Kauno technologijos universitetas

Informatikos fakultetas

T120B162 Programų sistemų testavimas

Todžės Warriors

Lab 2. Automated Testing

Ugnius Rinkevičius Studentas

Martynas Kuliešius Studentas

Nedas Liaudanskis Studentas

Paulius Osipauskas Studentas

Dėst. Lukas Arlauskas

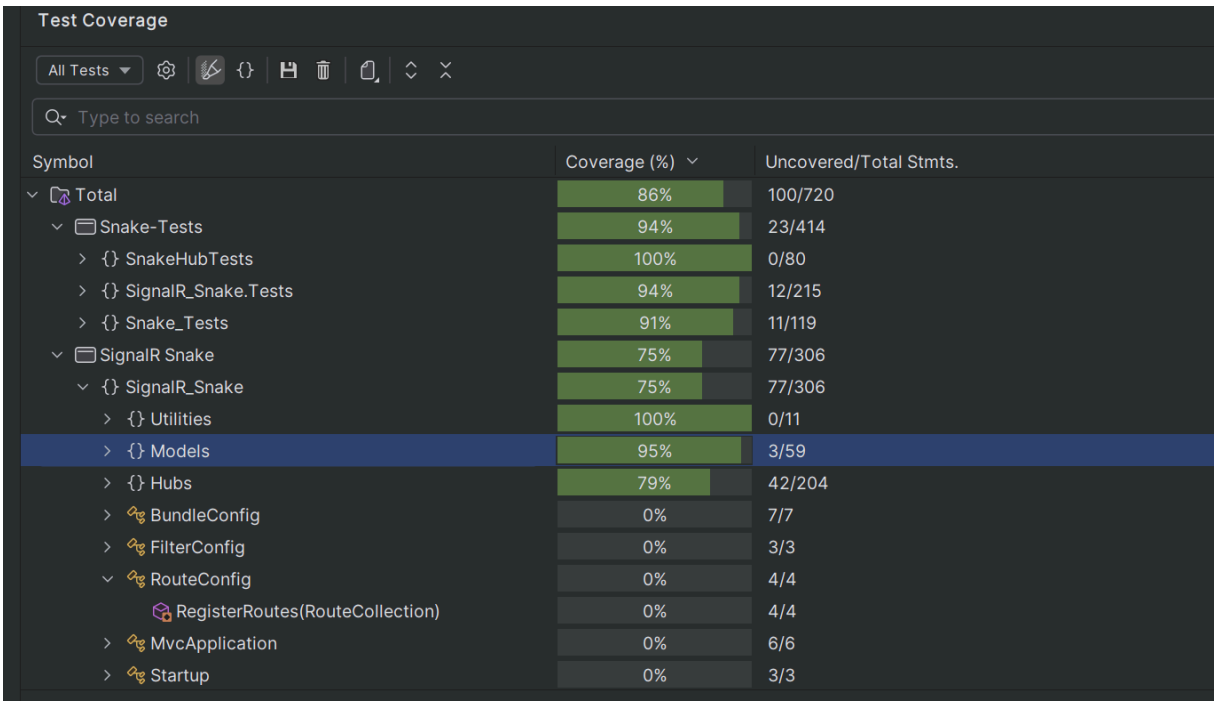
Turinys

Įvadas:

Laboratorinio darbo tikslas yra atlikti programinės įrangos testavimą, siekiant užtikrinti programos kokybę. Šiam tikslui įvykdyti pasitelkėme “NUnit” įrankį, kurio dėka galėjome apsirašyti šiuos testus bei patikrinti funkcionalumą. Pasitelkėme „JetBrain Rider “ įrankį „Test Coverage“, kuris parodo kiek procentų projekto apima mūsų aprašyti testai.

Testavimo apimtis:

Šį laboratorinį darbą mes gavome tiktai 86% kodo testavimo apimtį. Buvo parašyti 66 testai.



Testavimo kodas:

MovementStrategyTests.cs

Aprašas:

`MovementStrategyTests` klasė apima vieneto testų rinkinį, kurie tikrina skirtingų judėjimo strategijų veikimą naudojamų `SignalR_Snake` programoje. Testai užtikrina, kad judėjimo skaičiavimai, priklausomai nuo krypties ir greičio, būtų atliekami teisingai.

Kodas:

```
using NUnit.Framework;
using SignalR_Snake.Models.Strategies;
using System.Drawing;

namespace Snake_Tests
{
    [TestFixture]
    public class MovementStrategyTests
    {
        [Test]
        [TestCase(0, 10, 10, 0)]
        [TestCase(90, 10, 0, 10)]
        [TestCase(180, 10, -10, 0)]
        [TestCase(270, 10, 0, -10)]
        [TestCase(45, 10, 7, 7)]
        public void NormalMovementStrategy_Move_ShouldUpdatePosition(double direction, int
speed, int expectedX, int expectedY)
        {
            var normalMovementStrategy = new NormalMovementStrategy();
            var startPosition = new Point(0, 0);
            Point result = normalMovementStrategy.Move(startPosition, direction, speed);
            Assert.AreEqual(new Point(expectedX, expectedY), result, $"The position should move as
expected for direction {direction} degrees and speed {speed}.");
        }

        [Test]
        [TestCase(0, 10, 20, 0)]
        [TestCase(90, 10, 0, 20)]
        [TestCase(180, 10, -20, 0)]
        [TestCase(270, 10, 0, -20)]
        [TestCase(45, 10, 14, 14)]
        public void BoostMovementStrategy_Move_ShouldDoubleTheDistance(double direction, int
speed, int expectedX, int expectedY)
        {
            var boostMovementStrategy = new BoostMovementStrategy();
            var startPosition = new Point(0, 0);
            Point result = boostMovementStrategy.Move(startPosition, direction, speed);
            Assert.AreEqual(new Point(expectedX, expectedY), result, $"The position should move twice
```

```
the distance as expected for direction {direction} degrees and speed {speed}.");
    }

    [Test]
    public void
NormalMovementStrategy_Move_WithNegativeSpeed_ShouldMoveInOppositeDirection()
    {
      var normalMovementStrategy = new NormalMovementStrategy();
      var startPosition = new Point(0, 0);
      double direction = 0;
      int speed = -10;
      Point result = normalMovementStrategy.Move(startPosition, direction, speed);
      Assert.AreEqual(new Point(-10, 0), result, "The position should move in the opposite
direction due to negative speed.");
    }

    [Test]
    public void BoostMovementStrategy_Move_WithZeroSpeed_ShouldNotMove()
    {
      var boostMovementStrategy = new BoostMovementStrategy();
      var startPosition = new Point(0, 0);
      double direction = 90;
      int speed = 0;
      Point result = boostMovementStrategy.Move(startPosition, direction, speed);
      Assert.AreEqual(new Point(0, 0), result, "The position should not change when speed is
zero.");
    }
  }
}
```

