

Kauno Technologijos Universitetas

Informatikos Fakultetas

**T120B162 Programų sistemų testavimas**

Antro laboratorinio darbo aprašymas

**Atliko:**

IFF-6/8 gr. stud.

Tadas Laurinaitis

Kaunas 2019

# Darbo užduotis

Lab work tasks:

1. Generate software unit tests in order to evaluate software quality for chosen software. The JTest, C++ Test or dot Tests programs could be used. Create some unit tests manually.
2. Generate tests for a whole software application (100% code coverage)
3. Research mocks, stubs, drivers, use them while creating unit tests where appropriate.
4. Research parametrized tests, use them while creating unit tests where appropriate.
5. Research tests set-up, tear-down phases, use them while creating unit tests where appropriate.
6. Create tests using graphical editor for one chosen class to test.
7. Create unit tests in code for one chosen class to test.
8. Evaluate software tests coverage.

# Darbo užduočių vykdymas/sprendimas

## Darbo Aprašymas

Laboratonio darbo metu pagrinde buvo naudojama xUnit testavimo įrankis ir Moq biblioteka.

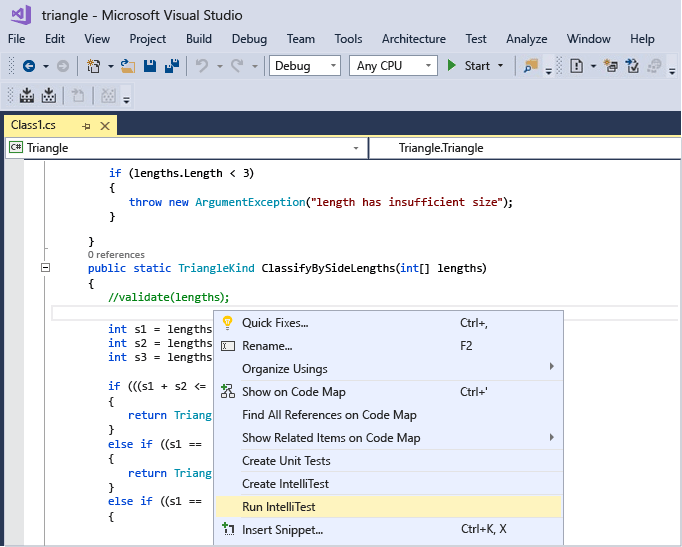
Laboratorniam darbui buvo pasirinktas projektas žaidimas „Bomberman“, kuris buvo kuriamas Objektinių programų projektavimo modulyje. Projektas buvo sukurtas naudojantis .NET Core.

Unit testavimas – programinės įrangos testavimo rūšis, kur kiekvienas individualus komponentas yra testuojamas. Pagrindinė paskirtis – užtikrinti, kad kiekvienas komponentas atlieka visus savo veiksmus kaip numatyta. Unit – mažiausia galima testuoti programinės įrangos dalis, dažniausiai turinti tik kelias įvestis ir vieną išvestį.

## Unit testų generavimas naudojant įrankį

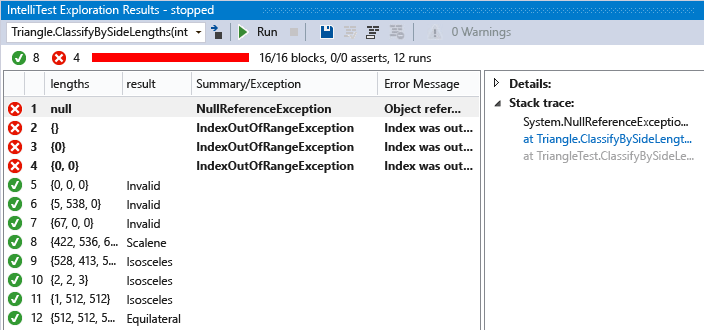
Kadangi testuojamas projektas yra parašytas C# kalba, todėl unit testų generavimui optimaliausias įrankis yra IntelliTest. IntelliTest įrankis ištiria kodą ir sugeneruoja testinius duomenis bei unit test‘ų rinkinius. Kiekvienam kodo sakiniui yra sugeneruojamas testinė įvestis, kuri įvykdys tą sakinį. Ši kodo analizė yra naudojama testinių duomenų, naudojamų parametrizuotuose unit testuose sukūrimui kiekvienam metodui, užtikrinant didelį kodo padengimą. Paleidus IntelliTest galima lengvai matyti nepavykusius testus ir pridėti reikiamą kodą kad jie veiktų.

