**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 1

**З дисципліни:** *“Конструювання програмного забезпечення”*

**На тему:** *“Розробка алгоритму руху роботів”*

**Лектор:**

доцент каф. ПЗ

Сердюк П. В.

**Виконала:**

ст. гр. ПЗ-31

Башинська Е.І.

**Прийняв:**

асистент каф. ПЗ

Яценко Р.О.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

∑= \_\_\_\_\_ .

Львів – 2024

**Тема роботи:** Розробка алгоритму руху роботів.

**Мета роботи:** Ознайомлення з засобами розробки Visual Studio та Resharper. Після виконання лабораторної студент повинен уміти створювати проекти, підключати бібліотеки, відлагоджувати програми.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Об'єктно-орієнтоване програмування або ООП (object - oriented programming) - методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів кожен з яких являється реалізацією визначеного типу, що використовує механізм пересилки повідомлень і класи організовані в ієрархію наслідування.

Абстрагування (abstraction) - метод рішення задачі, при якому об'єкти різного роду об'єднуються загальним поняттям (концепцією), а потім згруповані суті розглядаються як елементи єдиної категорії. Абстрагування дозволяє відокремити логічний сенс фрагмента програми від проблеми його реалізації, розділивши зовнішній опис (інтерфейс) об'єкту і його внутрішню організацію (реалізацію).

Інкапсуляція (encapsulation) - техніка, при якій несуттєва з точки зору інтерфейсу об'єкту інформація ховається усередині нього.

Спадкоємство (inheritance) - властивість об'єктів, за допомогою якої екземпляри класу дістають доступ до даних і методів класів-предків без їх повторного визначення. Спадкоємство дозволяє різним типам даних спільно використовувати один і той же код, призводячи до зменшення його розміру і підвищення функціональності.

Поліморфізм (polymorphism) - властивість, що дозволяє використовувати один і той же інтерфейс для різних дій; поліморфній змінній, наприклад, може відповідати декілька різних методів.

Протокол або інтерфейс є звичайним засобом для незв'язаних об'єктів спілкуватися один з одним. Це визначення методів та цінностей, з яким об'єкти погоджуються для співпраці. Наприклад, у Java, де протоколи називаються інтерфейсами, інтерфейс Comparable визначає метод compareTo(), який реалізовані класи повинні виконувати. Це означає, що окремий метод сортування, наприклад, може сортувати будь-який об'єкт, який реалізує інтерфейс Comparable, без необхідності знати будь-що про внутрішню природу класу (крім того, що два з цих об'єктів можна порівняти за допомогою compareTo()).

Абстрактний клас - це базовий клас, від якого не можна створити екземпляру. На практиці абстрактні класи реалізують один з принципів ООП - поліморфізм. В абстрактному класі можна описати (або не визначити) абстрактні методи та властивості.

У числі основних ознак абстрактного класу - наявність реалізації. При цьому в ньому може бути присутнім один або кілька її методів. В абстрактному класі є властивості, що мають модифікатори доступу. Інтерфейс, в свою чергу, не містить реалізації. У ньому присутній опис в чистому вигляді - як сукупність методів, що підлягають реалізації в конкретному класі, за допомогою якого здійснюється імплементація того чи іншого інтерфейсу.

Додавання подій до коду є доцільним, оскільки вони забезпечують гнучкість, модульність і можливість інтеграції з іншими компонентами системи без зміни основної логіки алгоритму. Події дозволяють сповіщати зовнішні компоненти (наприклад, графічний інтерфейс, логгер або систему моніторингу) про ключові дії робота, такі як рух, створення нового робота чи збір енергії. Це зменшує залежності між класами, полегшує логування для дебагінгу та підтримує координацію в багатокористувацьких сценаріях. У коді додано три події: RobotMoved (рух робота до нової позиції), RobotCreated (створення нового робота) і EnergyCollected (збір енергії). Ці події викликаються в методі DoStep залежно від виконаної команди, не змінюючи логіку алгоритму. Зовнішні компоненти можуть підписатися на ці події для обробки, наприклад, для виведення інформації про дії робота чи оновлення стану гри.

**ЗАВДАННЯ**

* Інсталювати Visual Studio та Resharper.
* Створити новий проект, у якому реалізувати алгоритм за яким його роботи рухатимуться, збиратимуть ресурси та створюватимуть нових роботів.
* Написати 10 Unit-тестів.
* Використати основи мови C# (List, event, додаткові класи і тд).

На карті розміром 100 на 100 є ресурси енергії та роботи які їх збирають. Кожен студент реалізує алгоритм за яким його роботи рухатимуться, збиратимуть ресурси та створюватимуть нових роботів. Лабораторна є конкурсною, тобто на одній карті будуть одночасно роботи усіх студентів, що здають лабораторну. Кількість балів за лабораторну залежить від кількості ресурсів, який зібрали роботи за 50 ходів усіх роботів.

Варіант 2. Станцій у 5 разів більше ніж усіх роботів на початку змагання, кожна з яких генерує від 50 до 100 одиниць енергії за хід. Максимальне значення енергії станції 20000. Роботи

можуть збирати енергію зі всіх станцій на відстані 2 клітинок, але не більше 40 з одної станції за хід. Енергія втрат при створенні нового робота рівна 100. За кожні 15 000 очок ставиться 1 бал.

**ХІД ВИКОНАННЯ**

Лістинг файлу EvelinaBashynskaAlgorithm.cs:

using Robot.Common;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EvelinaBashynska

{

public class DistanceHelper

{

public static int FindDistance(Position a, Position b)

{

return (int)(Math.Pow(a.X - b.X, 2) + Math.Pow(a.Y - b.Y, 2));

}

}

public class EvelinaBashynskaAlgorithm : IRobotAlgorithm

{

public string Author => "Evelina Bashynska";

// Визначення делегатів для подій

public delegate void RobotMovedEventHandler(Robot.Common.Robot robot, Position newPosition);

public delegate void RobotCreatedEventHandler(Robot.Common.Robot robot, int newRobotEnergy);

public delegate void EnergyCollectedEventHandler(Robot.Common.Robot robot);

// Події

public event RobotMovedEventHandler RobotMoved;

public event RobotCreatedEventHandler RobotCreated;

public event EnergyCollectedEventHandler EnergyCollected;

public Position FindNearestFreeStation(Robot.Common.Robot movingRobot, Map map, IList<Robot.Common.Robot> robots)

{

EnergyStation nearest = null;

int minDistance = int.MaxValue;

foreach (var station in map.Stations)

{

if (IsStationFree(station, movingRobot, robots))

{

int d = DistanceHelper.FindDistance(station.Position, movingRobot.Position);

if (d < minDistance)

{

minDistance = d;

nearest = station;

}

}

}

return nearest == null ? null : nearest.Position;

}

public Position FindTrueNearestFreeStation(Robot.Common.Robot movingRobot, Map map, IList<Robot.Common.Robot> robots)

{

EnergyStation nearest = null;

int minDistance = int.MaxValue;

foreach (var station in map.Stations)

{

if (IsCellNotUs(station.Position, movingRobot, robots))

{

int d = DistanceHelper.FindDistance(station.Position, movingRobot.Position);

if (d < minDistance)

{

minDistance = d;

nearest = station;

}

}

}

return nearest == null ? null : nearest.Position;

}

public bool IsStationFree(EnergyStation station, Robot.Common.Robot movingRobot, IList<Robot.Common.Robot> robots)

{

return IsCellFree(station.Position, movingRobot, robots);

}

public bool IsCellFree(Position cell, Robot.Common.Robot movingRobot, IList<Robot.Common.Robot> robots)

{

foreach (var robot in robots)

{

if (robot.Position == cell)

{

if (cell == movingRobot.Position)

{

return true;

}

if (robot.OwnerName == movingRobot.OwnerName)

{

return false;

}

return true;

}

}

return true;

}

public bool IsCellNotUs(Position cell, Robot.Common.Robot movingRobot, IList<Robot.Common.Robot> robots)

{

foreach (var robot in robots)

{

if (robot.Position == cell)

{

if (robot.OwnerName == movingRobot.OwnerName)

{

return false;

}

return true;

}

}

return true;

}

private Position FindNextPositionToMove(Robot.Common.Robot myRobot, Position destination)

{

Position nextPosition = new Position();

int distance = DistanceHelper.FindDistance(destination, myRobot.Position);

if (distance \* distance < myRobot.Energy)

{

nextPosition = destination.Copy();

}

else

{

// Обмежимо крок на основі наявної енергії

int maxDistance = (int)Math.Floor(Math.Sqrt(myRobot.Energy)); // Максимальна кількість клітинок, яку робот може пройти

int deltaX = destination.X - myRobot.Position.X;

int deltaY = destination.Y - myRobot.Position.Y;

// Рахуємо, як далеко можна пересунутись на осях

int stepX = 0;

int stepY = 0;

if (Math.Abs(deltaX) > 0)

{

stepX = Math.Min(Math.Abs(deltaX), maxDistance) \* Math.Sign(deltaX);

}

if (Math.Abs(deltaY) > 0)

{

stepY = Math.Min(Math.Abs(deltaY), maxDistance - Math.Abs(stepX)) \* Math.Sign(deltaY);

}

// Створюємо нову позицію на основі розрахованих кроків

nextPosition.X = myRobot.Position.X + stepX;

nextPosition.Y = myRobot.Position.Y + stepY;

}

return nextPosition;

}

int movingIndex = 5;

public RobotCommand DoStep(IList<Robot.Common.Robot> robots, int robotToMoveIndex, Map map)

{

Robot.Common.Robot movingRobot = robots[robotToMoveIndex];

Position stationPosition = FindNearestFreeStation(movingRobot, map, robots);

if (stationPosition == movingRobot.Position)

{

int nearestFreeStationDistance = DistanceHelper.FindDistance(FindTrueNearestFreeStation(movingRobot, map, robots), map.FindFreeCell(stationPosition, robots));

if (movingRobot.Energy >= (nearestFreeStationDistance / movingIndex) \* movingIndex \* movingIndex + (movingIndex \* movingIndex - movingIndex) + 40)

{

var command = new CreateNewRobotCommand() { NewRobotEnergy = (nearestFreeStationDistance / movingIndex) \* movingIndex + (movingIndex \* movingIndex - movingIndex) + 40 };

// Виклик події RobotCreated

RobotCreated?.Invoke(movingRobot, command.NewRobotEnergy);

return command;

}

else

{

// Виклик події EnergyCollected

EnergyCollected?.Invoke(movingRobot);

return new CollectEnergyCommand();

}

}

var moveCommand = new MoveCommand() { NewPosition = FindNextPositionToMove(movingRobot, FindTrueNearestFreeStation(movingRobot, map, robots)) };

// Виклик події RobotMoved

RobotMoved?.Invoke(movingRobot, moveCommand.NewPosition);

return moveCommand;

}

}

}

Лістинг файлу UnitTest1.cs:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Robot.Common;

using System;

using System.Collections.Generic;

using System.Linq;

using EvelinaBashynska;

namespace EvelinaBashynska.Tests

{

[TestClass]

public class DistanceHelperTests

{

[TestMethod]

public void FindDistance\_SamePosition\_ReturnsZero()

{

// Arrange

var positionA = new Position(5, 5);

var positionB = new Position(5, 5);

// Act

int distance = DistanceHelper.FindDistance(positionA, positionB);

// Assert

Assert.AreEqual(0, distance);

}

[TestMethod]

public void FindDistance\_DifferentPositions\_ReturnsCorrectSquaredDistance()

{

// Arrange

var positionA = new Position(0, 0);

var positionB = new Position(3, 4);

// Act

int distance = DistanceHelper.FindDistance(positionA, positionB);

// Assert

Assert.AreEqual(25, distance); // 3^2 + 4^2 = 9 + 16 = 25

}

[TestMethod]

public void FindDistance\_NegativeCoordinates\_ReturnsCorrectSquaredDistance()

{

// Arrange

var positionA = new Position(-2, -3);

var positionB = new Position(1, 1);

// Act

int distance = DistanceHelper.FindDistance(positionA, positionB);

// Assert

Assert.AreEqual(25, distance); // (1-(-2))^2 + (1-(-3))^2 = 3^2 + 4^2 = 25

}

}

[TestClass]

public class EvelinaBashynskaAlgorithmTests

{

private EvelinaBashynskaAlgorithm algorithm;

private Map map;

private List<Robot.Common.Robot> robots;

private Robot.Common.Robot movingRobot;

[TestInitialize]

public void Setup()

{

algorithm = new EvelinaBashynskaAlgorithm();

// Створюємо карту з енергетичними станціями

var stations = new List<EnergyStation>

{

new EnergyStation { Position = new Position(0, 0), Energy = 1000, RecoveryRate = 2 },

new EnergyStation { Position = new Position(5, 5), Energy = 1000, RecoveryRate = 2 },

new EnergyStation { Position = new Position(10, 0), Energy = 1000, RecoveryRate = 2 }

};

map = new Map { Stations = stations };

// Створюємо роботів

movingRobot = new Robot.Common.Robot

{

Position = new Position(2, 2),

Energy = 100,

OwnerName = "Evelina Bashynska"

};

robots = new List<Robot.Common.Robot> { movingRobot };

}

[TestMethod]

public void Author\_ReturnsCorrectName()

