk

who we are

we are sphero – we put fun first
we are passionate about the present, and focused on the future
we are humble, hustle hard, and check our egos at the door
we are committed to customers, experiences, and results
we are proud of our work, but never satisfied

Michael T. Starks

SENIOR SOFTWARE ENGINEER AT *SPHERO*

I write *Unity* apps to control toys.

I have a Master's in physics from Indiana University.

I've worked as a teacher and a developer.

I've helped create the MMOG *LEGO Universe*.

I produce and direct movies.

https://www.linkedin.com/in/michaeltstarks

https://www.facebook.com/michael.t.starks

Test Guided Development:

How to realize the benefits of TDD with Unity

–MICHAEL T. STARKS

Test Guided Development

Introduction

Hands-On Testing and Development

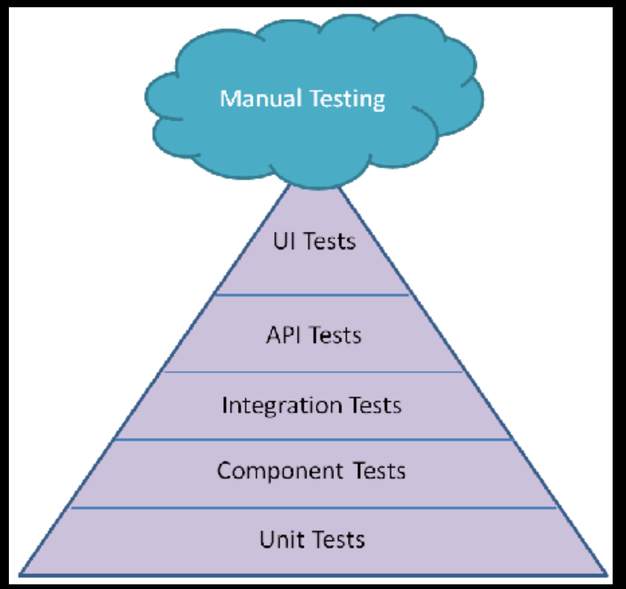
- Simple Examples

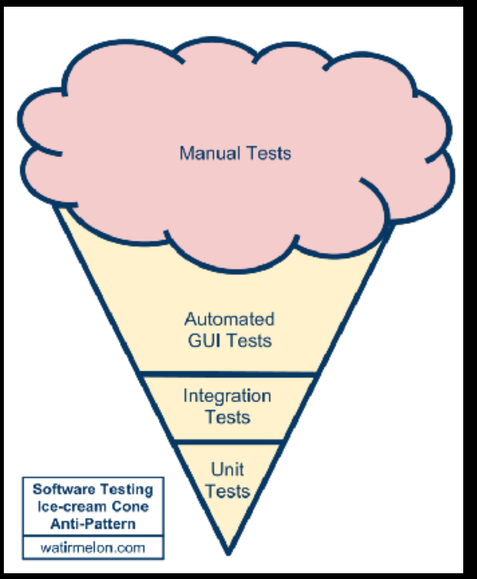
- Building the Game Objects

- Mocks and Stubs

- More Examples

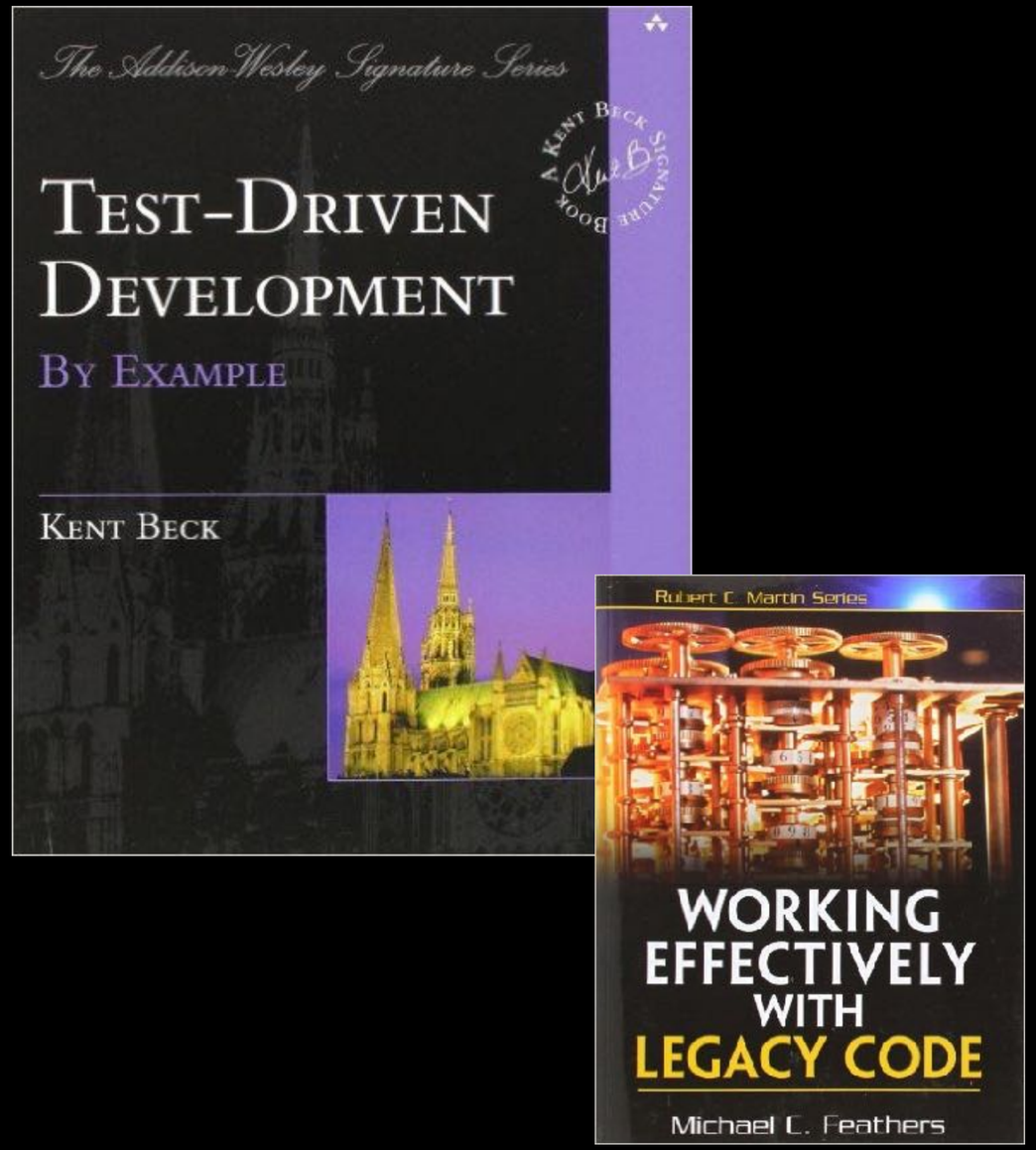
Hints





http://james-willett.com/2016/09/the-evolution-of-the-testing-pyramid/

Test Driven Development



Use tests to drive every line of code you write.

TDD is championed by the *Agile* community.

https://www.agilealliance.org/glossary/tdd/

http://david.heinemeierhansson.com/2014/tdd-is-dead-long-live-testing.html

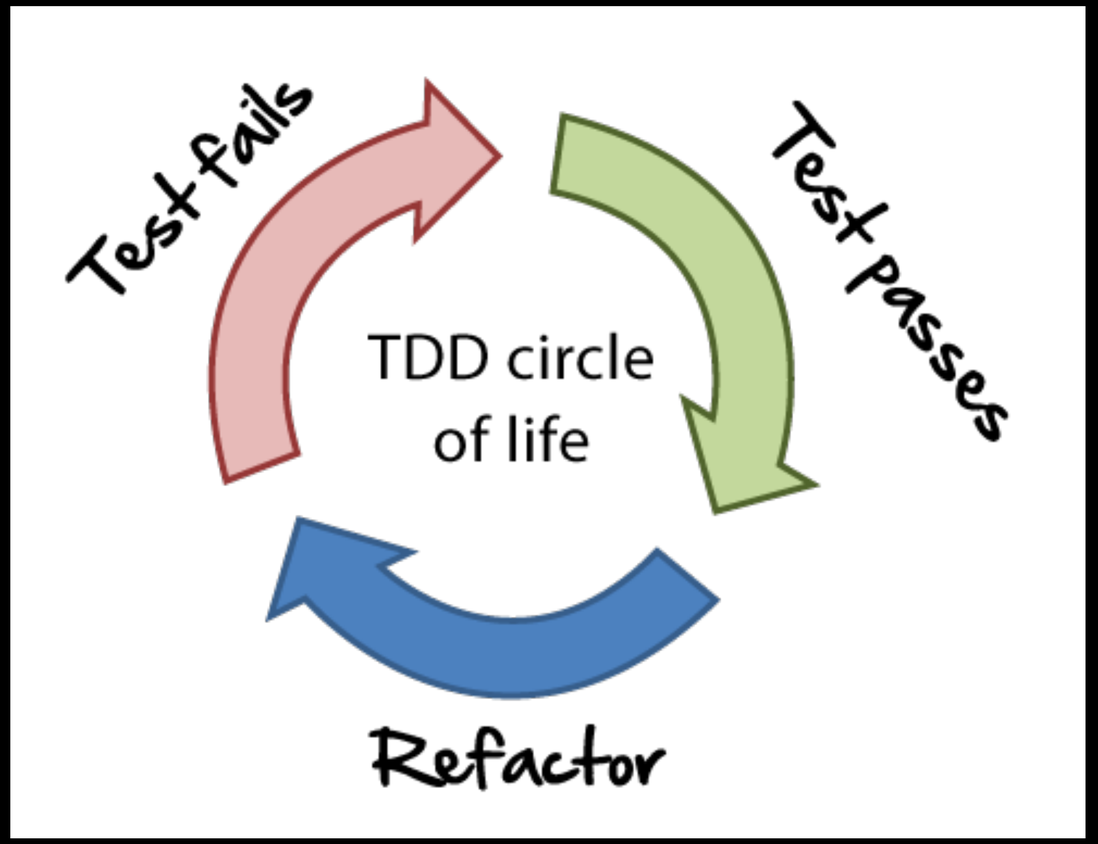
Michael Feathers defines “Legacy Code” as code without tests.

Test Driven Development

APPROACH

Write a failing test.
(Arrange, act, assert.)
Write the code needed to get the test to pass.
(Avoid premature optimizations.)
Refactor your code to eliminate duplication.

Repeat.



Test Driven Development

WHY USE TDD?

The app is written for testability.

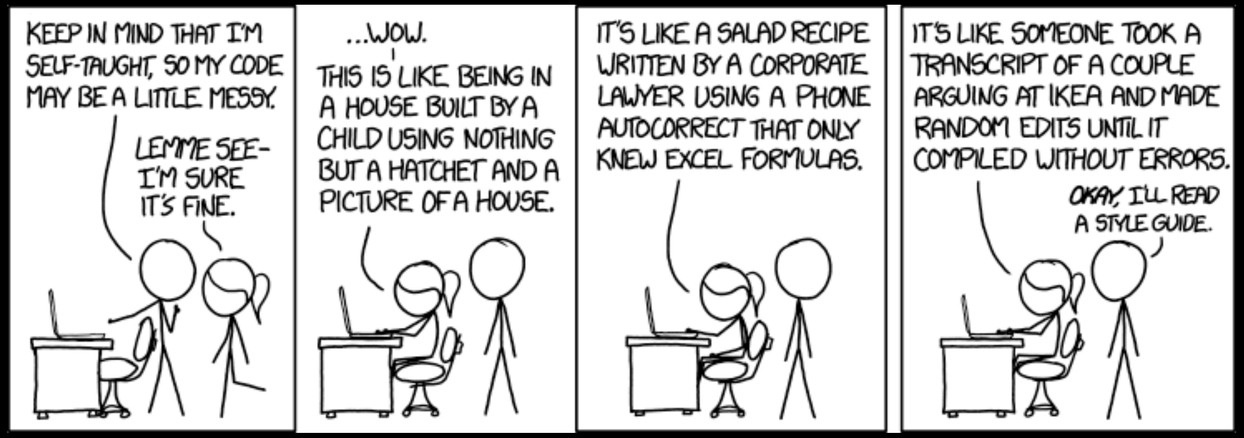
Every important feature gets a test.

Important conversations about the product happen early.

The code quality increases.

Test Driven Development

TWO STORIES



http://xkcd.com/1513/

In about two years, *Scrum* and TDD enabled us to improve a critical code base.

http://homeguides.sfgate.com/substandard-housing-55618.html

For another project, the tests were kept in a separate repository from the production code and not run very often.

