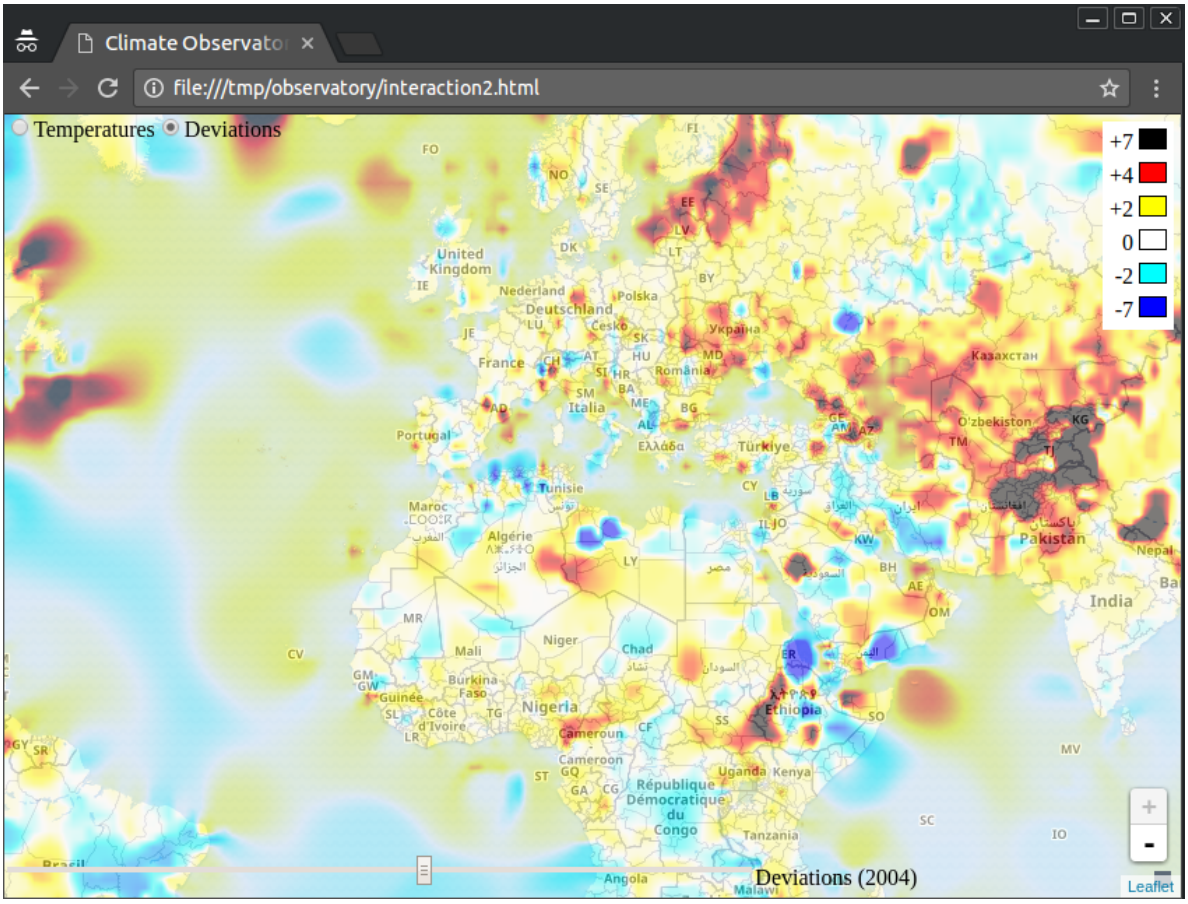
Milestone overview

This (last!) milestone consists in implementing an interactive user interface so that users can select which data set (either the temperatures or the deviations) as well as which year they want to observe. You will have to complete the file **Interaction2.scala**. But first, remember to update the grading milestone number:

```
1  val milestone: Int = 6
```



We provide a sub-project, **capstoneUI**, which contains the actual user interface implementation. This sub-project uses the methods you are going to implement.

Last but not least, you will use the tiles that you previously generated for the temperatures and the deviations, so check that you had not deleted them from your file system!