1. Pasirenkamas metodas kuriam norime sugeneruoti unit testus, spaudžiamas antras pelės mygtukas metodo viduje ir pasirenkamas Run IntelliTest pasirinkimas.



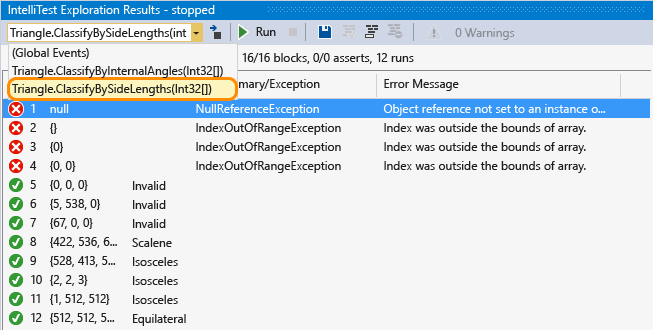
1 pav. „Run Intellitest“

2) IntelliTest vykdo kodą daug kartų su daug skirtingų duomenų ir parodo kiekvieno vykdymo atvejo rezultatus lentelėje.



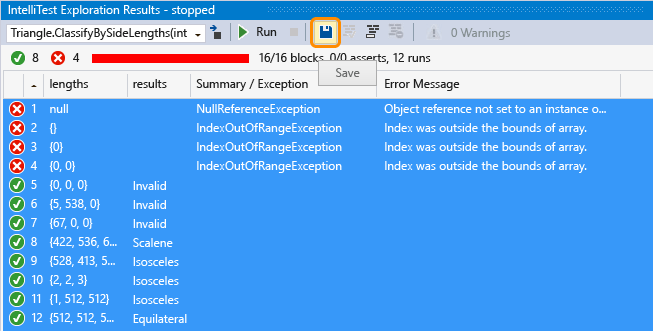
2 pav. Sugeneruotų duomenų ir rezultatų lentelė

3) Norint sugeneruoti testus visiem klasės metodams, reikia žingsnį nr. 1 vykdyti ne metodo o klasės viduje. Kai IntelliTest baigia generuoti, galima pasirinkti kurio metodo rezultatus norima matyti.



3 pav. Sugeneruotų visų klasės metodų duomenų ir rezultatų lentelė

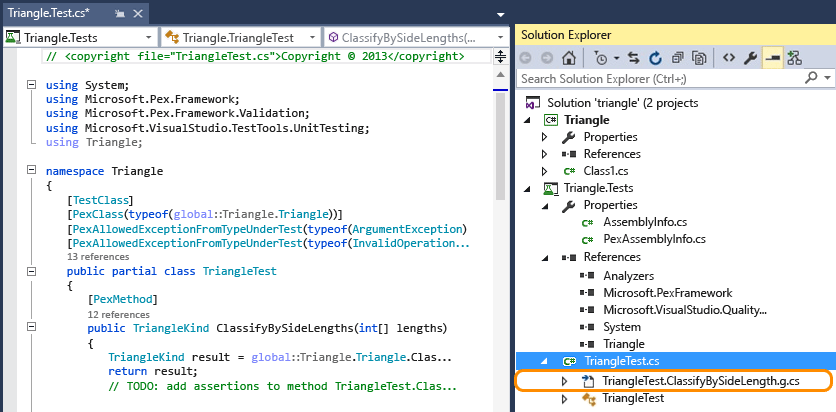
4) Pasirenkamos testų eilutės, kurias norime išsaugoti ir spaudžiame „Save“ mygtuką.



4 pav. Norimų išsaugoti testų pasirinkimas lentelėje ir „Save“ mygtukas

5) Išsaugoti testai mato „Solution Explorer“ lange. Individualūs unit testai yra išsaugomi failuose, kurie baigiasi „.g.cs“, o parametrizuoti unit testai yra išsaugomi atitinkamose atskirose klasėse.

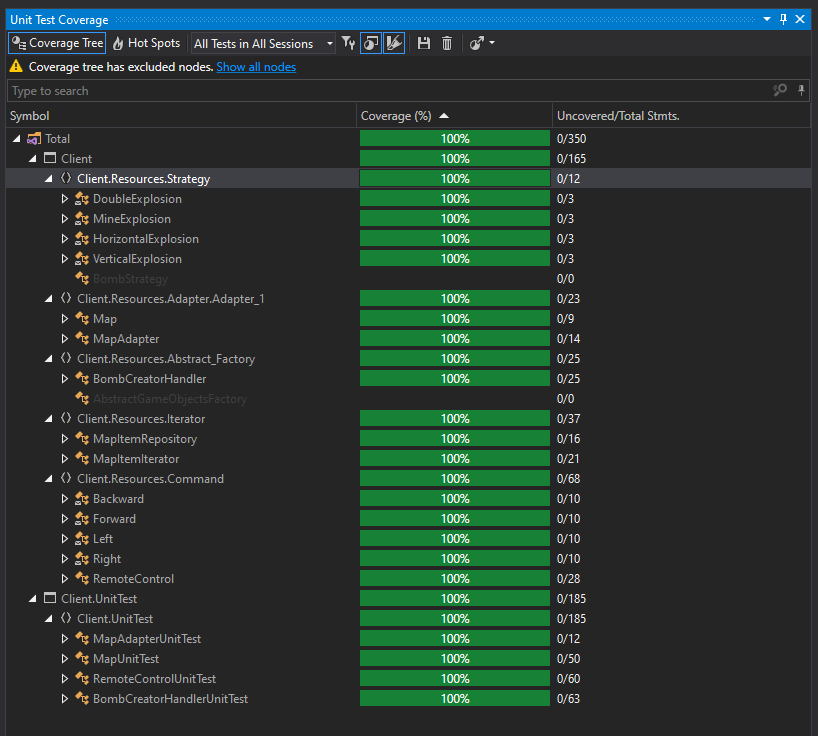
Šiuos testus galima vykdyti kaip ir paprastus, ranka sukurtus testus – per Test Explorer.



5 pav. Matomi išsaugoti testai „Solution Explorer“ lange

## Sugeneruoti testus visai aplikacijai (100% kodo padengimas)

Prieš generuojant ir rašant testus, buvo atfiltruotos klasės, kurių testavimas neatneštų jokios naudos: POCO klasės (modeliai ir t.t.), controller‘iai, taip pat klasės kuriose nenaudojama jokia validacija, o tiesiog gražinami duomenys. Didžiają dalį testų rašiau ranka, nes generavimo įrankis yra prieinamas tik tiems, kas turi Visual Studio Enterprise. Galiausiai buvo pasiektas 100% kodo padengimas. 100% kodo padengimas tikrai nėra rodiklis, kad kodas yra visiškai idealus ir viekiantis taip kaip reikia. Kartais programinė įranga, turinti 70% kodo padengimo veikia teisingiau negu programinė įranga, turinti 100% kodo padengimą.



6 pav. „Unit Test Coverage“

## Mocks, Stubs, Drivers

Mock‘ai – objektai, kurie simuliuoja tikrų objektų elgseną. Testuojamas objektas gali turėti pirklausomybių nuo kitu, sudėtingesnių objektų, todėl siekiant izoliuoti testuojamą objektą, kiti objektai yra pakeičiami mock’ais, kurie atkartoja tikrų objektų elgseną. Mock’ai yra ypač naudingi kai tikrų objektų įtraukimas į unit test’ą yra nepraktiškas.

Stub’ai – kodo gabalas, kuris pakeičia kitą komponentą testavimo metu suteikdamas jam netikrą impelementaciją. Pagrindinis stub’o pliusas yra tas, kad jis gražina tolygius rezultataus, šitaip palengvindamas testo rašymą. Taip pat, stub’ai leidžia vykdyti testus netgi kai kiti komponentai nėra veikiantys.

Driver’iai – naudojami užpildyti trūkstamų ar nepilnai realizuotų komponentų ar modulių reikalavimus. Driver’iai yra naudojami, kai pagrindinis modulis dar nėra užbaigtas. Dažniausiai sudėtingesni negu stub’ai, naudojami vykdant Bottom-Up integracijos testų metodą.

.

## Parameterizuoti testai

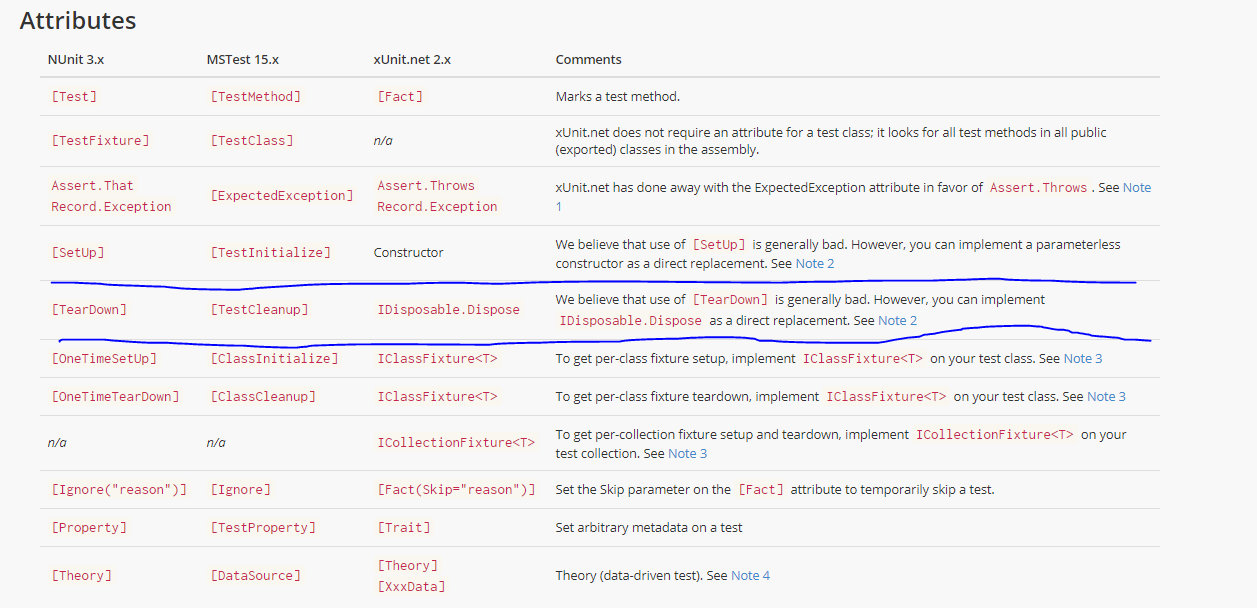
Parametrizuoti testai – testų rūšis, kada reikia tą patį testą vykdyti su keleta arba daugiau duomenų rinkinių. Duomenys yra perduodami per unit test‘o parametrus ir naudojami testo viduje. xUnit naudoja [Theory] antraštę virš testo, bei [InlineData], [ClassData] ir [MemberData] antraštes kiekvienam atskiram duomenų rinkiniui aprašyti. [InlineData] – perduoda duomenis skliausteliuose (Konstantos). [ClassData] – perduoda klasės pavadinimą, iš kurios paimami duomenys (Klasės iš kurios norima paimti duomenis turi implementuoti Ienumerable<object[]>). [MemberData] – perduoda duomenis iš statinio metodo ar property.

## Set-up, tear-down fazės

Set-up metodas – specialus metodas, kuris yra vykdomas prieš kiekvieną unit testą rinkinyje. Tokie metodai dažniausiai naudojami paruošti aplinką prieš unit testo pradžią

Tear-down metodas – specialus metodas, kuris yra vykdomas po kiekvieno unit testo rinkinyje ir yra skirtas apsivalymui testui pasibaigus.

Kadangi testavimui naudojau xUnit, o xUnit kūrėjų nuomone Set-up ir Tear-down metodų naudojimas yra blogas, tai šių metodų nenaudojau.



6 pav. Xunit palyginimas su kitais karkasais

## Sukurti unit test‘ai

Iš pradžių bus įdedadamas testuojamas kodas, o po to lentelių pavidalu bus įdeti unit test metodai ir jų aprašymai. Lentelės struktūra:

|  |
| --- |
| Metodo Pavadinimas |
| Metodo Aprašymas |
| //Metodo kodas |

**Testuojamas BombCreatorHandler klasės metodas:**

public static IBombCreator GetBombCreator(string strategy, string type)

{

IBombCreator bombCreator = null;

if (bombDictionary.ContainsKey(strategy + type))

bombCreator = bombDictionary[strategy + type];

else

{

BombStrategy bombStrategy = null;

switch (strategy)

{

case "double": bombStrategy = new DoubleExplosion(); break;

case "horizontal": bombStrategy = new HorizontalExplosion(); break;

case "mine": bombStrategy = new MineExplosion(); break;

case "vertical": bombStrategy = new VerticalExplosion(); break;

}

switch (type)

{

case "player": bombCreator = new SimplePlayerBomb(bombStrategy);break;

case "enemy": bombCreator = new SimpleEnemyBomb(bombStrategy);break;

}

bombDictionary.Add(strategy + type, bombCreator);

}

return bombCreator;

}

**Metodo testavimui parašyti unit test‘ai:**

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimplePlayerBomb\_HorizontalExplosion |
| Tikrinama ar metodas sukuria SimplePlayerBomb tipo su HorizontalExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimplePlayerBomb\_HorizontalExplosion()  {  string strategy = "horizontal";  string type = "player";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimplePlayerBomb>((SimplePlayerBomb)bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_HorizontalExplosion |
| Tikrinama ar metodas sukuria SimpleEnemyBomb tipo su HorizontalExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_HorizontalExplosion()  {  string strategy = "horizontal";  string type = "enemy";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimpleEnemyBomb>(bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimplePlayerBomb\_MineExplosion |
| Tikrinama ar metodas sukuria SimplePlayerBomb tipo su MineExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimplePlayerBomb\_HorizontalExplosion()  {  string strategy = "horizontal";  string type = "player";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimplePlayerBomb>((SimplePlayerBomb)bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_MineExplosion |
| Tikrinama ar metodas sukuria SimpleEnemyBomb tipo su MineExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_MineExplosion()  {  string strategy = "mine";  string type = "enemy";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimpleEnemyBomb>(bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimplePlayerBomb\_DoubleExplosion |
| Tikrinama ar metodas sukuria SimplePlayerBomb su DoubleExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimplePlayerBomb\_DoubleExplosion()  {  string strategy = "double";  string type = "player";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimplePlayerBomb>((SimplePlayerBomb)bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_DoubleExplosion |
| Tikrinama ar metodas sukuria SimpleEnemyBomb tipo su DoubleExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_DoubleExplosion()  {  string strategy = "double";  string type = "enemy";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimpleEnemyBomb>(bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimplePlayerBomb\_VerticalExplosion |
| Tikrinama ar metodas sukuria SimplePlayerBomb tipo su VerticalExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimplePlayerBomb\_VerticalExplosion()  {  string strategy = "vertical";  string type = "player";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimplePlayerBomb>((SimplePlayerBomb)bombCreator);  } |

|  |
| --- |
| GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_VerticalExplosion |
| Tikrinama ar metodas sukuria SimpleEnemyBomb tipo su VerticalExplosion strategy bombą |
| [Fact]  public void GetBombCreator\_Should\_Return\_SimpleEnemyBomb\_VerticalExplosion()  {  string strategy = "vertical";  string type = "enemy";  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  Assert.IsType<SimpleEnemyBomb>(bombCreator);  } |

Kiekvienas iš šių unit testų buvo skirtas patikrinti, ar padavus skirtingas „strategy“ ir „type“ reikšmes metodas gražins teisingus rezultatus ir padengti visus galimus variantus.

|  |
| --- |
| GetBombCreator\_Should\_Return\_CorrectType |
| Tikrinama ar metodas sukuria reikiamo tipo bombą, o jeigu paduodamos tuščios arba neteisingos reikšmės, gražina null. |
| [Theory]  [InlineData("player", "double")]  [InlineData("enemy", "double")]  [InlineData("", "")]  public void GetBombCreator\_Should\_Return\_CorrectType(string type, string strategy)  {  var bombCreator = BombCreatorHandler.GetBombCreator(strategy, type);  if (type == "player")  {  Assert.IsType<SimplePlayerBomb>(bombCreator);  }  else if (type == "enemy")  {  Assert.IsType<SimpleEnemyBomb>(bombCreator);  }  else  {  Assert.Null(bombCreator);  }  } |

Šis parametrizuotas testas buvo skirtas išbandyti parametrizuotų testų veikimą bei patikrinti ar GetBombBcreator gražina teisingo tipo objektą priklausomai nuo paduodamo “type”.

**Testuojama RemoteControl klasė:**

public class RemoteControl

{

private List<ICommand> commands = new List<ICommand>();

public void undo()

{

if (commands.Count > 0)

{

commands.Last().undo();

commands.RemoveAt(commands.Count - 1);

}

}

public void MoveForward(Player player)

{

ICommand forward = new Forward(player);

forward.move();

commands.Add(forward);

}

public void MoveBackward(Player player)

{

ICommand backward = new Backward(player);

backward.move();

commands.Add(backward);

}

public void MoveLeft(Player player)

{

ICommand left = new Left(player);

left.move();

commands.Add(left);

}

public void MoveRight(Player player)

{

ICommand right = new Right(player);

right.move();

commands.Add(right);

}

}

**Klasės metodų testavimui parašyti unit test’ai:**

|  |
| --- |
| MoveForward\_Should\_Move\_Player\_Forward |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos y koordintatę |
| [Fact]  public void MoveForward\_Should\_Move\_Player\_Forward()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.y - 2;  remoteControl.MoveForward(player);  Assert.Equal(expectedCoordinate, player.y);  } |

|  |
| --- |
| MoveForward\_And\_Then\_Undo |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos y koordintatę ir tada atgražina atgal. |
| [Fact]  public void MoveForward\_And\_Then\_Undo()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.y;  remoteControl.MoveForward(player);  remoteControl.undo();  Assert.Equal(expectedCoordinate, player.y);  } |

|  |
| --- |
| MoveBackward\_Should\_Move\_Player\_Backward |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos y koordintatę |
| [Fact]  public void MoveBackward\_Should\_Move\_Player\_Backward()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.y + 2;  remoteControl.MoveBackward(player);  Assert.Equal(expectedCoordinate, player.y);  } |

|  |
| --- |
| MoveBackward\_And\_Then\_Undo |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos y koordintatę ir tada atgražina atgal. |
| [Fact]  public void MoveBackward\_And\_Then\_Undo()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.y;  remoteControl.MoveBackward(player);  remoteControl.undo();  Assert.Equal(expectedCoordinate, player.y);  } |

|  |
| --- |
| MoveLeft\_Should\_Move\_Player\_Left |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos x koordintatę |
| [Fact]  public void MoveLeft\_Should\_Move\_Player\_Left()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.x - 2;  remoteControl.MoveLeft(player);  Assert.Equal(expectedCoordinate, player.x);  } |

|  |
| --- |
| MoveLeft\_And\_Then\_Undo |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos x koordintatę ir tada atgražina atgal. |
| [Fact]  public void MoveLeft\_And\_Then\_Undo()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.x;  remoteControl.MoveLeft(player);  remoteControl.undo();  Assert.Equal(expectedCoordinate, player.x);  } |

|  |
| --- |
| MoveRight\_Should\_Move\_Player\_Right |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos x koordintatę |
| [Fact]  public void MoveRight\_Should\_Move\_Player\_Right()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.x + 2;  remoteControl.MoveRight(player);  Assert.Equal(expectedCoordinate, player.x);  } |

|  |
| --- |
| MoveRight\_And\_Then\_Undo |
| Tikrinama ar metodas teisingai pakeičia Player tipo objekto pozicijos x koordintatę ir tada atgražina atgal. |
| [Fact]  public void MoveRight\_And\_Then\_Undo()  {  Player player = new Player();  RemoteControl remoteControl = new RemoteControl();  int expectedCoordinate = player.x;  remoteControl.MoveRight(player);  remoteControl.undo();  Assert.Equal(expectedCoordinate, player.x);  } |

Kiekvienas iš šių testų tikrina ar teisingai kinta Player klasės objekto koordinatės x ir y priklausomai nuo to, kokia komanda duota. Taip pat patikrinamas undo metodas, kuris gražina Player klasės objekto coordinates į buvusias prieš vykdant komandą.

