# Design Patterns RESUME

User and Developer Manual

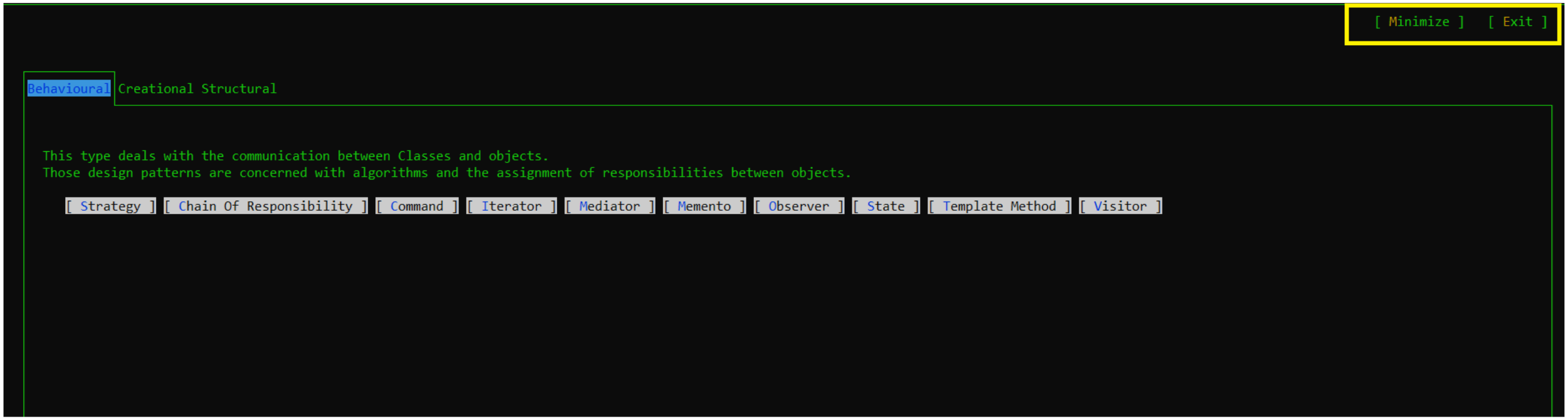
# Console Project – application overview

The application is simply divided in 3 tab sections in which the main design patterns categories are presented.

For each category there are the list (buttons) of available design patterns

# Console Project – fixed buttons

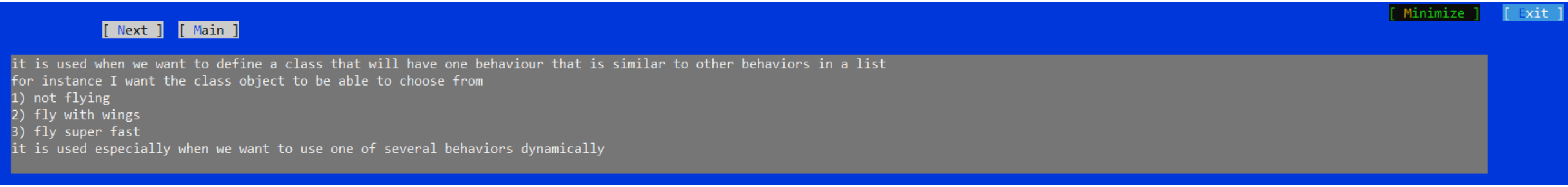
Exit and minimize buttons are always available in the context: you can minimize the application (which is set to full screen) or exit from it at anytime



# Console Project – page description

For each of the available pattern there are 2 types of pages. The first one in the description page, in which only a main description is shown.

The description page has the following view (blue screen and gray characters):



# Console Project – example description

The second available page is a page of example. In this context a brief description of the example is given and a second view renders the class of the example at hand.

The example page has the following characteristics:

[ Prev ] [ Next ] [ Main ] [ Wrong ]

how to give to bird and other animals (which extends animal interface) the ability to fly?
1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal
I don't want to give all the animals the ability to fly and override that method

**brief description for the case**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
  /// <summary>
  /// This is a class representing a generic animal in a first implementation
  /// </summary>
  public class Animal_Implementation1
  {
    private string _name;
    private double _height;
    private int _weight;
    private string _favFood;
    private double _speed;
    private string _sound;

    public string Name { get => _name; set => _name = value; }
```

**code example (from a project class)**

# Console Project – wrong button

When the current example has a counterexample, it is often showed clicking on the wrong button.

Another instance of the application will start showed a console with the wrong class case:

```
[ Prev ]  [ Next ]  [ Main ]  [ Wrong ]                                  [ Minimize ]

how to give to bird and other animals (which extends animal interface) the ability to fly?
1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal
I don't want to give all the animals the ability to fly and override that method
```

**new instance for the application**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
  /// <summary>
  /// This is a class representing a generic animal in a first implementation
  /// </summary>
  public class Animal_Implementation1
  {
    private string _name;
    private double _height;
    private int _weight;
    private string _favFood;
    private double _speed;
    private string _sound;

    public string Name { get => _name; set => _name = value; }

    public double Height { get => _height; set => _height = value; }
```

```
D:\GIT\DesignPatterns_RESUME\DesignPatterns\bin\Debug\DesignPatterns.exe
    private int _weight;
    private string _favFood;
    private double _speed;
    private string _sound;

    public string Name { get => _name; set => _name = value; }

    public double Height { get => _height; set => _height = value; }

    private int Weight { get => _weight; set => _weight = value; }

    private string FavFood { get => _favFood; set => _favFood = value; }

    private double Speed { get => _speed; set => _speed = value; }

    public string Sound { get => _sound; set => _sound = value; }

    /// <summary>
    /// The "wrong" implementation
    /// </summary>
    public void Fly()
    {
      Console.WriteLine("I'm flying");
    }
  }
}
```

**RED color for the wrong lines**

# Console Project – prev – next buttons

For moving forward and behind in the demo, the previous and next page buttons are rendered in the page (previous on left and next on right). Sometimes they change a bit when passing from a description to an example page and viceversa:

```
[ Prev ] [ Next ] [ Main ]                                  [ Wrong ]

how to give to bird and other animals (which extends animal interface) the ability to fly?
1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal
I don't want to give all the animals the ability to fly and override that method

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
  /// <summary>
  /// This is a class representing a generic animal in a first implementation
  /// </summary>
  public class Animal_Implementation1
  {
    private string _name;
    private double _height;
    private int _weight;
    private string _favFood;
    private double _speed;
    private string _sound;

    public string Name { get => _name; set => _name = value; }
```

