— Code review P&O CW 2019–2020 Task ID2 —
Department of Computer Science – KU Leuven April 2, 2020

Team <number>		
Toon Sauvillers	2h	270 LOC
<team member 2>	15h	30 LOC

Demo: https://penocw.cs.kotnet.kuleuven.be:80#/demo-task-44?0xjwkkslam9
Files reviewed: RGBBarcodeScanner.js; PixelIterator.js

1 RGBBarcodeScanner.js

- Om de gehele afbeelding te scannen wordt er gebruik gemaakt van een iterator. Dit is een goede keuze om een *nulpointer exception* te vermeiden. Het wordt gebruikt om, ook al staat het scherm niet gelijk met de afbeelding, toch de het scherm van links naar rechts horizontaal te overlopen. Het in een iterator te schrijven vergroot de leesbaarheid en correctheid.

- Voor het filteren van noise wordt er gebruikt gemaakt van een while loop om elke rij (volgens de iterator) apart te filteren. Zie figuur 2 Duidelijker zou zijn dat de foto in zijn geheel wordt gefilterd in een aparte functie. Deze functie zou dan deze while lus bevatten.

- Vervolgens wordt in de noiseFilter voor elke rij dezelfde bewerking gedaan, met dezelfde gegevens van spectrum. Zie figuur 1. Aangezien het spectrum steeds hetzelfde is, kunnen deze bewerkingen éénmalig worden gedaan en worden meegegeven.

- Ook in de while-lus in figuur 2 wordt er gekeken naar de barcodes per rij. Dit lijkt ook beter en duidelijker moest er eerst gefilterd worden en vervolgens, in een aparte lus, gescanned worden op barcodes. Deze verandering maakt het ook mogelijk om in de toekomst gemakkelijker aanpassingen te doen voor het filteren en het scannen van barcodes apart.

- In scanRow, waar de barcodes worden gescanned, wordt heel vaak $value/255$ gedaan. Dit is echter redundant, het oogt mooier doordat je later kan checken op $== 1$, waarden zijn namelijk of 0 of 255. Als men echter controleert op $== 255$ moet men niet steeds hierdoor delen.

- Voor het vinden van het procentueel aantal juiste barcodes wordt er gebruikt gemaakt van lambdafuncties. Dit oogt zeer mooi en overzichtelijk! Zie figuur 3. Hier zou echter wel mogen bijstaan in de comments om dit eventueel weg te halen. Het wordt nu enkel gebruikt om te loggen.

- Ook voor *getHighestCode* wordt er ook gebruikt gemaatk van een lambdafunctie. Zie figuur 4

```
let grey = spectrum[0]
let distance = Math.round( x: (spectrum[1] - spectrum[2]) / 2)
let black = [grey[0] - distance, grey[1] - distance, grey[2] - distance]
let white = [grey[0] + distance, grey[1] + distance, grey[2] + distance]
let kSize = 15
```

Figure 1: Deel van de noiseFilter die voor elke lijn herhaald wordt.

```
while (iterator.hasNextRow()) {
  let row = iterator.nextRow()
  let filteredRow = this.noiseFilter(imageData, row, spectrum)
  barcodes = this.scanRow(filteredRow, barcodes)
}
```

Figure 2: While-lus in de scan functie

```
let values = Object.keys(barcodes).map(function(key : string ) {
  return barcodes[key]
})
let totalScanned = values.reduce((a, b) => a + b, 0)
```

Figure 3: Lambda functies om de juiste code te vinden.

```
static getHighestCode(barcodes) {
  return Object.keys(barcodes).reduce((a : string , b : string ) =>
    barcodes[a] > barcodes[b] ? a : b
  )
}
```

Figure 4: Lambdafuncties in *getHighestCode*