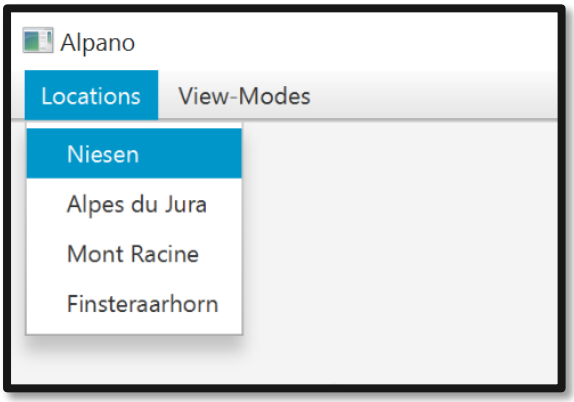
Alpano – improvements

Natal Willisch 30.05.2017

What I added

1. I implemented the possibility to choose from (some of) the predefined Panoramas in a menu “Locations”.
2. I also added some ImagePainters beside of the “rainbow”-painter, they can be selected with a toggle menu “View-Modes”. The views are:
    - Reuse of the two Painters which we used for the drawings before: I called “rainbow” (default) and sketch
    - A Painter that shows isohypses (“altitude lines”) for every 200m of altitude, called “lig. de hauteur”
    - A black-grey-and-with mode called “gray”
    - And 3 photorealistic mode, which try to reconstruct the “real” colours of the landscape.
3. The labels fade out and disappear (almost) if the mouse is not moved, so the view can be enjoyed.

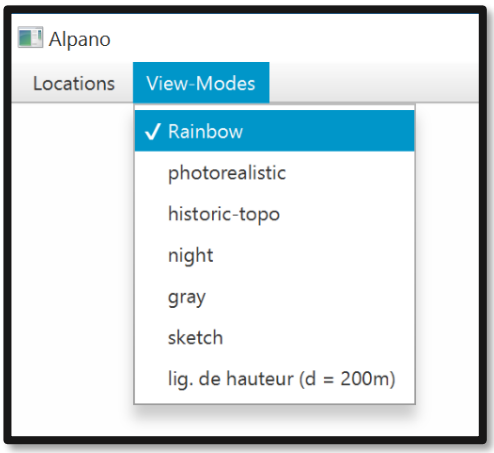
1. Chose other predefined Panoramas




```
Consumer<PanoramaUserParameters> set = param -> {

    parameterBean.observerLatitudeProperty()
            .setValue(param.observerLatitude());
    parameterBean.observerLongitudeProperty()
            .setValue(param.observerLongitude());
    parameterBean.observerelevationProperty()
            .setValue(param.observerElevation());
    parameterBean.centerAzimuthProperty()
            .setValue(param.centerAzimuth());
    parameterBean.horizontalFieldOfViewProperty()
            .setValue(param.horizontalFieldOfView());
};

MenuBar bar = new MenuBar();
Menu menuLocation = new Menu("Locations");

BiConsumer<String, PanoramaUserParameters> add = (s, p) -> {
    RadioMenuItem item = new RadioMenuItem(s);

    item.setOnAction(e -> set.accept(p));
    menuLocation.getItems().add(item);
};
add.accept("Niesen", PredefinedPanoramas.NIESEN);
add.accept("Alpes du Jura", PredefinedPanoramas.ALPES_DU_JURA);
add.accept("Mont Racine", PredefinedPanoramas.MONT_RACINE);
add.accept("Finsteraarhorn", PredefinedPanoramas.FINSTERAARHORN);

        //after I add the menu to the menu bar
```

I created the menu and the menu bar. After that I defined a local BiConsumer that adds the predefined panoramas to the menu and sets them on action. For the action itself I defined another Consumer called “set” that replaces the properties of the parameter-bean, in the same way it already happens when the user changes manually the parameters in the user-interface, so I didn’t have to change anything for that in another class. I don’t replace all parameters; I keep the parameters that relate to the quality and size of the rendering and only change the parameters related to the position and view. I defined this function not-anonymous because the function could be maybe reused for other functionalities in the menu in the future.