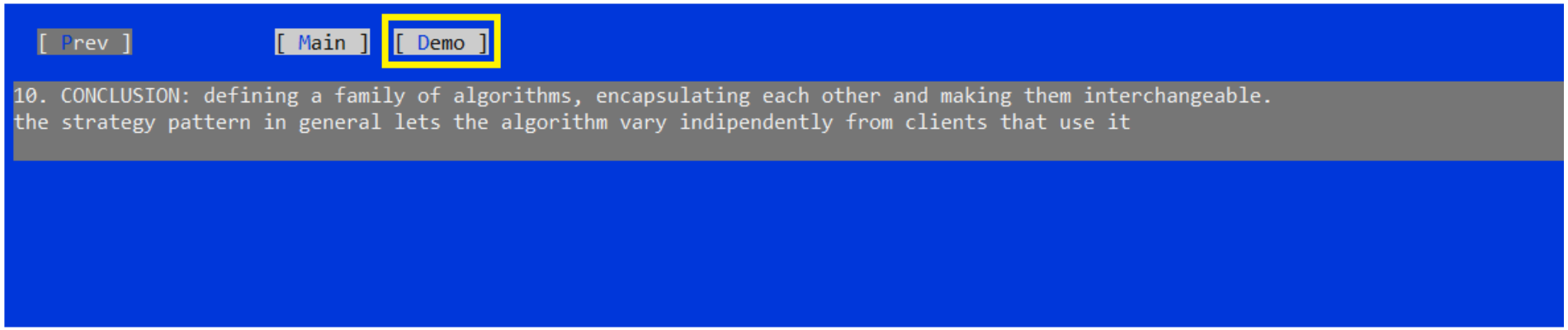
# Console Project – Main button

For the presentation the main button is always available: with it, it is always possible to come back to the original context (the tabular page with all the alternatives available):

```
[ Prev ]  [ Next ]  [ Main ]                                  [ Wrong ]

how to give to bird and other animals (which extends animal interface) the ability to fly?
1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal
I don't want to give all the animals the ability to fly and override that method

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
  /// <summary>
  /// This is a class representing a generic animal in a first implementation
  /// </summary>
  public class Animal_Implementation1
  {
    private string _name;
    private double _height;
    private int _weight;
    private string _favFood;
    private double _speed;
    private string _sound;

    public string Name { get => _name; set => _name = value; }

    public double Height { get => _height; set => _height = value; }
```

# Console Project – demo button

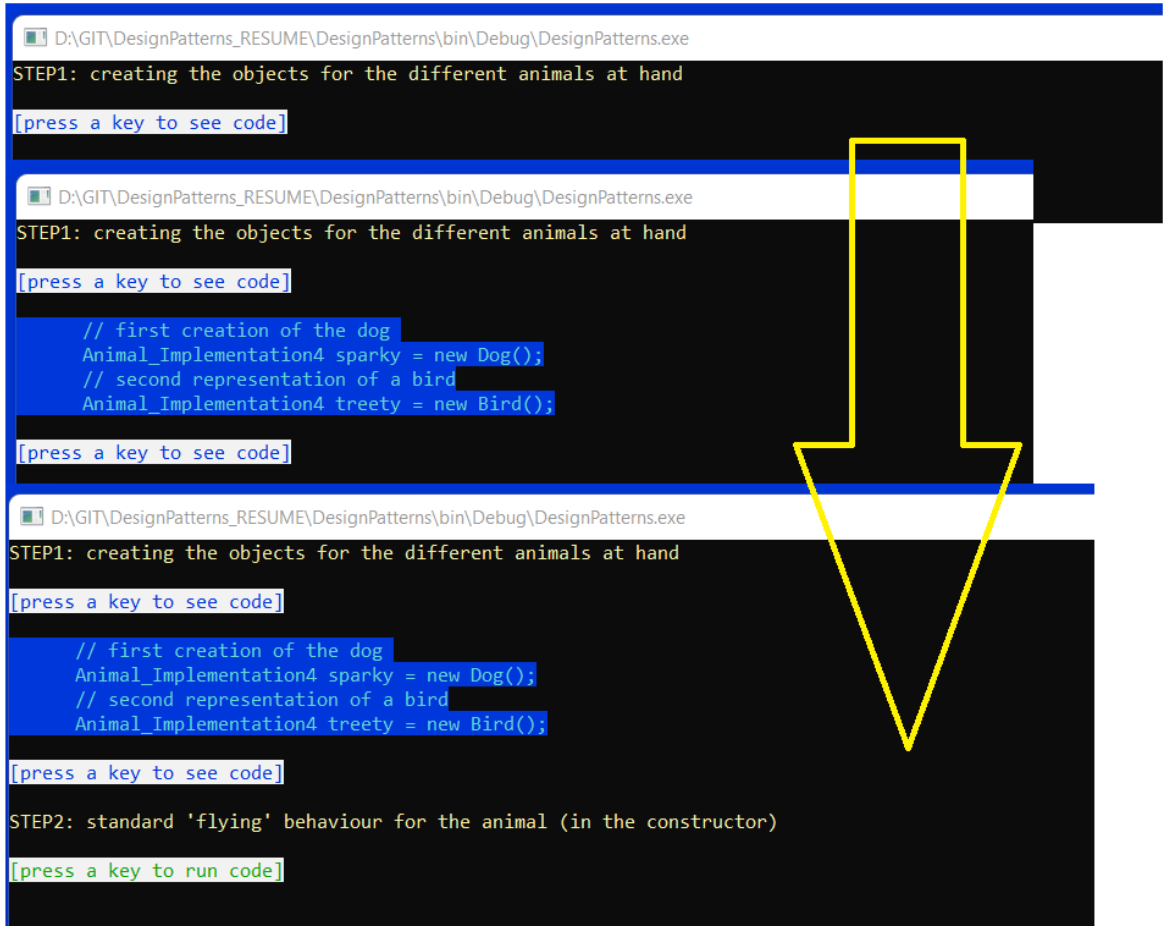
When concluding the presentation part, it will be possible seeing the demo too. This is available on the last page of each of the presented patterns:



