Übung 3

Table of Contents

1. Simple Mandelbrot generator

Mandelbrot generation was already finished in the lesson. This is the code for the synchronous image generation.

Listing 1. Image generation

```
using System;
using System.Diagnostics;
using System.Drawing;

namespace MandelbrotGenerator.Generators
{
    public class SyncImageGenerator : IImageGenerator
    {
        #region Public Events

        public event EventHandler<ImageGeneratedEventArgs> ImageGenerated;

        #endregion

        #region Public Methods

        public void GenerateImage(Area area)
        {
            Stopwatch stopWatch = Stopwatch.StartNew();

            int maxIterations;
            double zBorder;
            double cReal, cImg, zReal, zImg, zNewReal, zNewImg;

            maxIterations = Settings.DefaultSettings.MaxIterations;
            zBorder = Settings.DefaultSettings.ZBorder * Settings.DefaultSettings.ZBorder;

            Bitmap bitmap = new Bitmap(area.Width, area.Height);

            for (int x = 0; x < area.Width; x++)
            {
                cReal = area.MinReal + x * area.PixelWidth;
                for (int y = 0; y < area.Height; y++)
                {
                    cImg = area.MinImg + y * area.PixelHeight;
                    zReal = 0.0;
                    zImg = 0.0;
                    int iteration = 0;
                    while (iteration < maxIterations // Check if smaller max iterations
                        && zReal * zReal + zImg * zImg < zBorder) // Check if in border
                    {
                        zNewReal = zReal * zReal - zImg * zImg + cReal;
                        zNewImg = zReal * zImg * 2 + cImg;
                        zReal = zNewReal;
                        zImg = zNewImg;
                        iteration += 1;
                    }
                    bitmap.SetPixel(x, y, ColorSchema.GetColor(iteration));
                }
```

```
            }

            stopWatch.Stop();
            ImageGenerated?.Invoke(this, new ImageGeneratedEventArgs(bitmap, area, stopWatch
.Elapsed));

        }

        #endregion
    }
}
```

2. Worker thread

2.1. Solution with thread

One possible solution would be to create each time a new thread which starts the image execution. If the method is currently executing an image generation, the previous generation gets cancelled via `CancellationTokenSource`. Additionally if the image was generated successfully, an event gets fired where the new image, area and time gets passed with `EventArgs`.

Listing 2. Image generator with thread

```
using System;
using System.Diagnostics;
using System.Drawing;
using System.Threading;

namespace MandelbrotGenerator.Generators
{
    public class AsyncThreadImageGenerator : IImageGenerator
    {
        #region Private Fields

        private CancellationTokenSource cancellationTokenSource = new CancellationTokenSource();
        private Thread thread;

        #endregion

        #region Public Events

        public event EventHandler<ImageGeneratedEventArgs> ImageGenerated;

        #endregion

        #region Public Methods

        public static Bitmap GenerateImage(Area area, CancellationToken cancellationToken)
        {
            int maxIterations;
            double zBorder;
            double cReal, cImg, zReal, zImg, zNewReal, zNewImg;

            maxIterations = Settings.DefaultSettings.MaxIterations;
            zBorder = Settings.DefaultSettings.ZBorder * Settings.DefaultSettings.ZBorder;

            Bitmap bitmap = new Bitmap(area.Width, area.Height);

            for (int x = 0; x < area.Width; x++)
            {
                if (cancellationToken.IsCancellationRequested)
                {
                    return null;
                }
```

```
                cReal = area.MinReal + x * area.PixelWidth;
                for (int y = 0; y < area.Height; y++)
                {
                    cImg = area.MinImg + y * area.PixelHeight;
                    zReal = 0.0;
                    zImg = 0.0;
                    int iteration = 0;
                    while (iteration < maxIterations // Check if smaller max iterations
                        && zReal * zReal + zImg * zImg < zBorder) // Check if in border
                    {
                        zNewReal = zReal * zReal - zImg * zImg + cReal;
                        zNewImg = zReal * zImg * 2 + cImg;
                        zReal = zNewReal;
                        zImg = zNewImg;
                        iteration += 1;
                    }
                    bitmap.SetPixel(x, y, ColorSchema.GetColor(iteration));
                }
            }

            return bitmap;
        }

        public void GenerateImage(Area area)
        {
            cancellationTokenSource.Cancel(); // Cancel previous calculation
            cancellationTokenSource = new CancellationTokenSource(); // Create new cancellation
source
            var token = cancellationTokenSource.Token;
            thread = new Thread(() =>
            {
                var watch = Stopwatch.StartNew();
                var bitmap = GenerateImage(area, token);
                var args = new ImageGeneratedEventArgs(bitmap, area, watch.Elapsed);
                if (!token.IsCancellationRequested)
                {
                    ImageGenerated?.Invoke(this, args);
                }
            });
            thread.Start();
        }

        #endregion
    }
}
```

2.2. Solution with background worker

BackgroundWorker is used for image generation. Previous worker gets cancelled if new worker gets created and started. Two callback methods are used: `DoWork` and `RunWorkerCompleted`. Additionally the flag `WorkerSupportsCancellation` has to be set.