2. ImagePainters




```
Menu menuMode = new Menu("View-Modes");
ToggleGroup group = new ToggleGroup();
BiConsumer<String, Function<Panorama, ImagePainter>> addPainters = (s, p) -> {
    RadioMenuItem item = new RadioMenuItem(s);
    item.setToggleGroup(group);
    item.setUserData(p);
    item.setId(s);
    menuMode.getItems().add(item);

};
addPainters.accept("Rainbow", p->ImagePainter.rainbow(p));
addPainters.accept("photorealistic", p->ImagePainter.photorealisticBlue(p));
addPainters.accept("historic", p->ImagePainter.photorealisticHisto(p));
addPainters.accept("night", p->ImagePainter.photorealisticNight(p));
addPainters.accept("gray", p->ImagePainter.grayish(p));
addPainters.accept("sketch", p->ImagePainter.draw(p));
addPainters.accept("lig. de hauteur (d = 200m)", p->ImagePainter.layer(p));

group.getToggles().get(0).setSelected(true);

group.selectedToggleProperty().addListener((t, old, updated) -> {
    if (updated != null){
        computerBean.labelColorProperty().setValue(((RadioMenuItem)updated).getId()
                =="night"?true:false);
        computerBean.imagePainterProperty()
                .setValue((Function<Panorama, ImagePainter>) updated.getUserData());

    }
});
```

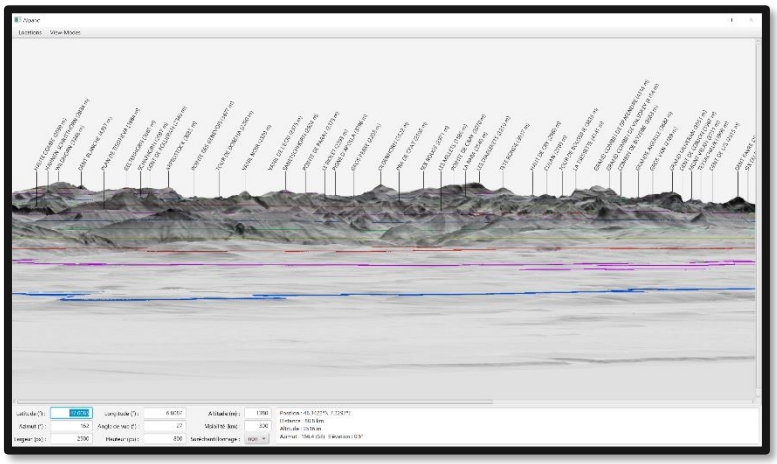
This menu works in a similar way as the first menu, but it uses a ToggleGroup, as always one of the ImagePainters should be selected.
As UserProperty, I use here a function to a ImagePainter. In the event listener, I set the imagePainterProperty of the computerBean which stores the actual ImagePainter to the selected ImagePainter and change the Property that defines the colour of the labels.

To change the ImagePainter without recalculating the Panorama, I outsourced this part of the function into the function draw(). In the function the colour of the labels is also changed if needed. (the Boolean “change” helps to decide if the function has to change back to black or if it is already black. The recalculation of the panorama sets the boolean to false)

```
private void draw() {
    if (getPanorama() != null) {

        boolean bool = false;
        if (labelColorProperty.getValue())
            bool = true;

        if (bool)
            labelList.forEach(n -> {
                if (n instanceof Text)
                    ((Text) n).setFill(Color.WHITE);
                if (n instanceof Line)
                    ((Line) n).setStroke(Color.WHITE);
            });
        else if (change)
            labelList.forEach(n -> {
                if (n instanceof Text)
                    ((Text) n).setFill(Color.BLACK);
                if (n instanceof Line)
                    ((Line) n).setStroke(Color.BLACK);
            });
        change = bool;

        image.setValue(PanoramaRenderer.renderPanorama(getPanorama(),
                imagePainterProperty.getValue().apply(getPanorama())));
    }

}
```

The ImagePainters

Rainbow and sketch are the two painters that were given for testing the Panorama-calculations. The ImagePainter grayish() (“gray”), works similar like them. Also about layer() is not a lot to say: depending on the altitude at a pixel it paints in colour or not. The result looks like the picture on the right (coloured lines all 200m of altitude)

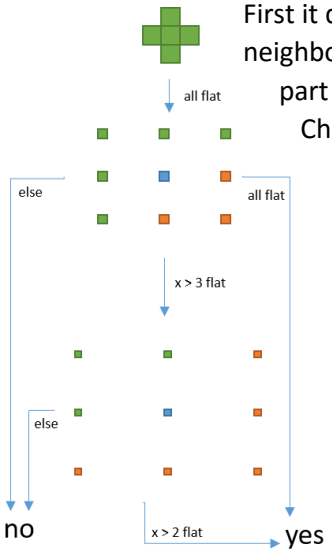


The photorealistic painters work all more or less the same and base on the following two ImagePainters that create together a photorealistic Panorama:

sky() is a simple filter that replaces the transparent areas (the sky) with a colour: light blue, pale yellow, or black with white spots (imitates the night sky)

The ImagePainter photorealistic() is not special at all, but it uses the following special ChannelPainters: photorealistic(), fade() and snowB(), that all take a Panorama as parameter.

