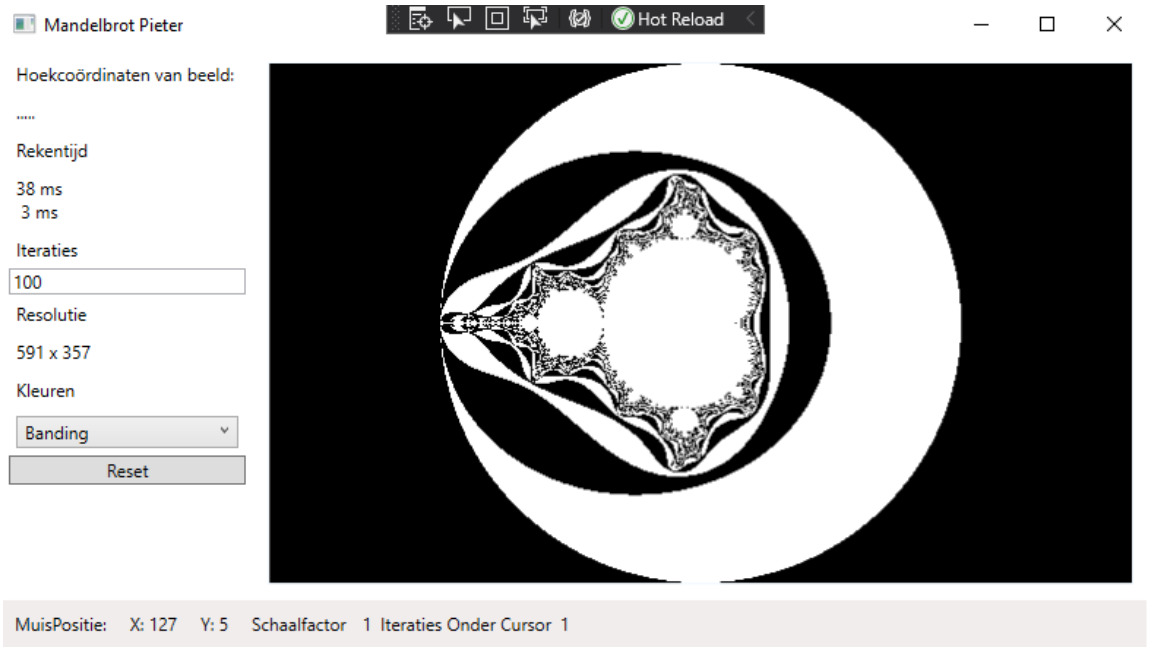
Verslag AP: Mandelbrot

Korte inleiding

De eerste opdracht van het vak aplied programming was het maken van een WPF applicatie die een Mandelbrot weergeeft samen met instelbare parameters.



GUI

De variabelen in het MainWindow.xaml worden bediend via DataBinding. DataBinding creëert een simpele manier om data gemakkelijk tussen verschillende classes te laten vloeien. Mijn Bindings zijn gelinkt aan properties deze properties maken gebruik van een setProperty() methode. Deze methode vergelijkt de lokale private variabele met het nieuwe value, als deze verschillen dan veranderd deze methode de lokale variabele. Het gebruik van properties zorgt dus voor een gemakkelijke manier om data te verplaatsen tussen verschillende klassen.

```
public double ScalingFactor
{
    get => scalingFactor;
    private set
    {
        SetProperty(ref scalingFactor, value);
    }
}
```

Methoden MainViewModel.cs

ResetMandelbrot()

Hersteld alle variabelen om het origineel Mandelbrot bitmap te verkrijgen.

CreateBitmap()

Maakt een start bitmap aan. Deze methode kan meerdere keren opgeroepen worden als de application window veranderd van grootte en dus ook de resolutie veranderd.

SetBitmap(`uint[,] colors`**)**

Deze methode moet een array meekrijgen die even groot is als de resolutie van de bitmap. Dit array bezit de kleur info voor elke pixel. Uiteindelijk word de WritePixels() methode opgeroepen.

```
Task RecalculateMandelbrot()
```

Hier word een Stopwatch opgeroepen die de rekentijd zal bijhouden. Het iterationArray word hier opgevuld. Dit is een asyncrone methode

RedrawBitap()

Maakt ook gebruik van een Stopwatch en een switch case methode. Deze roept andere methoden op om de iterationArray te verrekenen naar een kleur array die dan later gebruikt word om de bitmap mee op te vullen.

Banding()

Hier check ik gewoon of de restdeling van een iteratie op de plaats van een pixel gelijk is aan nul of niet. Als deze gelijk is aan nul geeft ik het een witte waarde anders een zwarte waarde.

Grayscale()

De kleur grijs word gemaakt door 3 dezelfde R, G en B waarden te hebben. Eerst stel ik een schaal in omdat mijn iterations meestal groter zijn dan de maxiamale waarde van R, G en B(255). Ik zet daarna mijn iteratie simpelweg om naar een waarde < 255 en vul het colourArray er mee op.

```
public async void MouseLeftButtonUp(Point q)
```

De bedoeling van deze methode is om tijdens het pannen moet er enkel maar herrekend worden als de muisknop is losgelaten door de gebruiker. Dit maakt het een pak gemakkelijker voor de CPU. Daarbij genereer ik een offset die op zijn beurt meegegeven word aan de logicalaag, de offset heeft invloed op mijn a en b waarde van de logica laag.

MouseMoved()

Deze is verantwoordelijk voor 2 verschillende dingen. De cursorpositie weer te geven aan de gebruiker en hoeveel iteraties er onder de muis zijn.

```
public async void MouseWheelMoved(double delta)
```

Deze methode bekijk waar het muiswielte naartoe word gedraaid. Als de schaalfactor <1 dan word deze beperkt we mogen niet uit ons main beeld zoomen.

```
public void ChangeResolution(double height, double width)
```

Deze methode word telkens opgeroepen als de window veranderd van grootte deze zorgt er voor dat er een nieuwe, juiste bitmap gemaakt word die past in de window. Daarna word opnieuw het mandelbrot herberekend en geven we de gebruiker een zich van hoeveel de resolutie is.

Methoden Logic.cs

CalulateMandelbrot()

Deze methode maakt gebruikt van paparell tasks, for. Iedere thread krijgt een loop van de for loop. Daarnaast wroden nog enkele wiskundige formules opgeroepen.

GenerateMandelbrot()

Hier bereken ik met een Reële formule hoeveel iteraties op een bepaald punt zijn.