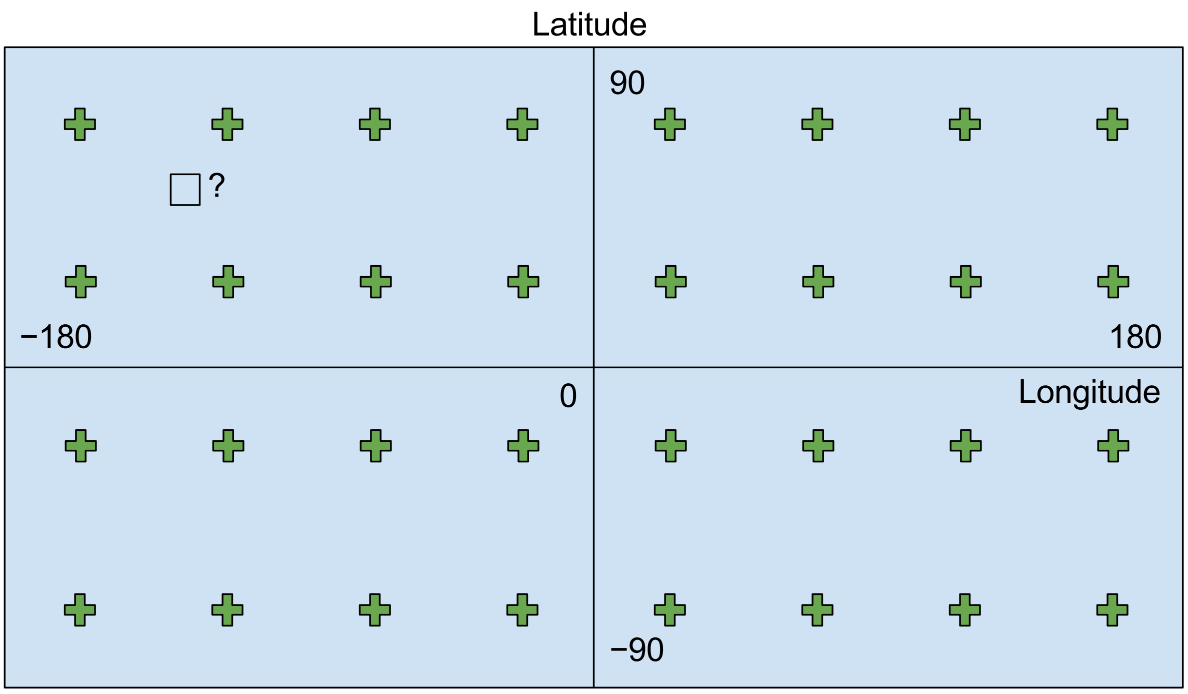
Milestone overview

The goal of this milestone is to produce tile images from the grids generated at the previous milestone. You will have to complete the file Visualization2.scala. But first, remember to update the grading milestone number:

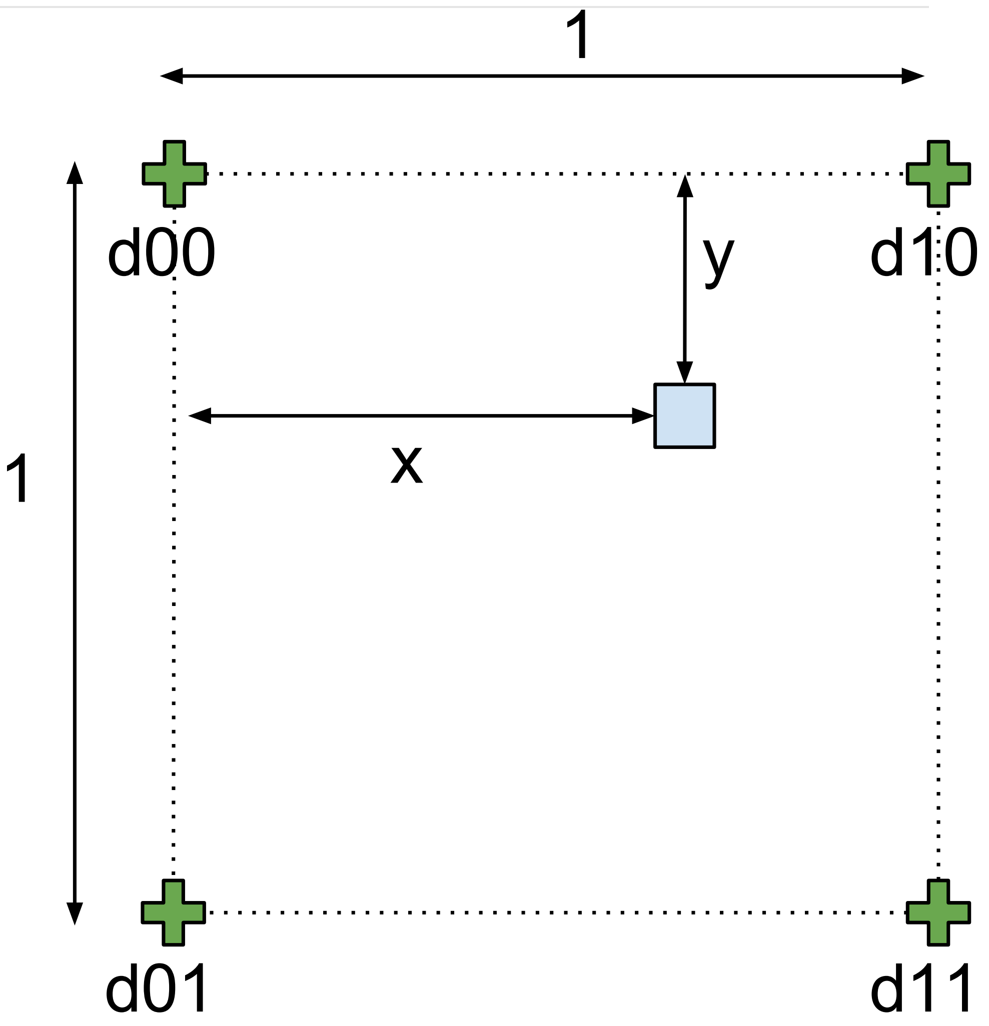
```scala
val milestone: Int = 5
```

As in the 3rd milestone, you will have to compute the color of every pixel of the tiles. But now the situation has changed: instead of working with a set of scattered points, you have a regular grid of points:



In this figure, the square in the middle materializes a pixel that you want to compute. You can leverage the grid to use a faster interpolation algorithm: you can now use bilinear interpolation rather than inverse distance weighting.

More precisely, you will implement a simplified form of bilinear interpolation:



In this form of bilinear interpolation, the location of the point to estimate is given by coordinates x and y, which are numbers between 0 and 1. The algorithm considers that the four known points, d00, d01, d10 and d11, form a *unit square* whose origin is its top-left corner. As such, the coordinates of a pixel inside of a grid cell can be described by the following case class, defined in **models.scala**:

```scala
case class CellPoint(x: Double, y: Double)
```

You will also have to decide on a color scale to use, to represent the temperature deviations. You can for instance use one like the following:

+7	⬛
+4	🟥
+2	🟨
0	⬜
-2	🟦
-7	🟦

Here are the RGB values of these colors:

Temperature (°C)	Red	Green	Blue
7	0	0	0
4	255	0	0

2	255	255	0
0	255	255	255
-2	0	255	255
-7	0	0	255

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.

Visualization

You will have to implement the following methods:

```scala
def bilinearInterpolation(
  point: CellPoint,
  d00: Temperature,
  d01: Temperature,
  d10: Temperature,
  d11: Temperature
): Temperature
```

This method takes the coordinates (**x** and **y** values between 0 and 1) of the location to estimate the temperature at, and the 4 known temperatures as shown in the above figure, and returns the estimated temperature at location **(x, y)**.

```scala
def visualizeGrid(
  grid: GridLocation => Temperature,
  colors: Iterable[(Temperature, Color)],
  tile: Tile
): Image
```

This method takes a grid, a color scale and the coordinates of a tile, and returns the 256×256 image of this tile, where each pixel has a color computed according to the given color scale applied to the grid values.

Hint: remember that our grid is a rectangular projection of a sphere, so **IndexOutOfBoundsExceptions** on coordinates should not be possible!

Deviation tiles generation

Once you have implemented the above methods, you are ready to generate the tiles showing the deviations for all the years between 1990 and 2015, so that the final application (in the last milestone) will nicely display them:

- Compute normals from yearly temperatures between 1975 and 1989 ;
- Compute deviations for years between 1990 and 2015 ;
- Generate tiles for zoom levels going from 0 to 3, showing the deviations. Use the output method of Image to write the tiles on your file system, under a location named according to the following scheme: **target/deviations/<year>/<zoom>/<x>-<y>.png**.

Note that this process is going to be very CPU consuming, or might even crash if your implementation tries to load too much data into memory. That being said, even a smart solution performing incremental data manipulation and parallel computations might take a lot of time (several days). You can reduce this time by using some of these ideas:

- Identify which parts of the process are independent and perform them in parallel ;
- Reduce the quality of the tiles. For instance, instead of computing 256×256 images, compute 128×128 images (that's going to be 4 times fewer pixels to compute) and then scale them to fit the expected tile size ;
- Reduce the quality of the spatial interpolation. For instance, instead of having grids with 360×180 points, you can use a grid with 120×60 points (that's going to be 9 times fewer points to compute).

Mark as completed