These ChannelPainters analyse the Panorama and using information like the slope or the altitude they can determine if it is simple green vegetation, stone (steep areas), or possible snow (in higher areas) or if it is a lake. Depending on the found information the ChannelPainters fade() and snowB() (don't detect lakes) manipulate the channels and photorealistic() chooses a colour that fits to the situation. Since the code is not that special I only want to explain and show how the ChannelPainter detects the lakes.

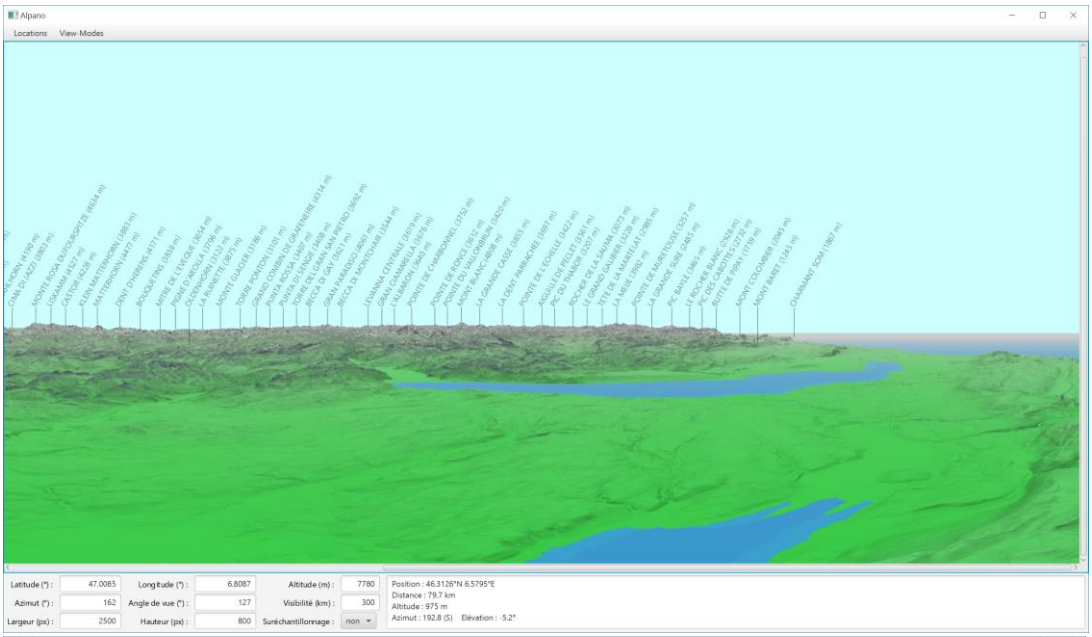


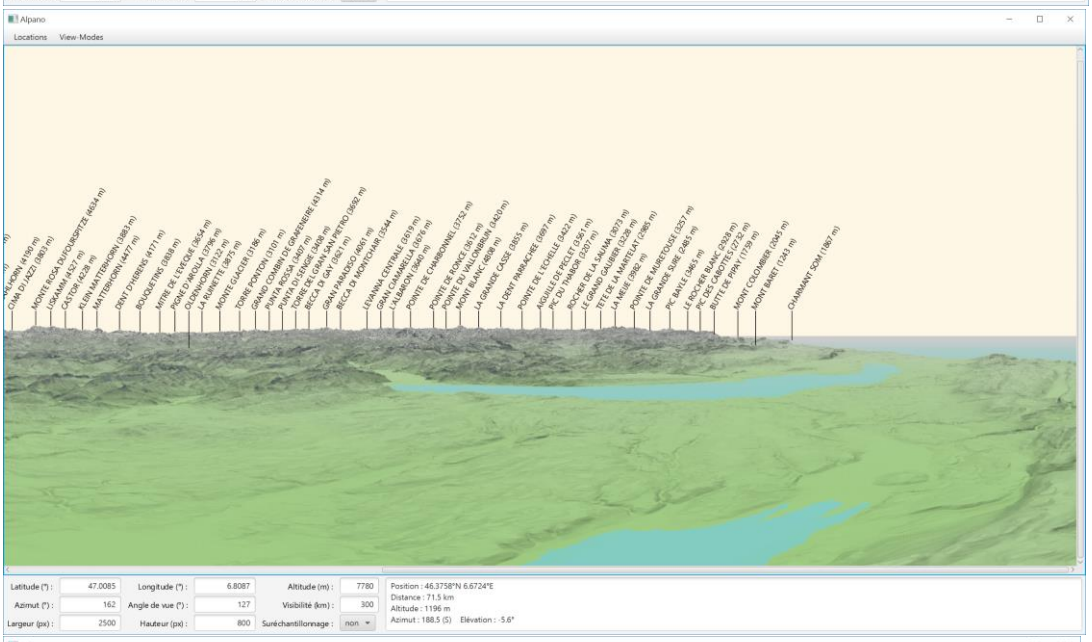
First it detects if the point itself and its direct neighbours are flat. If yes it could be maybe a part of a lake. To be sure that is a lake, the ChannelPainter also tests if some not direct neighbours are flat. The distance of pixel is defined in vFactor and hFactor, which depend both on the distance and the horizontal field of view and the size of the panorama, as the same lake is smaller far away from the observer as near to the observer. If all points around the point are flat too and have the same altitude, the point is recognized as part of a lake, if not enough points fulfil this property, but still some, the algorithm does the same evaluation for points which are even further away to find out if the point is maybe on the border of the lake-area or near an island.

