Platformy programistyczne .Net i Java

Laboratorium 2

Filip Zioło (272543)

1 Wstęp

Zadanie polegało na zaimplementowaniu programu rozwiązującego problem plecakowy. Pierwszym etapem było stworzenie generatora, który losowo przydzielał wartości dla poszczególnych przedmiotów.
W pierwszej części projektu należało opracować aplikację konsolową realizującą algorytm problemu plecakowego. Następnie wymagane było napisanie pięciu testów jednostkowych – trzech zgodnych z podaną instrukcją oraz dwóch autorskich.
Ostatnim etapem projektu była implementacja graficznej wersji aplikacji, która umożliwiała interaktywne korzystanie z programu.

2 Konfiguracja urządzeń

```
public int[] Weights { get; set; }
public int[] Values { get; set; }
public int MaxCapacity { get; set; }
public int ItemCount { get; set; }
```

W klasie posaiadamy 4 pola dwie tablice przechowywujące wagi i wartości oraz dwie liczby całkowite.

```
public KnapsackProblem()
{
    Console.WriteLine("Enter number of items:");
    int nItems = int.Parse(Console.ReadLine());

    Console.WriteLine("Enter the seed:");
    int seed = int.Parse(Console.ReadLine());

    Console.WriteLine("Enter the maxCapacity:");
    MaxCapacity = int.Parse(Console.ReadLine());

    Initialize(nItems, seed);
}
```

Domyślny konstruktor używany przez aplikacje konsolową do podania wartości

```
public KnapsackProblem()
{
public KnapsackProblem(int itemCount, int seed, int maxCapacity)
{
    MaxCapacity = maxCapacity;
    Initialize(itemCount, seed);
}
}
```

konstruktor przeciążony umożliwia stworzenie obiektu bez wpiswania wartości w konsoli, używany np. do unittestów.

```
public KnapsackProblem()
{
public Result Solve(int capacity)
{
    List<Item> items = new List<Item>();
    for (int i = 0; i < ItemCount; i++)
    {
        items.Add(new Item(i, Values[i], Weights[i]));
    }

    items = items.OrderByDescending(item => item.Ratio).ToList();

    List<int> chosenItems = new List<int>();
    int totalValue = 0;
    int totalWeight = 0;

    foreach (var item in items)
    {
        if (totalWeight + item.Weight <= capacity)
        {
            chosenItems.Add(item.Index);
            totalWeight += item.Weight;
            totalValue += item.Value;
        }
    }

    return new Result(chosenItems, totalValue, totalWeight);
}
}
```

Główna metoda klasy polegająca na rozwiązaniu problemu plecakowego. Używamy tutaj algorytmu zachłannego. Czyli najpierw sortujemy liste przedmiotów po stosunku wagi do wartości. Następnie dobieramy przedmioty do póki starczy nam miejsca

3 Unit Tests

```
[Test]
public void KnapsackSolver_PoprawneRozwiazanieDlaInstancji()
{
    KnapsackProblem problem = new KnapsackProblem(5, 42, 15);
    Result result = problem.Solve(problem.MaxCapacity);

    Assert.IsTrue(result.BestValue == 21);
}
```

Test sprawdzający Poprawne rozwiązanie dla podanej instancji

```
[Test]
public void KnapsackProblem_DlaBrakuPasujacych()
{
    int itemCount = 5;
    int seed = 42;
    int maxCapacity = 1;

    KnapsackProblem problem = new KnapsackProblem(itemCount, seed,
       maxCapacity);
    for (int i = 0; i < problem.ItemCount; i++)
    {
        problem.Weights[i] = maxCapacity + 1;
    }
    Result result = problem.Solve(maxCapacity);

    Assert.IsEmpty(result.ChosenItems);
}
```

Test sprawdzający czy dla braku pasujących rozwiązań tablica ChosenItems będzie pusta

```
[Test]
public void KnapsackProblem_DlaJednegoPasujacego()
{
    int itemCount = 1;
    int seed = 1;
    int maxCapacity = 3;

    KnapsackProblem problem = new KnapsackProblem(itemCount, seed,
       maxCapacity);
    Result result = problem.Solve(maxCapacity);

    Assert.IsTrue(result.TotalWeight == problem.Weights[0]);
}
```

Test sprawdzający czy jeżeli jeden przedmiot będzie spełniał warunki waga przedmiotu będzie równa wadze całkowitej przedmiotów w plecaku.

