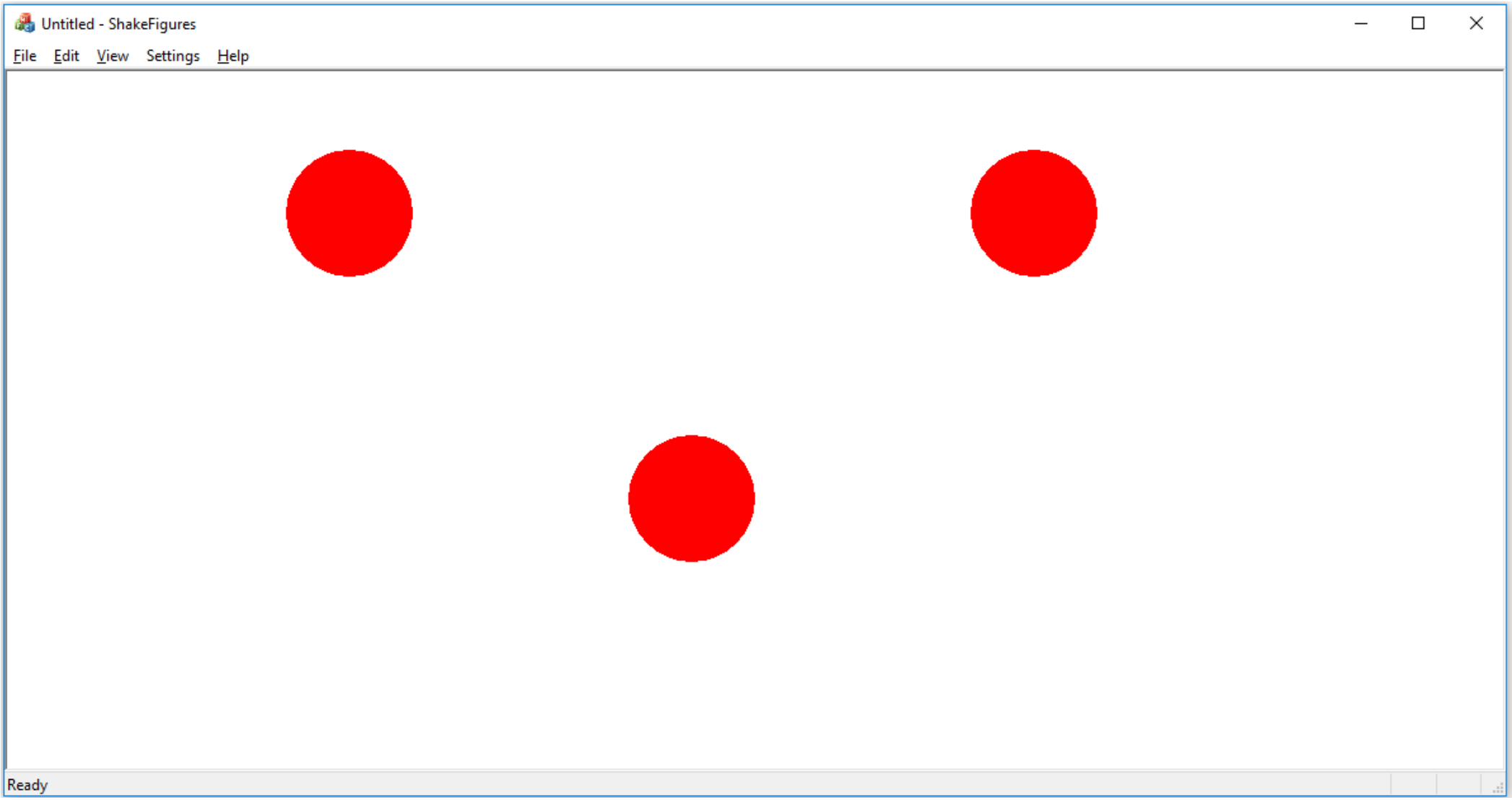
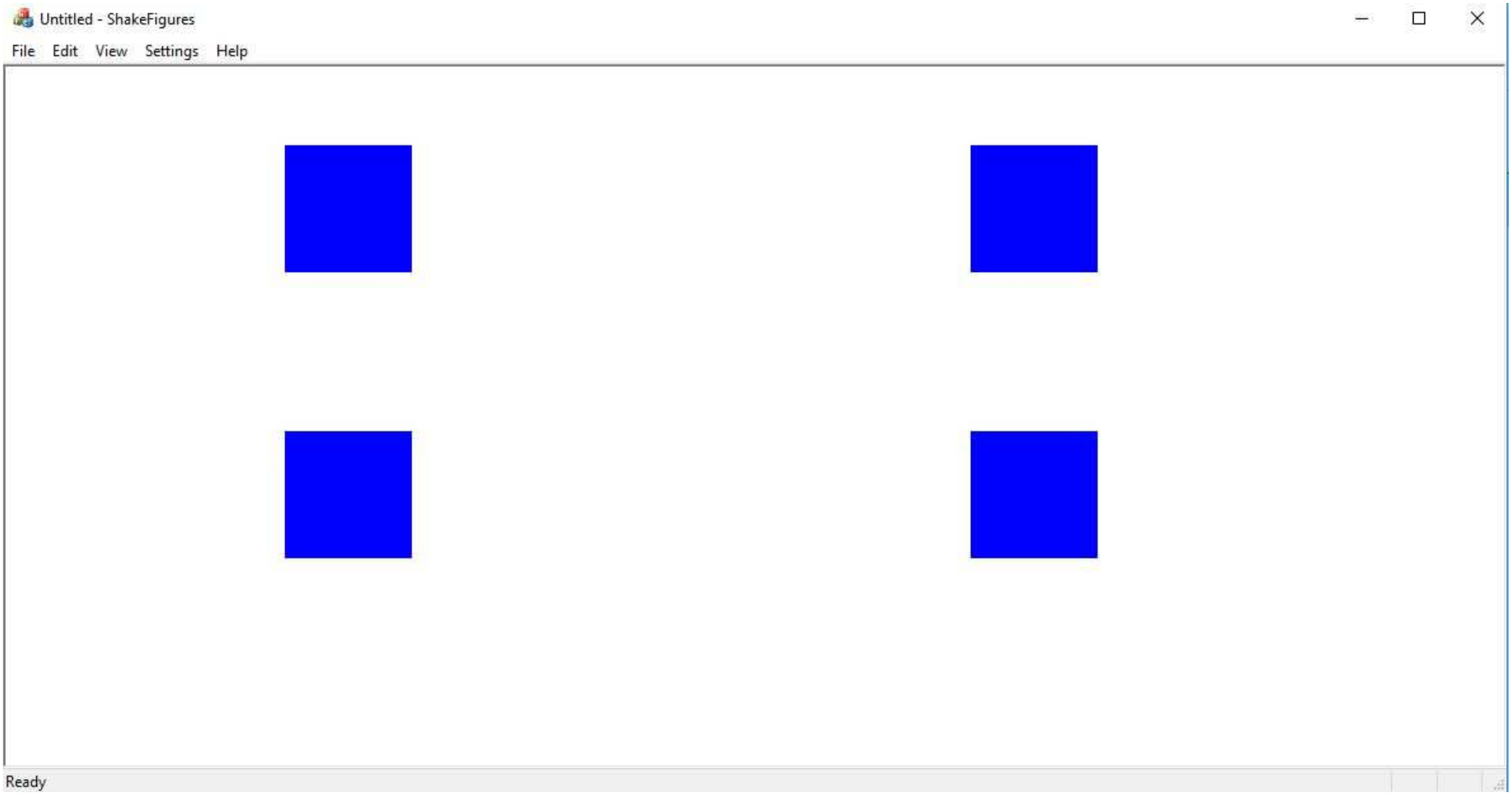
ShakeFigures