Test Guided Development

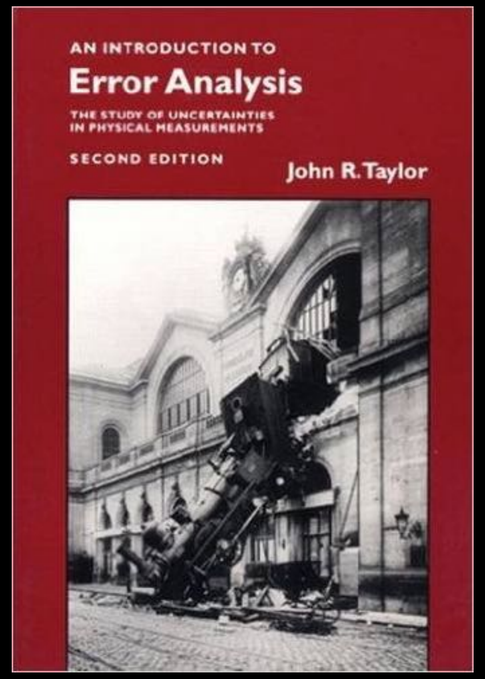
TDD IS TOUGH WITH *UNITY*.

Unity provides a ton of functionality.
It's not our job to test the physics, rendering, or UI systems.

Unity encourages UI-first development.
We need to embrace this.

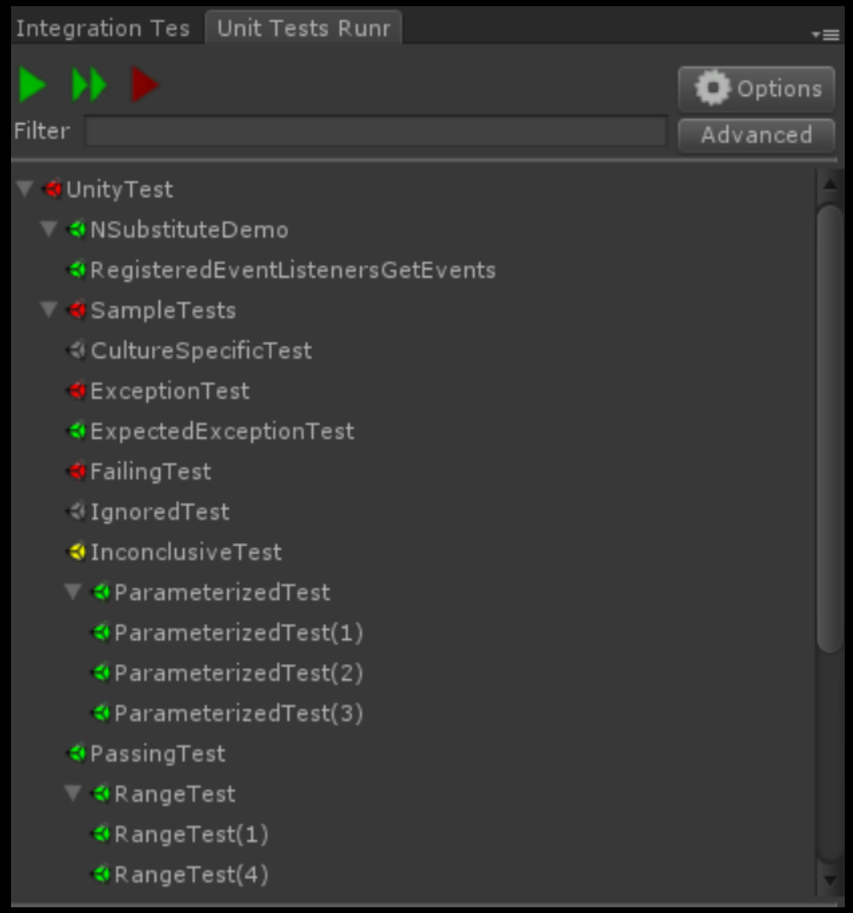
MonoBehaviour objects are not easy to test.
Some objects require testing on a native device.

INSTEAD, LET TESTS GUIDE YOUR DEVELOPMENT WORK.



Unity Test Tools
https://unity3d.com/learn/tutorials/topics/production/unity-test-tools

Install the *Unity* Test Tools



Search for the Unity Test Tools in the app store.

Install them.

Write your tests in the Editor folder.

Run your tests with the Editor Tests Runner.

https://blogs.unity3d.com/2013/12/18/unity-test-tools-released/

Get to know *NUnit*

Create a class for a collection of tests with the [TestFixture] attribute.

Write several test methods, using the [Test] attribute for each.

NUnit can be used to write both unit tests and system tests

We are going to see more functionality as we practice.

```
using UnityEngine;
using NUnit.Framework;


[TestFixture]
public class MazeTests
{
   [Test]
   public void DefaultValuesMakeSenseTest ()
   {
      // Arrange
      const int mazeLength = 10;
      const int mazeWidth = 6;
      const int numberOfAvailableRewards = 3;

      // Act
      Maze m1 = new Maze ();

      // Assert
      Assert.That (m1.Size.x, Is.EqualTo (mazeLength));
      Assert.That (m1.Size.y, Is.EqualTo (mazeWidth));
      Assert.That (m1.NumberOfAvailableRewards,
                        Is.EqualTo (numberOfAvailableRewards));
   }

}
```

Hands-On Testing and Development

CREATE A SIMPLE GAME USING TEST GUIDED DEVELOPMENT

Let’s create a maze that a player can navigate.

Include a few rewards that can be collected and hazards that should be avoided.

Show the player’s progress.

Tests can drive the specifics as we dig deeper.

To follow along, you can grab the content from GitHub.
https://github.com/mstarks/unite-talk

Hands-On Testing and Development

Initial Stories

As a player, I begin the game with maximum Health and zero Currency.

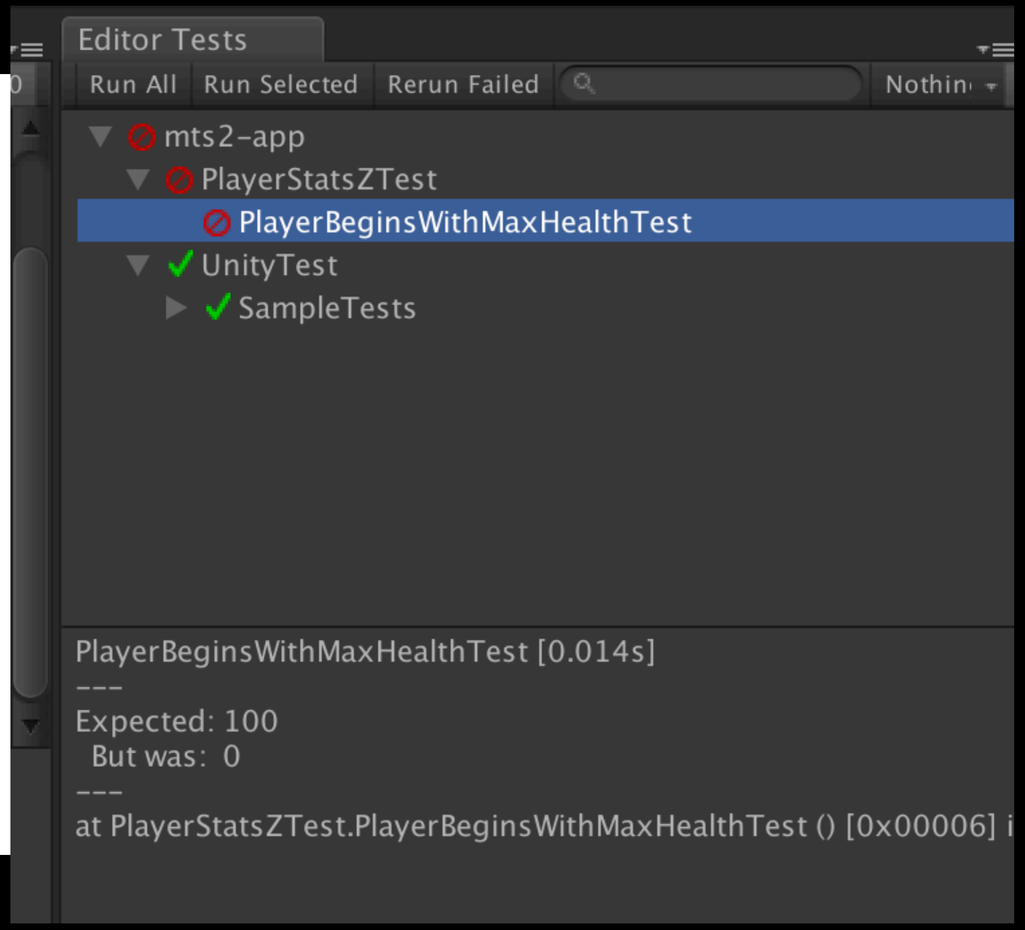
As a player, I can collect loot to increase my Currency.

As a player, my Health goes down when getting damage.

```
// TODO List
//
// Player Stats are initialized with maximum Health and zero Currency.
// Currency in Player Stats can be increased or decreased.
// Health in Player Stats can be increased or decreased.
```

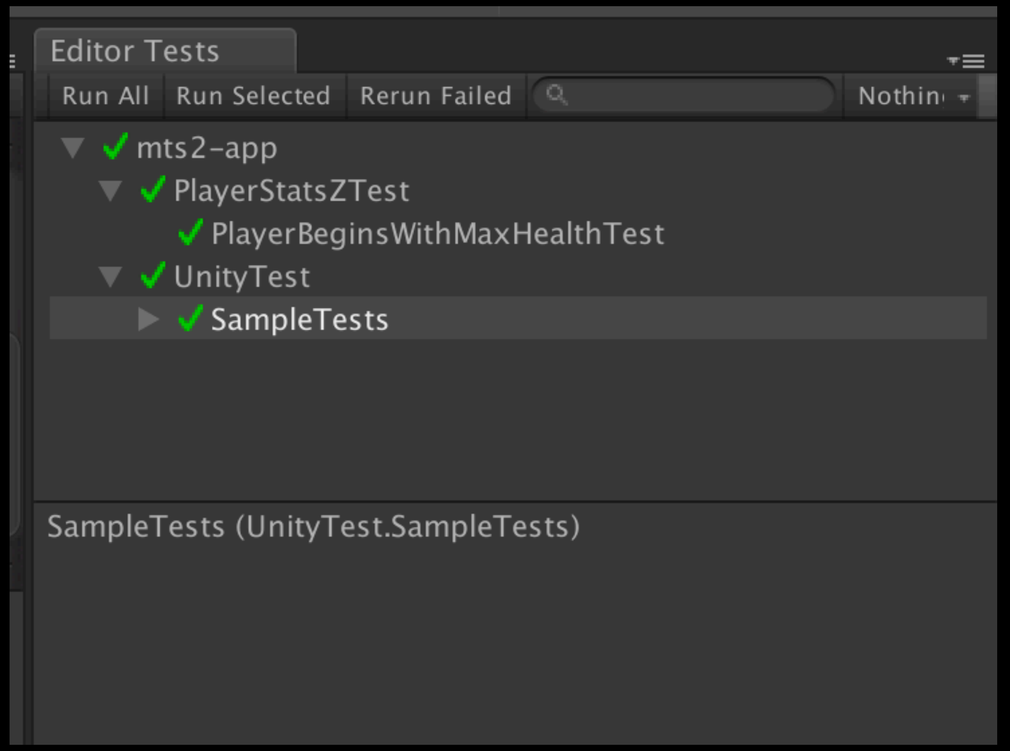
Hands-On Testing and Development

```using UnityEngine;
using System.Collections;
using NUnit.Framework;

[TestFixture]
public class PlayerStatsZTest
{
    [Test]
    public void PlayerBeginsWithMaxHealthTest()
    {
        // Arrange and Act
        const int maxHealth = 100;
        PlayerStatsZ playerStats = new PlayerStatsZ();

        // Assert
        Assert.That(playerStats.CurrentHealth, Is.EqualTo(maxHealth));
    }
}
```