RandomColorSingletonHelperTests.cs

Aprašas:

Ši testų klasė yra skirta patikrinti `RandomColorSingletonHelper` klasės funkcionalumą. Ši klasė atsakinga už atsitiktinės spalvos generavimą HEX formato spalvų kodo pavidalu bei už singletono dizaino šablono įgyvendinimą.

Kodas:

```
using NUnit.Framework;
using System.Text.RegularExpressions;
using SignalR_Snake.Utilities;

namespace Snake_Tests
{
  public class RandomColorSingletonHelperTests
```

```
  {
    [Test]
    public void Instance_ShouldReturnSameInstance()
    {
      var instance1 = RandomColorSingletonHelper.Instance;
      var instance2 = RandomColorSingletonHelper.Instance;

      Assert.AreSame(instance1, instance2, "Instance method should return the same singleton
instance.");
    }

    [Test]
    public void GenerateRandomColor_ShouldReturnValidHexColor()
    {
      var helper = RandomColorSingletonHelper.Instance;
      var hexColorPattern = @"^#[0-9A-Fa-f]{6}$";
      var color = helper.GenerateRandomColor();

      Assert.IsTrue(Regex.IsMatch(color, hexColorPattern),
        $"Generated color '{color}' does not match the hex color format.");
    }

    [Test]
    public void GenerateRandomColor_ShouldGenerateDifferentColors()
    {
      var helper = RandomColorSingletonHelper.Instance;

      var color1 = helper.GenerateRandomColor();
      var color2 = helper.GenerateRandomColor();

      Assert.AreNotEqual(color1, color2,
        "Two consecutive calls to GenerateRandomColor should produce different colors.");
    }

  }
}
```

ObserverTests.cs

Aprašas:

`ObserverTests` klasė skirta testuoti stebėtojo (Observer) ir subjekto (Subject) sąveiką pagal stebėtojo šabloną. Testuojamos situacijos, kai stebėtojai registruojami, pašalinami ir informuojami apie įvykusius įvykius, susijusius su "Snake" objektais.

Kodas:

```
using NUnit.Framework;
using System.Collections.Generic;
using SignalR_Snake.Models.Observer;
using SignalR_Snake.Models;

namespace Snake_Tests
{
   public class ObserverTests
   {
      public class SnakeHub : ISnakeSubject
      {
         private List<ISnakeObserver> observers = new List<ISnakeObserver>();

         public void RegisterObserver(ISnakeObserver observer)
         {
            observers.Add(observer);
         }

         public void RemoveObserver(ISnakeObserver observer)
         {
            observers.Remove(observer);
         }

         public void NotifySnakeUpdated(SignalR_Snake.Models.Snake snake)
         {
            foreach (var observer in observers)
            {
               observer.OnSnakeUpdated(snake);
            }
         }

         public void NotifySnakeDied(SignalR_Snake.Models.Snake snake)
         {
            foreach (var observer in observers)
            {
               observer.OnSnakeDied(snake);
            }
         }
      }

      public class SnakeObserver : ISnakeObserver
      {
         public bool WasSnakeUpdatedCalled { get; private set; }
         public bool WasSnakeDiedCalled { get; private set; }

         public void Reset()
         {
            WasSnakeUpdatedCalled = false;
            WasSnakeDiedCalled = false;
```

```
    }

    void ISnakeObserver.OnSnakeUpdated(SignalR_Snake.Models.Snake snake)
    {
        WasSnakeUpdatedCalled = true;
    }

    void ISnakeObserver.OnSnakeDied(SignalR_Snake.Models.Snake snake)
    {
        WasSnakeDiedCalled = true;
    }
}

[TestFixture]
public class SnakeHubTests
{
    private SnakeHub _snakeHub;
    private SnakeObserver _observer;

    [SetUp]
    public void SetUp()
    {
        _snakeHub = new SnakeHub();
        _observer = new SnakeObserver();
    }

    [Test]
    public void RegisterObserver_ShouldNotifyOnSnakeUpdated()
    {
        var snake = new Snake { Name = "Snek"};
        _snakeHub.RegisterObserver(_observer);

        _snakeHub.NotifySnakeUpdated(snake);

        Assert.IsTrue(_observer.WasSnakeUpdatedCalled);
    }

    [Test]
    public void RegisterObserver_ShouldNotifyOnSnakeDied()
    {
        var snake = new Snake { Name = "Snek"};
        _snakeHub.RegisterObserver(_observer);

        _snakeHub.NotifySnakeDied(snake);

        Assert.IsTrue(_observer.WasSnakeDiedCalled);
    }

    [Test]
    public void RemoveObserver_ShouldNotNotifyAfterRemoval()
    {
```

```
            var snake = new Snake { Name = "Snek"};
            _snakeHub.RegisterObserver(_observer);
            _snakeHub.RemoveObserver(_observer);

            _snakeHub.NotifySnakeUpdated(snake);

            Assert.IsFalse(_observer.WasSnakeUpdatedCalled);
        }

        [Test]
        public void NotifySnakeUpdated_ShouldNotifyAllObservers()
        {
            var snake = new Snake { Name = "Snek" };
            var observer2 = new SnakeObserver();
            _snakeHub.RegisterObserver(_observer);
            _snakeHub.RegisterObserver(observer2);

            _snakeHub.NotifySnakeUpdated(snake);

            Assert.IsTrue(_observer.WasSnakeUpdatedCalled);
            Assert.IsTrue(observer2.WasSnakeUpdatedCalled);
        }

        [Test]
        public void NotifySnakeDied_ShouldNotifyAllObservers()
        {
            var snake = new Snake { Name = "Snek" };
            var observer2 = new SnakeObserver();
            _snakeHub.RegisterObserver(_observer);
            _snakeHub.RegisterObserver(observer2);

            _snakeHub.NotifySnakeDied(snake);

            Assert.IsTrue(_observer.WasSnakeDiedCalled);
            Assert.IsTrue(observer2.WasSnakeDiedCalled);
        }
    }
  }
}
```

