Platformy programistyczne .Net i Java

Projekt aplikacji .Net z prostym problemem optymalizacyjnym

Mateusz Marciak 272599

Opis Projektu

Utworzona aplikacja w języku C# jest rozwiązaniem problemu optymalizacyjnego, na podstawie znanego problemu plecakowego. Następnie dla zaimplantowanego rozwiązania utworzono test oraz proste GUI umożliwiające użytkownikowi wpisaniu liczby przedmiotów, maksymalną wagę oraz seed dla losowania wartości. W projekcie użyto:

- Platformę .NET 8.0
- Framework do testów MSTest Project
- Windows Forms App dla GUI

Problem plecakowy

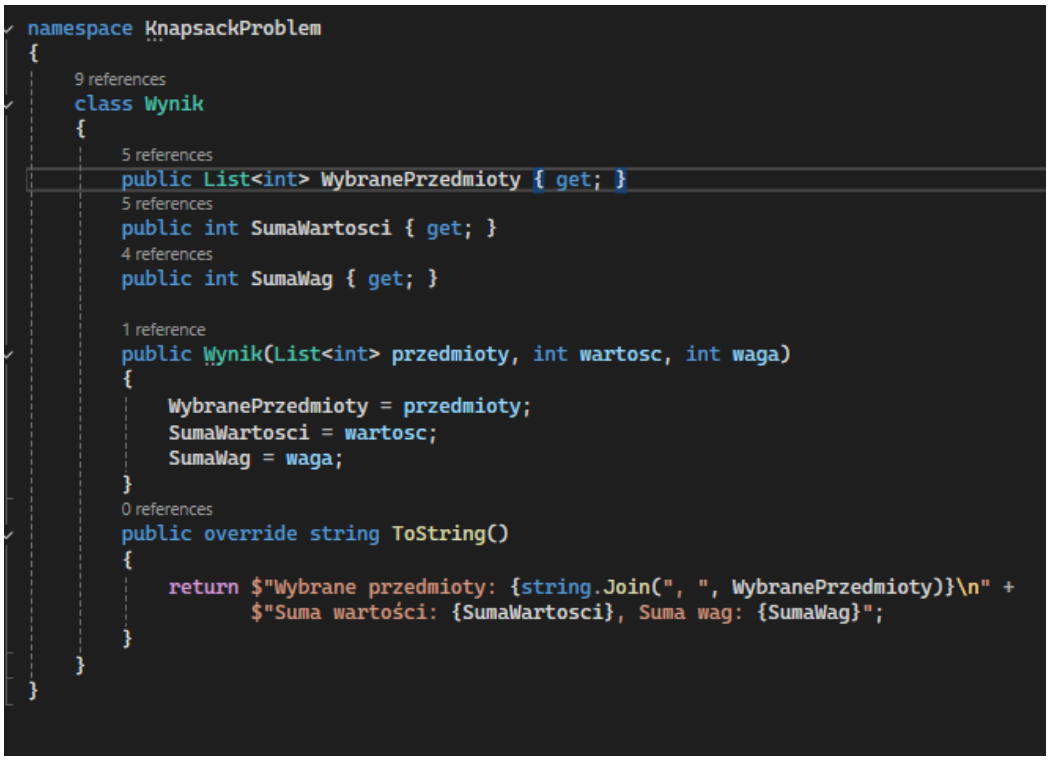
Problem w projekcie został zdefiniowany przy pomocy 3 klas, przedmiotu , wyniku oraz problemu. Algorytm opiera się na podaniu przez użytkownika liczby przedmiotów, maksymalnej pojemności plecaka oraz seed dla losowania instancji problemu.

Klasa przedmiot ma przypisaną swoją wagę oraz wartość, które są generowane losowo.

```
namespace KnapsackProblem
{
    4 references
    class Przedmiot
    {
        4 references
        public int Numer { get; }
        7 references
        public int Waga { get; }
        5 references
        public int Wartosc { get; }

        1 reference
        public Przedmiot(int numer, int waga, int wartosc)
        {
            Numer = numer;
            Waga = waga;
            Wartosc = wartosc;
        }

        0 references
        public override string ToString()
        {
            return $"Przedmiot {Numer}: Waga = {Waga}, Wartość = {Wartosc}";
        }

    }
}
```