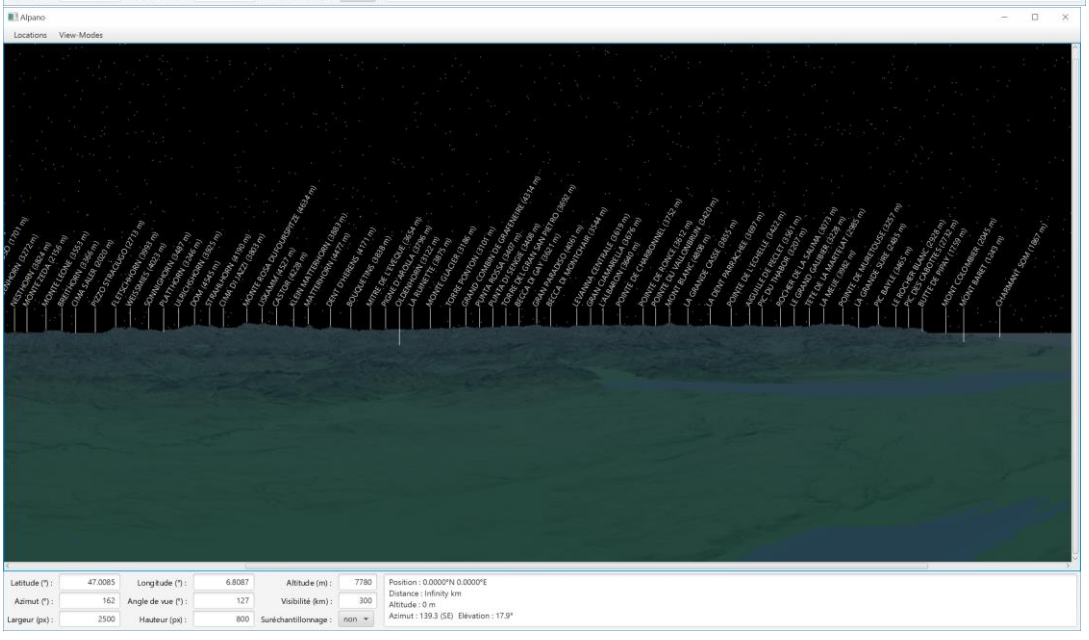
So, with the help of that algorithm, it is possible to detect lakes without a lot of wrong detections.