Listing 3. Image generator with background worker

```
using MandelbrotGenerator.Generators;
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Drawing;

namespace MandelbrotGenerator
{
    public class BackgroundWorkerImageGenerator : IImageGenerator
    {
        #region Private Fields

        private BackgroundWorker backgroundWorker;

        #endregion

        #region Public Constructors

        public BackgroundWorkerImageGenerator()
        {
            InitializeBackgroundWorker();
        }

        #endregion

        #region Public Events

        public event EventHandler<ImageGeneratedEventArgs> ImageGenerated;

        #endregion

        #region Private Methods

        private static Bitmap GenerateImage(Area area, BackgroundWorker worker, DoWorkEventArgs
e)
        {
            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return null;
            }

            int maxIterations;
            double zBorder;
            double cReal, cImg, zReal, zImg, zNewReal, zNewImg;

            maxIterations = Settings.DefaultSettings.MaxIterations;
            zBorder = Settings.DefaultSettings.ZBorder * Settings.DefaultSettings.ZBorder;

            Bitmap bitmap = new Bitmap(area.Width, area.Height);

            for (int x = 0; x < area.Width; x++)
            {
                if (worker.CancellationPending)
                {
                    e.Cancel = true;
                    return null;
                }
                cReal = area.MinReal + x * area.PixelWidth;
                for (int y = 0; y < area.Height; y++)
```

```
            {
                cImg = area.MinImg + y * area.PixelHeight;
                zReal = 0.0;
                zImg = 0.0;
                int iteration = 0;
                while (iteration < maxIterations // Check if smaller max iterations
                    && zReal * zReal + zImg * zImg < zBorder) // Check if in border
                {
                    zNewReal = zReal * zReal - zImg * zImg + cReal;
                    zNewImg = zReal * zImg * 2 + cImg;
                    zReal = zNewReal;
                    zImg = zNewImg;
                    iteration += 1;
                }
                bitmap.SetPixel(x, y, ColorSchema.GetColor(iteration));
            }
        }

        return bitmap;
    }

    private void Completed(object sender, RunWorkerCompletedEventArgs e)
    {
        if (e.Cancelled)
        {
            return;
        }

        ImageGenerated?.Invoke(this, (ImageGeneratedEventArgs)e.Result);
    }

    private void InitializeBackgroundWorker()
    {
        backgroundWorker = new BackgroundWorker();

        backgroundWorker.DoWork += Run;
        backgroundWorker.WorkerSupportsCancellation = true;
        backgroundWorker.RunWorkerCompleted += Completed;
    }

    private void Run(object sender, DoWorkEventArgs e)
    {
        var area = (Area)e.Argument;

        var watch = Stopwatch.StartNew();
        var bitmap = GenerateImage(area, sender as BackgroundWorker, e);
        var args = new ImageGeneratedEventArgs(bitmap, area, watch.Elapsed);

        e.Result = args;
    }

    #endregion

    #region Public Methods

    public void GenerateImage(Area area)
    {
        if (backgroundWorker.IsBusy)
        {
            backgroundWorker.CancelAsync();
```

```
                InitializeBackgroundWorker();
            }

            backgroundWorker.RunWorkerAsync(area);
        }

        #endregion
    }
}
```

3. Parallelize work

The whole areal gets separated into multiple columns (depending on the worker count setting). Each part gets calculated separately. That means that for each each part an separate thread gets created. After the generation of each part they get merged into one bitmap.