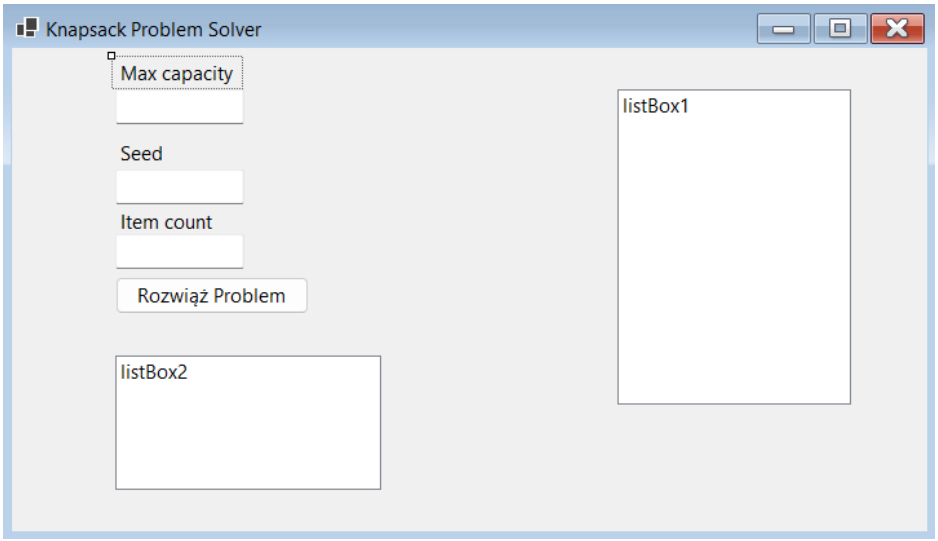
```
[Test]
public void KnapsackProblem_WszystkiePrzedmiotyMieszczaSie()
{
    int itemCount = 4;
    int seed = 10;
    int maxCapacity = 100;

    KnapsackProblem problem = new KnapsackProblem(itemCount, seed,
       maxCapacity);

    problem.Weights = new int[] { 10, 20, 30, 40 };
    problem.Values = new int[] { 60, 100, 120, 140 };

    Result result = problem.Solve(maxCapacity);

    Assert.AreEqual(itemCount, result.ChosenItems.Count);
}
```

Test sprawdzajacy czy wszystkie przedmioty podane do algorytmu mieszczą się w plecaku.

```
[Test]
public void KnapsackProblem_PoprawneLosowanie()
{
    List<int> sizes = new List<int>() { 10, 20, 30, 40, 50 };
    foreach (int n in sizes)
    {
        KnapsackProblem problem = new KnapsackProblem(n, 42, 50);
        Assert.AreEqual(n, problem.Values.Length);
    }
    Assert.Pass();
}
```

Test sprawdzający czy losowanie przedmiotów przebiega poprawnie.

4 Aplikacja okienkowa



Rysunek 1: Wygląd okna apliakcji

Aplikacja posiadała 3 pola do wpisania wartości, kolejno maksymalnej pojemności plecaka, seed'u do generatora liczb pseudo losowych oraz ilości przedmiotów które zostaną wygenerowane. ListBox1 przechowywał wszystkie wygenerowane przedmioty natomiast ListBox2 przechowywał wybrane przedmioty sume ich wartości oraz sume wag wszystkich wybranych przedmiotów.

```
if (!int.TryParse(textBox1.Text, out int itemCount) || itemCount <= 0)
{
    errorMessage += "Error dla 1\n";
    isValid = false;
}

if (!int.TryParse(textBox2.Text, out int seed) || seed < 0)
{
    errorMessage += "Error dla 2\n";
    isValid = false;
}

if (!int.TryParse(textBox3.Text, out int maxCapacity) || maxCapacity <=
   0)
{
    errorMessage += "Error dla 3\n";
    isValid = false;
}
```

Fragment kodu odpowiadający za walidacje danych wejściowych. Sprawdza on każdą wartość i jeżeli jakaś jest nie poprawna dodaje do ErrorMessage odpowiedniąwiadomość.

```
listBox1.Items.Clear();
for (int i = 0; i < knapsackProblem.ItemCount; i++)
{
    listBox1.Items.Add($"Item {i}: Value={knapsackProblem.Values[i]},
       Weight={knapsackProblem.Weights[i]}");
}
```

Wyświetlanie wygenerowanych przedmiotów poprzez dodawanie każdego jako "Item" w listBoxie.