Documentação do Teste: Pong

Os testes empregados na aplicação são testes unitários utilizando o framework Nunit.

Um teste unitário é uma técnica de teste de software que visa verificar o funcionamento correto de unidades individuais do código.

Dessa forma, o teste unitário isola a unidade a ser testada e fornece entradas específicas para verificar se a saída é a esperada. Um teste unitário é uma técnica de teste de software que visa verificar o funcionamento correto de unidades individuais do código. Dessa forma, o teste unitário isola a unidade a ser testada e fornece entradas específicas para verificar se a saída é a esperada.

No código fonte do projeto dentro da pasta “Assets” há uma pasta denominada “Tests” que contém os arquivos fonte “aquivo.cs” (“Csharp” ou “C#”).

Os Testes estão estruturados de modo que verificam a forma como um objeto da aplicação deve se comportar durante a gameplay e validam uma ação principal do objeto testado.

Dentro do Pong há os seguintes objetos a ser testado:

- Player (que são nossos Jogadores);
- Ball (Bola);
- Goal (gol).

As classes pertencentes a estes objetos foram marcadas com a anotação [TestFixture].

A anotação [TestFixture] é usada no framework NUnit para marcar uma classe como uma classe de teste. Ela indica que a classe contém um ou mais métodos de teste que serão executados pelo NUnit durante a execução dos testes unitários.

Teste nos jogadores:

Foram escritas 5 funções que testam os players na cena do game, sendo elas:

1. TestExistingPlayer

```
[Test]
    public void TestExistingPlayer()
    {
        var PlayerInScene = GameObject.FindAnyObjectByType<ControllerPlayers>
();      Assert.IsNotNull(PlayerInScene) ;
    }
```

   - Verifica a existência de jogadores na cena do game, essa verificação ocorre mapeando quais “GameObject” contém o script responsável pelos jogadores.

2. AmountPlayer

```
[Test]
   public void AmountPlayer()
   {
       var playersInScene = GameObject.FindObjectsOfType<ControllerPlayers>
();     int playerCount = playersInScene.Length;

       Assert.AreEqual(2, playerCount);

   }
```

   - Verifica se a quantidade de jogadores na tela é no máximo 2.

3. isPlayerOne

```
 [Test]
    public void isPlayerOne()
    {
        var playersInScene = GameObject.FindObjectsOfType<ControllerPlayers>
();     var playerOne = playersInScene[1];

        Assert.IsTrue(playerOne.playerOne);

    }
```

   - Verifica se um dos 2 “GameObject” encontrados na cena é o playerOne.

4. isPlayerTwo

```
[Test]
    public void isPlayerTwo()
    {
        var playersInScene = GameObject.FindObjectsOfType<ControllerPlayers>
();     var playerTwo = playersInScene[0];

        Assert.IsFalse(playerTwo.playerOne);

    }
```

   - Verifica se um dos 2 “GameObject” encontrados na cena é o playerTwo.

5. MovePlayer

```
public void MovePlayer()
    {
        var playersInScene = GameObject.FindObjectsOfType<ControllerPlayers>
();      var playerOne = playersInScene[1];
        var playerTwo = playersInScene[0];

        playerOne.transform.position = new Vector3(0, 5, 0);
        Assert.IsTrue(playerOne.transform.position.y != 0);

        playerTwo.transform.position = new Vector3(0, 5, 0);
        Assert.IsTrue(playerTwo.transform.position.y != 0);

    }
```

   - Verifica se os dois jogadores conseguem se movimentar na tela.

Teste na bolinha:

Foram escritas 3 funções para testa a bolinha do Pong, sendo elas:

1. TestExistingBall

```
[Test]
    public void TestExistingBall()
    {
        var BallInScene = GameObject.FindAnyObjectByType<Ball>
();      Assert.IsNotNull(BallInScene);
    }
```

   - Verifica a existência da bolinha na cena do game, essa verificação ocorre mapeando qual “GameObject” contém o script responsável pela bolinha.

2. AmountBall

```
[Test]
    public void AmountBall()
    {
        var BallInScene = GameObject.FindObjectsOfType<Ball>();
        int BallCount = BallInScene.Length;

        Assert.AreEqual(1, BallCount);

    }
```

- Verifica se há apenas uma bolinha na cena.

3. MoveBall

```
 [Test]
    public void MoveBall()

    {
        var BallInScene = GameObject.FindAnyObjectByType<Ball>
();      BallInScene.transform.position = new Vector3(4, 5, 0);
        Assert.IsTrue(BallInScene.transform.position.x != 0);
        Assert.IsTrue(BallInScene.transform.position.y != 0);

    }
```

- Verifica a movimentação da bolinha na cena.

Testes nos gols:

Foram escritas 5 funções que testa o funcionamento correto dos gols, sendo elas:

1. TestExistingGoal

```
[Test]
    public void TestExistingGoal()
    {
        var GoalInScene = GameObject.FindAnyObjectByType<Goal>
();      Assert.IsNotNull(GoalInScene);
    }
```

   - Verifica a existência de Gols na cena do game, essa verificação ocorre mapeando qual “GameObject” contém o script responsável pelo gol.

2. AmountGoal

```
[Test]
    public void AmountGoal()
    {
        var GoalInScene = GameObject.FindObjectsOfType<Goal>();
        int GoalCount = GoalInScene.Length;

        Assert.AreEqual(2, GoalCount);

    }
```

   - Verifica se a dois gols na cena.

3. isGoalOne

```
[Test]
    public void isGoalOne()
    {
        var GoalInScene = GameObject.FindObjectsOfType<Goal>();
        var GoalOne = GoalInScene[1];
        Assert.IsTrue(GoalOne.goalPlayerOne);

    }
```

- Verifica se o gol pertence ao jogador um.

4. isGoalTwo

```
[Test]
    public void isGoalTwo()
    {
        var GoalInScene = GameObject.FindObjectsOfType<Goal>();
        var GoalTwo = GoalInScene[0];
        Assert.IsFalse(GoalTwo.goalPlayerOne);

    }
```

- Verifica se o gol pertence ao jogador dois.

5. MakeGoal

```
[Test]
    public void MakeGoal()
    {
        var GoalOne =  FindAnyObjectByType<GameManager>
();     var GoalTwo = FindAnyObjectByType<GameManager>();

        GoalOne.scorePlayerOne = 1;
        GoalTwo.scorePlayerOne = 1;

        Assert.NotZero(GoalOne.scorePlayerOne);
        Assert.NotZero(GoalTwo.scorePlayerOne);
    }
```

- A Verificação aqui é feita validando se o score é alterado, validando tanto o gol como o sistema de score.