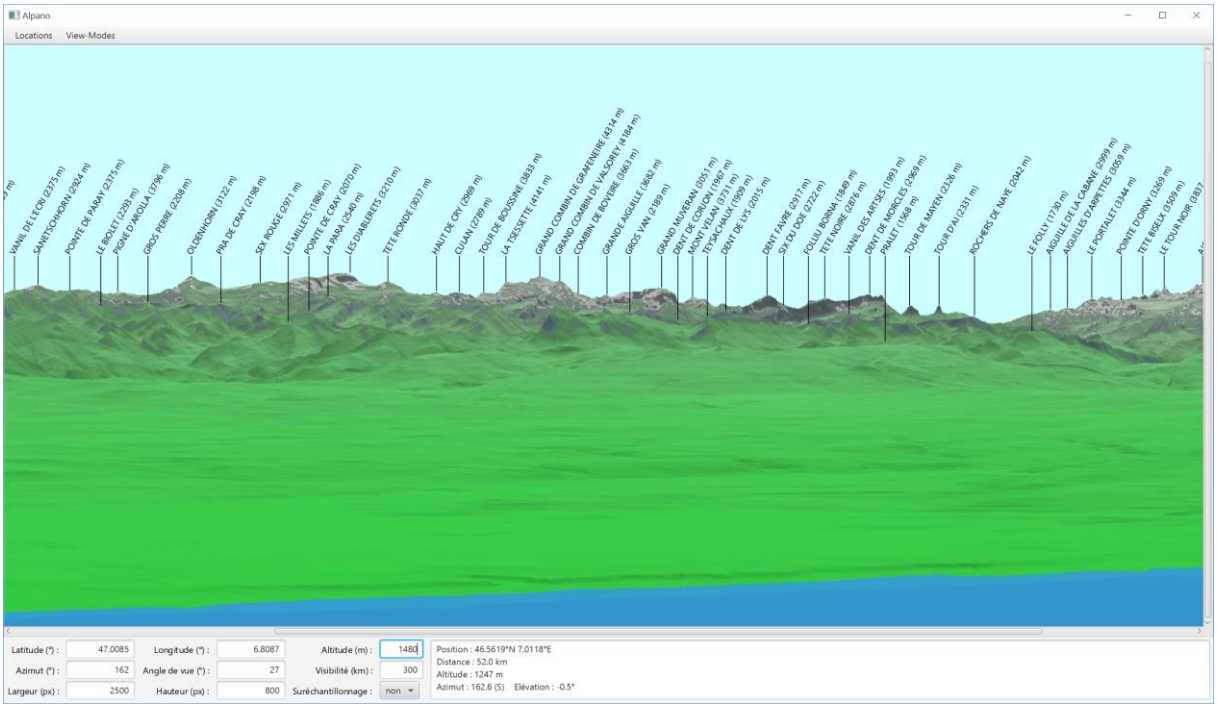
```
public static ImagePainter photorealistic(Panorama panorama) {

        ChannelPainter distance = panorama::distanceAt;
        ChannelPainter slope = panorama::slopeAt;
        ChannelPainter h = ChannelPainter.photorealistic(panorama).mul(360);
        ChannelPainter s = distance.div(200000).clamped().inverted()
                .fade(panorama);

        ChannelPainter b = slope.mul(2 / Math.PI).inverted().mul(0.7).add(0.3)
                .snowB(panorama).clamped();
        ChannelPainter o = distance
                .map(d -> d == Float.POSITIVE_INFINITY ? 0 : 1);

        return ImagePainter.hsb(h, s, b, o);

}

public default ImagePainter sky(int mode) {

        return (x, y) -> {
            if (this.colorAt(x, y).getOpacity() < 1) {
                if (mode == 0)

                    return Color.hsb(180, 0.2, 1, 1);

                if (mode == 1)
                    return Color.hsb(70, 0.2, 1, 1);

                if (mode == 2) {
                    if (Math.random() < 0.9985)
                        return Color.hsb(70, 0, 0, 1);

                    return Color.hsb(70, 0, 1, 1);
                }
            }
            return this.colorAt(x, y);
        };

}
```

```
double facteur = (panorama.parameters().width()
        / panorama.parameters().horizontalFieldOfView()
        / panorama.distanceAt(x, y) * 1000);

int vFactor = (int) (Math
        .sqrt((panorama.parameters().altitudeForY(y) / Math2.PI2)
                * facteur))
        + 1;

int hFactor = (int) facteur + 1;
IntUnaryOperator checkAround = (multi) -> {
    int xLeft = x_cut.applyAsInt(x - multi * hFactor);
    int xRight = x_cut.applyAsInt(x + multi * hFactor);
    int yTop = y_cut.applyAsInt(y - multi * vFactor);
    int yButtom = y_cut.applyAsInt(y + multi * vFactor);
    int count = 0;

    if (slopF.apply(xLeft, y) < flat
            && elevF.apply(xLeft, y) == elevation)
        count++;
    if (slopF.apply(xLeft, yTop) < flat
            && elevF.apply(xLeft, yTop) == elevation)
        count++;
    if (slopF.apply(xLeft, yButtom) < flat
            && elevF.apply(xLeft, yButtom) == elevation)
        count++;
    if (slopF.apply(x, yTop) < flat
            && elevF.apply(x, yTop) == elevation)
        count++;
    if (slopF.apply(x, yButtom) < flat
            && elevF.apply(x, yButtom) == elevation)
        count++;
    if (slopF.apply(xRight, yTop) < flat
            && elevF.apply(xRight, yTop) == elevation)
        count++;
    if (slopF.apply(xRight, yButtom) < flat
            && elevF.apply(xRight, yButtom) == elevation)
        count++;
    if (slopF.apply(xRight, y) < flat
            && elevF.apply(xRight, y) == elevation)
        count++;

    return count;
};

 ...
else if (slope < flat
        && slopF.apply(x_cut.applyAsInt(x + 1), y) < flat
        && slopF.apply(x_cut.applyAsInt(x - 1), y) < flat
        && (slopF.apply(x, y_cut.applyAsInt(y + 1)) < flat || slopF
                .apply(x, y_cut.applyAsInt(y - 1)) < flat)) {

    int count = checkAround.applyAsInt(1);

    if (count == 8)
        return 0.56;
    else if (count > 3) {
        count = checkAround.applyAsInt(3);

        if (count > 2)
            return 0.55;
    }
}
```


