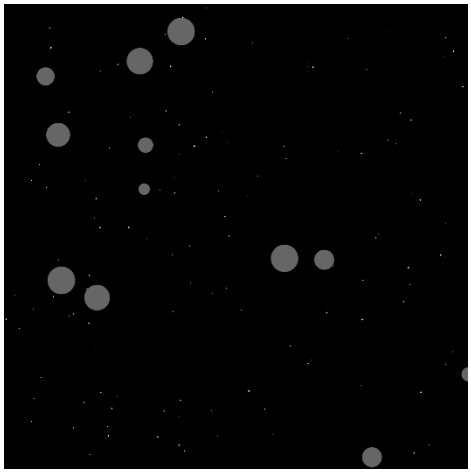
CMPT 381 Assignment 4:
Multiple Views, Transforms, and Selection

Due: Sunday, Dec. 10, 11:59pm

Overview

In this assignment you will build an interactive visual system that shows a simulation of asteroids floating in space. The system will demonstrate your understanding of MVC, multiple views, graphics transformations, and object selection. The system shows procedurally-generated asteroids on a simulated starfield, and allows the user to select asteroids with the mouse and fling them in a new direction. The assignment is divided into four parts, covering basic MVC architecture, creation and animation of the asteroids, additional views, and user interaction.

Part 1: MVC Basics (10 marks)



Set up a standard MVC architecture for your system, and include the following classes: SpaceApp (the application class), MainUI, SpaceModel, SpaceView, SpaceController, InteractionModel, and PublishSubscribe. These classes will be similar to what you have seen in labs and previous assignments.

Create class Asteroid to store each of the objects in the model: include variables for location and radius, and store asteroid positions using normalized coordinates (0.0-1.0). Create class Star to store the objects in the background starfield (the model should create 100 stars at startup). In MainUI, create 10 asteroids using the model's createAsteroid() method.

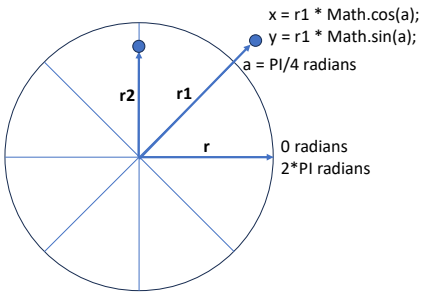
Your view class should take constructor parameters that define the size of the canvas (assume that the canvas will always be square). Use immediate-mode graphics to draw the starfield as white dots on a black background, and the asteroids as circles with the correct location and radius.

The result of these steps should display a view such as the picture at right (asteroid positions will differ).

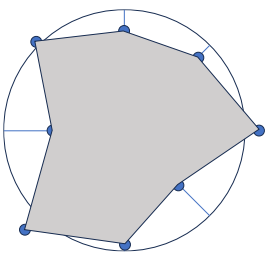
Part 2: Procedural Asteroids and Animation (20 marks)

Add to your Asteroids class to create a different polygon for each asteroid. Use the following generation method:

- Choose a radius for the asteroid (a random value within a range)
- Divide the 2*PI circle into 4-8 sections; for each angle:
    - Choose a new radius for this section (a random value based on the original radius)
    - Find the point on the circle of the chosen radius (using the equations in the figure at right)
    - Add the x and y values to two arrays for that asteroid (store the points as "home coordinates" centred at the origin, as discussed in class).

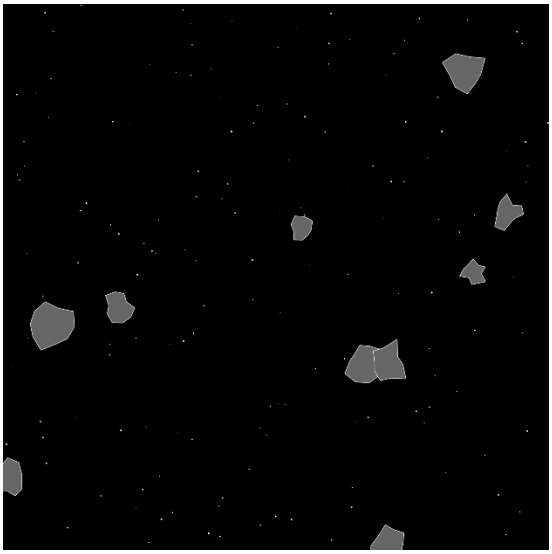


The resulting set of points (such as the example at right) can be drawn as a polygon using GraphicsContext.fillPolygon().



Add instance variables to Asteroid to store the x and y translation of the object (i.e., its position), and its angle (so that it can spin around its centre point). Add instance variables xVelocity,

yVelocity, and aVelocity to represent the amount that should be added to the asteroid's x, y, and angle variables in each animation frame. Initial values for the velocity variables should be randomly generated.



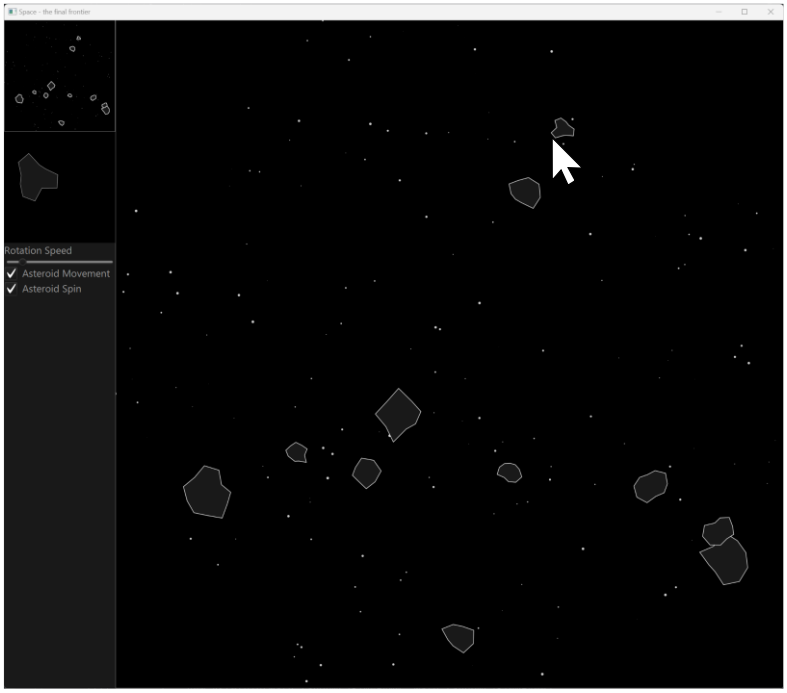
In class MainUI, add a JavaFX AnimationTimer that will call method handleAnimationTick() in your controller. Add methods moveAsteroids() and spinAsteroids() to your model that will be called on each animation tick. (Do not use any actual physics to move or spin the asteroids – just add the asteroid's velocity variables to its current position and spin; similarly, do not consider collisions). When an asteroid moves off screen, move it to the opposite side of the screen (i.e., wrap-around).

Add an instance variable worldRotation to your InteractionModel to track the overall rotation of the view. In your handler for each animation tick, also increment the world rotation so that the entire scene rotates around the centre. With all of these changes, make sure that you are communicating between the MVC components in the correct way, and that you are maintaining clear separation between the modules.

In your view class, use the transformation methods of GraphicsContext to correctly draw the moving and spinning asteroids, and the overall rotation of the world (static picture shown at right).