Rysunek 1 Klasa przedmiot

Klasa wynik posiada listę wybranych przedmiotów, ich łączną wagę oraz łączną wartość.



```
namespace KnapsackProblem
{
    9 references
    class Wynik
    {
        5 references
        public List<int> WybranePrzedmioty { get; }
        5 references
        public int SumaWartosci { get; }
        4 references
        public int SumaWag { get; }

        1 reference
        public Wynik(List<int> przedmioty, int wartosc, int waga)
        {
            WybranePrzedmioty = przedmioty;
            SumaWartosci = wartosc;
            SumaWag = waga;
        }
        0 references
        public override string ToString()
        {
            return $"Wybrane przedmioty: {string.Join(", ", WybranePrzedmioty)}\n" +
                   $"Suma wartości: {SumaWartosci}, Suma wag: {SumaWag}";
        }
    }
}
```

Rysunek 2 Klasa wynik

Klasa problem definiuje sposób rozwiązania problemu, który opiera się na posortowaniu wartości rosnąco i wyszukanie tych które dla najmniejszej wagi mają największą wartość.

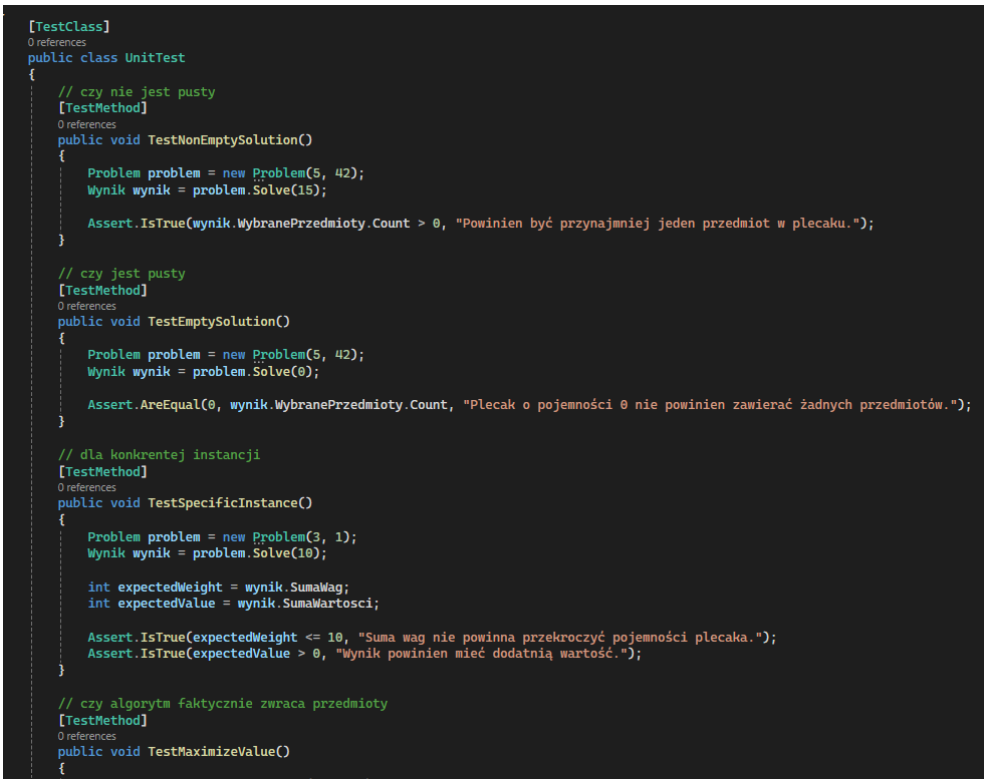
```
class Problem
{
    1 reference
    public int LiczbaPrzedmiotów { get; }
    5 references
    public List<Przedmiot> Przedmioty { get; } = new List<Przedmiot>();

    7 references
    public Problem(int n, int seed)
    {
        LiczbaPrzedmiotów = n;
        Random random = new Random(seed);

        for (int i = 1; i <= n; i++)
        {
            int waga = random.Next(1, 11);
            int wartosc = random.Next(1, 11);
            Przedmioty.Add(new Przedmiot(i, waga, wartosc));
        }
    }

    0 references
    public override string ToString()
    {
        return string.Join("\n", Przedmioty);
    }

    6 references
    public Wynik Solve(int capacity)
    {
        var posortowane = Przedmioty.OrderByDescending(p => (double)p.Wartosc / p.Waga).ToList();
        List<int> wybranePrzedmioty = new List<int>();
        int sumaWag = 0;
        int sumaWartosci = 0;

        foreach(var przedmiot in posortowane)
        {
            if (sumaWag+przedmiot.Waga <= capacity)
            {
                wybranePrzedmioty.Add(przedmiot.Numer);
                sumaWag += przedmiot.Waga;
                sumaWartosci += przedmiot.Wartosc;
            }
        }

        return new Wynik(wybranePrzedmioty,sumaWartosci,sumaWag);
```