Hands-On Testing and Development

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class PlayerStatsZBehaviour : MonoBehaviour
5  {
6      public PlayerStatsZ PlayerStats;
7
8      void Start()
9      {
10         //
11     }
12
13 }
14
15 public class PlayerStatsZ
16 {
17     public int CurrentHealth;
18
19     public PlayerStatsZ()
20     {
21         CurrentHealth = 100;
22     }
23
24 }
```

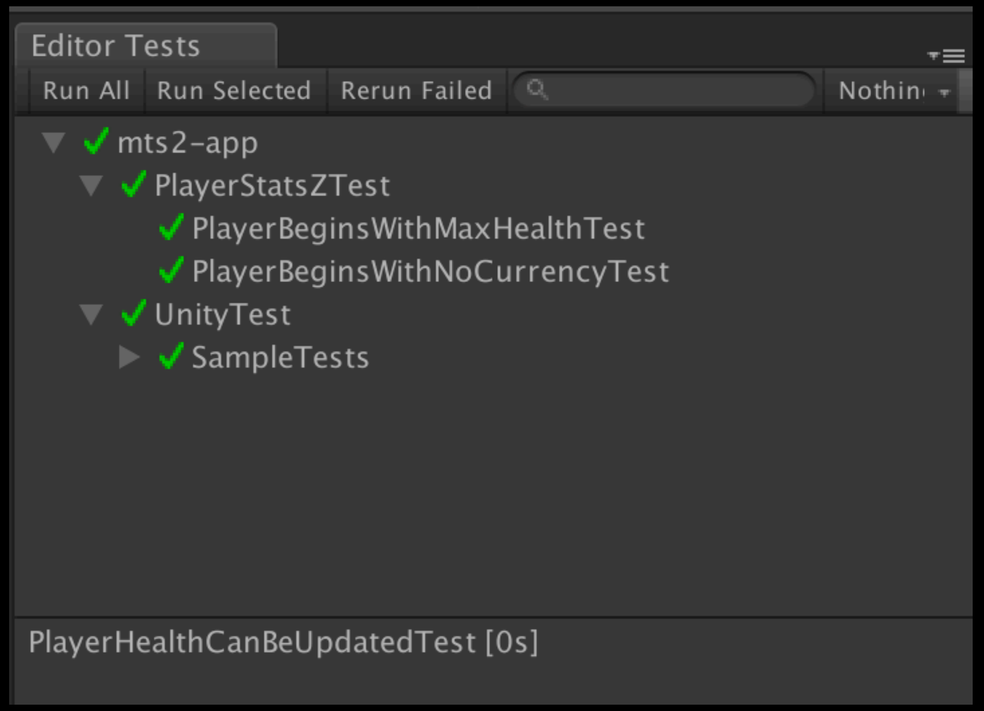


Hands-On Testing and Development

```
 5 [TestFixture]
 6 public class PlayerStatsZTest
 7 {
 8     [Test]
 9     public void PlayerBeginsWithMaxHealthTest()
10     {
11         // Arrange and Act
12         const int maxHealth = 100;
13         PlayerStatsZ playerStats = new PlayerStatsZ();
14
15         // Assert
16         Assert.That(playerStats.CurrentHealth, Is.EqualTo(maxHealth));
17     }
18
19     [Test]
20     public void PlayerBeginsWithNoCurrencyTest()
21     {
22         // Arrange and Act
23         const int initialCurrency = 0;
24         PlayerStatsZ playerStats = new PlayerStatsZ();
25
26         // Assert
27         Assert.That(playerStats.CurrentCurrency, Is.EqualTo(initialCurrency));  Type `PlayerStatsZ' does not contain a definition for `CurrentCurrency' and no extension method `...
28     }
29 }
30
```

Hands-On Testing and Development

```
public class PlayerStatsZ
{
    public int CurrentHealth;
    public int CurrentCurrency;

    public PlayerStatsZ()
    {
        CurrentHealth = 100;
        CurrentCurrency = 0;
    }

}
```



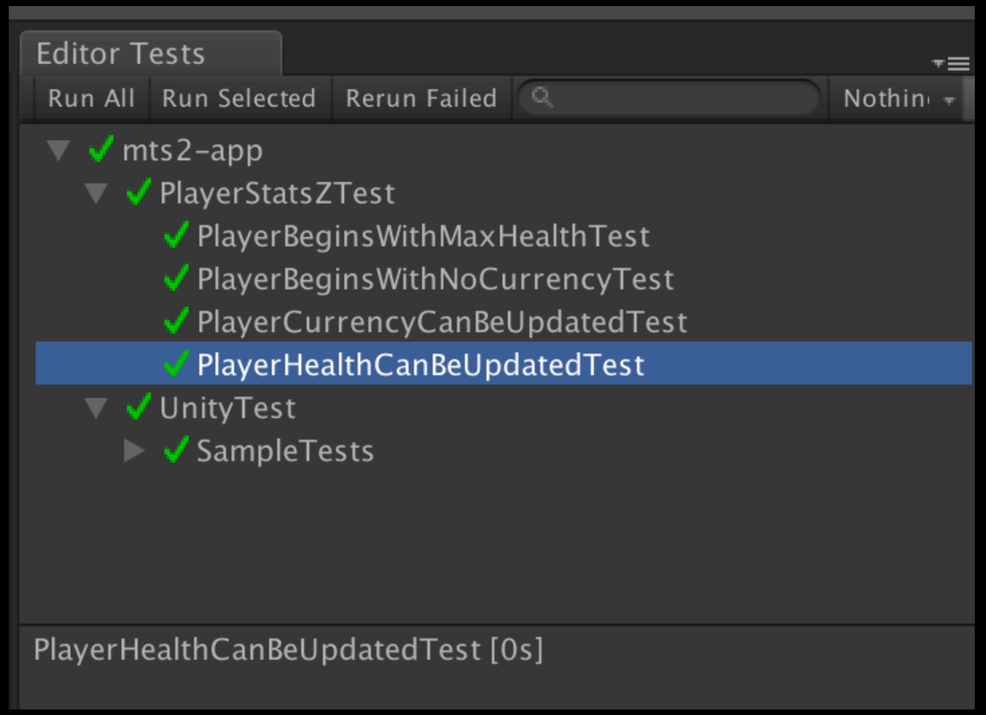
Hands-On Testing and Development

```
[Test]
public void PlayerHealthCanBeUpdatedTest()
{
    // Arrange
    const int updatedHealth = 50;
    PlayerStatsZ playerStats = new PlayerStatsZ();

    // Act
    playerStats.UpdateHealth(-25); Type `PlayerStatsZ' does not contain a definition for `UpdateHealth' and no extension method `UpdateHealth' of type `PlayerStatsZ' could be found. Are you
    playerStats.UpdateHealth(-25); Type `PlayerStatsZ' does not contain a definition for `UpdateHealth' and no extension method `UpdateHealth' of type `PlayerStatsZ' could be found. Are you

    // Assert
    Assert.That(playerStats.CurrentHealth, Is.EqualTo(updatedHealth));
}

[Test]
public void PlayerCurrencyCanBeUpdatedTest()
{
    // Arrange
    const int updatedCurrency = 50;
    PlayerStatsZ playerStats = new PlayerStatsZ();

    // Act
    playerStats.UpdateCurrency(25); Type `PlayerStatsZ' does not contain a definition for `UpdateCurrency' and no extension method `UpdateCurrency' of type `PlayerStatsZ' could be found.
    playerStats.UpdateCurrency(25); Type `PlayerStatsZ' does not contain a definition for `UpdateCurrency' and no extension method `UpdateCurrency' of type `PlayerStatsZ' could be found.

    // Assert
    Assert.That(playerStats.CurrentCurrency, Is.EqualTo(updatedCurrency));
}
```

Hands-On Testing and Development

```
public class PlayerStatsZ
{
    public int CurrentHealth;
    public int CurrentCurrency;

    public PlayerStatsZ()
    {
        CurrentHealth = 100;
        CurrentCurrency = 0;
    }

    public void UpdateHealth(int deltaHealth)
    {

    }

    public void UpdateCurrency(int deltaCurrency)
    {

    }
}
```

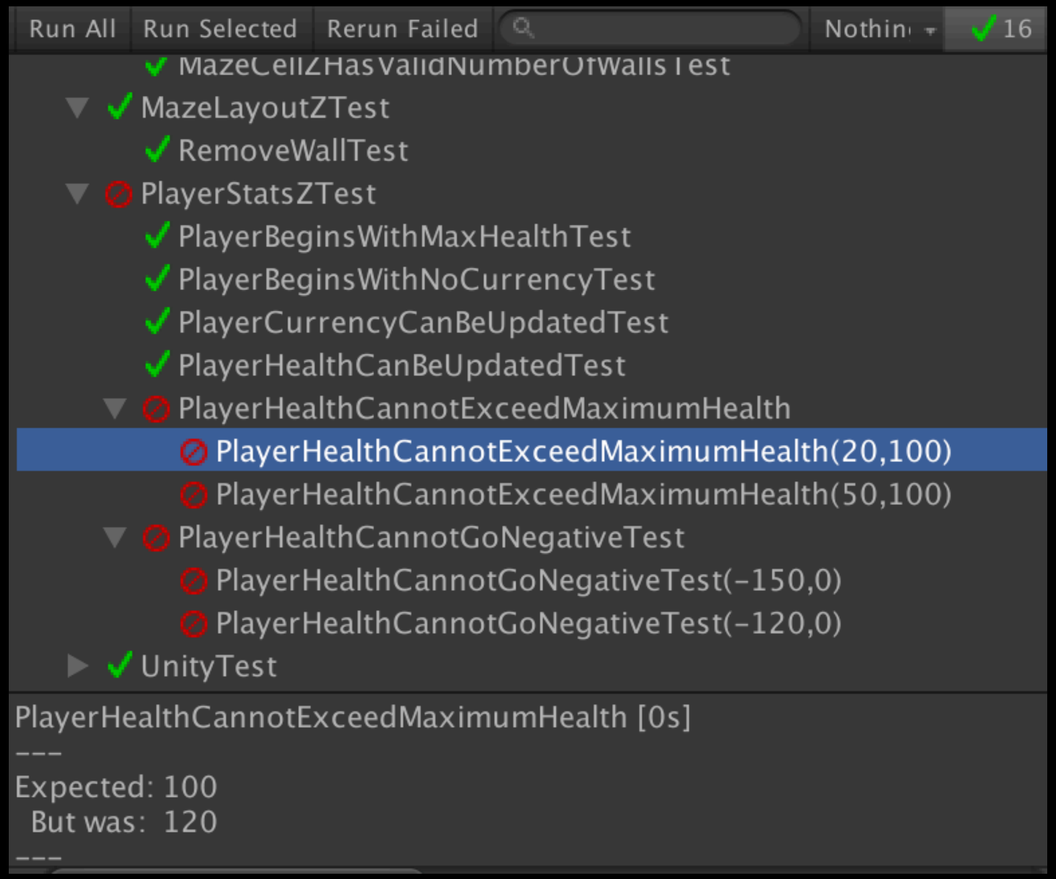
Hands-On Testing and Development




```
public class PlayerStatsZ
{
    public int CurrentHealth;
    public int CurrentCurrency;

    public PlayerStatsZ()
    {
        CurrentHealth = 100;
        CurrentCurrency = 0;
    }

    public void UpdateHealth(int deltaHealth)
    {
        CurrentHealth += deltaHealth;
    }

    public void UpdateCurrency(int deltaCurrency)
    {
        CurrentCurrency += deltaCurrency;
    }

}
```

Hands-On Testing and Development

BE SURE TO TEST THE EXCEPTIONAL CASES (FIRST)!



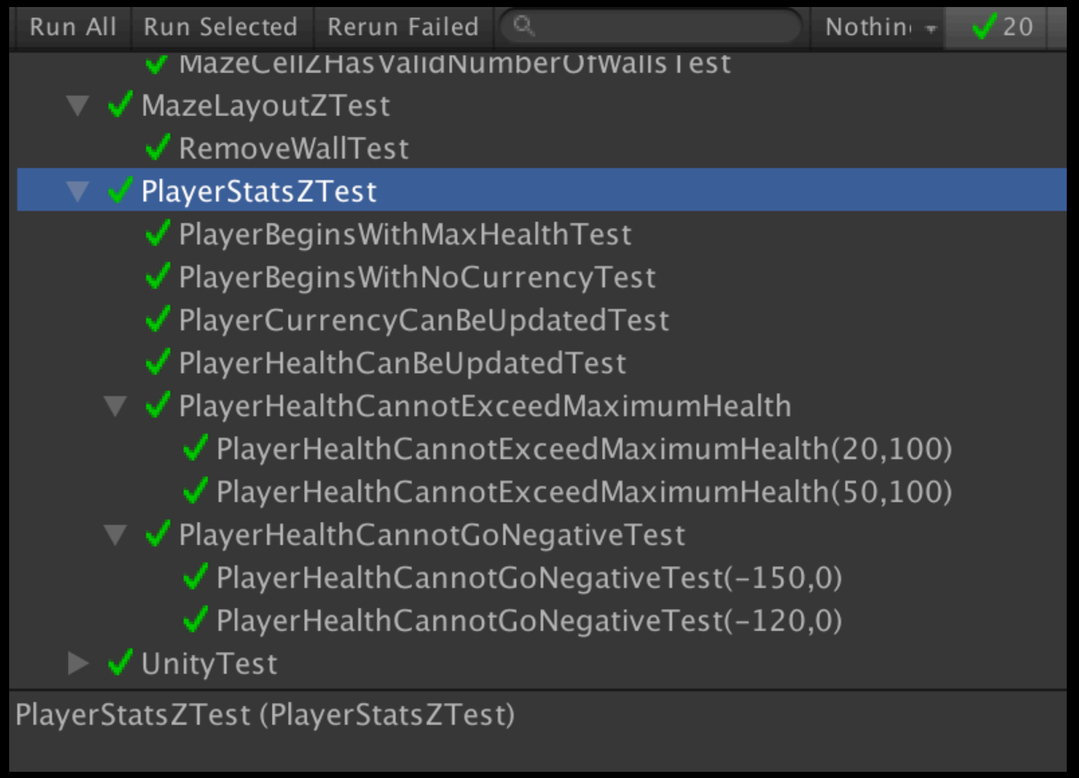
Hands-On Testing and Development

```
[TestCase(-120, 0)]
[TestCase(-150, 0)]
public void PlayerHealthCannotGoNegativeTest(int deltaHealth, int expectedHealth)
{
    // Arrange
    PlayerStatsZ playerStats = new PlayerStatsZ();

    // Act
    playerStats.UpdateHealth(deltaHealth);

    // Assert
    Assert.That(playerStats.CurrentHealth, Is.EqualTo(expectedHealth));
}

[TestCase(20, 100)]
[TestCase(50, 100)]
public void PlayerHealthCannotExceedMaximumHealth(int deltaHealth, int expectedHealth)
{
    // Arrange
    PlayerStatsZ playerStats = new PlayerStatsZ();

    // Act
    playerStats.UpdateHealth(deltaHealth);

    // Assert
    Assert.That(playerStats.CurrentHealth, Is.EqualTo(expectedHealth));
}
```

Hands-On Testing and Development

REMEMBER TO REFACTOR YOUR CODE AND TESTS.