HomeControllerTests.cs

Aprašas:

Testų tikslas yra patikrinti `HomeController` kontrolerio veikimą programoje. Tikrinama, ar `Index` metodo grąžinamas vaizdas (`ViewResult`) yra teisingai sukurtas ir grąžina numatytą vaizdo pavadinimą. Šie testai padeda įsitikinti, kad kontrolerio metodai veikia tvarkingai.

Kodas:

```
using System.Web.Mvc;
using NUnit.Framework;
using SignalR_Snake.Controllers;
using SignalR_Snake.Models;

namespace Snake_Tests
{
    public class HomeControllerTests
    {

        private HomeController _controller;

        [SetUp]
        public void SetUp()
        {
            _controller = new HomeController();
        }

        [Test]
        public void Index_Get_Returns_View()
        {
            var result = _controller.Index() as ViewResult;

            Assert.IsNotNull(result);
            Assert.AreEqual("Index", result.ViewName);
        }
    }

    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View("Index");
        }

        [HttpPost]
        public ActionResult Index(Snake snake)
        {
            return RedirectToAction("Snek", "Home", snake);
        }

        public ActionResult Snek(Snake snake)
        {
            return View(snake);
        }
    }
}
```

FieldTests.cs :

Aprašas:

`FieldTests` klasė patikrina `Field` objekto savybių (`Width` ir `Height`) pradinį inicijavimą ir jų tinkamą nustatymą bei gavimą. Testai užtikrina, kad `Field` objektas būtų teisingai inicializuotas pagal numatytas reikšmes ir kad `Width` bei `Height` savybės galėtų būti tinkamai nustatomos ir skaitomos.

Kodas:

```
using NUnit.Framework;
using SignalR_Snake.Models;

namespace SignalR_Snake.Tests
{
   [TestFixture]
   public class FieldTests
   {
      [Test]
      public void DefaultConstructor_ShouldInitializeWidthTo3000()
      {
         var field = new Field();

         Assert.AreEqual(3000, field.Width);
      }

      [Test]
      public void DefaultConstructor_ShouldInitializeHeightTo3000()
      {
         var field = new Field();

         Assert.AreEqual(3000, field.Height);
      }

      [Test]
      public void Width_ShouldAllowSettingAndGettingValue()
      {
         var field = new Field();
         var newWidth = 4000;

         field.Width = newWidth;

         Assert.AreEqual(newWidth, field.Width);
      }

      [Test]
      public void Height_ShouldAllowSettingAndGettingValue()
      {
```

```
        var field = new Field();
        var newHeight = 4000;

        field.Height = newHeight;
        Assert.AreEqual(newHeight, field.Height);
    }
  }
}
```

SnakeHubTests.cs :

Aprašas:

`SnakeHubTests` klasė skirta testuoti `SnakeHub` komponento funkcionalumą ir patikrinti, kaip jis sąveikauja su programos vartotojais, stebėtojais (Observers) bei kitais susijusiais komponentais. Naudojant `Moq` biblioteka, testai apima `SnakeHub` funkcijas, tokias kaip naujų gyvačių registravimas, stebėtojų informavimas apie įvykius, greičio keitimas, krypties atnaujinimas ir taškų suvestinės siuntimas klientams.