Listing 4. Image generator with parallel generator

```
using System;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Threading;

namespace MandelbrotGenerator.Generators
{
    public class MultiAsyncThreadImageGenerator : IImageGenerator
    {
        #region Private Fields

        private Bitmap[] bitmaps;

        private CancellationTokenSource cancellationToken;

        #endregion

        #region Public Events

        public event EventHandler<ImageGeneratedEventArgs> ImageGenerated;

        #endregion

        #region Private Methods

        private static Bitmap GenerateImagePart(Area area, int startWidth, int endWidth,
CancellationToken token)
        {
            if (token.IsCancellationRequested)
            {
                return null;
            }

            Bitmap bitmap = new Bitmap(endWidth - startWidth, area.Height);
            int maxIterations = Settings.DefaultSettings.MaxIterations;
            double zBorder = Settings.DefaultSettings.ZBorder * Settings.DefaultSettings.
ZBorder;
            double cReal, cImg, zReal, zImg, zNewReal, zNewImg;

            for (int x = startWidth; x < endWidth; x++)
            {
                if (token.IsCancellationRequested)
                {
                    return null;
                }
```

```
            cReal = area.MinReal + x * area.PixelWidth;
            for (int y = 0; y < area.Height; y++)
            {
                cImg = area.MinImg + y * area.PixelHeight;
                zReal = 0.0;
                zImg = 0.0;
                int iteration = 0;
                while (iteration < maxIterations // Check if smaller max iterations
                       && zReal * zReal + zImg * zImg < zBorder) // Check if in border
                {
                    zNewReal = zReal * zReal - zImg * zImg + cReal;
                    zNewImg = zReal * zImg * 2 + cImg;
                    zReal = zNewReal;
                    zImg = zNewImg;
                    iteration += 1;
                }

                bitmap.SetPixel(x - startWidth, y, ColorSchema.GetColor(iteration));
            }
        }

        return bitmap;
    }

    private void GenerateImagePart(object obj)
    {
        var tuple = (Tuple<Area, int, int, int, CancellationToken>)obj;
        var area = tuple.Item1;
        var index = tuple.Item4;
        var cancellationToken = tuple.Item5;

        var sw = Stopwatch.StartNew();
        var bitmap = GenerateImagePart(area, tuple.Item2, tuple.Item3, cancellationToken);
        sw.Stop();

        OnImageGenerated(area, bitmap, sw.Elapsed, index);
    }

    private Bitmap MergeBitmaps(Area area)
    {
        var result = new Bitmap(area.Width, area.Height);
        using (Graphics graphics = Graphics.FromImage(result))
        {
            var startWidth = 0;
            for (var index = 0; index < bitmaps.Length; index++)
            {
                graphics.DrawImage(bitmaps[index], startWidth, 0);
                startWidth += bitmaps[index].Width;
            }
        }

        return result;
    }

    private void OnImageGenerated(Area area, Bitmap bitmap, TimeSpan elapsed, int index)
    {
        bitmaps[index] = bitmap;
        if (bitmaps.Any(map => map == null))
        {
```

```
                return;
            }

            var resultingBitmap = MergeBitmaps(area);
            var handler = ImageGenerated;
            handler?.Invoke(this, new ImageGeneratedEventArgs(resultingBitmap, area, elapsed));
        }

        #endregion

        #region Public Methods

        public void GenerateImage(Area area)
        {
            // Cancel previous calculations
            cancellationToken?.Cancel(false);
            cancellationToken = new CancellationTokenSource();

            int cols = Settings.DefaultSettings.Workers;
            int fractionWidth = area.Width / cols;

            bitmaps = new Bitmap[cols];

            int startWidth = 0;
            for (int i = 0; i < cols; i++)
            {
                int endWidth = startWidth + fractionWidth;

                if (cols > 1 && i == cols - 1) // Fix problems with rounding for last column
                {
                    endWidth += area.Width % cols;
                }

                var thread = new Thread(GenerateImagePart);
                thread.Start(new Tuple<Area, int, int, int, CancellationToken>(area, startWidth,
endWidth, i, cancellationToken.Token));
                startWidth += fractionWidth;
            }
        }

        #endregion
    }
}
```

4. Runtime measurement

The performance of the parallel generator is better than the synchronized one, already with 4 workers. It looks like there is some overhead with the image merging and creating of multiple threads and the calculation for each part. When the number of workers is changed to 8, the generation gets drastically faster. The standard deviation gets also smaller with the parallel execution with 4 workers and even smaller with eight workers.

Run	Syncronized execution	Parallel execution (4 Workers)	Parallel execution (8 workers)
1	643.75680	346.56250	203.52830
2	601.94520	323.52320	211.45730
3	582.95330	330.46690	213.83270
4	583.43150	326.17090	203.23550
5	589.63790	326.66550	204.42670
6	585.59210	325.69180	205.68360
7	595.38610	324.67610	210.55210
8	597.59320	322.12880	206.26040
9	593.22640	339.05350	214.74040
10	588.11790	326.37520	205.36260
STDDEV	17.84348013	7.736533803	4.328813482
AVG	596.16404	329.13144	207.90796