Alpano
Locations
View Modes
Latitude (°) : 47.0085
Longitude (°) : 6.8087
Altitude (m) : 7780
Azimut (°) : 162
Angle de vue (°) : 127
Visibilité (km) : 300
Largeur (px) : 2500
Hauteur (px) : 800
Suréchantillonnage : non
Position : 46.3126°N 6.5795°E
Distance : 79.7 km
Altitude : 975 m
Azimut : 192.8 (S) Élévation : -5.2°


Alpano
Locations
View Modes
Latitude (°) : 47.0085
Longitude (°) : 6.8087
Altitude (m) : 7780
Azimut (°) : 162
Angle de vue (°) : 127
Visibilité (km) : 300
Largeur (px) : 2500
Hauteur (px) : 800
Suréchantillonnage : non
Position : 46.3758°N 6.6724°E
Distance : 71.5 km
Altitude : 1196 m
Azimut : 188.5 (S) Élévation : -5.6°


Alpano
Locations
View Modes
Latitude (°) : 47.0085
Longitude (°) : 6.8087
Altitude (m) : 7780
Azimut (°) : 162
Angle de vue (°) : 127
Visibilité (km) : 300
Largeur (px) : 2500
Hauteur (px) : 800
Suréchantillonnage : non
Position : 0.0000°N 0.0000°E
Distance : Infinity km
Altitude : 0 m
Azimut : 139.3 (SE) Élévation : 17.9°



Remarque: Honestly I think that the historic mode is more photorealistic than the photorealistic-mode… ☺

3. Fading out of the labels

The labels fade slowly out after some seconds of no mouse movement in the picture pane and fade fast in again after a mouse movement is detected. For measuring the time I use the PauseTransition-class.

```
  Pane labelsPane = new Pane();

  FadeTransition fadeOut = new FadeTransition(Duration.millis(4444),
          labelsPane);
  FadeTransition fadeIn = new FadeTransition(Duration.millis(250),
          labelsPane);
  PauseTransition wait = new PauseTransition(Duration.seconds(2));

  {
      labelsPane.setMouseTransparent(true);
      labelsPane.prefWidthProperty().bind(parameterBean.widthProperty());
      labelsPane.prefHeightProperty()
              .bind(parameterBean.heightProperty());

      Bindings.bindContent(labelsPane.getChildren(),
              computerBean.getLabels());

      fadeOut.setFromValue(1.0);
      fadeOut.setToValue(0.08);
      fadeIn.setToValue(1.0);

      wait.setOnFinished(e -> fadeOut.play());

      computerBean.panoramaProperty()
              .addListener(e -> wait.playFromStart());
  }

…

  ScrollPane panoScrollPane = new ScrollPane();
   panoScrollPane.setContent(panoGroup);
   panoScrollPane.setOnMouseMoved(e -> {

      fadeOut.stop();
      fadeIn.play();
      labelsPane.setOpacity(1);
      wait.playFromStart();
  });
```

Modified classes

modified, added

- Alpano: panoPane, menu
- ChannelPainter: photorealistic, fade, snowB, snow1, snow2, stone
- ImagePainter: photorealistic, photorealisticHisto, photorealisticBlue, photorealisticNight, sky
- PanoramaComputerBean: Constructor & attributes, imagePainterProperty, labelColorProperty, synchronizeParameters, draw