# Console Project – demo

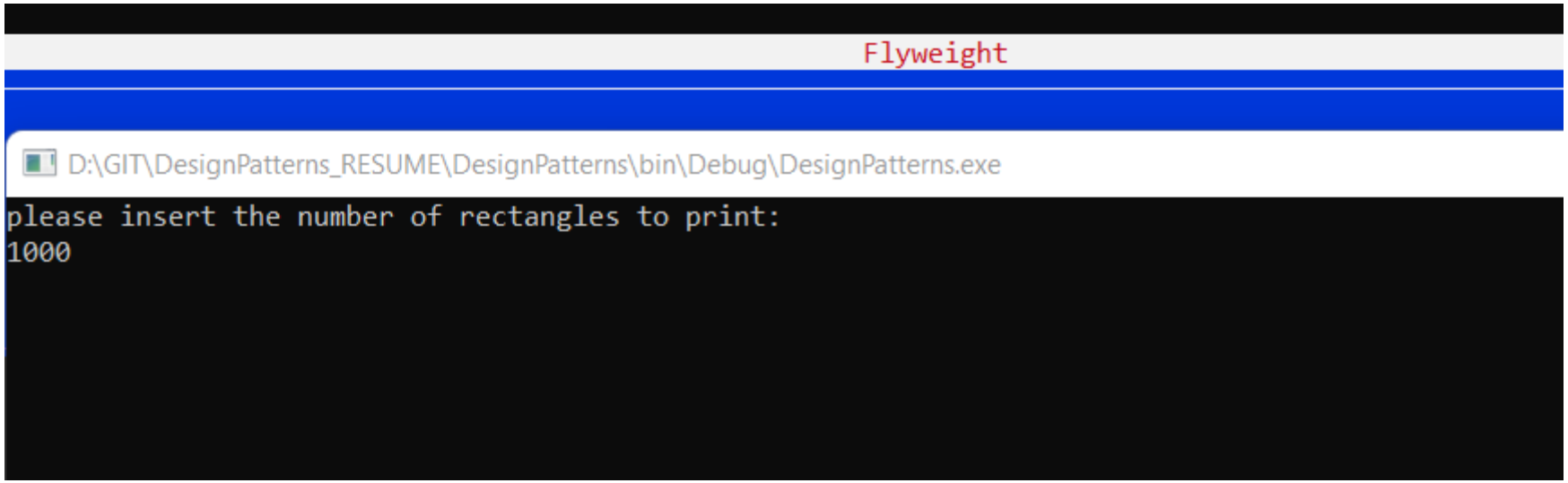
Demo is contained in another instance of the application: it could vary depending on the selected pattern.