```
public PlayerStatsZ()
{
    CurrentHealth = MAX_HEALTH;
    CurrentCurrency = 0;
}

public void UpdateHealth(int deltaHealth)
{
    CurrentHealth += deltaHealth;
    if (CurrentHealth > MAX_HEALTH)
    {
        CurrentHealth = MAX_HEALTH;
    }
    if (CurrentHealth < 0)
    {
        CurrentHealth = 0;
    }
}

public void UpdateCurrency(int deltaCurrency)
{
    CurrentCurrency += deltaCurrency;
    if (CurrentCurrency < 0)
    {
        CurrentCurrency = 0;
    }
}
```

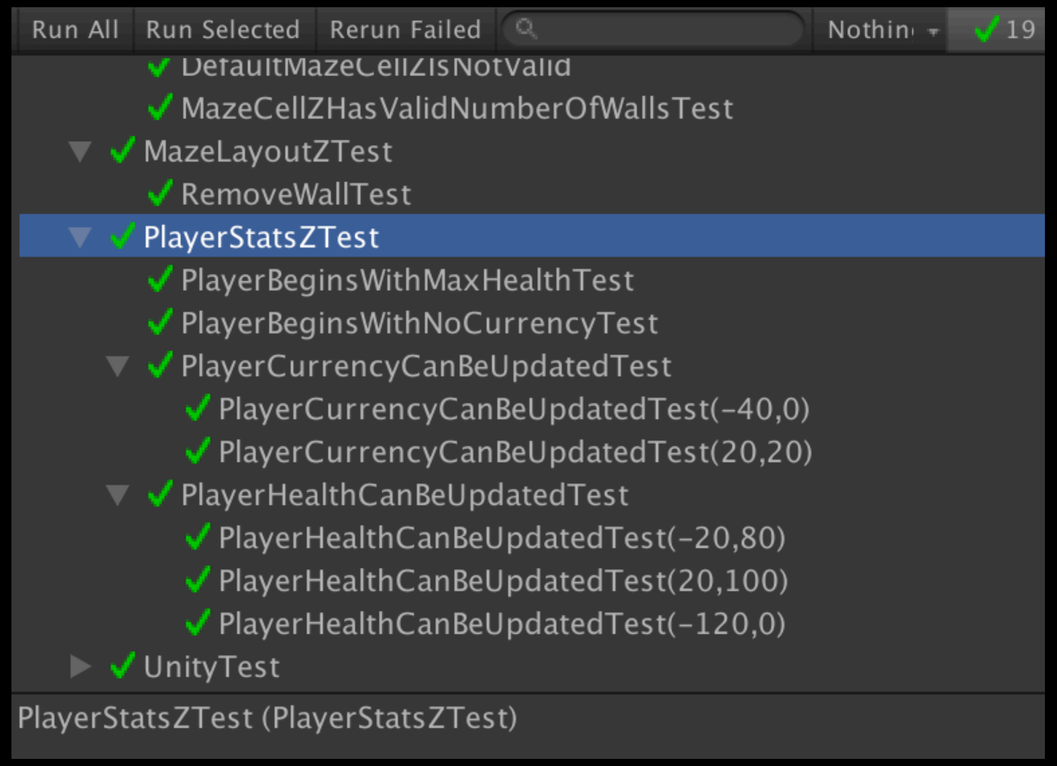


Hands-On Testing and Development

```
[TestCase(20, 100)]
[TestCase(-20, 80)]
[TestCase(-120, 0)]
public void PlayerHealthCanBeUpdatedTest(int deltaHealth, int expectedHealth)
{
    // Arrange
    PlayerStatsZ playerStats = new PlayerStatsZ();

    // Act
    playerStats.UpdateHealth(deltaHealth);

    // Assert
    Assert.That(playerStats.CurrentHealth, Is.EqualTo(expectedHealth));
}

[TestCase(20, 20)]
[TestCase(-40, 0)]
public void PlayerCurrencyCanBeUpdatedTest(int deltaCurrency, int expectedCurrency)
{
    // Arrange
    PlayerStatsZ playerStats = new PlayerStatsZ();

    // Act
    playerStats.UpdateCurrency(deltaCurrency);

    // Assert
    Assert.That(playerStats.CurrentCurrency, Is.EqualTo(expectedCurrency));
}
```

Hands-On Testing and Development

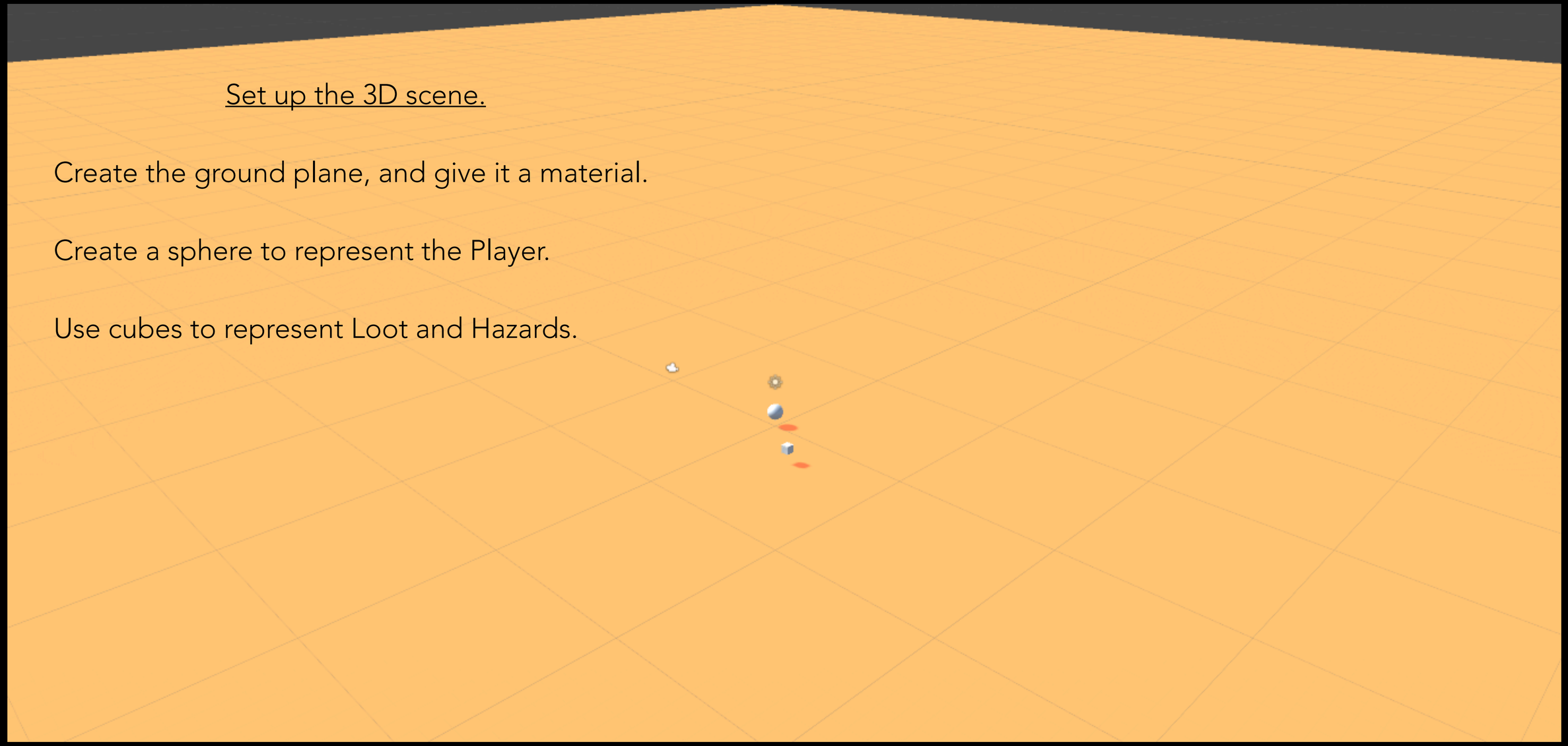
NOW LET'S COMPLETE THE INITIAL STORIES.



When you get into the TDD rhythm (fail, pass, refactor), the overhead of testing is not difficult.

```
//      DONE
// Player Stats are initialized with maximum Health and zero Currency.
// Currency in Player Stats can be increased or decreased.
// Health in Player Stats can be increased or decreased.
// Health cannot exceed Maximum Health (100).
// Health and Currency cannot go negative.
```

Hands-On Testing and Development



Set up the 3D scene.

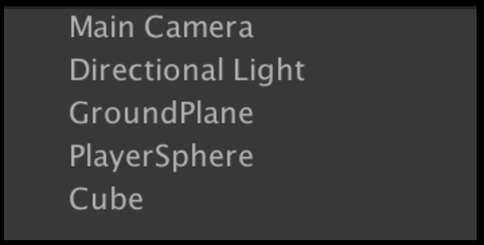
Create the ground plane, and give it a material.

Create a sphere to represent the Player.

Use cubes to represent Loot and Hazards.

Hands-On Testing and Development

```
using UnityEngine;

public class PlayerControlZBehaviour : MonoBehaviour
{

    private Rigidbody _rb;


    void Start()
    {
        _rb = GetComponent<Rigidbody>();
    }

    void Update()
    {
        // for game logic
    }

    void FixedUpdate()
    {
        // for physics logic
        const float speed = 1.5f;

        float thrustX = Input.GetAxis("Horizontal");
        float thrustZ = Input.GetAxis("Vertical");

        Vector3 thrust = new Vector3(thrustX, 0.0f, thrustZ);
        _rb.AddForce(thrust * speed);
    }

}
```



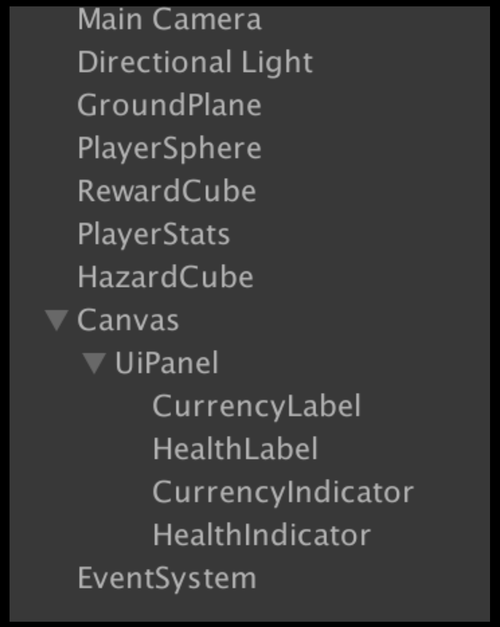
Provide a way for the user to control the Player.

https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial

No tests needed!

Hands-On Testing and Development

Update the Player Control to handle collisions.
Add a Canvas, and build the User Interface.
Adjust the UI to be a reasonable size.




```
void OnTriggerEnter(Collider other)
{
    DetermineCollisionResults(other);
}

public void DetermineCollisionResults(Collider aCollider)
{
    Debug.Log("DetermineCollisionResults reached.");
    const int wealthIncrement = 10;
    const int healthIncrement = -5;

    if (aCollider.gameObject.CompareTag("Reward"))
    {
        aCollider.gameObject.SetActive(false);

        _playerStats.UpdateCurrency(wealthIncrement);

    }

    if (aCollider.gameObject.CompareTag("Hazard"))
    {
        _playerStats.UpdateHealth(healthIncrement);
    }
}
```

Hands-On Testing and Development

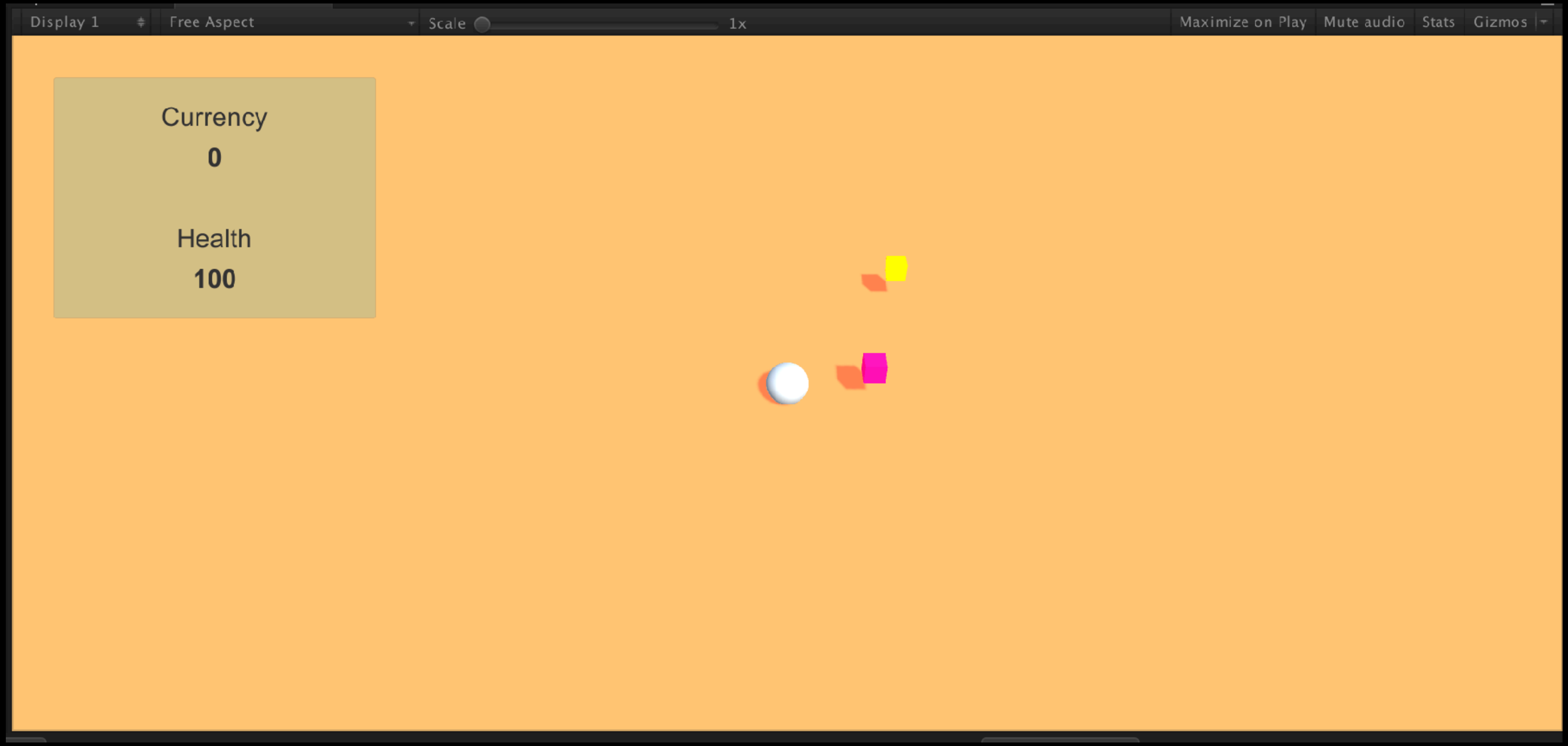
Let the camera follow the player.
Add materials to the cubes.
Modify the Player Stats to update the UI.

```
using UnityEngine;

public class CameraControlZBehaviour : MonoBehaviour
{
    [SerializeField]
    GameObject _player;

    Vector3 _offset;

    void Start()
    {
        _offset = transform.position - _player.transform.position;
        _offset += new Vector3(0.0f, 10.0f, 0.0f);
    }

    void LateUpdate()
    {
        transform.position = _player.transform.position + _offset;
        transform.LookAt(_player.transform);
    }

}
```

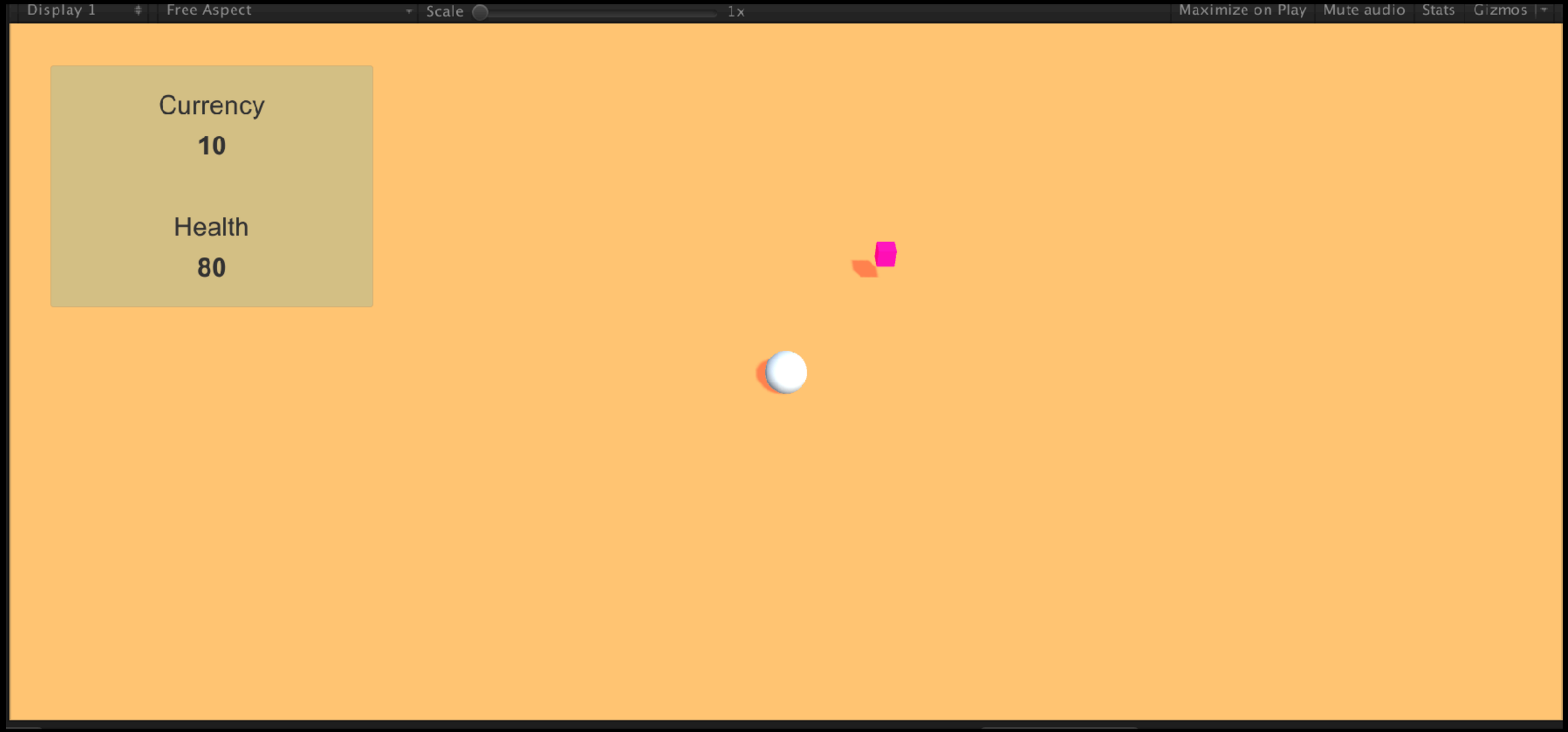
Hands-On Testing and Development

```
using UnityEngine;
using UnityEngine.UI;

public class PlayerStatsZBehaviour : MonoBehaviour
{
    [SerializeField]
    Text _healthIndicator;

    [SerializeField]
    Text _currencyIndicator;

    PlayerStatsZ _playerStats;

    void Awake()
    {
        _playerStats = new PlayerStatsZ();
    }

    public void UpdateHealth(int deltaHealth)
    {
        _playerStats.UpdateHealth(deltaHealth);
        _healthIndicator.text = _playerStats.CurrentHealth.ToString();
    }

    public void UpdateCurrency(int deltaCurrency)
    {
        _playerStats.UpdateCurrency(deltaCurrency);
        _currencyIndicator.text = _playerStats.CurrentCurrency.ToString();
    }
}
```

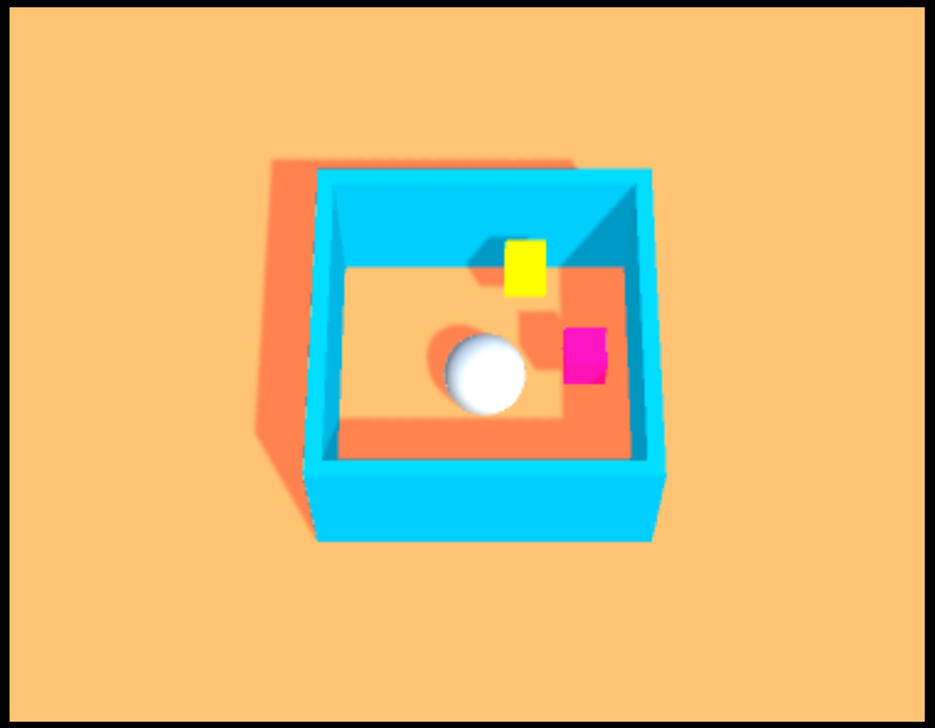
Hands-On Testing and Development



Hands-On Testing and Development



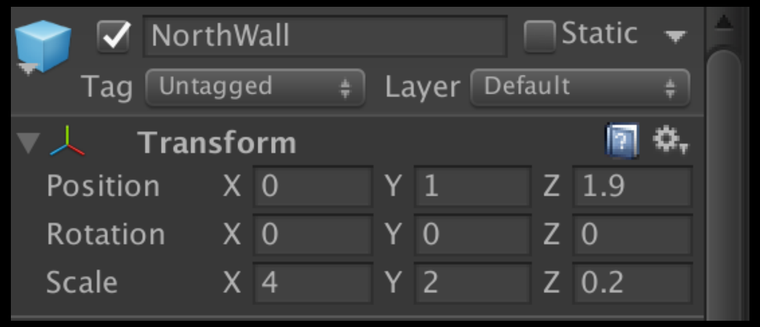
Hands-On Testing and Development

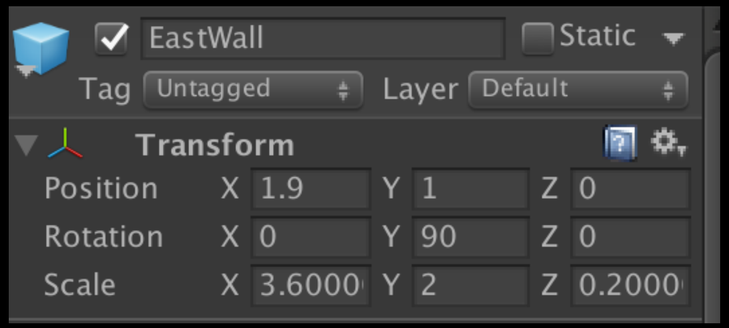


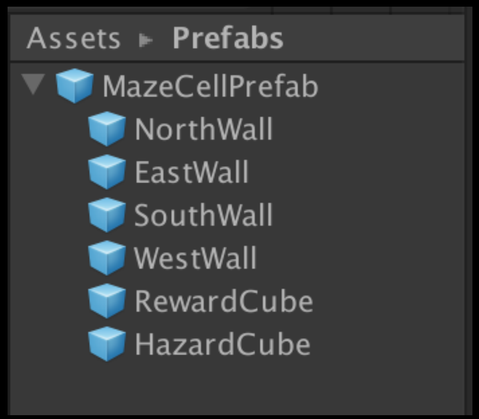
UNITY PREFABS

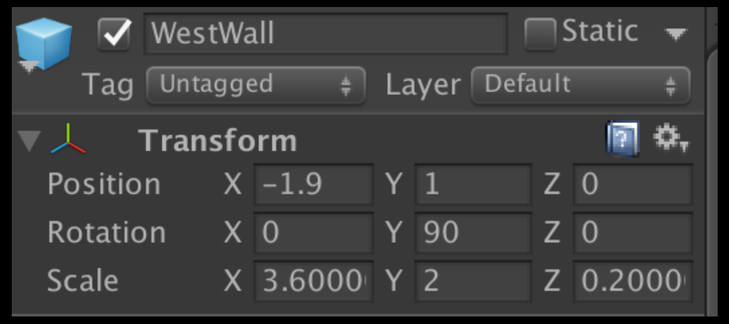
Now we take my advice to set up the scene
BEFORE writing more tests.

Hands-On Testing and Development

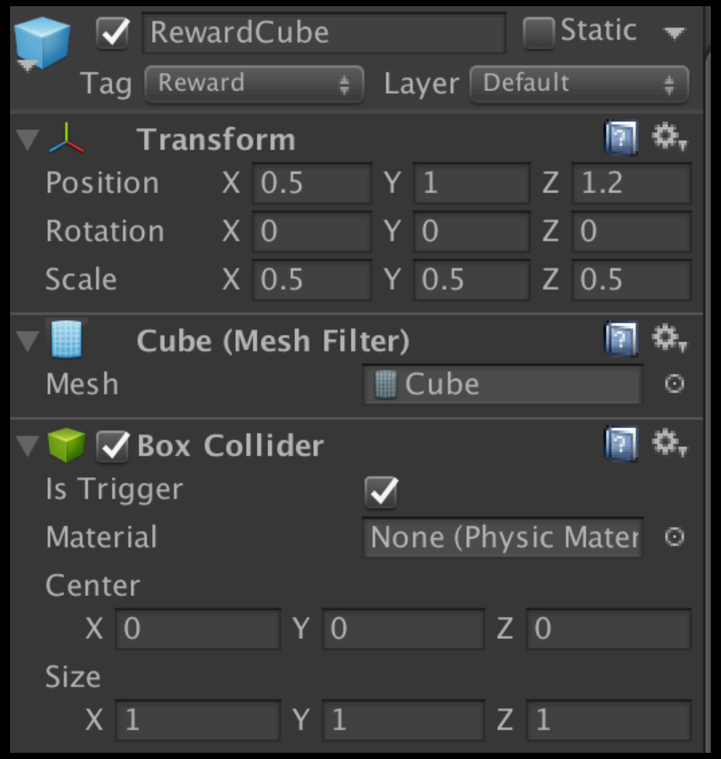


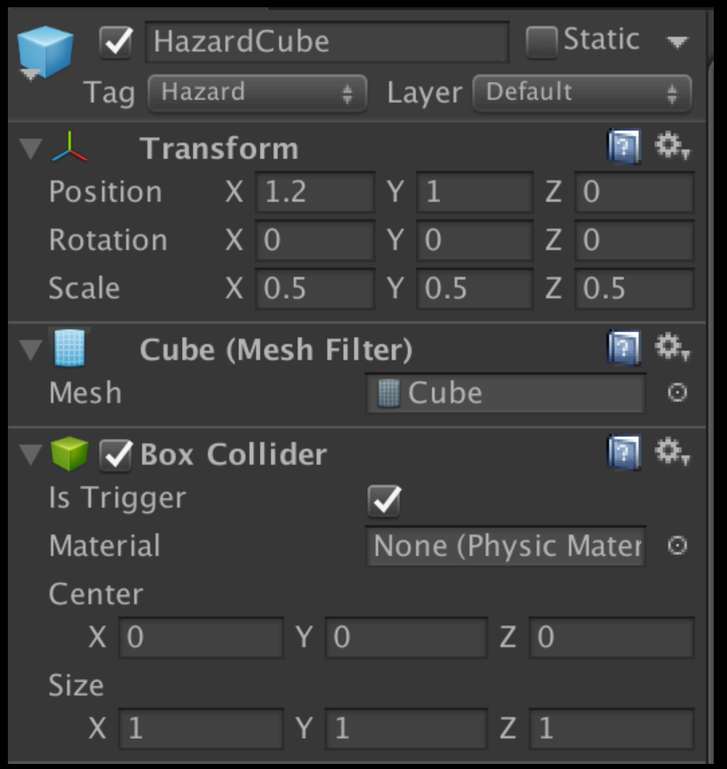




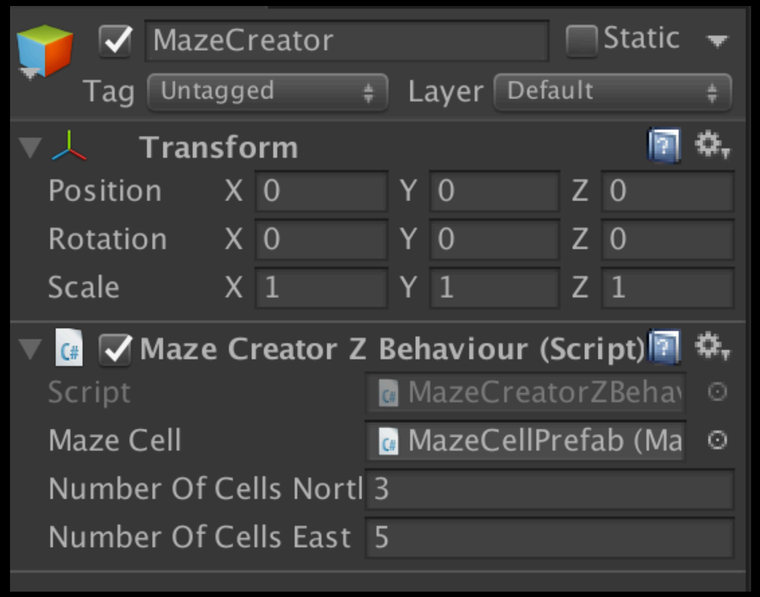


Hands-On Testing and Development





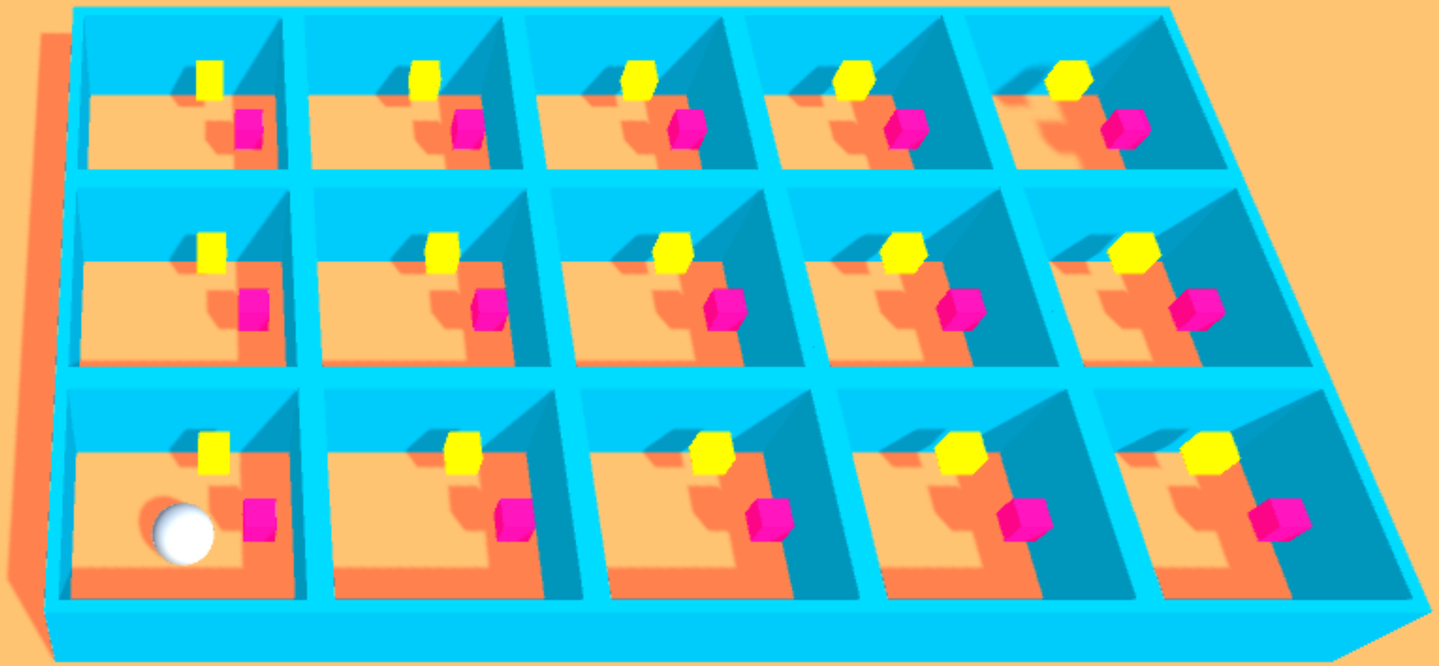
Hands-On Testing and Development



Note: Refactoring may be needed.

```
using UnityEngine;
using System.Collections;

public class MazeCreatorZBehaviour : MonoBehaviour
{
    [SerializeField]
    MazeCellZBehaviour _mazeCell;

    [SerializeField]
    int _numberOfCellsNorth;

    [SerializeField]
    int _numberOfCellsEast;


    void Start()
    {
        const float mazeCellSize = 4.0f;
        for (int i = 0; i < _numberOfCellsEast; ++i)
        {
            for (int j = 0; j < _numberOfCellsNorth; ++j)
            {
                Vector3 position = new Vector3(mazeCellSize * i, 0, mazeCellSize * j);
                Quaternion rotation = new Quaternion();
                Instantiate(_mazeCell, position, rotation);
            }
        }

    }

}
```

Hands-On Testing and Development



We need to remove some walls!

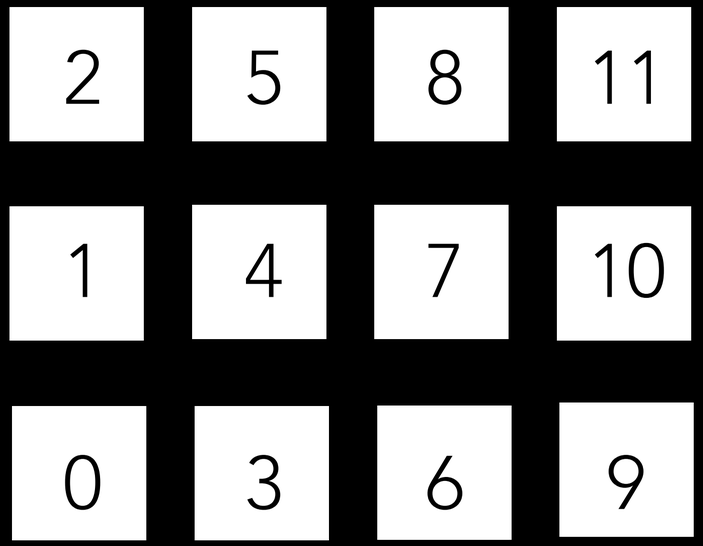
Hands-On Testing and Development

STORIES

As a player, I can reach each cell in the maze.
As a player, I never see missing Border walls (except for the entrance and exit).
As a player, I can enter and exit the maze.
As a player, I never see adjacent cells with only one adjacent wall missing.

```
// TODO
//
// A Default Maze Cell is not valid, because it has 4 walls.
// A valid Maze Cell must have 0, 1, 2, or 3 walls.
// A Maze Shaper can identify Border Cells.
// A Maze Shaper can find adjacent Maze Cells (if they exist).
```

IDENTIFY BORDER CELLS

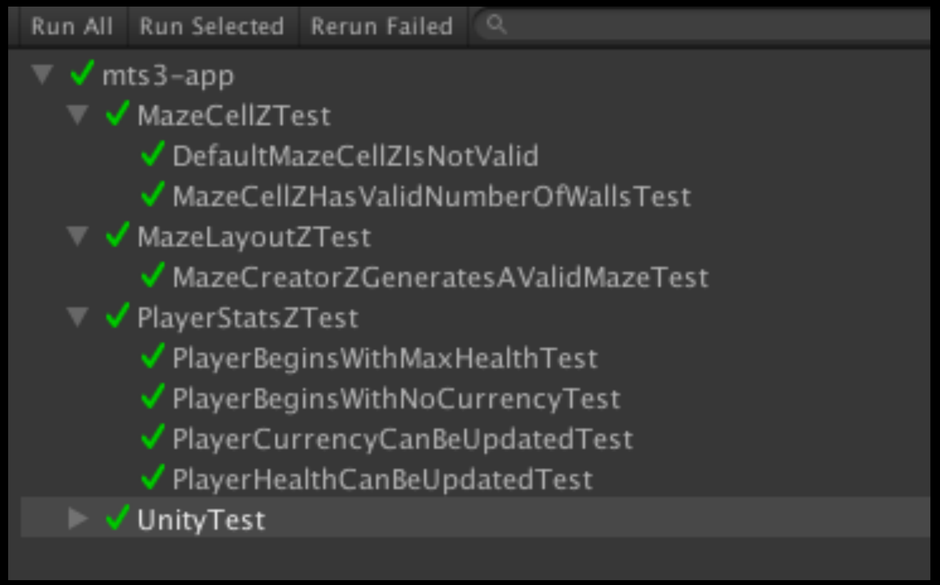


Hands-On Testing and Development

VALID MAZE CELLS CANNOT HAVE FOUR WALLS.

```
[Test]
public void DefaultMazeCellZIsNotValid()
{
    // Arrange and Act
    MazeCellZ mazeCell = new MazeCellZ();

    // Assert
    Assert.IsFalse(mazeCell.IsValid()); // A MazeCellZ is constructed with four walls
}

[Test]
public void MazeCellZHasValidNumberOfWallsTest()
{
    // Arrange
    MazeCellZ mazeCell = new MazeCellZ();

    // Act and Assert
    mazeCell.RemoveWall(WallIndex.NorthWall);
    Assert.IsTrue(mazeCell.IsValid());

    mazeCell.RemoveWall(WallIndex.EastWall);
    Assert.IsTrue(mazeCell.IsValid());

    mazeCell.RemoveWall(WallIndex.SouthWall);
    Assert.IsTrue(mazeCell.IsValid());

    mazeCell.RemoveWall(WallIndex.WestWall);
    Assert.IsTrue(mazeCell.IsValid());

}
```

After removing a wall, these tests all pass.



Hands-On Testing and Development

```
public class MazeShaperZ
{
    int _sizeNorth;
    int _sizeEast;

    public MazeShaperZ(int numberOfCellsNorth, int numberOfCellsEast)
    {
        _sizeNorth = numberOfCellsNorth;
        _sizeEast = numberOfCellsEast;
    }

    public bool IsNorthBorderCell(int cellIndex)
    {
        return false;
    }

    public bool IsEastBorderCell(int cellIndex)
    {
        return false;
    }

    public bool IsSouthBorderCell(int cellIndex)
    {
        return false;
    }

    public bool IsWestBorderCell(int cellIndex)
    {
        return false;
    }

}
```

```
[TestCase(4, 5, 3, WallIndex.NorthWall, true)]
public void IsBorderCellTest(int maxY, int maxX,
                int cellIndex, WallIndex wallIndex, bool expectedResult)
{
    // Arrange
    MazeShaperZ shaper = new MazeShaperZ(maxY, maxX);

    // Act
    bool result = false;
    if (wallIndex == WallIndex.NorthWall)
    {
        result = shaper.IsNorthBorderCell(cellIndex);
    }

    if (wallIndex == WallIndex.EastWall)
    {
        result = shaper.IsEastBorderCell(cellIndex);
    }

    if (wallIndex == WallIndex.SouthWall)
    {
        result = shaper.IsSouthBorderCell(cellIndex);
    }

    if (wallIndex == WallIndex.WestWall)
    {
        result = shaper.IsWestBorderCell(cellIndex);
    }

    // Assert
    Assert.That(expectedResult, Is.EqualTo(result));
}
```

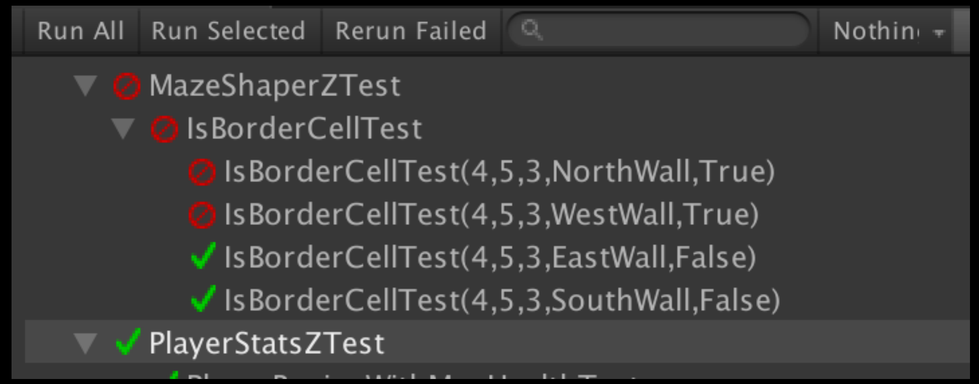
Hands-On Testing and Development

```
public class MazeShaperZ
{
    int _sizeNorth;
    int _sizeEast;

    public MazeShaperZ(int numberOfCellsNorth, int numberOfCellsEast)
    {
        _sizeNorth = numberOfCellsNorth;
        _sizeEast = numberOfCellsEast;
    }

    public bool IsNorthBorderCell(int cellIndex)
    {
        if ((cellIndex % _sizeNorth) == (_sizeNorth - 1) &&
            cellIndex != (_sizeNorth * _sizeEast - 1))
        {
            return true;
        }
        return false;
    }

    public bool IsEastBorderCell(int cellIndex)
    {
        int eastTarget = _sizeNorth * (_sizeEast - 1);
        if (cellIndex >= eastTarget)
        {
            return true;
        }
        return false;
    }
```

```
    public bool IsSouthBorderCell(int cellIndex)
    {
        if ((cellIndex % _sizeNorth) == 0 && cellIndex != 0)
        {
            return true;
        }
        return false;
    }

    public bool IsWestBorderCell(int cellIndex)
    {
        if (cellIndex < _sizeNorth)
        {
            return true;
        }
        return false;
    }

    bool IsBorderCell(int cellIndex)
    {
        if (IsNorthBorderCell(cellIndex) || IsEastBorderCell(cellIndex) ||
            IsSouthBorderCell(cellIndex) || IsWestBorderCell(cellIndex))
        {
            return true;
        }
        return false;
    }
```

Hands-On Testing and Development

DON'T REMOVE BORDER WALLS.




```
[TestCase(4, 5, 3, WallIndex.NorthWall, true)]
[TestCase(4, 5, 3, WallIndex.EastWall, false)]
[TestCase(4, 5, 3, WallIndex.SouthWall, false)]
[TestCase(4, 5, 3, WallIndex.WestWall, true)]
[TestCase(4, 5, 8, WallIndex.SouthWall, true)]
[TestCase(4, 5, 8, WallIndex.NorthWall, false)]
[TestCase(4, 5, 10, WallIndex.EastWall, false)]
[TestCase(4, 5, 10, WallIndex.WestWall, false)]
[TestCase(4, 5, 16, WallIndex.NorthWall, false)]
[TestCase(4, 5, 16, WallIndex.EastWall, true)]
[TestCase(4, 5, 16, WallIndex.SouthWall, true)]
[TestCase(4, 5, 16, WallIndex.WestWall, false)]
public void IsBorderCellTest(int maxY, int maxX,
                int cellIndex, WallIndex wallIndex, bool expectedResult)
{
    // Arrange
    MazeShaperZ shaper = new MazeShaperZ(maxY, maxX);

    // Act
```

Hands-On Testing and Development

LET THE MAZE CREATOR USE THE MAZE SHAPER.

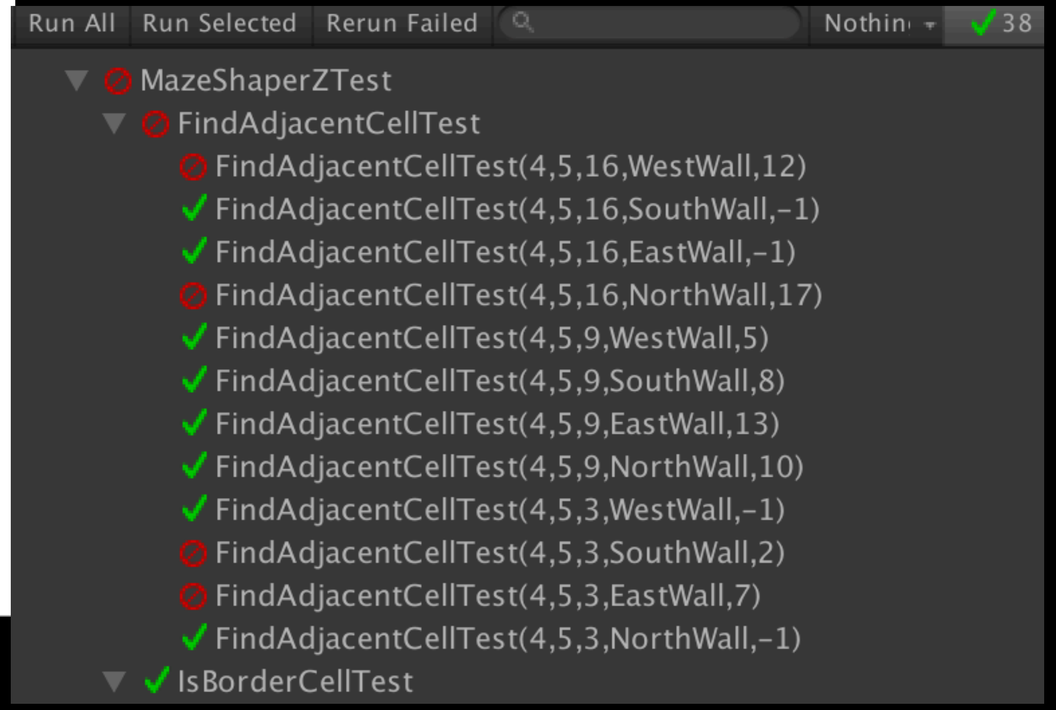
```
void SetOuterWalls()
{
    for (int i = 0; i < _mazeCells.Count; ++i)
    {
        // West
        if (_mazeShaper.IsWestBorderCell(i))
        {
            _mazeCells[i].SetOuterWallAsBorder(WallIndex.WestWall);
        }

        // South
        if (_mazeShaper.IsSouthBorderCell(i))
        {
            _mazeCells[i].SetOuterWallAsBorder(WallIndex.SouthWall);
        }

        // East
        if (_mazeShaper.IsEastBorderCell(i))
        {
            _mazeCells[i].SetOuterWallAsBorder(WallIndex.EastWall);
        }

        // North
        if (_mazeShaper.IsNorthBorderCell(i))
        {
            _mazeCells[i].SetOuterWallAsBorder(WallIndex.NorthWall);
        }

    }

}
```

Hands-On Testing and Development

BE ABLE TO FIND ADJACENT CELLS.