**Testuojama MapAdapter klasė:**

public class MapAdapter : IMap

{

public Map map { get; set; }

public MapAdapter()

{

map = new Map();

}

public void AddMapItem()

{

//map.mapItems.Add();

throw new **NotImplementedException**();

}

public void AddItemToMap(IGameObject item)

{

map.AddItemToMap(item);

}

public MapItemRepository GetMap()

{

return map.mapItems;

}

}

**Testuojama Map klasė:**

public class Map

{

public MapItemRepository mapItems { get; private set; }

public Map()

{

mapItems = new MapItemRepository();

}

public void AddItemToMap(IGameObject item)

{

mapItems.Add(item);

}

}

**Klasių metodų testavimui parašyti unit test’ai:**

|  |
| --- |
| AddItemToMap\_Add\_Item\_And\_Check |
| Tikrinama ar įdėjus IGameObject interfeisą realizuojančios klasės objektą į MapAdapter klasės objekto vidinę kolekciją, įdėjimas veikia teisingai ir gražina reikiamą reikšmę. |
| [Fact]  public void AddItemToMap\_Add\_Item\_And\_Check()  {  MapAdapter mapAdapter = new MapAdapter();  IGameObject gameObject = new Wall(5,5);  mapAdapter.AddItemToMap(gameObject);  MapItemRepository mapItems = mapAdapter.GetMap();  Assert.Equal(gameObject, mapItems.GetIterator().First());  } |

|  |
| --- |
| AddMapItem\_CheckForNotImplementedException |
| Tikrinama ar dar nerealizuotas metodas išmes reikiamo tipo exception. |
| [Fact]  public void AddMapItem\_CheckForNotImplementedException()  {  MapAdapter mapAdapter = new MapAdapter();  Assert.Throws<**NotImplementedException**>(() => mapAdapter.AddMapItem());  } |

Pirmasis unit test’as skirtas patikrinti, ar pridedant naują objektą į kolekciją jis tikrai gerai prisideda. Antrasis unit testas skirtas pabandyti, kaip veikia dar neimplementuotų metodų, kurie išmeta exception kai juos kvieti, testavimas.

**Testuojama MapItemRepository klasė:**

public class MapItemRepository : Container

{

private List<IGameObject> mapItems = new List<IGameObject>();

public Iterator GetIterator()

{

return new MapItemIterator(this);

}

public void Add(IGameObject mapItem)

{

mapItems.Add(mapItem);

}

public int Count

{

get { return mapItems.Count; }

}

// Indexer

public object this[int index]

{

get { return mapItems[index]; }

set { mapItems.Add((IGameObject)value); }

}

}

**Testuojama MapItemIterator klasė:**

public class MapItemIterator : Iterator

{

public MapItemRepository repository;

public MapItemIterator(MapItemRepository repository)

{

this.repository = repository;

}

public int index = 0;

public object First()

{

index = 0;

return repository[0];

}

public object Next()

{

if (index < repository.Count - 1)

{

return repository[++index];

}

return null;

}

public object CurrentItem()

{

return repository[index];

}

public bool IsDone()

{

return (index >= repository.Count-1);

}

}

**Klasių metodų testavimui parašyti unit test’ai:**

|  |
| --- |
| Return\_First\_Map\_Repository\_Item\_From\_Iterator |
| Tikrinama ar įdėjus objektą į MapItemRepository jis įsideda gerai. |
| [Fact]  public void Return\_First\_Map\_Repository\_Item\_From\_Iterator()  {  Mock gameObjectMock = new Mock<IGameObject>();  var gameObject = gameObjectMock.Object;  MapItemRepository mapItemRepository = new MapItemRepository();  mapItemRepository.Add((IGameObject)gameObject);  MapItemIterator iterator = new MapItemIterator(mapItemRepository);  var resultObject = iterator.First();  Assert.Equal(gameObject, resultObject);  } |

|  |
| --- |
| Return\_Next\_Map\_Repository\_Item\_From\_Iterator |
| Tikrinama ar įdėjus kelis objektus į MapItemRepository jie įsideda gerai ir ar gaunamas rezultatas paimant juos su Next metodu yra geras. |
| [Fact]  public void Return\_Next\_Map\_Repository\_Item\_From\_Iterator()  {  Mock gameObjectMock = new Mock<IGameObject>();  var gameObject1 = gameObjectMock.Object;  var gameObject2 = gameObjectMock.Object;  MapItemRepository mapItemRepository = new MapItemRepository();  mapItemRepository.Add((IGameObject)gameObject1);  mapItemRepository.Add((IGameObject)gameObject2);  MapItemIterator iterator = new MapItemIterator(mapItemRepository);  var resultObject = iterator.Next();  Assert.Equal(gameObject2, resultObject);  } |

|  |
| --- |
| Return\_NULL\_From\_Next\_In\_Iterator |
| Tikrinama ar iteratoriaus Next metodas gražins Null reikšmę kai bus praeiti visi kolekcijos objektai. |
| [Fact]  public void Return\_NULL\_From\_Next\_In\_Iterator()  {  Mock gameObjectMock = new Mock<IGameObject>();  var gameObject = gameObjectMock.Object;  MapItemRepository mapItemRepository = new MapItemRepository();  mapItemRepository.Add((IGameObject)gameObject);  MapItemIterator iterator = new MapItemIterator(mapItemRepository);  var resultObject = iterator.Next();  Assert.Null(resultObject);  } |

|  |
| --- |
| Return\_CurrentItem\_From\_Iterator |
| Tikrinama ar CurrentItem iteratoriaus metodas gražins teisingą esamą reikšmę |
| [Fact]  public void Return\_CurrentItem\_From\_Iterator()  {  Mock gameObjectMock = new Mock<IGameObject>();  var gameObject = gameObjectMock.Object;  MapItemRepository mapItemRepository = new MapItemRepository();  mapItemRepository.Add((IGameObject)gameObject);  MapItemIterator iterator = new MapItemIterator(mapItemRepository);  var resultObject = iterator.CurrentItem();  Assert.Equal(gameObject, resultObject);  } |

|  |
| --- |
| Check\_IsDone\_Iterator |
| Tikrinama ar metodas gražina teisingą reikšmę kai iteratoriaus count yra daugiau arba lygu kolekcijos objektų skaičių |
| [Fact]  public void Check\_IsDone\_Iterator()  {  Mock gameObjectMock = new Mock<IGameObject>();  var gameObject = gameObjectMock.Object;  MapItemRepository mapItemRepository = new MapItemRepository();  mapItemRepository[0] = (IGameObject) gameObject;  mapItemRepository.Add((IGameObject)gameObject);  MapItemIterator iterator = new MapItemIterator(mapItemRepository);  var resultObject = iterator.First();  resultObject = iterator.Next();  resultObject = iterator.Next();  Assert.True(iterator.IsDone());  } |

Kiekviename iš šių unit test‘ų yra naudojamas Mock klasės objektas iš Moq bibliotekos, pakeičiantis IGameObject. Jis naudojamas tam, kad nereikėtų kurti specialių IgameObject interface paveldinčių klasių objektų, kurių konstruktoriai yra sudėtingi. Šitaip yra supaprastinamas visas procesas, o esmė nepasikeičia. Šie unit testai skirti patikrinti MapItemIterator ir MapItemRepository klasių metodų veikimą, kadangi abi šios klasės yra labai svarbios ir yra būtina užtikrinti kad jie veikia taip kaip reikia. Kiekvienas metodas tikrina kiekvieno klasės metodo veikimą ir pereina per visus jo galimus kontrolinius kelius.

# Išvados

Prieš atliekant laboratorinį darbą, buvo perskaitytas nemažas kiekis teorinės medžiagos ir peržiūrėti keli video, kuriose buvo pasakojama kaip teisingai reikia kurti ir vykyti unit test‘us ir kaip pasiekti didelį kodo padengimą testais. Taip pat išanalizuotas parametrizuotų testų, mockų, stubų ir driverių, kūrimas ir panaudojimas. Laboratorinio darbo užduotims atlikti buvo surasti ir naudoti teisingi įrankiai ir bibliotekos, leidę darbą atlikti efektyviai. Naudojantis įrankiais buvo sukurti testai visam programos kodui, šitaip pasiekiant 100% padengimą.

# Šaltiniai

<https://docs.microsoft.com/en-us/visualstudio/test/generate-unit-tests-for-your-code-with-intellitest?view=vs-2019>

<https://spin.atomicobject.com/2017/08/07/intro-mocking-moq/>

<https://www.youtube.com/watch?v=lqNwdO9E-OQ>

<https://xunit.net/docs/getting-started/netfx/visual-studio>

<https://andrewlock.net/creating-parameterised-tests-in-xunit-with-inlinedata-classdata-and-memberdata/>

<https://dzone.com/articles/unit-testing-guidelines-what-to-test-and-what-not>

<http://www.professionalqa.com/stubs-and-drivers>

<https://www.c-sharpcorner.com/article/moq-mocking-framework-with-xunit-net-testing-fr/>

<https://spring.io/blog/2007/01/15/unit-testing-with-stubs-and-mocks/>