Usually a DEMO is made such that it is stopped at each line or block of lines of relevant code, given precise information to the user concerning what is happening:



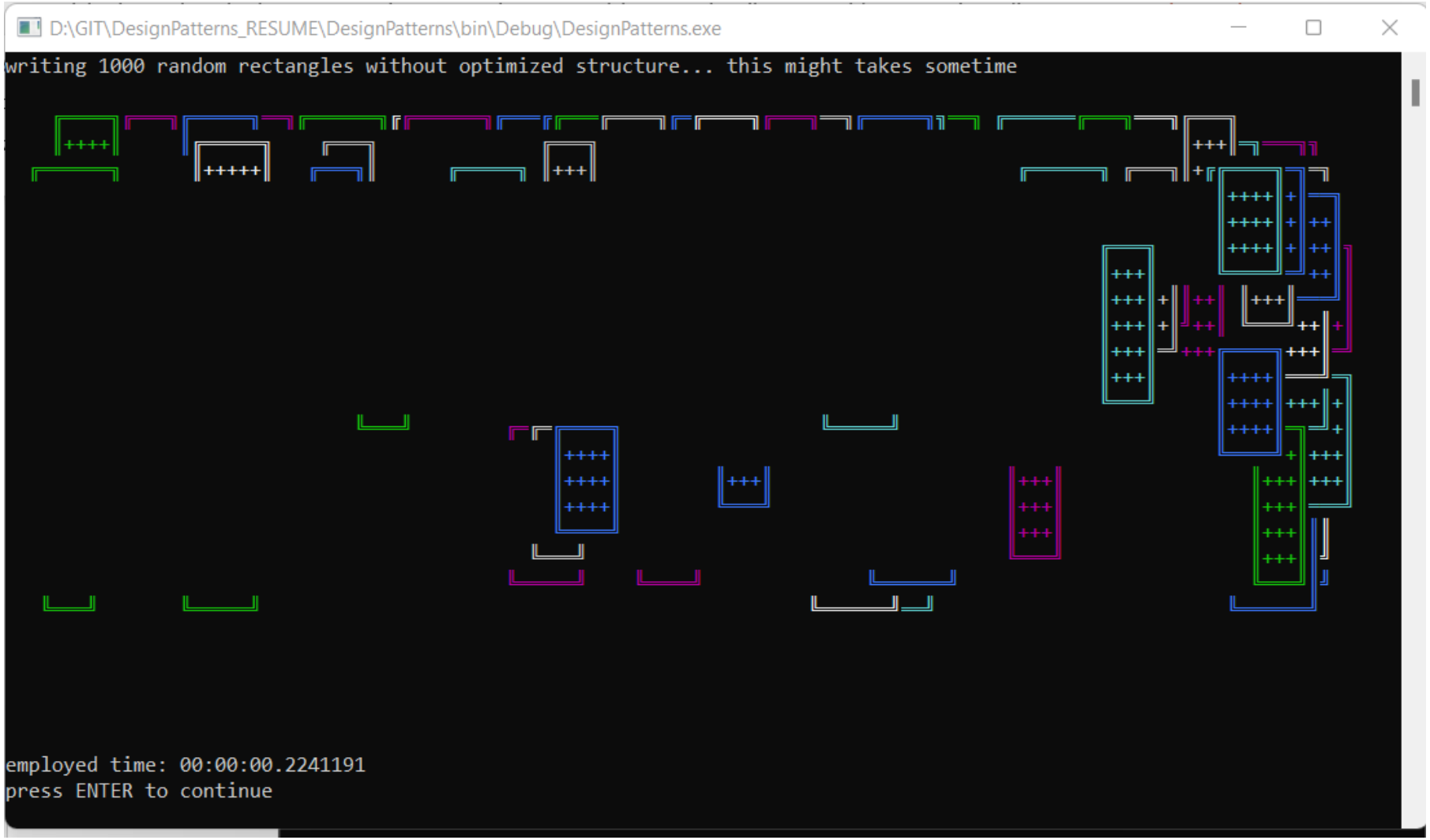
# Console Project – demo

But sometimes it could be also represented by a different application with a different logic. In any case instructions are presented to the user:



# Console Project – demo

But sometimes it could be also represented by a different application with a different logic. In any case instructions are presented to the user:



# Console Project – demo

But sometimes it could be also represented by a different application with a different logic. In any case instructions are presented to the user: