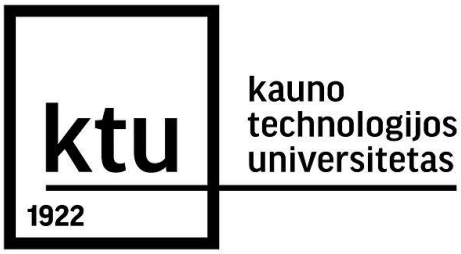


INFORMATIKOS FAKULTETAS

T120B162 Programų sistemų testavimas

2 laboratorinis darbas

Studentas: Gabija Šeškauskaitė, IFF-0/2
Dėstytojas: lekt. Dominykas Barisas

KAUNAS 2023

Turinys

1. Įvadas

2 laboratorinio darbo tikslas – ištestuoti kuriamos sistemos komponentus, sukuriant komponentų vienetų (angl. Unit) testus.

Darbo uždaviniai:

1. Susirasti bei tinkamai panaudoti testavimo priemones sistemos komponentų testavimui;
2. Išsiaiškinti, kaip aprašomi bei sukuriami vienetų testai;
3. Vienetų testais padengti 100% programos kodo.

2. Testavimo priemonės

Atsižvelgiant į kuriamos sistemos kūrimo priemones bei naudojamą karkasą, testų kūrimui pasirinkta naudotis tokias priemones:

1. Testų rašymo karkasas – xUnit;
2. Programos objektų funkcionalumo imitavimas – Moq;
3. Priemonė testų vykdymui – ReSharper;
4. Įrankis, skirtas nustatyti programos kodo padengimą testais – dotCover;

3. Testuojamos programos kodas

Žemiau pateikiamas programos kodas, kuris bus padengiamas komponentų testais.

1 lentelė. Game programos kodas

Game.cs

```
using TicTacToe.Interfaces;
using TicTacToe.Models;

namespace TicTacToe.GameObjects
{
    public class Game : IPrototype<Game>
    {
        public bool isFirstPlayersTurn;

        /// <summary>
        /// Creates a new game object.
        /// </summary>
        /// <param name="player1">The first player to join the
game.</param>
        /// <param name="player2">The second player to join the
game.</param>
        public Game(Player1Factory player1Factory, Player player1, string
roomName, int boardSize, bool obstacles)
        {
            Player1 = player1;
            GameRoomName = roomName;
            ToggleObstacles = obstacles;
            Board = BoardCreator.factoryMethod(boardSize);

            isFirstPlayersTurn = true;

            // Link the players to the game as well
            Player1.PlayingRoomName = GameRoomName;
```

```
            Player1.Piece =
player1Factory.CreatePiece(player1).ToString();
        }

        /// <summary>
        /// Creates a new game object.
        /// </summary>
        /// <param name="player1">The first player to join the
game.</param>
        /// <param name="player2">The second player to join the
game.</param>
        public Game(GameFactory gameFactory1, GameFactory gameFactory2,
Player player1, Player player2, string roomName, int boardSize, bool
obstacles)
        {
            Player1 = player1;
            Player2 = player2;
            GameRoomName = roomName;
            ToggleObstacles = obstacles;
            Board = BoardCreator.factoryMethod(boardSize);

            isFirstPlayersTurn = true;

            // Link the players to the game as well
            Player1.PlayingRoomName = GameRoomName;
            Player1.Piece = gameFactory1.CreatePiece(player1).ToString();

            if (Player2 != null)
            {
                Player2.PlayingRoomName = GameRoomName;
                Player2.Piece =
gameFactory2.CreatePiece(player2).ToString(); ;
            }
        }

        /// <summary>
        /// A unique identifier for this game.
        /// </summary>
        //public string Id { get; set; }

        /// <summary>
        /// Game room name identifier.
        /// </summary>
        public string GameRoomName { get; set; }

        /// <summary>
        /// One of two partipants of the game.
        /// </summary>
        public Player? Player1 { get; set; }

        /// <summary>
        /// One of two participants of the game.
        /// </summary>
        public Player? Player2 { get; set; }

        /// <summary>
        /// The board that represents the tic-tac-toe game.
        /// </summary>
        public Board Board { get; set; }

        /// <summary>
        /// Checks if obstacles are on for that game
        /// </summary>
        public bool ToggleObstacles { get; set; }
```

```
        /// <summary>
        /// Returns which player is currently allowed to place a piece
down.
        /// </summary>
        public Player WhoseTurn
        {
            get
            {
                return isFirstPlayersTurn ? Player1 : Player2;
            }
        }

        /// <summary>
        /// Returns whether the game is ongoing or has completed.
        /// Over states include either a tie or a player has won.
        /// </summary>
        public bool IsOver
        {
            get
            {
                return IsTie || Board.GameEnded;
            }
        }

        /// <summary>
        /// Returns whether the game is a tie.
        /// </summary>
        public bool IsTie
        {
            get
            {
                return !Board.AreSpacesLeft;
            }
        }

        /// <summary>
        /// Places a piece on the board. The game knows whose turn it is
so no need
        /// to identify the player. Will also update whose turn it is.
        /// </summary>
        /// <param name="row">The row where the piece will be
placed.</param>
        /// <param name="col">The column where the piece will be
placed.</param>
        public void PlacePiece(int row, int col)
        {
            string pieceToPlace = isFirstPlayersTurn ? Player1.Piece :
Player2.Piece;
            Board.PlacePiece(row, col, pieceToPlace);

            isFirstPlayersTurn = !isFirstPlayersTurn;
        }

        /// <summary>
        /// Returns whether or not the specified move is valid.
        /// A move is invalid if there is already a piece placed in the
location or
        /// the move is off the board.
        /// </summary>
        /// <param name="row">The row position of the move.</param>
        /// <param name="col">The column position of the move.</param>
        /// <returns>true if the move is valid; otherwise false.</returns>
        public bool IsValidMove(int row, int col)
```

```
        {
            // TODO: Make the board dimensions public properties
            bool cond1 = row < Board.Pieces.GetLength(0);
            bool cond2 = col < Board.Pieces.GetLength(1);
            bool cond3 = string.IsNullOrWhiteSpace(Board.Pieces[row,
col].Value);


            return
                cond1 &&
                cond2 &&
                cond3;
        }

        public override string ToString()
        {
            return string.Format("(Id={0}, Player1={1}, Player2={2},
Board={3})",
                GameRoomName, Player1, Player2, Board);
        }

        public Game ShallowCopy()
        {
            return (Game)this.MemberwiseClone();
        }

        public Game DeepCopy()
        {
            Player1Factory player1Factory = new Player1Factory();
            Player2Factory player2Factory = new Player2Factory();
            Player player1 = new Player(Player1.Name,
Player1.PlayingRoomName, Player1.Id);
            player1.Piece = this.Player1.Piece;
            Player player2 = new Player(Player2.Name,
Player2.PlayingRoomName, Player2.Id);
            player2.Piece = this.Player2.Piece;
            return new Game(player1Factory, player2Factory, player1,
player2, this.GameRoomName, this.Board.BoardSize, this.ToggleObstacles);
        }
    }
}
```

2 lentelė. Board3 programos kodas

Board3.cs

```
using TicTacToe.Models;

namespace TicTacToe.GameObjects;


public class Board3 : Board
{
    public IWinningStrategy winningStrategy;

    public Board3()
    {
        Set(3);
        winningStrategy = new ThreeByThreeWinningStrategy();
    }

    public bool IsThreeInRow
```

```
    {
        get
        {
            return winningStrategy.IsThreeInRow(Pieces);
        }
    }

    public override bool GameEnded
    {
        get
        {
            return this.IsThreeInRow;
        }
    }
}
```

3 lentelė. Board4 programos kodas

Board4.cs

```
using TicTacToe.Models;

namespace TicTacToe.GameObjects
{
    public class Board4 : Board
    {
        private IWinningStrategy winningStrategy;

        public Board4()
        {
            Set(4);
            winningStrategy = new FourByFourWinningStrategy();
        }

        public bool IsFourInRow
        {
            get
            {
                return winningStrategy.IsFourInRow(Pieces);
            }
        }

        public override bool GameEnded
        {
            get
            {
                return this.winningStrategy.IsFourInRow(Pieces);
            }
        }
    }
}
```

4 lentelė. BoardCreator programos kodas

BoardCreator.cs

```
using TicTacToe.GameObjects;

namespace TicTacToe.Models;


public class BoardCreator
{

    /// <summary>
    /// create specific board
    /// </summary>
    /// <param name="type">dimensions for the board</param>
    /// <returns></returns>
    public static Board factoryMethod(int type)
    {
        if(type == 3)
        {
            return new Board3();
        }
        else if (type == 4)
        {
            return new Board4();
        }
        else
        {
            return null; // wanted type is not implemented
        }
    }
         }
```

5 lentelė. FourByFourWinningStrategy programos kodas

FourByFourWinningStrategy.cs

```
using TicTacToe.GameObjects;

namespace TicTacToe.Models
{
    public class FourByFourWinningStrategy : IWinningStrategy
    {
        public bool IsFourInRow(Cell[,] Pieces)
        {
            // Check all rows
            for (int row = 0; row < Pieces.GetLength(0); row++)
            {
                if (Pieces[row, 0] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[row, 0].Value) &&
                    Pieces[row, 1] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[row, 1].Value) &&
                    Pieces[row, 2] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[row, 2].Value) &&
                    Pieces[row, 3] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[row, 3].Value) &&
                    Pieces[row, 0].Value == Pieces[row, 1].Value &&
                    Pieces[row, 1].Value == Pieces[row, 2].Value &&
                    Pieces[row, 2].Value == Pieces[row, 3].Value)
                {
                    return true;
                }
            }

            // Check all columns
            for (int col = 0; col < Pieces.GetLength(1); col++)
            {
                if (Pieces[0, col] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[0, col].Value) &&
                    Pieces[1, col] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[1, col].Value) &&
                    Pieces[2, col] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[2, col].Value) &&
                    Pieces[3, col] != null &&
                    !string.IsNullOrWhiteSpace(Pieces[3, col].Value) &&
                    Pieces[0, col].Value == Pieces[1, col].Value &&
                    Pieces[1, col].Value == Pieces[2, col].Value &&
                    Pieces[2, col].Value == Pieces[3, col].Value)
                {
                    return true;
                }
            }

            // Check forward-diagonal
            if (Pieces[1, 1] != null &&
                !string.IsNullOrWhiteSpace(Pieces[1, 1].Value) &&
                Pieces[2, 0] != null &&
                !string.IsNullOrWhiteSpace(Pieces[2, 0].Value) &&
                Pieces[0, 2] != null &&
                !string.IsNullOrWhiteSpace(Pieces[0, 2].Value) &&
                Pieces[3, 3] != null &&
                !string.IsNullOrWhiteSpace(Pieces[3, 3].Value) &&
                Pieces[2, 0].Value == Pieces[1, 1].Value &&
                Pieces[1, 1].Value == Pieces[0, 2].Value &&
```

```
                Pieces[0, 2].Value == Pieces[3, 3].Value)
            {
                return true;
            }

            // Check backward-diagonal
            if (Pieces[1, 2] != null &&
                !string.IsNullOrWhiteSpace(Pieces[1, 2].Value) &&
                Pieces[0, 3] != null &&
                !string.IsNullOrWhiteSpace(Pieces[0, 3].Value) &&
                Pieces[2, 1] != null &&
                !string.IsNullOrWhiteSpace(Pieces[2, 1].Value) &&
                Pieces[3, 0] != null &&
                !string.IsNullOrWhiteSpace(Pieces[3, 0].Value) &&
                Pieces[0, 3].Value == Pieces[1, 2].Value &&
                Pieces[1, 2].Value == Pieces[2, 1].Value &&
                Pieces[2, 1].Value == Pieces[3, 0].Value)
            {
                return true;
            }

            return false;
        }


        public bool IsBoardFull(Cell[,] Pieces)
        {
            for (int row = 0; row < Pieces.GetLength(0); row++)
            {
                for (int col = 0; col < Pieces.GetLength(1); col++)
                {
                    if (Pieces[row, col] == null ||
string.IsNullOrWhiteSpace(Pieces[row, col].Value))
                    {
                        return false;
                    }
                }
            }
            return true;
        }

        public bool IsGameOver(Cell[,] Pieces)
        {
            return IsFourInRow(Pieces) || IsBoardFull(Pieces);
        }
        public bool IsThreeInRow(Cell[,] pieces)
        {
            // Implement a method that always returns false for a 3x3
board.
            return false;
        }
    }
}
```

```
    {
        private readonly IOrganizationsRepository _organizationsRepository;
        private readonly IUsersRepository _usersRepository;
        private readonly IBranchUsersRepository _branchUsersRepository;

        public OrganizationsController(IOrganizationsRepository
organizationsRepository, IUsersRepository usersRepository,
IBranchUsersRepository branchUsersRepository)
        {
            _organizationsRepository = organizationsRepository;
            _usersRepository = usersRepository;
            _branchUsersRepository = branchUsersRepository;
        }
        [HttpGet]
        public async Task<IActionResult> GetOrganizationWithBranchesAsync()
        {
            var user =
_usersRepository.GetAuthenticatedUser(HttpContext.Request);
            if (user == null)
            {
                return BadRequest();
            }
            return Ok(await
_organizationsRepository.GetOrganizationWithBranches(user));
        }
        [HttpPost]
        public async Task<IActionResult>
CreateNewOrganizationAsync([FromBody] OrganizationForCreationDto
organization)
        {
            var user =
_usersRepository.GetAuthenticatedUser(HttpContext.Request);
            if (user == null)
            {
                return BadRequest();
            }
            if (organization == null)
            {
                return BadRequest();
            }
            var newBranch = Mapper.Map<Branch>(organization);
            _organizationsRepository.CreateOrganization(newBranch);
            if (!_organizationsRepository.Save())
            {
                return StatusCode(500, "A problem happened while handling
your request.");
            }
            var newBranchUser = Mapper.Map<BranchUser>(new
BranchUserForCreationDto
            {
                BranchId = newBranch.Id,
                UserId = user.Id
            });
            _branchUsersRepository.CreateBranchUser(newBranchUser);
            if (!_branchUsersRepository.Save())
            {
                return StatusCode(500, "A problem happened while handling
your request.");
            }
            return Ok(await
_organizationsRepository.GetOrganizationWithBranches(user));
        }
```

```
        [HttpPut("{organizationResourceID}")]
        public async Task<IActionResult> UpdateOrganizationAsync([FromBody]
OrganizationForCreationDto organization, string organizationResourceId)
        {
            var user =
_usersRepository.GetAuthenticatedUser(HttpContext.Request);
            if(user == null)
            {
                return BadRequest();
            }
            if (organization == null)
            {
                return BadRequest();
            }
            var organizationEntity = await
_organizationsRepository.GetOrganization(organizationResourceId);
            if (organizationEntity == null)
            {
                return NotFound();
            }
            Mapper.Map(organization, organizationEntity);
            if (!_organizationsRepository.Save())
            {
                return StatusCode(500, "A problem happened while handling
your request.");
            }
            return Ok(await
_organizationsRepository.GetOrganizationWithBranches(user));
        }
    }
}
```