You can monitor your progress by submitting your work at any time during the development of this milestone. Your submission token and the list of your graded submissions is available on this page.

Layers

This milestone introduces the concept of **Layer**s. A **Layer** describes the additional information shown on a map. In your case, you will have two layers: one showing the temperatures over time, and one showing the temperature deviations over time.

You will have to implement the following method:

```
def availableLayers: Seq[Layer]
```

This method returns the layers you want the user to be able to visualize. Each layer has a name, a color scale and a range of supported years.

The user interface implementation will use your **availableLayers** to build buttons allowing to choose which layer to enable. The value over time of the enabled layer is represented by a **Signal[Layer]** value (the same **Signal** as in the progfun2 course).

Signals

In this part of the assignment you are going to reuse the **Signal** abstraction introduced in the "Functional Program Design in Scala" course.

Reminder on Signals

In case you didn't follow this course or you need to refresh your memory, here is a short reminder on **Signal**s.

A **Signal** is a value that can change over time:

```
val x = Signal(0)  // Initialize x's value to "0"
println(x())       // Read x's current value ("0")
x()= x() + 1       // Change x's value
println(x())       // "1"
```

Signals can depend on other **Signal**s. In such a case, when the value of a **Signal** changes, the **Signal**s that depend on it are automatically updated:

```
val x = Signal(0)
val y = Signal(x() * 2)  // y depends on x's value
println(y())             // "0"
x()= x() + 1
println(y())             // "2"
```

Note that, in the above example, if we didn't want to introduce a dependency between **x** and **y** we would have to first capture the current value of **x** in a usual **val**:

```
val x = Signal(0)
val currentX = x()            // currentX is a stable value
val y = Signal(currentX * 2)  // y is initialized with currentX's value, but
  does not depend on x
x()= x() + 1
println(y())                  // "0" (the update of x did not trigger an update
  on y)
```

Methods to implement

You will have to implement the following signals:

```
def yearBounds(selectedLayer: Signal[Layer]): Signal[Range]
```

This method takes the selected layer signal and returns the years range supported by this layer.

```
def yearSelection(
  selectedLayer: Signal[Layer],
  sliderValue: Signal[Year]
): Signal[Year]
```

This method takes the selected layer and the year slider value and returns the actual selected year, so that this year is not out of the layer bounds (remember that **Year** is just a type alias for **Int**).

```
def layerUrlPattern(
  selectedLayer: Signal[Layer],
  selectedYear: Signal[Year]
): Signal[String]
```

This method takes the selected layer and the selected year and returns the pattern of the URL to use to retrieve the tiles. You will return a relative URL (starting by **target/**). Note that the **LayerName id** member corresponds to the sub-directory name you had generated the tiles in. This URL pattern is going to be used by the mapping library to retrieve the tiles, so it must follow a special syntax, as described here (you can ignore the **"s"** parameter).

```
def caption(
  selectedLayer: Signal[Layer],
  selectedYear: Signal[Year]
): Signal[String]
```

This method takes the selected layer and the selected year and returns the text information to display. The text to display should be the name of the layer followed by the selected year, between parenthesis. For instance, if the selected layer is the temperatures layer and the selected year is 2015, it should return "Temperatures (2015)".

Running the Web application

Once you have implemented the above methods, you are ready to finally run the whole application. Execute the following sbt command:

```
capstoneUI/fastOptJS
```

This will compile part of your Scala code to JavaScript instead of JVM bytecode, using Scala.js. To see it in action, just open the **interaction2.html** file in your browser!

Note that some of the source files you had written are going to be shared with the capstoneUI sub-project. That's the case for **Interaction2.scala**, obviously, but also **models.scala** (which contains the definition of the **Color** data type). Why does that matter? Because while Scala.js supports a wealth of libraries, many others just aren't compatible. This means that your code in this milestone must not depend on libraries that have not been compiled for JavaScript (like the scrimage library, for instance).

If you'd like to know more after this short introduction on Scala.js, you can check out their resources.

Mark as completed