|  |  |
| --- | --- |
| Naam | Swen Meeuwes |
| Studentnummer | 0887127 |
| Naam | Jesse van Breda |
| Studentnummer | 0902729 |
| Klas | INF2A |
| Docent | Jan Kroon |
| Datum | 12-6-2016 |

De volledige beschrijving van de opdracht vind je hier:

<https://confluence.hr.nl/display/CMIP/Visualisatie+-+inleveropdracht+3>

## 

## 

## Opdracht 3A. preprocessing

Maak de hoogtedata geschikt voor visualisatie en geef hier die code:

|  |
| --- |
| Code:  public void getDataAsyncFast(Canvas canvas) {  int linesToSkip = 25;  Thread thread = new Thread(new Runnable() {  public void run() {  try {  ClassLoader classLoader = getClass().getClassLoader();  File fileWest = new File(classLoader.getResource("rotterdamopendata\_hoogtebestandtotaal\_west.csv").getFile());  BufferedReader br = new BufferedReader(new InputStreamReader(fileWest.toURI().toURL().openStream()));  Iterator lines = br.lines().iterator();  // Skip first line, these are the column names  lines.next();  while (lines.hasNext()) {  String line = (String) lines.next();  String[] values = line.split(",");  Vector3<Float> vector = new Vector3<Float>();  vector.setX(Float.parseFloat(values[0]));  vector.setY(Float.parseFloat(values[1]));  vector.setZ(Float.parseFloat(values[2]));  canvas.data.add(vector);  for (int i = 0; i < linesToSkip; i++) {  if (lines.hasNext()) {  lines.next();  }  }  }  } catch (IOException e) {  System.err.println("IO Exception occured while reading the data!");  } finally {  canvas.state = SimulationState.DONE;  }  }  });  thread.setPriority(Thread.MIN\_PRIORITY);  Thread thread2 = new Thread(new Runnable() {  public void run() {  try {  ClassLoader classLoader = getClass().getClassLoader();  File fileOost = new File(classLoader.getResource("rotterdamopendata\_hoogtebestandtotaal\_oost.csv").getFile());  BufferedReader br = new BufferedReader(new InputStreamReader(fileOost.toURI().toURL().openStream()));  Iterator lines = br.lines().iterator();  // Skip first line, these are the column names  lines.next();  while (lines.hasNext()) {  String line = (String) lines.next();  String[] values = line.split(",");  Vector3<Float> vector = new Vector3<Float>();  vector.setX(Float.parseFloat(values[0]));  vector.setY(Float.parseFloat(values[1]));  vector.setZ(Float.parseFloat(values[2]));  canvas.data.add(vector);  for (int i = 0; i < linesToSkip; i++) {  if (lines.hasNext()) {  lines.next();  }  }  }  } catch (IOException e) {  System.err.println("IO Exception occured while reading the data!");  }  }  });  thread2.setPriority(Thread.MIN\_PRIORITY);  thread.start();  thread2.start();  } |

## 

## 

## Opdracht 3B. Visualisatie van omgeving

Maak een visualisatie van de omgeving van de school (Wijnhaven 107, gebied van 500x500 meter), waarbij de hoogte van het gebied in kaart gebracht wordt.

|  |
| --- |
| Screenshot: Het oranje stipje is Wijnhaven 107. Het midden is het beeld van zadkine, zoals dit op confluence beschreven stond. |

|  |
| --- |
| Code:  De code is te lang om hier weergegeven te worden. Onderstaande classes zijn van belang bij de visualisatie van deze data. De classes staan allemaal in de FloodSimulation/src/main/java/com/jesseenswen/floodsimulation:   * data/Dataprovider.java (methode voor het inlezen van data) * models/Vector3.java (model voor het opslaan van de data) * Canvas.java (het canvas waar de data op wordt gevisualiseerd) * de hele ui-folder (de elementen voor de User interface zijn hierin gedefineerd)   Link naar de repo: https://github.com/swenmeeuwes/Dev8-Visualisation-Opdracht3 |

## 

## 

## Opdracht 3C. Waterstijging

Visualiseer vervolgens een waterstijging:

* laat het waterniveau in een simulatie stijgen en kleur de gebieden die onder water komen te staan blauw.
* Laat de waterhoogte met een duidelijk font zien, midden boven.
* Laat in je verslag 6 toestanden zien. Neem aan dat het water een halve meter per uur stijgt en zet de tijd bij de afbeeldingen.

|  |
| --- |
| Screenshots van de visualisatie |

|  |
| --- |
| Code of jouw git repository:  ui.pushElement(new Button(this, "Save this image", new Vector2<>(simulationArea.getX() + simulationArea.getWidth()+ 12, simulationArea.getY() + simulationArea.getHeight()+ 12)){  @Override  public void onClick(){  if (unixNow == 0)  unixNow = System.currentTimeMillis();  else {  unixNow += interval;  }  PGraphics graphic = createGraphics(simulationArea.getWidth(), simulationArea.getHeight());  PImage mapImage = get(simulationArea.getX(), simulationArea.getY(), simulationArea.getWidth(), simulationArea.getHeight());  graphic.image(mapImage, 0, 0);      graphic.textAlign(CENTER, TOP);  graphic.textSize(22);  graphic.text("Height map of Wijnhaven Rotterdam (" + (mappingArea.getWidth() - mappingArea.getX()) + " m\u00B2", graphic.width / 2, 20);  graphic.textSize(20);  graphic.text("Water Level: " + waterLevel, graphic.width / 2, 45);  graphic.text("Time: " + new Date(unixNow), graphic.width / 2, 65);      graphic.save("/target/screenshot.jpg");  JOptionPane.showMessageDialog(applet, "Your screenshot is saved in the /target folder.", "Success", JOptionPane.INFORMATION\_MESSAGE);    waterLevel += 0.5f;  }  }); |

## 

## 

## Opdracht 3D. Interactie

Bouw interactie in.

Hints:

* Processing heeft mogelijkheden om de muis of het keyboard uit te lezen.
  + Processing > reference > input
  + Voor deze opdracht mag je ook UI-widgets gebruiken

Requirements:

* gebruiker kan gebied kiezen (500x500 m2 of 1000x1000 m2)
* laat gebied langzaam/snel onder water lopen
* gebruiker kan op play-pauze-stop drukken
* gebruiker kan visualisatie opslaan als jpg-bestand

|  |
| --- |
| Screenshots van de visualisatie (inclusief interactie-elementen) -> zie volgende pagina |

|  |
| --- |
| Code van interactie:  De code is te lang om hier weergegeven te worden. Onderstaande classes zijn van belang bij de interactie van deze data. De classes staan allemaal in de FloodSimulation/src/main/java/com/jesseenswen/floodsimulation:   * Canvas.java (het canvas waar de data op wordt gevisualiseerd) * de hele ui-folder (de elementen voor de User interface zijn hierin gedefineerd)   In com.jesseenswen.floodsimulation.ui zitten verschillende ui-elementen die interactie mogelijk maken. Zo zijn er bijvoorbeeld buttons en toggleable buttons. Deze worden in canvas.java geïnstantieerd en toegevoegd aan de uioverlay. De uioverlay klasse tekent en handled de ui-elementen.  Voorbeeld button (die het water level verlaagt):  new Button(this, "Decrease water level", new Vector2<>(simulationArea.getX() + simulationArea.getWidth() + 12, 420)) {  @Override  public void onClick() {  waterLevel -= 0.5f;  }  });  Verder is er nog sprake van inheritance:    Link naar de repo: https://github.com/swenmeeuwes/Dev8-Visualisation-Opdracht3 |