6. lentelė. ThreeByThreeWinningStrategy programos kodas

ThreeByThreeWinningStrategy.cs

```
using System;
using TicTacToe.GameObjects;

namespace TicTacToe.Models
{
    public class ThreeByThreeWinningStrategy : IWinningStrategy
    {
        public bool IsThreeInRow(Cell[,] Pieces)
        {
            // Check all rows
            for (int row = 0; row < Pieces.GetLength(0); row++)
            {
                if (Pieces[row, 0] != null &&
!string.IsNullOrWhiteSpace(Pieces[row, 0].Value) &&
                    Pieces[row, 0].Value == Pieces[row, 1]?.Value &&
                    Pieces[row, 1]?.Value == Pieces[row, 2]?.Value)
                {
                    return true;
                }
            }

            // Check all columns
            for (int col = 0; col < Pieces.GetLength(1); col++)
            {
                if (Pieces[0, col] != null &&
!string.IsNullOrWhiteSpace(Pieces[0, col].Value) &&
                    Pieces[0, col].Value == Pieces[1, col]?.Value &&
```

```
                    Pieces[1, col]?.Value == Pieces[2, col]?.Value)
                {
                    return true;
                }
            }

            // Check forward-diagonal
            if (Pieces[1, 1] != null &&
!string.IsNullOrWhiteSpace(Pieces[1, 1].Value) &&
                Pieces[2, 0]?.Value == Pieces[1, 1].Value &&
                Pieces[1, 1].Value == Pieces[0, 2]?.Value)
            {
                return true;
            }

            // Check backward-diagonal
            if (Pieces[1, 1] != null &&
!string.IsNullOrWhiteSpace(Pieces[1, 1].Value) &&
                Pieces[0, 0]?.Value == Pieces[1, 1].Value &&
                Pieces[1, 1].Value == Pieces[2, 2]?.Value)
            {
                return true;
            }

            return false;
        }

        public bool IsBoardFull(Cell[,] Pieces)
        {
            for (int row = 0; row < Pieces.GetLength(0); row++)
            {
                for (int col = 0; col < Pieces.GetLength(1); col++)
                {
                    if (Pieces[row, col] == null ||
string.IsNullOrWhiteSpace(Pieces[row, col].Value))
                    {
                        return false;
                    }
                }
            }
            return true;
        }

        public bool IsGameOver(Cell[,] Pieces)
        {
            return IsThreeInRow(Pieces) || IsBoardFull(Pieces);
        }
        public bool IsFourInRow(Cell[,] Pieces)
        {
            // Implement a method that always returns false for a 3x3
board.
            return false;
        }
    }
}
```

7. lentelė. Piece programos kodas

Piece.cs

```
namespace TicTacToe.GameObjects
{
    public class Piece : Cell
    {

        public Piece(string value) : base(value)
        {
            Set(value);
        }

        public override string ToString()
        {
            return Value;
        }

        public string getStatus()
        {
            return "piece";
        }
    }
}
```

8. lentelė. Cell programos kodas

Cell.cs

```
namespace TicTacToe.GameObjects
{
    public class Cell
    {
        public string Value { get; private set; }

        /// <summary>
        /// constructor
        /// </summary>
        public Cell()
        {
            Value = "";
        }

        /// <summary>
        /// constructor
        /// </summary>
        /// <param name="value"></param>
        public Cell(string value)
        {
            Value = value;
        }

        /// <summary>
        /// sets value outside contructor
        /// </summary>
        /// <param name="value">cell value</param>
        public void Set(string value)
        {
            Value = value;
        }


        /// <summary>
        /// get status if cell is itself or it's child classes
        /// </summary>
        /// <returns>status that it's Cell class</returns>
```

```
        public string getStatus()
        {
            return "general";
        }

        /// <summary>
        /// get cell value
        /// </summary>
        /// <returns>cell value</returns>
        public override string ToString()
        {
            return Value;
        }
    }
         }
```

9. lentelė. GameSubject programoskodas

GameSubject.cs

```
using TicTacToe.GameObjects;

namespace TicTacToe.Models
{
    public class GameSubject :Subject
    {
        private bool state; //state of the game instance activity

        /// <summary>
        /// constructor
        /// </summary>
        public GameSubject() : base()
        {
            observers = new List<Obstacle>();
            state = true;
        }


        /// <summary>
        /// get state if game is still active
        /// </summary>
        /// <returns>state if game is still active</returns>
        public bool getState()
        {
            return state;
        }

        /// <summary>
        /// set state to know if game is active
        /// </summary>
        /// <param name="value">state if game is still active</param>
        public void setState(bool value)
        {
            state = value;
        }
    }
}
```

10. Pasiruošimas testavimui

Prieš aprašant testus, reikia tinkamai paruošti testavimo failų struktūrą, bei sukurti reikiamus elementus. Pirmiausia sukuriamos komponentų testų:

1 lentelė. Game testavimo klasė

GameTests.cs

```
using Moq;
using System;
using TicTacToe.GameObjects;
using TicTacToe.Models;
using Xunit;

namespace TestProject3.GameObjects
{
    public class GameTests
    {
        private MockRepository mockRepository;

        private Mock<Player1Factory> mockPlayer1Factory;
        private Mock<Player> mockPlayer;

        public GameTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);

            this.mockPlayer1Factory =
this.mockRepository.Create<Player1Factory>();
            this.mockPlayer = this.mockRepository.Create<Player>();
        }

        private Game CreateGame()
        {
            return new Game(
                this.mockPlayer1Factory.Object,
                this.mockPlayer.Object,
                TODO,
                TODO,
                TODO);
        }

        [Fact]
        public void PlacePiece_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var game = this.CreateGame();
            int row = 0;
            int col = 0;

            // Act
            game.PlacePiece(
                row,
                col);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsValidMove_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var game = this.CreateGame();
            int row = 0;
            int col = 0;
```

```
            // Act
            var result = game.IsValidMove(
                row,
                col);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void ToString_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var game = this.CreateGame();

            // Act
            var result = game.ToString();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void ShallowCopy_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var game = this.CreateGame();

            // Act
            var result = game.ShallowCopy();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void DeepCopy_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var game = this.CreateGame();

            // Act
            var result = game.DeepCopy();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

2 lentelė. Board3 testavimo klasė

Board3Tests.cs

```
using Moq;
using System;
using TicTacToe.GameObjects;
using Xunit;
```

```
namespace TestProject3.GameObjects
{
    public class Board3Tests
    {
        private MockRepository mockRepository;



        public Board3Tests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private Board3 CreateBoard3()
        {
            return new Board3();
        }

        [Fact]
        public void TestMethod1()
        {
            // Arrange
            var board3 = this.CreateBoard3();

            // Act


            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

3 lentelė. Board4 testavimo klasė

Board4Tests.cs

```
using Moq;
using System;
using TicTacToe.GameObjects;
using Xunit;

namespace TestProject3.GameObjects
{
    public class Board4Tests
    {
        private MockRepository mockRepository;



        public Board4Tests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private Board4 CreateBoard4()
        {
            return new Board4();
        }

        [Fact]
        public void TestMethod1()
        {
            // Arrange
            var board4 = this.CreateBoard4();

            // Act


            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

4 lentelė. BoardCreator testavimo klasė

BoardCreatorTests.cs