Kodas:

```
using Moq;
using NUnit.Framework;
using SignalR_Snake.Hubs;
using SignalR_Snake.Models;
using System.Collections.Generic;
using Microsoft.AspNet.SignalR.Hubs;
using SignalR_Snake.Models.Observer;
using System.Drawing;

namespace SnakeHubTests
{
  [TestFixture]
  public class SnakeHubTests
  {
    private SnakeHub _snakeHub;
    private Mock<IHubCallerConnectionContext<dynamic>> _mockClients;
    private Mock<HubCallerContext> _mockContext;

    [SetUp]
    public void SetUp()
    {
      _snakeHub = new SnakeHub();
      _mockClients = new Mock<IHubCallerConnectionContext<dynamic>>();
      _mockContext = new Mock<HubCallerContext>();

      _snakeHub.Clients = _mockClients.Object;
      _snakeHub.Context = _mockContext.Object;
    }
```

```
[TearDown]
public void TearDown()
{
    SnakeHub.Sneks.Clear();
    SnakeHub.Foods.Clear();
}


[Test]
public void NewSnek_ShouldAddNewSnake()
{
    string snakeName = "TestSnake";
    _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");

    _snakeHub.NewSnek(snakeName);

    Assert.AreEqual(1, SnakeHub.Sneks.Count);
    Assert.AreEqual(snakeName, SnakeHub.Sneks[0].Name);
    Assert.AreEqual("test-connection-id", SnakeHub.Sneks[0].ConnectionId);
}

[Test]
public void RegisterObserver_ShouldAddObserver()
{
    var observerMock = new Mock<ISnakeObserver>();


    _snakeHub.RegisterObserver(observerMock.Object);

    Assert.Contains(observerMock.Object, _snakeHub.observers);
}

[Test]
public void RemoveObserver_ShouldRemoveObserver()
{
    var observerMock = new Mock<ISnakeObserver>();
    _snakeHub.RegisterObserver(observerMock.Object);


    _snakeHub.RemoveObserver(observerMock.Object);

    Assert.IsFalse(_snakeHub.observers.Contains(observerMock.Object));
}

[Test]
public void NotifySnakeUpdated_ShouldNotifyAllObservers()
{
    var observerMock = new Mock<ISnakeObserver>();
    _snakeHub.RegisterObserver(observerMock.Object);
    var snake = new Snake { Name = "TestSnake" };
```

```
        _snakeHub.NotifySnakeUpdated(snake);

        observerMock.Verify(o => o.OnSnakeUpdated(snake), Times.Once);
    }

    public interface IPositionClient
    {
        void AllPos(List<SnekPart> snakeParts, Point myPoint, List<Food> foods);
    }

    [Test]
    public void AllPos_ShouldSendAllPositionsToCaller()
    {
        string snakeName = "TestSnake";

        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek(snakeName);

        var mockCaller = new Mock<IPositionClient>();
        _mockClients.Setup(clients => clients.Caller).Returns(mockCaller.Object);
        _snakeHub.Clients = _mockClients.Object;

        _snakeHub.AllPos();

        mockCaller.Verify(c => c.AllPos(It.IsAny<List<SnekPart>>(), It.IsAny<Point>(),
It.IsAny<List<Food>>()), Times.Once);
    }


    [Test]
    public void SendDir_ShouldUpdateSnakeDirection()
    {
        string snakeName = "TestSnake";
        double newDirection = 90;

        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek(snakeName);

        _snakeHub.SendDir(newDirection);

        Assert.AreEqual(newDirection, SnakeHub.Sneks[0].Dir);
    }


    [Test]
    public void Speed_ShouldToggleSnakeSpeed()
```

```
    {
        string snakeName = "TestSnake";

        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek(snakeName);

        bool initialSpeed = SnakeHub.Sneks[0].Fast;

        _snakeHub.Speed();

        Assert.AreNotEqual(initialSpeed, SnakeHub.Sneks[0].Fast);
    }



    public interface IScoreClient
    {
        void Score(IEnumerable<SnekScore> scores);
    }

    [Test]
    public void Score_ShouldSendOrderedScoresToCaller()
    {
        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id-1");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek("Snake1");

        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id-2");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek("Snake2");

        SnakeHub.Sneks[0].Parts.Add(new SnekPart());

        var mockCaller = new Mock<IScoreClient>();
        _mockClients.Setup(clients => clients.Caller).Returns(mockCaller.Object);
        _snakeHub.Clients = _mockClients.Object;

        _snakeHub.Score();

        mockCaller.Verify(c => c.Score(It.IsAny<IEnumerable<SnekScore>>()), Times.Once);
    }


  }
}
```

FoodTests.cs :

Aprašas:

`FoodTests` klasė skirta testuoti `Food` modelio objektų savybes (`Position` ir `Color`). Testai tikrina, ar savybės tinkamai nustatomos ir gaunamos, taip pat ar `Food` objektas inicializuojamas su numatytomis reikšmėmis naudojant numatytąjį konstruktorių.

Kodas:

```
using NUnit.Framework;
using SignalR_Snake.Models;
using System.Drawing;

namespace SignalR_Snake.Tests
{
  [TestFixture]
  public class FoodTests
  {
    [Test]
    public void PositionProperty_ShouldSetAndGetCorrectly()
    {
      var food = new Food();
      var expectedPosition = new Point(50, 100);

      food.Position = expectedPosition;

      Assert.AreEqual(expectedPosition, food.Position);
      Assert.AreEqual(50, food.Position.X);
      Assert.AreEqual(100, food.Position.Y);
    }

    [Test]
    public void ColorProperty_ShouldSetAndGetCorrectly()
    {
      var food = new Food();
      var expectedColor = "Red";

      food.Color = expectedColor;

      Assert.AreEqual(expectedColor, food.Color);
    }

    [Test]
    public void DefaultConstructor_ShouldInitializePropertiesToNullOrDefault()
    {
      var food = new Food();

      Assert.IsNull(food.Color);
      Assert.AreEqual(new Point(0, 0), food.Position);
    }
```

```
    }
}
```

SnekPartTest.cs :

Aprašas:

Šis `SnekPartTests` testų rinkinys patikrina `SnekPart` klasės savybių priskyrimą ir numatytąsias reikšmes. Testai apima `Position`, `Color` ir `Name` savybių nustatymą ir gavimą, taip pat tikrina, ar konstruktorius inicializuoja `Position` į `Point(0,0)`, o `Color` ir `Name` nustato į `null`.