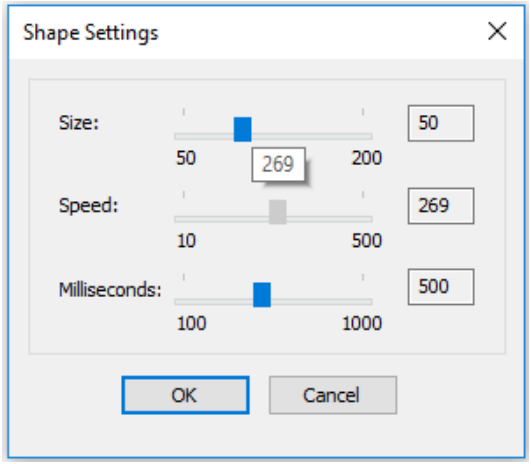
ShakeFigures



The Settings menu

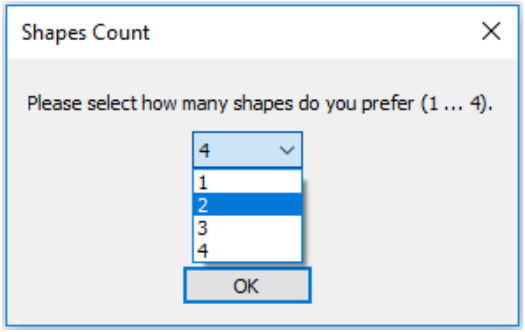
The Settings menu exposes two options:

- Shapes settings
- Shapes count



The parameters are:

- The size width and height of the shapes in pixels
- The shape's move "speed" – the shape's overhead after a timer tick
- The timer execution frequency in milliseconds.



The Shapes Count dialog allows user to specify the number of shapes want to be generated. By default, this number is 4.

Implementation details

The SDI's view class (**CShakeFiguresView**) contains an instance of **CShapesManager**, a shapes manager class that is responsible with the shapes instantiation, shapes movement operation and parameters changing.

Depending on the shapes count and the type the shape manager receives commands from the viewer, a shape factory generates the shapes: ellipses and fulfilled rectangles. In case the shapes hierarchy is extended, the factory must be extended in order to supports generation of such shapes types.

```
std::unique_ptr<CShape> CShapesManager::ShapesFactory(SHAPE_TYPE shape_type, CPoint shapeCenterPoint)
{
    switch (shape_type)
    {
        case SHAPE_TYPE::ellipse:
            return std::make_unique<CEllipseShape>(shapeCenterPoint,
                                                   m_shapeWidth, m_shapeHeight,
                                                   m_nInitialSpeed, m_moveAreaRect);
            break;

        case SHAPE_TYPE::rectangle:
        default:
            return std::make_unique<CRectangleShape>(shapeCenterPoint,
                                                     m_shapeWidth, m_shapeHeight,
                                                     m_nInitialSpeed, m_moveAreaRect);
            break;
    }
}
```

The shapes class hierarchy

The generated shapes are inherited from an abstract class **CShape**.
This class contains the common attributes and method, but also enforce the polymorphism implementation for the drawing action.
Base on that, each derived class that implements an effective shape have to Implement the drawing action, in a different specific way.

In case we want to generate other shapes we have to implement different class (ex. **COtherShape**) derived from **CShape** and to implement different drawing action.

Even if I am using **std::unique_ptr**, the CShape class contains copy constructor and assign operator in case the class instantiation is done by classical raw pointers, old style.