{

// Assert

Assert.AreEqual("Evelina Bashynska", algorithm.Author);

}

[TestMethod]

public void IsCellFree\_EmptyCell\_ReturnsTrue()

{

// Arrange

var emptyPosition = new Position(1, 1);

// Act

bool isFree = algorithm.IsCellFree(emptyPosition, movingRobot, robots);

// Assert

Assert.IsTrue(isFree);

}

[TestMethod]

public void IsCellFree\_OccupiedByMovingRobot\_ReturnsTrue()

{

// Act

bool isFree = algorithm.IsCellFree(movingRobot.Position, movingRobot, robots);

// Assert

Assert.IsTrue(isFree);

}

[TestMethod]

public void IsCellFree\_OccupiedByFriendlyRobot\_ReturnsFalse()

{

// Arrange

var friendlyRobot = new Robot.Common.Robot

{

Position = new Position(3, 3),

Energy = 50,

OwnerName = "Evelina Bashynska"

};

robots.Add(friendlyRobot);

// Act

bool isFree = algorithm.IsCellFree(friendlyRobot.Position, movingRobot, robots);

// Assert

Assert.IsFalse(isFree);

}

[TestMethod]

public void IsCellFree\_OccupiedByEnemyRobot\_ReturnsTrue()

{

// Arrange

var enemyRobot = new Robot.Common.Robot

{

Position = new Position(3, 3),

Energy = 50,

OwnerName = "Enemy"

};

robots.Add(enemyRobot);

// Act

bool isFree = algorithm.IsCellFree(enemyRobot.Position, movingRobot, robots);

// Assert

Assert.IsTrue(isFree);

}

[TestMethod]

public void IsCellNotUs\_EmptyCell\_ReturnsTrue()

{

// Arrange

var emptyPosition = new Position(1, 1);

// Act

bool isNotUs = algorithm.IsCellNotUs(emptyPosition, movingRobot, robots);

// Assert

Assert.IsTrue(isNotUs);

}

[TestMethod]

public void IsCellNotUs\_OccupiedByFriendlyRobot\_ReturnsFalse()

{

// Arrange

var friendlyRobot = new Robot.Common.Robot

{

Position = new Position(3, 3),

Energy = 50,

OwnerName = "Evelina Bashynska"

};

robots.Add(friendlyRobot);

// Act

bool isNotUs = algorithm.IsCellNotUs(friendlyRobot.Position, movingRobot, robots);

// Assert

Assert.IsFalse(isNotUs);

}

[TestMethod]

public void IsCellNotUs\_OccupiedByEnemyRobot\_ReturnsTrue()

{

// Arrange

var enemyRobot = new Robot.Common.Robot

{

Position = new Position(3, 3),

Energy = 50,

OwnerName = "Enemy"

};

robots.Add(enemyRobot);

// Act

bool isNotUs = algorithm.IsCellNotUs(enemyRobot.Position, movingRobot, robots);

// Assert

Assert.IsTrue(isNotUs);

}

[TestMethod]

public void FindNearestFreeStation\_NoFreeStations\_ReturnsNull()

{

// Arrange - блокуємо всі станції дружніми роботами

var friendlyRobot1 = new Robot.Common.Robot { Position = new Position(0, 0), OwnerName = "Evelina Bashynska" };

var friendlyRobot2 = new Robot.Common.Robot { Position = new Position(5, 5), OwnerName = "Evelina Bashynska" };

var friendlyRobot3 = new Robot.Common.Robot { Position = new Position(10, 0), OwnerName = "Evelina Bashynska" };

robots.AddRange(new[] { friendlyRobot1, friendlyRobot2, friendlyRobot3 });

// Act

Position nearestStation = algorithm.FindNearestFreeStation(movingRobot, map, robots);

// Assert

Assert.IsNull(nearestStation);

}

[TestMethod]

public void FindNearestFreeStation\_MultipleFreeStations\_ReturnsNearestOne()

{

// Act

Position nearestStation = algorithm.FindNearestFreeStation(movingRobot, map, robots);

// Assert - найближча станція до позиції (2,2) має бути (0,0)

Assert.IsNotNull(nearestStation);

Assert.AreEqual(0, nearestStation.X);

Assert.AreEqual(0, nearestStation.Y);

}

[TestMethod]

public void FindTrueNearestFreeStation\_AllStationsAvailable\_ReturnsNearestOne()

{

// Act

Position nearestStation = algorithm.FindTrueNearestFreeStation(movingRobot, map, robots);

// Assert - найближча станція до позиції (2,2) має бути (0,0)

