— Code review P&O CW 2019–2020 Task ID2 —
Department of Computer Science – KU Leuven April 3, 2020

Team 12		
Toon Sauvillers	2h	270 LOC
Bert Van den Bosch	2h	389 LOC

Demo: https://penocw.cs.kotnet.kuleuven.be:80#/demo-task-44?0xjwkkslam9
Files reviewed: RGBBarcodeScanner.js; PixelIterator.js; Animation.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 vermijden. Het wordt gebruikt om, ook al staat het scherm niet gelijk met de afbeelding, toch 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 het 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 gemaakt 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*

2 Animation.js

- Over de triangulatie van de verschillende schermen lopen dieren achter elkaar om de synchrone animatie over de verschillende schermen te tonen. Dit was deel van de opdracht van het eerste semester, maar werd geupdate in het begin van dit semester.

- Over het algemeen mist de gehele klasse documentatie. Van de instructor tot grote methodes. Enkele functies zijn van zelfsprekend, maar het grotendeel is het voor een gebruiker dat de code nog nooit heeft gezien een gokwerk van de verschillende parameters en resultaat van de functie in kwestie.

- De functie $go(dx, dy)$ gebruikt twee function calls om eerst verticaal te verplaatsen en vervolgens horizontaal. Dit oogt mooi, maar is niet efficiënt aangezien deze methodes om volgens de x- en respectievelijke y-as te transleren enkel door de algemene $go()$ functie worden opgeroepen. Dit maakt dat deze extra methodes redundant zijn en de algemen functie efficiënter kan door de volledige translatie in één keer uit te voeren. (Zie figuur 5.)

- De code om de verschillende dieren af te beelden is niet voorbedachte raden geschreven op uitbreiding. De afbeelding van de twee gebruikte dieren worden geïnisialiseerd in de constructor. Beide dieren hebben een standaard en een kerst versie. Voor twee dieren gaat dit nog net in de constructor, maar de verschillende $if - statements$ maken de constructor onnodig lang. Deze initialisatie zou zich beter in een eigen methode bevinden om de constructor niet storend lang te maken bij het toevoegen van meerdere dieren.
  Ook de methode $drawAnimal()$ is nu gebasseerd op een boolean van welk dier er getekend moet worden en heeft dus een herschrijving nodig om meer dan twee dieren aan te kunnen. (Zie afbeelding 6.)
  Om de verschillende dieren hun frame te updaten wordt er nu een groot deel code twee maal met maar een minimale aanpassing gebruikt in $drawAnimals()$. Ook hier kan er gerefactord worden om meerdere dieren aan te kunnen zonder dat er veel aangepast moet worden en repetitieve code te vermijden. (Zie afbeelding 7.)

- De volledige klasse kan ook een opkuis gebruiken. Er zijn hier en daar nog overtollige *console.log* restanten van het debuggen, volledig gecommentarieerde functies en redundante parameters in een methode

```
81  goHorizontal(dx) {
82    this.setPosition(this.position.x + dx, this.position.y)
83    return this.getPosition()
84  }
85
86  goVertical(dy) {
87    this.setPosition(this.position.x, this.position.y + dy)
88    return this.getPosition()
89  }
90
91  go(dx, dy) {
92    this.goVertical(dy)
93    this.goHorizontal(dx)
94    return this.getPosition()
95  }
```

Figure 5: Redundate methodes voor translatie.

```
151  drawAnimal(canvas, x, y, angle, frame, right, cat) {
152    if (cat) {
153      var image = this.catImage
154    } else {
155      var image = this.mouseImage
156    }
157    let ctx = canvas.getContext('2d')
```

Figure 6: Maar twee soorten dieren supported door gebruik van boolean als parameter.

```
121   drawAnimals(nbCats, distCats, canvas, x, y, angle, frame, right) {
122     this.stack.push([x, y, angle, frame, right])
123     if (this.stack.length > distCats) {
124       for (let i = 0; i <= nbCats - 1; i++) {
125         if (i === 0) {
126           this.drawAnimal(
127             canvas,
128             this.stack[i * Math.round(distCats / nbCats)][0],
129             this.stack[i * Math.round(distCats / nbCats)][1],
130             this.stack[i * Math.round(distCats / nbCats)][2],
131             this.stack[i * Math.round(distCats / nbCats)][3],
132             this.stack[i * Math.round(distCats / nbCats)][4],
133             true
134           )
135         } else {
136           this.drawAnimal(
137             canvas,
138             this.stack[i * Math.round(distCats / nbCats)][0],
139             this.stack[i * Math.round(distCats / nbCats)][1],
140             this.stack[i * Math.round(distCats / nbCats)][2],
141             this.stack[i * Math.round(distCats / nbCats)][3],
142             this.stack[i * Math.round(distCats / nbCats)][4],
143             false
144           )
145         }
146       }
147       this.stack.shift()
148     }
149   }
```

Figure 7: Repetitieve code.