Rysunek 3 Klasa problem

Testy

W projekcie wykonano 5 testów. W tym celu tworzono konkretne instancje problemu, a następnie używając funkcji Assert, sprawdzano czy zwrotny wynik jest zgodny z oczekiwaniami. Sprawdzone zostały aspekty:

- Czy plecak jest pusty
- Czy plecak nie jest pusty
- Czy zwrócona wartość jest zgodna z prawdą
- Czy zwrócone przedmioty rzeczywiście istnieją
- Czy przedmiot nie ma ujemnej wagi



```
[TestClass]
0 references
public class UnitTest
{
    // czy nie jest pusty
    [TestMethod]
    0 references
    public void TestNonEmptySolution()
    {
        Problem problem = new Problem(5, 42);
        Wynik wynik = problem.Solve(15);

        Assert.IsTrue(wynik.WybranePrzedmioty.Count > 0, "Powinien być przynajmniej jeden przedmiot w plecaku.");
    }

    // czy jest pusty
    [TestMethod]
    0 references
    public void TestEmptySolution()
    {
        Problem problem = new Problem(5, 42);
        Wynik wynik = problem.Solve(0);

        Assert.AreEqual(0, wynik.WybranePrzedmioty.Count, "Plecak o pojemności 0 nie powinien zawierać żadnych przedmiotów.");
    }

    // dla konkrentej instancji
    [TestMethod]
    0 references
    public void TestSpecificInstance()
    {
        Problem problem = new Problem(3, 1);
        Wynik wynik = problem.Solve(10);

        int expectedWeight = wynik.SumaWag;
        int expectedValue = wynik.SumaWartosci;

        Assert.IsTrue(expectedWeight <= 10, "Suma wag nie powinna przekroczyć pojemności plecaka.");
        Assert.IsTrue(expectedValue > 0, "Wynik powinien mieć dodatnią wartość.");
    }

    // czy algorytm faktycznie zwraca przedmioty
    [TestMethod]
    0 references
    public void TestMaximizeValue()
    {
```

Rysunek 4 Fragment implementacji testów

GUI

W tym punkcie dla zaprojektowania interfejsu rozmieszczono widżety takie jak przyciski, pole tekstowe (pokazujące wynik) oraz pole do wpisywania wartości. W tym celu użyte zostało narzędzie **Form Designer,** które pozwala między innymi na ręczne ustawienie layoutu.

Funkcjonalność backendowa uruchamiała się po wciśnięciu przycisku. Najpierw była przeprowadzana walidacja wartości wpisanych (czy są nieujemne i tylko wartości liczbowe całkowite), a następnie wykonywał się algorytm zaimplantowany w pierwszym punkcie.

```
1 reference
private void btnSolve_Click(object sender, EventArgs e)
{
    try
    {

        if (!int.TryParse(txtItemCount.Text, out int itemCount) || itemCount <= 0)
        {
            MessageBox.Show("Wprowadź poprawną liczbę przedmiotów!", "Błąd", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        if (!int.TryParse(txtSeed.Text, out int seed))
        {
            MessageBox.Show("Wprowadź poprawną wartość seeda!", "Błąd", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        if (!int.TryParse(txtCapacity.Text, out int capacity) || capacity < 0)
        {
            MessageBox.Show("Wprowadź poprawną pojemność plecaka!", "Błąd", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }


        Problem problem = new Problem(itemCount, seed);


        txtInstanceResult.Clear();
        txtInstanceResult.AppendText("Wygenerowane przedmioty:" + Environment.NewLine + Environment.NewLine);
        foreach (var item in problem.Przedmioty)
        {
            txtInstanceResult.AppendText($"Numer: {item.Numer}, Waga: {item.Waga}, Wartość: {item.Wartosc}" + Environment.NewLine);
        }

        Wynik wynik = problem.Solve(capacity);

        txtResult.Clear();
        txtResult.AppendText("Rozwiązanie problemu plecakowego:" + Environment.NewLine + Environment.NewLine);

        foreach (var item in wynik.WybranePrzedmioty)
        {
            txtResult.AppendText("Wybrane przedmioty: "+ item + Environment.NewLine);
        }


        txtResult.AppendText(Environment.NewLine + $"Łączna wartość: {wynik.SumaWartosci}" + Environment.NewLine);
        txtResult.AppendText($"Łączna waga: {wynik.SumaWag}" + Environment.NewLine);
    }
    catch (Exception ex)
```

Rysunek 5 Fragment funkcji rozwiązywania

Rezultat

Ostateczny rezultaty prezentował się tak: