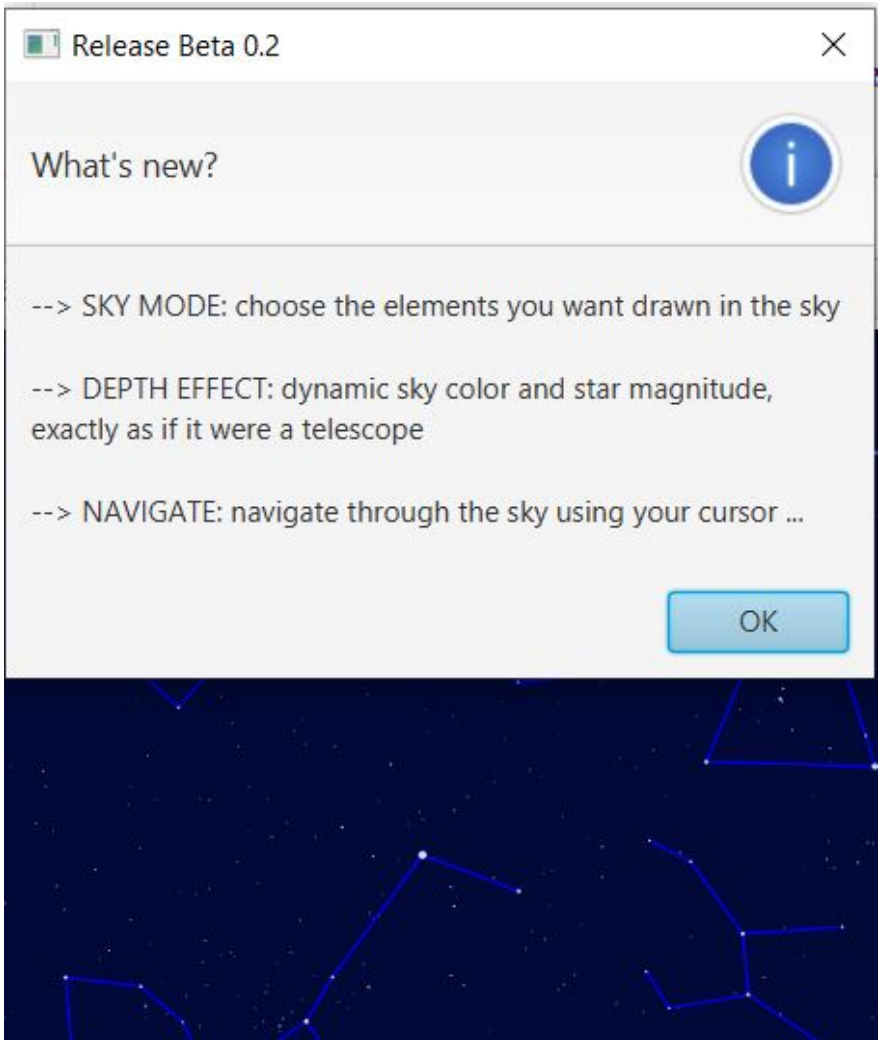
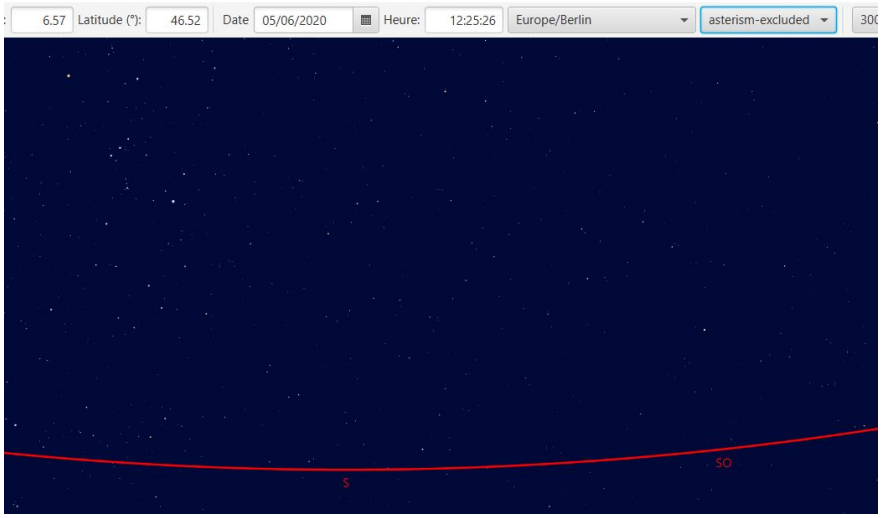
Etape 12 Code Report

User Perspective

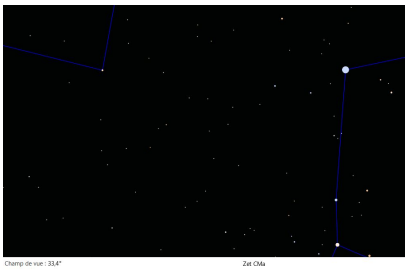
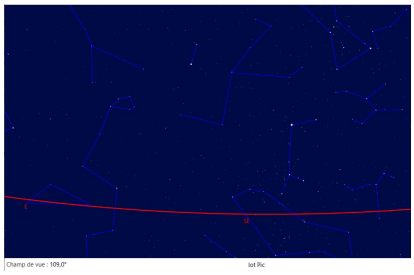
To ensure that the user is able to easily pick up the new bonuses we have added, we inform them of the new features through a pop-up display that alerts the user of the new capabilities:



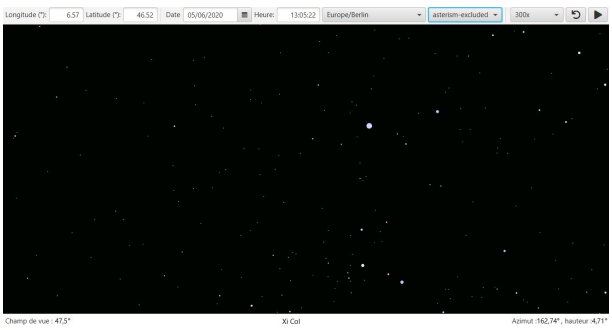
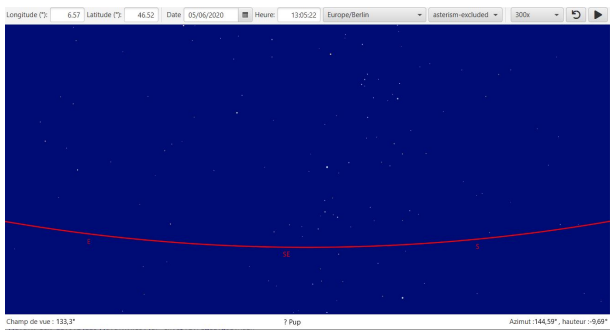
The first one is the Sky Mode and allows the user to remove from the screen the asterisms, the horizon and its annotations or the planets respectively. For instance, below the user has chosen a sky rendering without asterisms:



The second improvement is called “Depth Effect” and consists of two parts. The main purpose of the depth effect is to make the application feel like a telescope. First of all, the user finds that upon increasing field of view through scrolling out, the sky becomes increasingly blue. This has the purpose of enabling a more realistic user experience as if one were to use a telescope to observe the sky then indeed the sky would appear lighter at a larger field of view. This effect is seen below:

Sky color at minimal field of view	Sky color at larger field of view
	

The second effect is that of having dynamic star magnitude display. If the user is at a larger field of view, then fewer stars should be visible. To make this possible, we set a maximum star magnitude to display and this maximum value increases as the user scrolls in.

More stars shown upon zoom in	Fewer stars shown at zoom out
	

Finally, the last extension is that the user is able to change the projection center by a right mouse click. This is meant to make the program more intuitive for the user as most users are used to using the mouse click as a substitute for keyboard navigation.

Implementations in Java

Implementing SKYMODE

To implement the Sky Mode we created a bean storing the property on the form of a simple string property and the correspondent simple boolean properties isNonAsterism, isNonHorizon, isNonPlanet.

```
// EXAMPLE CODE SNIPPET(ONLY PORTIONS TAKEN FROM PRINCIPAL CODE)
private SimpleStringProperty mode = new SimpleStringProperty();
private SimpleBooleanProperty isNonAsterism, isNonHorizon, isNonPlanet;
 public boolean isIsNonAsterism() {
        if(!modeProperty().get().equals("asterism-excluded")){
            return false;
        }
        else{
            return true;
        }
    }
```