Kodas:

```
using NUnit.Framework;
using SignalR_Snake.Models;
using System.Drawing;

namespace SignalR_Snake.Tests
{
    [TestFixture]
    public class SnekPartTests
    {
        [Test]
        public void PositionProperty_ShouldSetAndGetCorrectly()
        {
            var snekPart = new SnekPart();
            var expectedPosition = new Point(10, 20);

            snekPart.Position = expectedPosition;

            Assert.AreEqual(expectedPosition, snekPart.Position);
            Assert.AreEqual(10, snekPart.Position.X);
            Assert.AreEqual(20, snekPart.Position.Y);
        }

        [Test]
        public void ColorProperty_ShouldSetAndGetCorrectly()
        {
            var snekPart = new SnekPart();
            var expectedColor = "Blue";

            snekPart.Color = expectedColor;

            Assert.AreEqual(expectedColor, snekPart.Color);
        }

        [Test]
        public void NameProperty_ShouldSetAndGetCorrectly()
        {
```

```
        var snekPart = new SnekPart();
        var expectedName = "SnekHead";

        snekPart.Name = expectedName;

        Assert.AreEqual(expectedName, snekPart.Name);
      }

      [Test]
      public void DefaultConstructor_ShouldInitializePropertiesToNullOrDefault()
      {
        var snekPart = new SnekPart();

        Assert.AreEqual(new Point(0, 0), snekPart.Position);
        Assert.IsNull(snekPart.Color);
        Assert.IsNull(snekPart.Name);
      }
    }
}
```

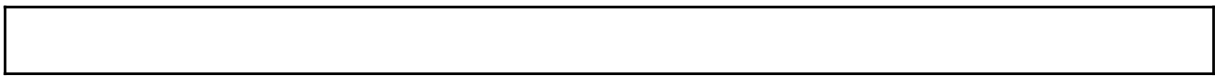
SnekScoreTests.cs :

Aprašas:

Šis `SnekScoreTests` testų rinkinys tikrina `SnekScore` klasės savybių priskyrimą ir numatytąsias reikšmes. Testai apima `SnakeName` ir `Length` savybių nustatymą bei gavimą, taip pat tikrina, ar konstruktorius inicializuoja `SnakeName` į `null` ir `Length` į `0`.

Kodas:

```
using NUnit.Framework;
using SignalR_Snake.Models;

namespace SignalR_Snake.Tests
{
    [TestFixture]
    public class SnekScoreTests
    {
        [Test]
        public void SnakeNameProperty_ShouldSetAndGetCorrectly()
        {
            var snekScore = new SnekScore();
            var expectedName = "TestSnake";

            snekScore.SnakeName = expectedName;

            Assert.AreEqual(expectedName, snekScore.SnakeName);
        }

        [Test]
        public void LengthProperty_ShouldSetAndGetCorrectly()
        {
            var snekScore = new SnekScore();
            var expectedLength = 5;

            snekScore.Length = expectedLength;

            Assert.AreEqual(expectedLength, snekScore.Length);
        }

        [Test]
        public void DefaultConstructor_ShouldInitializePropertiesToNullOrDefault()
        {
            var snekScore = new SnekScore();

            Assert.IsNull(snekScore.SnakeName);
            Assert.AreEqual(0, snekScore.Length);
        }
    }
}
```



SnakeTests.cs :

Aprašas:

Šis `SnakeTests` testų rinkinys tikrina `Snake` klasės savybių priskyrimą, numatytąsias reikšmes ir judėjimo strategijos keitimą. Testai apima `Width`, `Fast`, `Speed`, `Dir`, `SpeedTwo` ir `MovementStrategy` pradinį nustatymą konstruktoriuje, taip pat savybių `Name`, `ConnectionId`, `Color` ir `Parts` nustatymą ir gavimą. Be to, testai tikrina, kaip `ToggleMovementStrategy` metodas keičia `MovementStrategy` pagal `Fast` reikšmę, įskaitant perjungimą tarp `NormalMovementStrategy` ir `BoostMovementStrategy`.