```
using Moq;
using System;
using TicTacToe.Models;
using Xunit;

namespace TestProject3.Models
{
    public class BoardCreatorTests
    {
        private MockRepository mockRepository;



        public BoardCreatorTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private BoardCreator CreateBoardCreator()
        {
            return new BoardCreator();
        }

        [Fact]
        public void factoryMethod_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var boardCreator = this.CreateBoardCreator();
            int type = 0;

            // Act
            var result = boardCreator.factoryMethod(
                type);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

5 lentelė. FourByFourWinningStrategy testavimo klasė

FourByFourWinningStrategyTests.cs

```
using Moq;
using System;
using TicTacToe.Models;
using Xunit;

namespace TestProject3.Models
{
    public class FourByFourWinningStrategyTests
    {
        private MockRepository mockRepository;



        public FourByFourWinningStrategyTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private FourByFourWinningStrategy CreateFourByFourWinningStrategy()
        {
            return new FourByFourWinningStrategy();
        }

        [Fact]
        public void IsFourInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var fourByFourWinningStrategy =
this.CreateFourByFourWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = fourByFourWinningStrategy.IsFourInRow(
                Pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsBoardFull_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var fourByFourWinningStrategy =
this.CreateFourByFourWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = fourByFourWinningStrategy.IsBoardFull(
                Pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsGameOver_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
```

```
            var fourByFourWinningStrategy =
this.CreateFourByFourWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = fourByFourWinningStrategy.IsGameOver(
                Pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsThreeInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var fourByFourWinningStrategy =
this.CreateFourByFourWinningStrategy();
            Cell[,] pieces = null;

            // Act
            var result = fourByFourWinningStrategy.IsThreeInRow(
                pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

6 lentelė. ThreeByThreeWinningStrategy testavimo klasė

ThreeByThreeWinningStrategyTests.cs

```
using Moq;
using System;
using TicTacToe.Models;
using Xunit;

namespace TestProject3.Models
{
    public class ThreeByThreeWinningStrategyTests
    {
        private MockRepository mockRepository;



        public ThreeByThreeWinningStrategyTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private ThreeByThreeWinningStrategy CreateThreeByThreeWinningStrategy()
        {
            return new ThreeByThreeWinningStrategy();
        }
```

```
        [Fact]
        public void IsThreeInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = threeByThreeWinningStrategy.IsThreeInRow(
                Pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsBoardFull_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = threeByThreeWinningStrategy.IsBoardFull(
                Pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsGameOver_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = threeByThreeWinningStrategy.IsGameOver(
                Pieces);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void IsFourInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = null;

            // Act
            var result = threeByThreeWinningStrategy.IsFourInRow(
                Pieces);

            // Assert
```

```
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

7 lentelė. Piece testavimo klasė

PieceTests.cs

```
using Moq;
using System;
using TicTacToe.GameObjects;
using Xunit;

namespace TestProject3.GameObjects
{
    public class PieceTests
    {
        private MockRepository mockRepository;



        public PieceTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private Piece CreatePiece()
        {
            return new Piece(
                TODO);
        }

        [Fact]
        public void ToString_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var piece = this.CreatePiece();

            // Act
            var result = piece.ToString();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void getStatus_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var piece = this.CreatePiece();

            // Act
            var result = piece.getStatus();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
```

```
    }
}
```

8 lentelė. Cell testavimo klasė

CellTests.cs

```
using Moq;
using System;
using TicTacToe.GameObjects;
using Xunit;

namespace TestProject3.GameObjects
{
    public class CellTests
    {
        private MockRepository mockRepository;



        public CellTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private Cell CreateCell()
        {
            return new Cell();
        }

        [Fact]
        public void Set_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var cell = this.CreateCell();
            string value = null;

            // Act
            cell.Set(
                value);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void getStatus_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var cell = this.CreateCell();

            // Act
            var result = cell.getStatus();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
```

```
        public void ToString_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var cell = this.CreateCell();

            // Act
            var result = cell.ToString();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

9 lentelė. GameSubject testavimo klasė

GameSubjectTests.cs

```
using Moq;
using System;
using TicTacToe.Models;
using Xunit;

namespace TestProject3.Models
{
    public class GameSubjectTests
    {
        private MockRepository mockRepository;



        public GameSubjectTests()
        {
            this.mockRepository = new MockRepository(MockBehavior.Strict);


        }

        private GameSubject CreateGameSubject()
        {
            return new GameSubject();
        }

        [Fact]
        public void getState_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var gameSubject = this.CreateGameSubject();

            // Act
            var result = gameSubject.getState();

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }

        [Fact]
        public void setState_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var gameSubject = this.CreateGameSubject();
            bool value = false;
```

```
            // Act
            gameSubject.setState(
                value);

            // Assert
            Assert.True(false);
            this.mockRepository.VerifyAll();
        }
    }
}
```

11. Testavimo atvejai ir jų kodas

Šiame skyriuje pateikiami aprašyti komponentų testai. Kiekvienas testas yra pateikiamas lentele, kurią sudaro 3 dalys: 1-oji nurodo komponento testo pavadinimą, antroji – ką testuoja atitinkamas testas, o trečioje lentelės dalyje pateikiamas testo kodas. Iš viso buvo parašyti 71 komponentų testas.

a. Game komponentų testai:

NewGame_ShouldHaveFirstPlayersTurnTrue
Ar naujas žaidimas pradžioje priklauso pirmajam žaidėjui.

```
[Fact]
        public void NewGame_ShouldHaveFirstPlayersTurnTrue()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player1Factory player1Factory = new Player1Factory();

            // Act
            Game game = new Game(player1Factory, player1, "Room1", 3, false);

            // Assert
            Assert.True(game.isFirstPlayersTurn);
         }
```

b. AuthController komponentų testai:

PlacePiece_ShouldAlternateTurns
Ar eilės kaita vyksta teisingai žaidžiant žaidimą.

```
[Fact]
        public void PlacePiece_ShouldAlternateTurns()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player player2 = new Player("Player2", "Room1", "2");
            Player1Factory player1Factory = new Player1Factory();
            Player2Factory player2Factory = new Player2Factory();

            // Act
            Game game = new Game(player1Factory, player2Factory, player1,
player2, "Room1", 3, false);

            // Assert
            Assert.True(game.isFirstPlayersTurn);

            // Act
            game.PlacePiece(0, 0);

            // Assert
            Assert.False(game.isFirstPlayersTurn);
```

```
            // Act
            game.PlacePiece(1, 1);

            // Assert
            Assert.True(game.isFirstPlayersTurn);
                }
```

IsValidMove_ShouldReturnTrueForValidMove

Ar teisingai nustatoma, ar ėjimas yra galiojantis.

```
[Fact]
        public void IsValidMove_ShouldReturnTrueForValidMove()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player1Factory player1Factory = new Player1Factory();

            // Act
            Game game = new Game(player1Factory, player1, "Room1", 3, false);

            // Assert
            Assert.True(game.IsValidMove(0, 0));
                }
```

IsValidMove_ShouldReturnFalseForInvalidMove

Ar teisingai nustatoma, ar ėjimas yra negaliojantis.

```
[Fact]
        public void IsValidMove_ShouldReturnFalseForInvalidMove()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player1Factory player1Factory = new Player1Factory();

            // Act
            Game game = new Game(player1Factory, player1, "Room1", 3, false);

            // Act
            game.PlacePiece(0, 0);

            // Assert
            Assert.False(game.IsValidMove(0, 0));
        }
```

ToString_ShouldReturnFormattedString

Ar gaunamas tinkamai suformatuotas tekstas.

```
    [Fact]
       public void ToString_ShouldReturnFormattedString()
       {
           // Arrange
           Player player1 = new Player("Player1", "Room1", "1");
           Player1Factory player1Factory = new Player1Factory();
           Game game = new Game(player1Factory, player1, "Room1", 3, false);

           // Act
           string result = game.ToString();

           // Assert
           Assert.Equal($"(Id={game.GameRoomName}, Player1={game.Player1},
Player2={game.Player2}, Board={game.Board})", result);
                           }
```

ShallowCopy_ShouldReturnNewInstance

Ar gautas naujas objektas, kuris nėra toks pat kaip originalas.

```
  [Fact]
       public void ShallowCopy_ShouldReturnNewInstance()
       {
           // Arrange
           Player player1 = new Player("Player1", "Room1", "1");
           Player1Factory player1Factory = new Player1Factory();
           Game game = new Game(player1Factory, player1, "Room1", 3, false);

           // Act
           Game copy = game.ShallowCopy();

           // Assert
           Assert.NotSame(game, copy);
        }
```

ToggleObstacles_ShouldReturnSetValue

Ar gauta reikšmė yra teisingai nustatoma ir gali būti nustatyta.

```
        [Fact]
        public void ToggleObstacles_ShouldReturnSetValue()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player1Factory player1Factory = new Player1Factory();
            Game game = new Game(player1Factory, player1, "Room1", 3, false);

            // Act
            game.ToggleObstacles = true;

            // Assert
            Assert.True(game.ToggleObstacles);
         }
```

WhoseTurn_ShouldReturnCorrectPlayer
Ar grąžinamas teisingas žaidėjo objektas atitinkamai jo ėjimo metu.

```
        [Fact]
        public void WhoseTurn_ShouldReturnCorrectPlayer()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player player2 = new Player("Player2", "Room1", "2");
            Player1Factory player1Factory = new Player1Factory();
            Player2Factory player2Factory = new Player2Factory();
            Game game = new Game(player1Factory, player2Factory, player1,
player2, "Room1", 3, false);

            // Act
            Player currentTurnPlayer = game.WhoseTurn;

            // Assert
            Assert.Equal(player1, currentTurnPlayer);
        }
```

IsTie_ShouldReturnTrueWhenGameIsTie
Ar teisingai nustatoma, kad žaidimas baigėsi lygiosios.

```
        [Fact]
        public void IsTie_ShouldReturnTrueWhenGameIsTie()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player player2 = new Player("Player2", "Room1", "2");
            Player1Factory player1Factory = new Player1Factory();
            Player2Factory player2Factory = new Player2Factory();
            Game game = new Game(player1Factory, player2Factory, player1,
player2, "Room1", 3, false);

            // Act
            // Simulate a tie by filling the board without a winner
            game.PlacePiece(0, 0);
            game.PlacePiece(0, 1);
```

```
            game.PlacePiece(0, 2);
            game.PlacePiece(1, 0);
            game.PlacePiece(1, 1);
            game.PlacePiece(1, 2);
            game.PlacePiece(2, 0);
            game.PlacePiece(2, 1);
            game.PlacePiece(2, 2);

            // Assert
            Assert.True(game.IsTie);
        }       }
```

DeepCopy_ShouldReturnNewInstanceWithEqualProperties
Ar gauta gyli kopija yra naujas objektas su lygiomis savybėmis.

```
   [Fact]
        public void DeepCopy_ShouldReturnNewInstanceWithEqualProperties()
        {
            // Arrange
            Player1Factory player1Factory = new Player1Factory();
            Player2Factory player2Factory = new Player2Factory();
            Player player1 = new Player("Player1", "Room1", "1");
            player1.Piece = "X";
            Player player2 = new Player("Player2", "Room1", "2");
            player2.Piece = "O";

            Game game = new Game(player1Factory, player2Factory, player1,
player2, "Room1", 3, false);

            // Act
            Game deepCopy = game.DeepCopy();

            // Assert
            Assert.NotSame(game, deepCopy);
            Assert.Equal(game.GameRoomName, deepCopy.GameRoomName);
            Assert.Equal(game.Player1.Name, deepCopy.Player1.Name);
            Assert.Equal(game.Player1.Piece, deepCopy.Player1.Piece);
            Assert.Equal(game.Player2.Name, deepCopy.Player2.Name);
            Assert.Equal(game.Player2.Piece, deepCopy.Player2.Piece);
            Assert.Equal(game.Board.BoardSize, deepCopy.Board.BoardSize);
            Assert.Equal(game.ToggleObstacles, deepCopy.ToggleObstacles);
         }
```

IsOver_ShouldReturnTrueWhenIsTie
Ar teisingai nustatoma, kad žaidimas baigėsi lygiosios.

```
    [Fact]
        public void IsOver_ShouldReturnTrueWhenIsTie()
        {
            // Arrange
            Player1Factory player1Factory = new Player1Factory();
            Player player1 = new Player("Player1", "Room1", "1");
            Game game = new Game(player1Factory, player1, "Room1", 3, false);

            // Set up conditions for a tie
            game.Board.PlacePiece(0, 0, "X");
            game.Board.PlacePiece(0, 1, "O");
            game.Board.PlacePiece(0, 2, "X");
            game.Board.PlacePiece(1, 0, "O");
            game.Board.PlacePiece(1, 1, "X");
            game.Board.PlacePiece(1, 2, "O");
            game.Board.PlacePiece(2, 0, "O");
            game.Board.PlacePiece(2, 1, "X");
            game.Board.PlacePiece(2, 2, "O");

            // Act
            bool isOver = game.IsOver;

            // Assert
            Assert.True(isOver);
        }
```

c. Board3 komponentų testai:

IsThreeInRow_StateUnderTest_ExpectedBehavior
Ar teisingai nustatoma, ar yra trys iš eilės.

```
        [Fact]
        public void IsThreeInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var mockWinningStrategy = new Mock<IWinningStrategy>();
            var board3 = new Board3
            {
                winningStrategy = mockWinningStrategy.Object
            };

            // Set up the mock to return a specific value for IsThreeInRow
            mockWinningStrategy.Setup(ws =>
ws.IsThreeInRow(It.IsAny<Cell[,]>())).Returns(true);

            // Act
            var result = board3.IsThreeInRow;

            // Assert
            Assert.True(result);
            mockRepository.VerifyAll();
                }
```

GameEnded_StateUnderTest_ExpectedBehavior
Ar teisingai nustatoma, ar žaidimas baigėsi.

```
[Fact]
        public void GameEnded_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var mockWinningStrategy = new Mock<IWinningStrategy>();
            var board3 = new Board3
            {
                winningStrategy = mockWinningStrategy.Object
            };

            // Set up the mock to return a specific value for IsThreeInRow
            mockWinningStrategy.Setup(ws =>
ws.IsThreeInRow(It.IsAny<Cell[,]>())).Returns(true);

            // Act
            var result = board3.GameEnded;

            // Assert
            Assert.True(result);
            mockRepository.VerifyAll();
                }
```

d. Board4 komponentų testai:

IsFourInRow_Property_ReturnsCorrectValue
Ar teisingai nustatoma, ar yra keturi iš eilės.

```
  [Fact]
        public void IsFourInRow_Property_ReturnsCorrectValue()
        {
            // Arrange
            var board4 = this.CreateBoard4();

            // Act
            var result = board4.IsFourInRow;

            // Assert
            Assert.False(result); // Assuming the initial state is not four in a
row
            this.mockRepository.VerifyAll();
         }
```

GameEnded_Property_ReturnsCorrectValue
Ar teisingai nustatoma, ar žaidimas baigėsi.

```
[Fact]
        public void GameEnded_Property_ReturnsCorrectValue()
        {
            // Arrange
            var board4 = this.CreateBoard4();

            // Act
            var result = board4.GameEnded;

            // Assert
            Assert.False(result); // Assuming the initial state is not four in a
row
            this.mockRepository.VerifyAll();
                        }
```

e. **Piece** komponentų testai:

ToString_StateUnderTest_ExpectedBehavior
Ar teisingai konvertuojamas objektas į tekstą pagal nurodytą reikšmę.

```
 [Fact]
        public void ToString_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var piece = this.CreatePiece("X"); // Set an appropriate value for
the piece

            // Act
            var result = piece.ToString();

            // Assert
            Assert.Equal("X", result);
            this.mockRepository.VerifyAll();
                }
```

GetStatus_StateUnderTest_ExpectedBehavior
Ar teisingai grąžinama statuso reikšmė objektui.

```
[Fact]
        public void GetStatus_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var piece = this.CreatePiece("O"); // Set an appropriate value for
the piece

            // Act
            var result = piece.getStatus();

            // Assert
            Assert.Equal("piece", result);
            this.mockRepository.VerifyAll();
                    }
```

f. **BoardCreator** komponentų testai:

factoryMethod_CreateBoard3_ReturnsBoard3
Ar teisingai grąžinamas Board3 objektas naudojant factory metodą su tipo parametru 3.

```
[Fact]
        public void factoryMethod_CreateBoard3_ReturnsBoard3()
        {
            // Arrange
            var boardCreator = new BoardCreator();
            int type = 3;

            // Act
            var result = BoardCreator.factoryMethod(type);

            // Assert
            Assert.IsType<Board3>(result);
                    }
```

factoryMethod_CreateBoard4_ReturnsBoard4
Ar teisingai grąžinamas Board4 objektas naudojant factory metodą su tipo parametru 4.

```
[Fact]
        public void factoryMethod_CreateBoard4_ReturnsBoard4()
        {
            // Arrange
            var boardCreator = new BoardCreator();
            int type = 4;

            // Act
            var result = BoardCreator.factoryMethod(type);

            // Assert
            Assert.IsType<Board4>(result);
                    }internalServerError.Value.Should().Be("A problem happened while
            handling
your request.");
        }
```

factoryMethod_InvalidType_ReturnsNull
Ar grąžinamas null objektas naudojant factory metodą su neleistinu tipo parametru (0).

```
[Fact]
        public void factoryMethod_InvalidType_ReturnsNull()
        {
            // Arrange
            var boardCreator = new BoardCreator();
            int type = 0;

            // Act
            var result = BoardCreator.factoryMethod(type);

            // Assert
            Assert.Null(result);
                }
```

g. **GameSubject** komponentų testai:

getState_StateUnderTest_ExpectedBehavior
Ar teisingai grąžinama pradinė būsena (true) naudojant getState metodą.

```
 [Fact]
        public void getState_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var gameSubject = this.CreateGameSubject();

            // Act
            var result = gameSubject.getState();

            // Assert
            Assert.True(result); // Assuming the initial state is true
            this.mockRepository.VerifyAll();
                    }
```

setState_StateUnderTest_ExpectedBehavior
Ar teisingai nustatoma nauja būsena naudojant setState metodą.

```
[Fact]
        public void setState_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var gameSubject = this.CreateGameSubject();
            bool value = false;

            // Act
            gameSubject.setState(value);

            // Assert
            Assert.False(gameSubject.getState()); // Check if state is set
correctly
            this.mockRepository.VerifyAll();
                    }
```

h. **FourByFourWinningStrategy** komponentų testai:

IsFourInRow_NoWinningCondition_ReturnsFalse

Ar grąžinama false reikšmė, kai nėra keturių iš eilės.

```
 [Fact]
   public void IsFourInRow_NoWinningCondition_ReturnsFalse()
   {
       // Arrange
       var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
       Cell[,] Pieces = new Cell[4, 4];

       // Act
       var result = fourByFourWinningStrategy.IsFourInRow(Pieces);

       // Assert
       Assert.False(result);
       this.mockRepository.VerifyAll();
           }
```

IsFourInRow_WinningRow_ReturnsTrue

Ar grąžinama true reikšmė, kai yra keturių iš eilės.

```
   [Fact]
   public void IsFourInRow_WinningRow_ReturnsTrue()
   {
       // Arrange
       var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
       Cell[,] Pieces = new Cell[4, 4];

       // Create a winning row
       Pieces[0, 0] = new Cell("X");
       Pieces[0, 1] = new Cell("X");
       Pieces[0, 2] = new Cell("X");
       Pieces[0, 3] = new Cell("X");

       // Act
       var result = fourByFourWinningStrategy.IsFourInRow(Pieces);

       // Assert
       Assert.True(result);
       this.mockRepository.VerifyAll();
    }
```

IsFourInRow_WinningColumn_ReturnsTrue

Ar grąžinama true reikšmė, kai yra keturių iš eilės.

```
[Fact]
   public void IsFourInRow_WinningColumn_ReturnsTrue()
   {
       // Arrange
       var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
       Cell[,] Pieces = new Cell[4, 4];

       // Create a winning column with initialized Cell objects
       Pieces[0, 0] = new Cell("X");
       Pieces[1, 0] = new Cell("X");
```

```
        Pieces[2, 0] = new Cell("X");
        Pieces[3, 0] = new Cell("X");

        // Act
        var result = fourByFourWinningStrategy.IsFourInRow(Pieces);

        // Assert
        Assert.True(result);
        this.mockRepository.VerifyAll();
            }
```

IsBoardFull_EmptyBoard_ReturnsFalse

Ar grąžinama false reikšmė, kai lenta yra tuščia.

```
    [Fact]
    public void IsBoardFull_EmptyBoard_ReturnsFalse()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Act
        var result = fourByFourWinningStrategy.IsBoardFull(Pieces);

        // Assert
        Assert.False(result);
        this.mockRepository.VerifyAll();
            }
```

IsGameOver_NoWinningCondition_NotFull_ReturnsFalse

Ar grąžinama false reikšmė, kai nėra keturių iš eilės ir lenta nėra pilna.

```
    [Fact]
    public void IsGameOver_NoWinningCondition_NotFull_ReturnsFalse()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Act
        var result = fourByFourWinningStrategy.IsGameOver(Pieces);

        // Assert
        Assert.False(result);
        this.mockRepository.VerifyAll();
            }
```

IsThreeInRow_AnyBoard_ReturnsFalse

Ar grąžinama false reikšmė, nepriklausomai nuo to, kaip atrodo lenta.

```
[Fact]
    public void IsThreeInRow_AnyBoard_ReturnsFalse()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Act
        var result = fourByFourWinningStrategy.IsThreeInRow(Pieces);

        // Assert
        Assert.False(result);
        this.mockRepository.VerifyAll();
            }
```

IsFourInRow_WinningDiagonal_ReturnsTrue

Ar grąžinama true reikšmė, kai yra keturi iš eilės skersai.

```
    [Fact]
    public void IsFourInRow_WinningDiagonal_ReturnsTrue()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Create a winning diagonal with initialized Cell objects
        Pieces[1, 1] = new Cell("X");
        Pieces[2, 0] = new Cell("X");
        Pieces[0, 2] = new Cell("X");
        Pieces[3, 3] = new Cell("X");

        // Act
        var result = fourByFourWinningStrategy.IsFourInRow(Pieces);

        // Assert
        Assert.True(result);
        this.mockRepository.VerifyAll();
            }
```

IsFourInRow_WinningBackwardDiagonal_ReturnsTrue

Ar grąžinama true reikšmė, kai yra keturi iš eilės atvirkščiai.

```
[Fact]
    public void IsFourInRow_WinningBackwardDiagonal_ReturnsTrue()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Create a winning backward-diagonal with initialized Cell objects
        Pieces[1, 2] = new Cell("X");
        Pieces[0, 3] = new Cell("X");
        Pieces[2, 1] = new Cell("X");
        Pieces[3, 0] = new Cell("X");

        // Act
        var result = fourByFourWinningStrategy.IsFourInRow(Pieces);

        // Assert
        Assert.True(result);
        this.mockRepository.VerifyAll();
            }
```

IsBoardFull_PartiallyFullBoard_ReturnsFalse
Ar grąžinama false reikšmė, kai lenta yra tik dalinai užpildyta.

```
    [Fact]
    public void IsBoardFull_PartiallyFullBoard_ReturnsFalse()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Create a partially full board
        Pieces[0, 0] = new Cell("X");
        Pieces[1, 1] = new Cell("O");

        // Act
        var result = fourByFourWinningStrategy.IsBoardFull(Pieces);

        // Assert
        Assert.False(result);
        this.mockRepository.VerifyAll();
            }
```

IsBoardFull_CompletelyFullBoard_ReturnsTrue
Ar grąžinama true reikšmė, kai lenta yra visiškai užpildyta.

```
[Fact]
    public void IsBoardFull_CompletelyFullBoard_ReturnsTrue()
    {
        // Arrange
        var fourByFourWinningStrategy = this.CreateFourByFourWinningStrategy();
        Cell[,] Pieces = new Cell[4, 4];

        // Fill the entire board
        for (int row = 0; row < Pieces.GetLength(0); row++)
        {
            for (int col = 0; col < Pieces.GetLength(1); col++)
            {
                Pieces[row, col] = new Cell("X");
```

```
            }
        }

        // Act
        var result = fourByFourWinningStrategy.IsBoardFull(Pieces);

        // Assert
        Assert.True(result);
        this.mockRepository.VerifyAll();
                }
```

i. **ThreeByThreeWinningStrategy** komponentų testai:

IsThreeInRow_StateUnderTest_ExpectedBehavior
Ar grąžinama false reikšmė, kai nėra trijų iš eilės.

```
    [Fact]
        public void IsThreeInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Set up the Pieces array with your desired values for testing

            // Act
            var result = threeByThreeWinningStrategy.IsThreeInRow(Pieces);

            // Assert
            Assert.False(result); // Modify this based on your test scenario
            this.mockRepository.VerifyAll();
                }
```

IsBoardFull_StateUnderTest_ExpectedBehavior
Ar grąžinama false reikšmė, kai lenta nėra pilna.

```
    [Fact]
        public void IsBoardFull_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Set up the Pieces array with your desired values for testing

            // Act
            var result = threeByThreeWinningStrategy.IsBoardFull(Pieces);

            // Assert
            Assert.False(result); // Modify this based on your test scenario
            this.mockRepository.VerifyAll();
        }       this.mockRepository.VerifyAll();
     }
```

IsGameOver_StateUnderTest_ExpectedBehavior

Ar grąžinama false reikšmė, kai nėra trijų iš eilės ir lenta nėra pilna.

```
[Fact]
        public void IsGameOver_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Set up the Pieces array with your desired values for testing

            // Act
            var result = threeByThreeWinningStrategy.IsGameOver(Pieces);

            // Assert
            Assert.False(result); // Modify this based on your test scenario
            this.mockRepository.VerifyAll();
                }
```

IsFourInRow_StateUnderTest_ExpectedBehavior

Ar grąžinama false reikšmė, nes šioje strategijoje nėra keturių iš eilės.

```
[Fact]
        public void IsFourInRow_StateUnderTest_ExpectedBehavior()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Act
            var result = threeByThreeWinningStrategy.IsFourInRow(Pieces);

            // Assert
            Assert.False(result); // Modify this based on your test scenario
            this.mockRepository.VerifyAll();
                }
```

IsThreeInRow_WinningForwardDiagonal_ReturnsTrue

Ar grąžinama true reikšmė, kai yra trijų iš eilės

```
        [Fact]
        public void IsThreeInRow_WinningForwardDiagonal_ReturnsTrue()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Create a winning forward-diagonal with initialized Cell objects
            Pieces[0, 0] = new Cell("X");
            Pieces[1, 1] = new Cell("X");
            Pieces[2, 2] = new Cell("X");

            // Act
            var result = threeByThreeWinningStrategy.IsThreeInRow(Pieces);

            // Assert
            Assert.True(result);
                }
```

IsThreeInRow_WinningBackwardDiagonal_ReturnsTrue
Ar grąžinama true reikšmė, kai yra trijų iš eilės

```
        [Fact]
        public void IsThreeInRow_WinningBackwardDiagonal_ReturnsTrue()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Create a winning backward-diagonal with initialized Cell objects
            Pieces[0, 2] = new Cell("X");
            Pieces[1, 1] = new Cell("X");
            Pieces[2, 0] = new Cell("X");

            // Act
            var result = threeByThreeWinningStrategy.IsThreeInRow(Pieces);

            // Assert
            Assert.True(result);
                }
```

IsBoardFull_FullBoard_ReturnsTrue
Ar grąžinama true reikšmė, kai lenta yra visiškai užpildyta.

```
       [Fact]
        public void IsBoardFull_FullBoard_ReturnsTrue()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Fill the board with initialized Cell objects
            for (int row = 0; row < Pieces.GetLength(0); row++)
            {
                for (int col = 0; col < Pieces.GetLength(1); col++)
                {
                    Pieces[row, col] = new Cell("X");
                }
            }

            // Act
            var result = threeByThreeWinningStrategy.IsBoardFull(Pieces);

            // Assert
            Assert.True(result);
            this.mockRepository.VerifyAll();
                }
```

IsThreeInRow_WinningRow_ReturnsTrue

Ar grąžinama true reikšmė, kai yra trys iš eilės eilutėje.

```
        [Fact]
        public void IsThreeInRow_WinningRow_ReturnsTrue()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Create a winning row with initialized Cell objects
            Pieces[0, 0] = new Cell("X");
            Pieces[0, 1] = new Cell("X");
            Pieces[0, 2] = new Cell("X");

            // Act
            var result = threeByThreeWinningStrategy.IsThreeInRow(Pieces);

            // Assert
            Assert.True(result);
            this.mockRepository.VerifyAll();
                }
```

IsThreeInRow_WinningSpecificColumn_ReturnsTrue
Ar grąžinama true reikšmė, kai yra trijų iš eilės stulpelyje.

```
[Fact]
        public void IsThreeInRow_WinningSpecificColumn_ReturnsTrue()
        {
            // Arrange
            var threeByThreeWinningStrategy =
this.CreateThreeByThreeWinningStrategy();
            Cell[,] Pieces = new Cell[3, 3];

            // Create a winning column with initialized Cell objects
            Pieces[0, 0] = new Cell("X");
            Pieces[1, 0] = new Cell("X");
            Pieces[2, 0] = new Cell("X");

            // Act
            var result = threeByThreeWinningStrategy.IsThreeInRow(Pieces);

            // Assert
            Assert.True(result);
            this.mockRepository.VerifyAll();
                }
```

j. Integration testai:

PlayGame_ShouldReachEndGame

Ar žaidimas baigėsi ir nėra lygiosios

```
[Fact]
        public void PlayGame_ShouldReachEndGame()
        {
            // Arrange
            Player player1 = new Player("Player1", "Room1", "1");
            Player player2 = new Player("Player2", "Room1", "2");
            Player1Factory player1Factory = new Player1Factory();
            Player2Factory player2Factory = new Player2Factory();

            // Create the game
            Game game = new Game(player1Factory, player2Factory, player1,
player2, "Room1", 3, false);

            // Act
            // Simulate a sequence of moves that leads to the end of the game
            game.PlacePiece(0, 0); // Player 1
            game.PlacePiece(1, 1); // Player 2
            game.PlacePiece(0, 1); // Player 1
            game.PlacePiece(1, 0); // Player 2
            game.PlacePiece(0, 2); // Player 1

            // Assert
            // Check the game status or any other relevant assertions
            Assert.True(game.IsOver);
            Assert.False(game.IsTie);


            // Additional assertions based on the expected outcome
                }
```

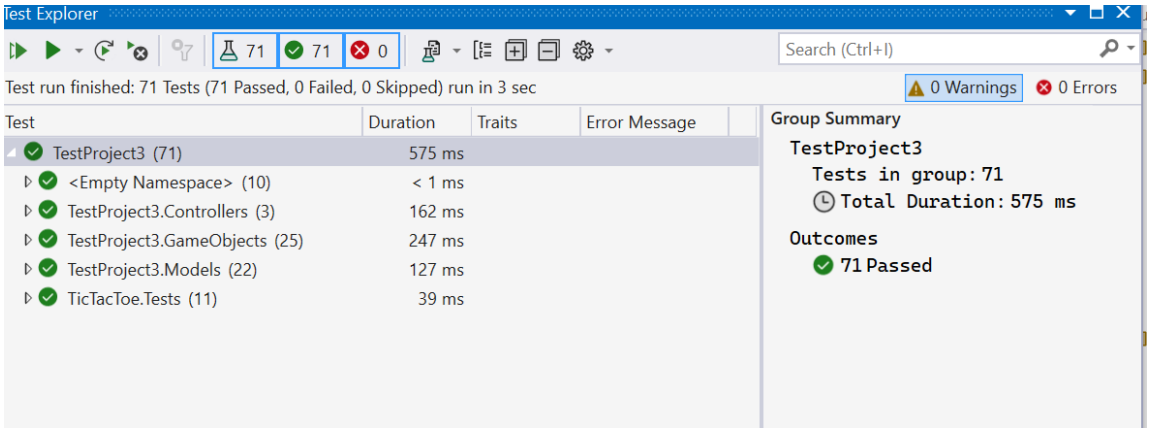
BoardCreator_CreateBoard_ShouldReturnCorrectBoardType

Ar sukurtos lentos yra Board3 ir Board4

```
[Fact]
      public void BoardCreator_CreateBoard_ShouldReturnCorrectBoardType()
      {
          // Arrange
          Board board3 = BoardCreator.factoryMethod(3);
          Board board4 = BoardCreator.factoryMethod(4);

          // Assert
          Assert.IsType<Board3>(board3);
          Assert.IsType<Board4>(board4);
               }
```

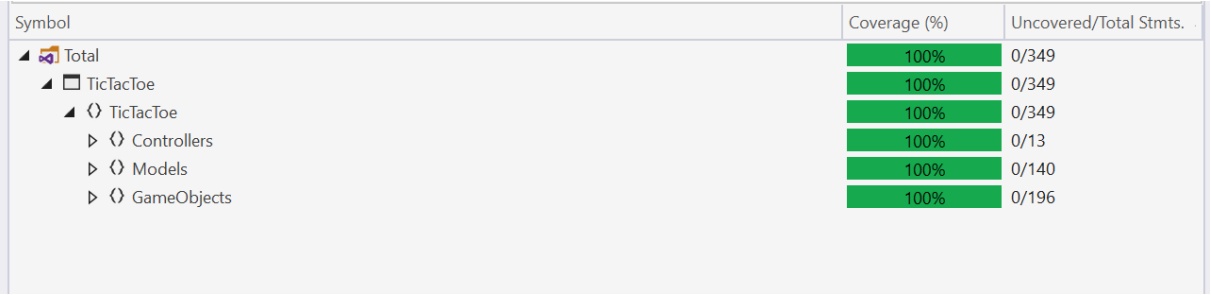
Naudojantis ReSharper testų paleidimo įrankiu nustatyta, kad visi testai yra teisingi (3 pav.)



3 pav. Sukurti komponentų testai yra teisingi

12. Programos kodo padengimas testais po testų sukūrimo

Sukūrus komponentų testus buvo panaudotas įrankis dotCover, kuris šįkart jau parodė, kad visi metodai yra 100% padengti testais (4 pav.).



4 pav. Testuojamo kodo padengimas testais

Taip pat prie metodų kodo eilučių yra rodomi žali laukeliai, kurie nurodo, kad API metodų kodo eilutės yra padengtos testais (5 pav.).

```
17                  !string.IsNullOrWhiteSpace(Pieces[row, 2].Value) &&
18                  Pieces[row, 3] != null &&
19                  !string.IsNullOrWhiteSpace(Pieces[row, 3].Value) &&
20                  Pieces[row, 0].Value == Pieces[row, 1].Value &&
21                  Pieces[row, 1].Value == Pieces[row, 2].Value &&
22                  Pieces[row, 2].Value == Pieces[row, 3].Value)
23              {
24                  return true;
25              }
26          }
27
28          // Check all columns
29          for (int col = 0; col < Pieces.GetLength(dimension: 1); col++)
30          {
31              if (Pieces[0, col] != null &&
32                  !string.IsNullOrWhiteSpace(Pieces[0, col].Value) &&
33                  Pieces[1, col] != null &&
34                  !string.IsNullOrWhiteSpace(Pieces[1, col].Value) &&
35                  Pieces[2, col] != null &&
36                  !string.IsNullOrWhiteSpace(Pieces[2, col].Value) &&
37                  Pieces[3, col] != null &&
38                  !string.IsNullOrWhiteSpace(Pieces[3, col].Value) &&
39                  Pieces[0, col].Value == Pieces[1, col].Value &&
40                  Pieces[1, col].Value == Pieces[2, col].Value &&
41                  Pieces[2, col].Value == Pieces[3, col].Value)
42              {
43                  return true;
44              }
45          }
46
```

5 pav. API programos kodo eilutės yra padengtos testais

13. Išvados

1. Laboratoriniam darbui atlikti buvo surastos, kaip vėliau paaiškėjo, tinkamos priemonės sistemos komponentų testavimui;
2. Prieš realizuojant testus buvo išsiaiškinta, kaip reikia tinkamai aprašyti komponentų vienetų testus bei kaip juos reikia tinkamai aprašyti programos kodu;
3. Komponentų testais buvo padengti visi metodai;
4. Kodo padengimas po testų sukūrimo rodo, kad komponentų vienetų testais buvo sėkmingai

padengta 100% programos kodo.