In the SkyCanvasManager, the observedSky object has the skyMode bean as a dependency which means that the paintSky method is called upon a change in skyMode. The the paintSky method simply checks the 3 booleans above the draw the sky as intended:

```
// EXAMPLE CODE SNIPPET(ONLY PORTIONS TAKEN FROM PRINCIPAL CODE)
private void paintSky(SkyModeBean skyMode, MagnitudeBean magnitude, BackgroundRgbBean color){

        painter.get().clear(color);
        if(!skyMode.isIsNonAsterism()){
            painter.get().drawAsterisms(sky.get(), projection.get(), planeToCanvas.get());
        }
        painter.get().drawStars(
            sky.get(), projection.get(), planeToCanvas.get(), magnitude.getMagnitude());
        if(!skyMode.isIsNonPlanet()){
            painter.get().drawPlanets(sky.get(), projection.get(), planeToCanvas.get());
        }
```

To this purpose we split the drawStars method and the drawAsterisms in order to control them independently.
To allow the user to control the sky mode we created a ChoiceBox inside our Main class whose text property is bound to the skyMode property of the skyModeBean:

```
// EXAMPLE CODE SNIPPET(ONLY PORTIONS TAKEN FROM PRINCIPAL CODE)
        ObservableList<String> skyDrawingOptions = FXCollections.observableArrayList(
            "asterism-excluded", "horizon-excluded", "planet-excluded", "normal");
        ChoiceBox<String> skyStructure = new ChoiceBox<>(skyDrawingOptions);
        skyStructure.setValue("normal");
        skyStructure.valueProperty().bindBidirectional(skyMode.modeProperty());
```

Implementing DEPTH EFFECT

The main challenge in implementing depth effect was finding an accurate mathematical model that would vary the maximum star magnitude and also control the RGB background color.
After various testing we created three different third degree polynomials that controls with an S-shaped curve the values of green, blue and magnitude (red remaining constant) as a function of the field of view. The blue goes roughly from 150 to 0, the green from 43 to 0. Overall the background color is dark blue when the field of view is wide and becomes black as we narrow down and focus, at the same time more stars appear as we rise the threshold for the magnitude of the stars displayed.
We implemented the BackgroundRgbBean to hold the simple object property Color and MagnitudeBean to hold the simple float property magnitude:

```
// EXAMPLE CODE SNIPPET(ONLY PORTIONS TAKEN FROM PRINCIPAL CODE)
        public class MagnitudeBean {
        .....
        private SimpleDoubleProperty magnitude = new SimpleDoubleProperty(60.0);
        .....
        }
        public class BackgroundRgbBean {
        .....
        private SimpleObjectProperty<Color> backgroundColor =
        new SimpleObjectProperty<>();
        .....
        }
```

These properties are set in the SkyCanvasManager and they change with the scroll of the mouse as the field of view. This manipulation is simply done by a method called `setStarMagAndRgb` which simply uses the polynomials we defined and calls their at method passing in the fieldOfView as an argument. Of course, a detail to notice is that the polynomials used in obtaining the green and blue RGB components return doubles but we really need an int. Hence we simply floor the result and pass in the floored result in `Color.rgb` of JavaFX.

```
// EXAMPLE CODE SNIPPET(ONLY PORTIONS TAKEN FROM PRINCIPAL CODE)
        private static final Polynomial RGBGREEN_CURVE =
        Polynomial.of(-0.000008384,0.00227894, -0.09159, 4.438179);
        .....
       private void setStarMagAndRgb(ScrollEvent e, MagnitudeBean magnitude,
       BackgroundRgbBean backgroundRgb){
        e.consume();
        magnitude.setMagnitude(MAGNITUDE_CURVE.at(viewParam.getFieldOfView()));
        int green = (int)Math.floor(
            RGBRANGE.clip(RGBGREEN_CURVE.at(viewParam.getFieldOfView())));
        int blue = (int)Math.floor(
            RGBRANGE.clip(RGBBLUE_CURVE.at(viewParam.getFieldOfView()/1.5)));
        int red = 0;
        backgroundRgb.setBackgroundColor(Color.rgb(red,green,blue));
    }
```

Implementing KEYBOARD CLICK

To implement this, we simply add an extra line of code to our listener from etape 11 that listened for a mouse click. When the mouseClick listener calls the setFocus method, we add the below line of code that ensures that now a right click on the mouse will modify the projection center:

```
viewParam.setCenter(HorizontalCoordinates.ofDeg(
    AZ_INTERVAL.reduce
    (mouseHorizontalCoordinates.get().azDeg()),
    ALT_INTERVAL.clip(mouseHorizontalCoordinates.get().altDeg())));
```

Alp Ozen (314542)
Jacopo Ferro (299301)
Laus, 05/06/2020