Kodas:

```
using NUnit.Framework;
using SignalR_Snake.Models;
using SignalR_Snake.Models.Strategies;
using System.Collections.Generic;
using System.Drawing;

namespace SignalR_Snake.Tests
{
    [TestFixture]
    public class SnakeTests
    {
        [Test]
        public void Constructor_ShouldInitializePropertiesWithDefaultValues()
        {
            var snake = new Snake();

            Assert.AreEqual(10, snake.Width);
            Assert.IsFalse(snake.Fast);
            Assert.AreEqual(4, snake.Speed);
            Assert.AreEqual(5, snake.Dir);
            Assert.AreEqual(8, snake.SpeedTwo);
            Assert.IsInstanceOf<NormalMovementStrategy>(snake.MovementStrategy);
        }

        [Test]
        public void NameProperty_ShouldSetAndGetCorrectly()
        {
            var snake = new Snake();
            var expectedName = "TestSnake";

            snake.Name = expectedName;

            Assert.AreEqual(expectedName, snake.Name);
        }

        [Test]
        public void ConnectionIdProperty_ShouldSetAndGetCorrectly()
```

```
    {
        var snake = new Snake();
        var expectedConnectionId = "test-connection-id";

        snake.ConnectionId = expectedConnectionId;

        Assert.AreEqual(expectedConnectionId, snake.ConnectionId);
    }

    [Test]
    public void ColorProperty_ShouldSetAndGetCorrectly()
    {
        var snake = new Snake();
        var expectedColor = "Green";

        snake.Color = expectedColor;

        Assert.AreEqual(expectedColor, snake.Color);
    }

    [Test]
    public void PartsProperty_ShouldSetAndGetCorrectly()
    {
        var snake = new Snake();
        var parts = new List<SnekPart> { new SnekPart { Position = new Point(0, 0), Color = "Green" }
};

        snake.Parts = parts;

        Assert.AreEqual(parts, snake.Parts);
        Assert.AreEqual(1, snake.Parts.Count);
        Assert.AreEqual("Green", snake.Parts[0].Color);
    }

    [Test]
    public void
ToggleMovementStrategy_ShouldSwitchToBoostMovementStrategy_WhenFastIsTrue()
    {
        var snake = new Snake();
        snake.Fast = true;

        snake.ToggleMovementStrategy();

        Assert.IsInstanceOf<BoostMovementStrategy>(snake.MovementStrategy);
    }

    [Test]
    public void
ToggleMovementStrategy_ShouldSwitchToNormalMovementStrategy_WhenFastIsFalse()
    {
```

```
        var snake = new Snake();
        snake.Fast = false;

        snake.ToggleMovementStrategy();

        Assert.IsInstanceOf<NormalMovementStrategy>(snake.MovementStrategy);
    }

    [Test]
    public void
ToggleMovementStrategy_ShouldRetainBoostMovementStrategy_WhenCalledMultipleTimesAnd
FastIsTrue()
    {
        var snake = new Snake();
        snake.Fast = true;

        snake.ToggleMovementStrategy();
        snake.ToggleMovementStrategy();

        Assert.IsInstanceOf<BoostMovementStrategy>(snake.MovementStrategy);
    }

    [Test]
    public void
ToggleMovementStrategy_ShouldSwitchBackToNormal_WhenFastToggledToFalse()
    {
        var snake = new Snake();
        snake.Fast = true;

        snake.ToggleMovementStrategy();
        snake.Fast = false;
        snake.ToggleMovementStrategy();

        Assert.IsInstanceOf<NormalMovementStrategy>(snake.MovementStrategy);
    }
  }
}
```

IntegrationTests.cs :

Aprašas:

`SnakeHubIntegrationTests` klasė skirta testuoti `SnakeHub` komponento funkcionalumą ir integraciją `SignalR_Snake` sistemoje. Naudojant `Moq` biblioteką, atliekami įvairūs testai, siekiant patikrinti, kaip `SnakeHub` valdo gyvates (sneks), tvarko jų pozicijas, greičio būsenas, kryptis ir taškus, taip pat sąveikauja su vartotojais ir praneša jiems apie svarbius įvykius.

Kodas:

```
using Moq;
using NUnit.Framework;
using SignalR_Snake.Hubs;
using SignalR_Snake.Models;
using Microsoft.AspNet.SignalR.Hubs;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using SignalR_Snake.Models.Strategies;

namespace SignalR_Snake.Tests
{
    public interface IScoreClient
    {
        void Score(IEnumerable<SnekScore> scores);
    }

    [TestFixture]
    public class SnakeHubIntegrationTests
    {
        private SnakeHub _snakeHub;
        private Mock<IHubCallerConnectionContext<dynamic>> _mockClients;
        private Mock<HubCallerContext> _mockContext;

        [SetUp]
        public void SetUp()
        {
            _snakeHub = new SnakeHub();
            _mockClients = new Mock<IHubCallerConnectionContext<dynamic>>();
            _mockContext = new Mock<HubCallerContext>();

            _snakeHub.Clients = _mockClients.Object;
            _snakeHub.Context = _mockContext.Object;
        }

        [Test]
        public void AddSnake_ShouldExistInHubSneksList()
        {
            string snakeName = "TestSnake";
            _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
```

```
        _snakeHub.NewSnek(snakeName);

        Assert.IsTrue(SnakeHub.Sneks.Exists(s => s.Name == snakeName && s.ConnectionId ==
"test-connection-id"));
    }

    [Test]
    public void ToggleSpeed_ShouldUpdateMovementStrategy()
    {
        string snakeName = "TestSnake";
        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek(snakeName);
        var snake = SnakeHub.Sneks.Find(s => s.Name == snakeName);

        Assert.IsInstanceOf<NormalMovementStrategy>(snake.MovementStrategy, "Expected
NormalMovementStrategy when Fast is false.");


        snake.Fast = true;
        snake.ToggleMovementStrategy();
        Assert.IsInstanceOf<BoostMovementStrategy>(snake.MovementStrategy, "Expected
BoostMovementStrategy when Fast is true.");

        snake.Fast = false;
        snake.ToggleMovementStrategy();
        Assert.IsInstanceOf<NormalMovementStrategy>(snake.MovementStrategy, "Expected
NormalMovementStrategy when Fast is false.");
    }



    [Test]
    public void Score_ShouldReturnOrderedScoresByLength()
    {
        _snakeHub.NewSnek("Snake1");
        _snakeHub.NewSnek("Snake2");

        var snake1 = SnakeHub.Sneks.Find(s => s.Name == "Snake1");
        snake1.Parts.Add(new SnekPart());

        var mockCaller = new Mock<IScoreClient>();
        _mockClients.Setup(clients => clients.Caller).Returns(mockCaller.Object);

        _snakeHub.Score();


        mockCaller.Verify(c => c.Score(It.Is<IEnumerable<SnekScore>>(scores =>
            scores.First().SnakeName == "Snake1" &&
```

```
            scores.First().Length == snake1.Parts.Count
        )), Times.Once);
    }

    [Test]
    public void AllPos_ShouldSendAllPositionsToCaller()
    {
        string snakeName = "TestSnake";
        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.NewSnek(snakeName);

        var mockCaller = new Mock<IPositionClient>();
        _mockClients.Setup(clients => clients.Caller).Returns(mockCaller.Object);

        _snakeHub.AllPos();

        mockCaller.Verify(c => c.AllPos(It.IsAny<List<SnekPart>>(), It.IsAny<Point>(),
It.IsAny<List<Food>>()), Times.Once);
    }

    [Test]
    public void SendDir_ShouldUpdateSnakeDirection()
    {

        string snakeName = "TestSnake";
        double newDirection = 90;

        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.Context = _mockContext.Object;

        _snakeHub.NewSnek(snakeName);


        _snakeHub.SendDir(newDirection);

        var snake = SnakeHub.Sneks.Find(s => s.ConnectionId == "test-connection-id");
        Assert.IsNotNull(snake, "Snake should exist in the Sneks list.");
        Assert.AreEqual(newDirection, snake.Dir);
    }


    [Test]
    public void Speed_ShouldToggleSnakeSpeed()
    {
        string snakeName = "TestSnake";
        _mockContext.Setup(c => c.ConnectionId).Returns("test-connection-id");
        _snakeHub.NewSnek(snakeName);
        var snake = SnakeHub.Sneks.Find(s => s.Name == snakeName);
        bool initialSpeed = snake.Fast;

        _snakeHub.Speed();
```

```
            Assert.AreNotEqual(initialSpeed, snake.Fast);
        }
    }

    public interface IPositionClient
    {
        void AllPos(List<SnekPart> snakeParts, Point myPoint, List<Food> foods);
    }
}
```

BundleConfigTests.cs :

Aprašas:

`BundleConfigTests` klasė patikrina, ar teisingai užregistruotas kiekvienas `Bundle`. Kiekvienas testas kviečia `BundleConfig.RegisterBundles`, kad sukonfigūruotų `BundleCollection`, tada patikrina kiekvieno `Bundle` tipą, ar teisingai sukurtas buvo objektas ir jo transformacijas.

Kodas:

```
using NUnit.Framework;
using System.Web.Optimization;
using SignalR_Snake;
using System.Linq;

namespace SignalR_Snake.Tests
{
    [TestFixture]
    public class BundleConfigTests
    {
        [SetUp]
        public void SetUp()
        {
            BundleTable.Bundles.Clear(); // Clear any previous bundles to ensure test isolation
        }

        [Test]
        public void RegisterBundles_ShouldAddJQueryBundle()
        {
            var bundles = new BundleCollection();

            BundleConfig.RegisterBundles(bundles);

            var jqueryBundle = bundles.GetBundleFor("~/bundles/jquery");
            Assert.IsNotNull(jqueryBundle, "jQuery bundle should be registered.");
            Assert.IsTrue(jqueryBundle is ScriptBundle, "jQuery bundle should be a ScriptBundle.");
            Assert.IsTrue(jqueryBundle.Transforms.Any(transform => transform is JsMinify), "jQuery
bundle should use JsMinify.");
        }
```

```
    [Test]
    public void RegisterBundles_ShouldAddJQueryValidationBundle()
    {
        var bundles = new BundleCollection();

        BundleConfig.RegisterBundles(bundles);

        var jqueryValBundle = bundles.GetBundleFor("~/bundles/jqueryval");
        Assert.IsNotNull(jqueryValBundle, "jQuery validation bundle should be registered.");
        Assert.IsTrue(jqueryValBundle is ScriptBundle, "jQuery validation bundle should be a
ScriptBundle.");
        Assert.IsTrue(jqueryValBundle.Transforms.Any(transform => transform is JsMinify), "jQuery
validation bundle should use JsMinify.");
    }

    [Test]
    public void RegisterBundles_ShouldAddModernizrBundle()
    {
        var bundles = new BundleCollection();

        BundleConfig.RegisterBundles(bundles);

        var modernizrBundle = bundles.GetBundleFor("~/bundles/modernizr");
        Assert.IsNotNull(modernizrBundle, "Modernizr bundle should be registered.");
        Assert.IsTrue(modernizrBundle is ScriptBundle, "Modernizr bundle should be a
ScriptBundle.");
    }

    [Test]
    public void RegisterBundles_ShouldAddBootstrapBundle()
    {
        var bundles = new BundleCollection();

        BundleConfig.RegisterBundles(bundles);

        var bootstrapBundle = bundles.GetBundleFor("~/bundles/bootstrap");
        Assert.IsNotNull(bootstrapBundle, "Bootstrap bundle should be registered.");
        Assert.IsTrue(bootstrapBundle is ScriptBundle, "Bootstrap bundle should be a
ScriptBundle.");
    }

    [Test]
    public void RegisterBundles_ShouldAddCssBundle()
    {
        var bundles = new BundleCollection();

        BundleConfig.RegisterBundles(bundles);

        var cssBundle = bundles.GetBundleFor("~/Content/css");
        Assert.IsNotNull(cssBundle, "CSS bundle should be registered.");
```

```
            Assert.IsTrue(cssBundle is StyleBundle, "CSS bundle should be a StyleBundle.");
        }
    }
}
```

FilterConfigTests.cs :

Aprašas:

Testų klasė `FilterConfigTests`, užtikrina, kad `RegisterGlobalFilters` prideda `HandleErrorAttribute` į pateiktą `GlobalFilterCollection`.

Kodas:

```
using NUnit.Framework;
using System.Web.Mvc;
using System.Linq;
using SignalR_Snake;

namespace SignalR_Snake.Tests
{
    [TestFixture]
    public class FilterConfigTests
    {
        [Test]
        public void RegisterGlobalFilters_AddsHandleErrorAttribute()
        {
            var filters = new GlobalFilterCollection();

            FilterConfig.RegisterGlobalFilters(filters);

            Assert.That(filters.Count, Is.EqualTo(1), "The filters collection should contain exactly one filter.");
            Assert.That(filters.Any(f => f.Instance is HandleErrorAttribute), "The filter should be of type HandleErrorAttribute.");
        }
    }
}
```

RouteConfigTests.cs :

Aprašas:

`RouterConfigTests` klasė užtikrinama, kad maršrutas ({controller}/{action}/{id}) būtų atvaizduojamas teisingai. Be to, ši testų klasė užtikrina, kad `RouterConfig` ignoruotų .axd failų formatus.

Kodas:

```
using NUnit.Framework;
using Moq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
using SignalR_Snake;

namespace SignalR_Snake.Tests
{
   [TestFixture]
   public class RouteConfigTests
   {
      [SetUp]
      public void SetUp()
      {
         RouteTable.Routes.Clear();
      }

      [Test]
      public void RegisterRoutes_ShouldIgnoreAxdRequests()
      {
         var routes = new RouteCollection();
         RouteConfig.RegisterRoutes(routes);

         // Use the correct format with leading slash ('/')
         var context = new Mock<HttpContextBase>();
         context.Setup(c =>
c.Request.AppRelativeCurrentExecutionFilePath).Returns("/resource.axd/somepath");

         RouteData routeData = routes.GetRouteData(context.Object);

         // Assert that no route matches the .axd path
         Assert.IsNull(routeData, "The .axd path should be ignored by the routing configuration.");
      }


      [Test]
      public void RegisterRoutes_DefaultRoute_ShouldMapToHomeIndex()
      {
         var routes = new RouteCollection();
         RouteConfig.RegisterRoutes(routes);

         var context = new Mock<HttpContextBase>();
         context.Setup(c => c.Request.AppRelativeCurrentExecutionFilePath).Returns("~/");

         RouteData routeData = routes.GetRouteData(context.Object);

         Assert.IsNotNull(routeData, "The default route should match the root URL.");
         Assert.AreEqual("Home", routeData.Values["controller"], "Default controller should be
```

```
'Home'.");
            Assert.AreEqual("Index", routeData.Values["action"], "Default action should be 'Index'.");
            Assert.AreEqual(UrlParameter.Optional, routeData.Values["id"], "Default id should be
optional.");
        }
    }
}
```

Išvados

Laboratorinio darbo metu buvo atlikta automatizuota SignalR_Snake programos vienetų ir integracinių testų analizė, siekiant įvertinti kodo kokybę ir aptikti galimas klaidas. Pateikiame apibendrintas išvadas:

1. **Vienetinių testų taikymas**:
    - Sukurti testai apima pagrindinius programos komponentus, tokius kaip judėjimo strategijos, spalvų generavimo mechanizmus, kontrolerius ir stebėtojo (Observer) šabloną. Tai leido detaliai patikrinti kiekvienos funkcijos veikimą ir užtikrinti, kad komponentai atlieka numatytas užduotis pagal specifikaciją.
2. **Dizaino šablonų veikimo patikra**:
    - Buvo patikrinti Singleton ir Observer dizaino šablonai. **Singletono** šablono testai įrodė, kad `RandomColorSingletonHelper` klasė grąžina tą pačią instanciją visose situacijose, o tai leidžia efektyviai valdyti atminties resursus. **Observer** šablono testai parodė, kad stebėtojai tinkamai registruojami, informuojami apie įvykius ir pašalinami, kai to reikia, kas užtikrina tinkamą realaus laiko informacijos sklaidą tarp komponentų.
3. **Programos funkcionalumo stabilumas**:
    - Atlikti testai patvirtino, kad kritiniai programos komponentai (pvz., judėjimo strategijos, gyvačių pranešimų sistema) veikia pagal numatytą logiką, net esant skirtingoms įėjimo sąlygoms (kryptims, greičiams ir kt.). Tai padidina pasitikėjimą programos stabilumu ir veikimo tikslumu realioje aplinkoje.
4. **Integracinių testų nauda**:
    - Integraciniai testai `SnakeHubTests`, padėjo patikrinti komponentų sąveiką su išorinėmis sistemomis (pvz., SignalR klientais). Tai įrodė, kad sistema gali efektyviai perduoti duomenis tarp serverio ir klientų, o gyvačių atnaujinimai vykdomi greitai ir be klaidų.
5. **Testų apimtis ir kokybės įvertinimas**:
    - Sukurti testai užtikrino didelę kodo apimtį, patikrinant tiek paprastus metodus, tiek sudėtingas veiksmų sekas. Nustatyta, kad didžioji dalis pagrindinio funkcionalumo buvo patikrinta automatizuotai, kas leidžia aptikti potencialias klaidas ankstyvame kūrimo etape ir sumažinti galimų problemų skaičių paleidimo metu.

Apibendrinant galima teigti, kad atliktas automatizuotas testavimas ženkliai prisidėjo prie programos kokybės gerinimo, aptiko klaidas ir užtikrino stabilų įvairių komponentų veikimą. Tolesnis testų plėtojimas ir papildoma apimtį padėtų dar labiau sumažinti klaidų riziką ir pagerintų sistemos patikimumą.