Assert.IsNotNull(nearestStation);

Assert.AreEqual(0, nearestStation.X);

Assert.AreEqual(0, nearestStation.Y);

}

[TestMethod]

public void IsStationFree\_FreeStation\_ReturnsTrue()

{

// Arrange

var station = map.Stations.First();

// Act

bool isFree = algorithm.IsStationFree(station, movingRobot, robots);

// Assert

Assert.IsTrue(isFree);

}

[TestMethod]

public void IsStationFree\_OccupiedByFriendlyRobot\_ReturnsFalse()

{

// Arrange

var station = map.Stations.First();

var friendlyRobot = new Robot.Common.Robot

{

Position = station.Position,

OwnerName = "Evelina Bashynska"

};

robots.Add(friendlyRobot);

// Act

bool isFree = algorithm.IsStationFree(station, movingRobot, robots);

// Assert

Assert.IsFalse(isFree);

}

[TestMethod]

public void DoStep\_RobotNotOnStation\_ReturnsMoveCommand()

{

// Arrange - робот не на станції

movingRobot.Position = new Position(2, 2);

// Act

RobotCommand command = algorithm.DoStep(robots, 0, map);

// Assert

Assert.IsInstanceOfType(command, typeof(MoveCommand));

}

[TestMethod]

public void DoStep\_MoveCommand\_HasValidNewPosition()

{

// Arrange

movingRobot.Position = new Position(2, 2);

movingRobot.Energy = 50;

// Act

RobotCommand command = algorithm.DoStep(robots, 0, map);

// Assert

Assert.IsInstanceOfType(command, typeof(MoveCommand));

var moveCommand = (MoveCommand)command;

Assert.IsNotNull(moveCommand.NewPosition);

// Перевіряємо, що нова позиція відрізняється від поточної (якщо є куди рухатись)

if (moveCommand.NewPosition.X != movingRobot.Position.X ||

moveCommand.NewPosition.Y != movingRobot.Position.Y)

{

// Перевіряємо, що рух не витрачає більше енергії, ніж є у робота

int energyNeeded = DistanceHelper.FindDistance(movingRobot.Position, moveCommand.NewPosition);

Assert.IsTrue(energyNeeded <= movingRobot.Energy,

$"Рух вимагає {energyNeeded} енергії, але у робота тільки {movingRobot.Energy}");

}

}

}

}

**РЕЗУЛЬТАТИ**

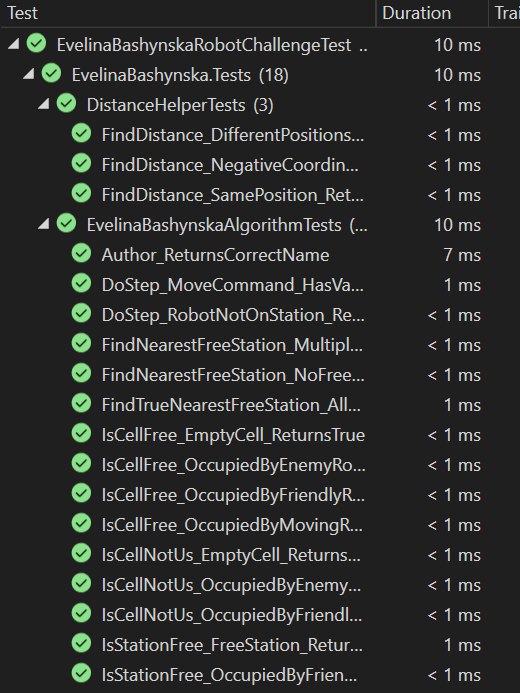


Рис. 1. Результат проходження юніт-тестів

**ВИСНОВКИ**

Виконуючи цю лабораторну роботу, я ознайомилась з засобами розробки Visual Studio та Resharper. Після виконання лабораторної я навчилась створювати проекти, підключати бібліотеки, відлагоджувати програми у середовищі розробки Visual Studio.

Для виконання цієї лабораторної роботи зокрема потрібно було пригадати основи мови програмування C#, що я і зробив за допомогою лекційних матеріалів, методичних вказівок та веб-ресурсів. У своєму проекті я застосував такі особливості мови C#, як: колекції, зокрема List, для зберігання списку роботів у тестових методах; попрацював з подіями, зокрема для визначення номера раунду.

Також я ознайомився із темою Unit-тестування, яку успішно зміг застосувати у своєму проекті та розробив 10 тестів, які перевіряли роботу розроблених мною функцій та роботу мого алгоритму загалом.