```
[TestCase(4, 5, 3, WallIndex.NorthWall, -1)]
[TestCase(4, 5, 3, WallIndex.EastWall, 7)]
[TestCase(4, 5, 3, WallIndex.SouthWall, 2)]
[TestCase(4, 5, 3, WallIndex.WestWall, -1)]
[TestCase(4, 5, 9, WallIndex.NorthWall, 10)]
[TestCase(4, 5, 9, WallIndex.EastWall, 13)]
[TestCase(4, 5, 9, WallIndex.SouthWall, 8)]
[TestCase(4, 5, 9, WallIndex.WestWall, 5)]
[TestCase(4, 5, 16, WallIndex.NorthWall, 17)]
[TestCase(4, 5, 16, WallIndex.EastWall, -1)]
[TestCase(4, 5, 16, WallIndex.SouthWall, -1)]
[TestCase(4, 5, 16, WallIndex.WestWall, 12)]
public void FindAdjacentCellTest(int maxY, int maxX,
                                 int cellIndex, WallIndex wallIndex, int expected)
{
    // Arrange
    MazeShaperZ shaper = new MazeShaperZ(maxY, maxX);

    // Act
    int adjacentResult = shaper.FindAdjacentCellIndex(cellIndex, wallIndex);

    // Assert
    Assert.That(adjacentResult, Is.EqualTo(expected));
}
```

Sometimes tests catch bugs!



Hands-On Testing and Development

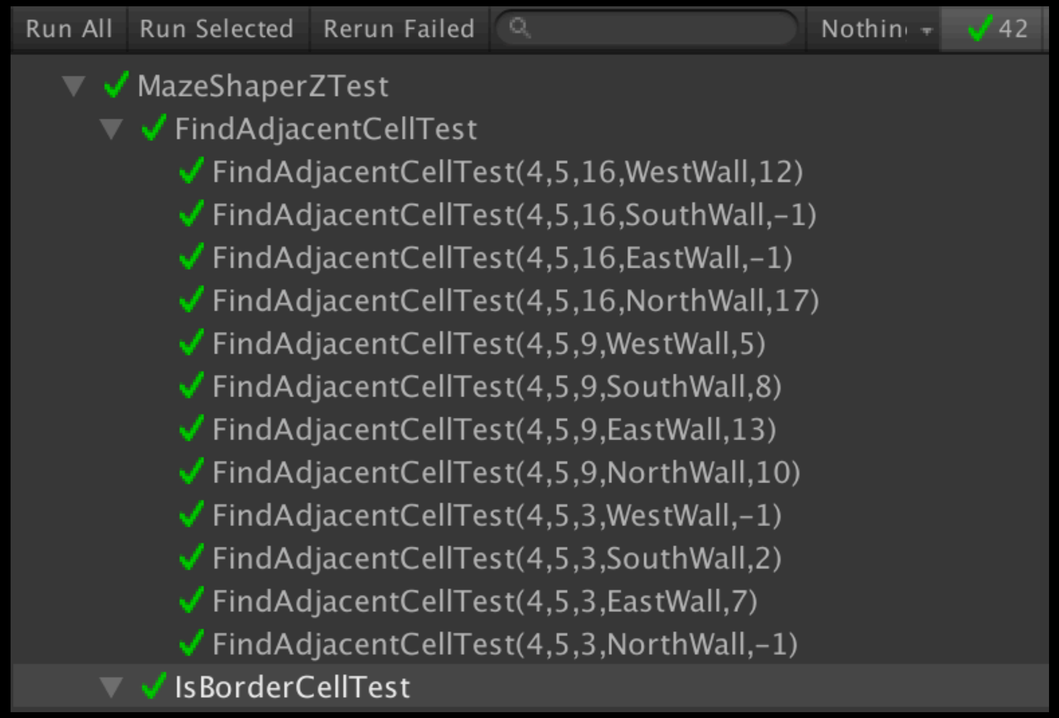
```
public int FindAdjacentCellIndex(int cellIndex, WallIndex wallIndexOfInterest)
{
    int oppositeCellIndex = -1;

    if (!IsBorderCell(cellIndex))
    {
        switch ((int)wallIndexOfInterest)
        {
            case 0:
                {
                    oppositeCellIndex = cellIndex + 1;
                    break;
                }
            case 1:
                {
                    oppositeCellIndex = cellIndex + _sizeNorth;
                    break;
                }
            case 2:
                {
                    oppositeCellIndex = cellIndex - 1;
                    break;
                }
            case 3:
                {
                    oppositeCellIndex = cellIndex - _sizeNorth;
                    break;
                }

        }

    }

    return oppositeCellIndex;
}
```

Ooops!

Hands-On Testing and Development

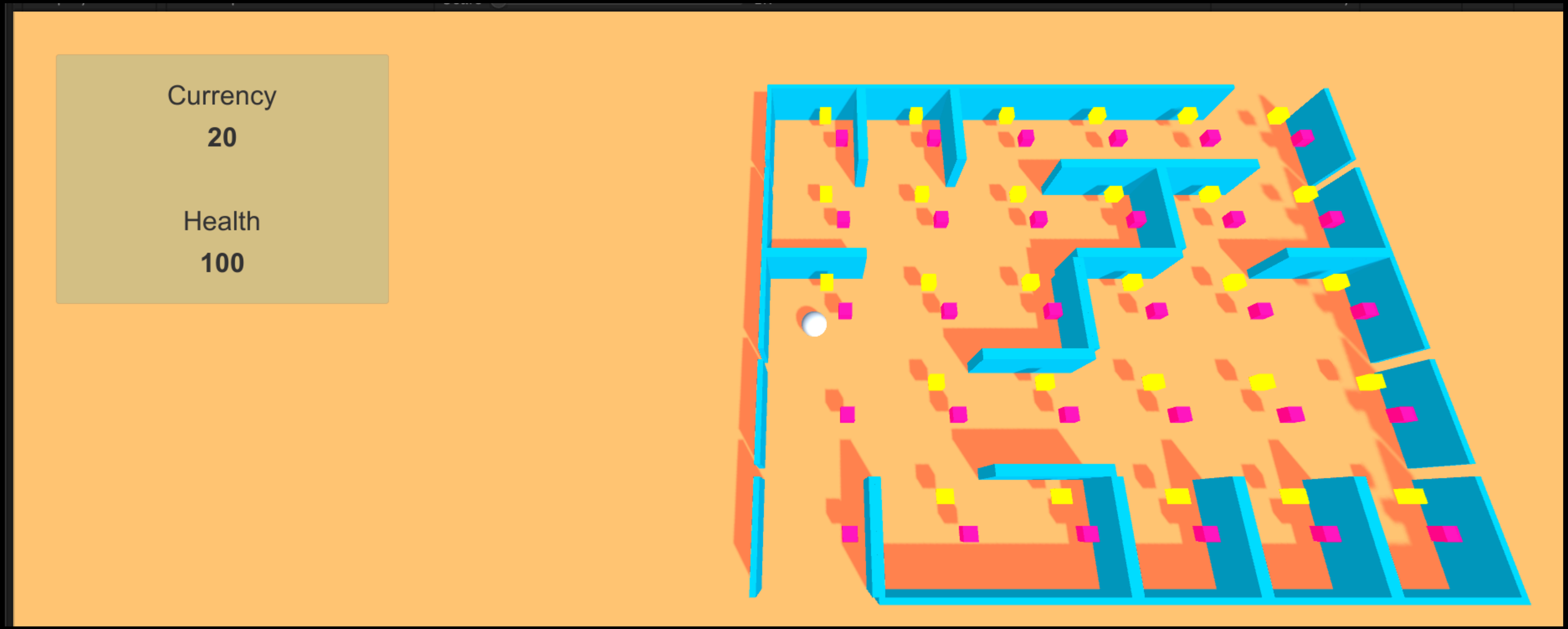
```
public int FindAdjacentCellIndex(int cellIndex, WallIndex wallIndexOfInterest)
{
    int oppositeCellIndex = -1;

    switch ((int)wallIndexOfInterest)
    {
        case 0:
            {
                if (!IsNorthBorderCell(cellIndex))
                {
                    oppositeCellIndex = cellIndex + 1;
                }
                break;
            }
        case 1:
            {
                if (!IsEastBorderCell(cellIndex))
                {
                    oppositeCellIndex = cellIndex + _sizeNorth;
                }
                break;
            }
        case 2:
            {
                if (!IsSouthBorderCell(cellIndex))
                {
                    oppositeCellIndex = cellIndex - 1;
                }
                break;
            }
        case 3:
            {
                if (!IsNorthBorderCell(cellIndex))
                {
                    oppositeCellIndex = cellIndex - _sizeNorth;
                }
                break;
            }

    }

    return oppositeCellIndex;
}
```

WE'RE OK AGAIN.



Hands-On Testing and Development

WE HAVE AN OK MAZE.



Hands-On Testing and Development

TESTING MONOBEHAVIOUR OBJECTS IS OK.

```
[Test]
public void Behaviour1Test()
{
    // Arrange
    SimpleBehaviour1 simple1 = new SimpleBehaviour1();

    // Act
    simple1.HandleCollision();
    simple1.HandleCollision();

    // Assert
    Assert.That(simple1.CollisionCount, Is.EqualTo(2));
}
```

```
public class SimpleBehaviour1 : MonoBehaviour
{
    public int CollisionCount { get; set; }

    void Awake()
    {
        CollisionCount = 0;
    }

    public void HandleCollision()
    {
        CollisionCount++;
    }

}
```

Hands-On Testing and Development

TESTING MONOBEHAVIOUR OBJECTS WITH SERIALIZED FIELDS IS NOT OK.

```
[Test]
public void Behaviour2Test()
{
    // Arrange
    SimpleBehaviour2 simple2 = new SimpleBehaviour2();

    // Act
    simple2.HandleCollision();
    simple2.HandleCollision();

    // Assert
    Assert.That(simple2.GetCollisionCount(), Is.EqualTo(2));
}
```

```
public class SimpleBehaviour2 : MonoBehaviour
{
    [SerializeField]
    int _collisionCount = 5;


    public void HandleCollision()
    {
        _collisionCount++;
    }

    public int GetCollisionCount()
    {
        return _collisionCount;
    }

}
```

Hands-On Testing and Development

MOCKS AND STUBS

Mocks and stubs are “fake” objects that are substituted for real objects to enable unit testing.

“**Stubs** provide canned answers to calls made during the test.”

“**Mocks** are… objects pre-programmed with expectations… mocks insist upon behavior verification.”

http://martinfowler.com/articles/mocksArentStubs.html

Steps

Write the minimum object Interface you need.
Use *NSubstitute* to create a substitute for your object.
Use *NSubstitute* to specify the behavior you need for your test.
Complete the test.

Hands-On Testing and Development

```
using UnityEngine;
using NUnit.Framework;
using NSubstitute;
using System.Collections.Generic;

[TestFixture]
public class DataManipulationTests
{
   [Test]
   public void LoadDataTest ()
   {
      // Arrange
      const int dataCount = 210;
      const int expectedGroupCount = 3;

      var dataLoaderSub = Substitute.For<IDataLoader> ();

      // Act
      DataHandler dh = new DataHandler (dataLoaderSub);
      dataLoaderSub.GetDataCount().Returns(dataCount);

      // Assert
      Assert.That (dh.GetGroupCount (),
                           Is.EqualTo(expectedGroupCount));

   }

}
```

USE *NSUBSTITUTE* TO WRITE MOCKS AND STUBS.

Sometimes you want to write a test without instantiating the real dependencies of a class.

NSubstitute, which comes with the Unity Test Tools, lets you do this.

Use mocks and stubs judiciously.

Hints

WHAT IF IT'S DIFFICULT TO WRITE THE TEST YOU WANT?

Do you really need that specific test?

Prefer Unit Tests over System Tests, but realize there is more than one way to test important functionality.

Can you use the approaches we've seen?

Find out what the Gurus would do.

REFACTORING LEGACY CODE

Wrap the functionality with system tests.
Extract a Method.
Extract a Class.

Write Unit Tests as you go, one step at a time.

Hints

TESTS CAN HELP YOU...

Verify that important functionality doesn't change.

Document how the code is supposed to function.

Prevent bugs.

Decide what code to write.

TESTS CANNOT HELP YOU...

Write code faster.

Become a better problem-solver.

Beware of "all or nothing" approaches.

How much testing should you do?
Consider the risks and rewards.

Hints

Is it bad to only have something public for a test?
It may be a sign of where you can refactor.

How do you reach a good design using TGD?
Incrementally…
Don’t forget the refactor step!
Don’t let TGD be an excuse not to think about design.

Tests should interact with your object the way you expect collaborators to.

Bugs are most expensive to fix when discovered late in development. TGD helps you prevent bugs and fix any that do happen early on.

Keep your test code in a repository.
Run your tests every time you check your code in.
Let the build server run the tests often and automatically.

Hints

Whenever you find a bug, write a failing test that passes after you fix the bug.

Avoid testing *Unity* and third-party systems that already work.
- Focus on testing the game logic code you want to develop.
- Pay attention to when mocks can make your life easier.

Do not develop back-end functionality independently of the UI.
- *Unity* encourages UI-first development — embrace this.
- Identify back-end functionality as you discover it, and switch into TDD mode.

Never hesitate to refactor important functionality using TDD.
- Rely on SOLID principles to update the technical design.
  https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
- If possible, create a component level test before getting started
- Add unit tests as you improve the system bit-by-bit.

Special Thanks

Unity
https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial

Catlike Coding
http://catlikecoding.com/unity/tutorials/
http://catlikecoding.com/unity/tutorials/maze/

Sphero

My Family

Questions and Answers

- THANKS, SCOTT ADAMS