Part 3: Multiple Views (30 marks)

Create three additional views and add them to a left-side panel in your UI. First, add a miniature view at the top of the left panel – the miniature view is another instance of your SpaceView class, but with a smaller size. Second, add a CursorView to the panel – this view is the same size as the miniature view, but shows a 2X zoomed region (assuming the main view is 1X), centred on the cursor's location; the CursorView does not show world rotation. Third, the ControlPanelView is a collection of the following widgets: a Label, a Slider, and two CheckBox buttons. The slider controls how fast the world rotates, and the checkboxes control whether asteroid movement and spin are shown (i.e., when these are switched off, your animation tick handler in the controller should not call model methods to update the position or spin of the asteroids).



The miniature view is simply another instance of SpaceView, so no additional code needs to be written (storing all values as normalized coordinates should allow a SpaceView to display correctly at any size). The CursorView needs to be a separate class, but should share most of the code written for SpaceView. The additional views should look similar to those shown in the picture.

Note that the dark theme on the widgets in the control panel was achieved using the following CSS in MainUI:

```
this.setStyle("-fx-base: #191919; -fx-background-color: #191919");
```

Part 4: Selection and Interaction (40 marks)

Add the ability to select asteroids using the mouse. Add event handling so that SpaceView sends mousePressed, mouseDragged, and mouseReleased events to the controller. The first step for the selection is to transform the mouse

location from the screen reference frame to the reference frame of the asteroids, which involves two rotations (rotating to account for the world rotation, and rotating to account for the asteroid's rotation) and one translation (to account for the asteroid's position). The following methods will be useful:

```
private double rotateX(double x, double y, double radians) { return Math.cos(radians) * x - Math.sin(radians) * y; }
private double rotateY(double x, double y, double radians) { return Math.sin(radians) * x + Math.cos(radians) * y; }
```

The first point-based selection method will use the bitmap approach described in lectures (using a secondary Canvas, a WriteableImage, and a PixelReader).

- Add to the constructor of class Asteroid to create a bitmap for the asteroid. The bitmap should be large enough to draw the entire asteroid in pixel coordinates at 1X (note that you will now need to pass in the main-view width and height so that you can create the bitmap at the correct size)
- Fill the bitmap with black
- Draw the asteroid on the bitmap, centred at the middle of the bitmap, filled with white

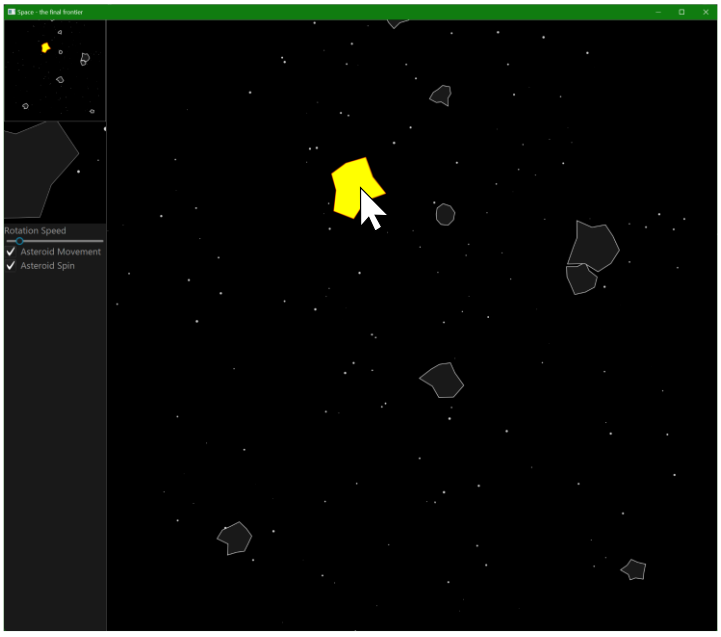
Then write method `public boolean contains(double x, double y)` in class Asteroid that takes in a cursor location (that has already been transformed based on the world rotation), transfroms the location to the reference frame of the asteroid, and checks the color of the pixel under the x,y position using the PixelReader. If the color is equal to Color.WHITE, then the x,y position is on the asteroid.

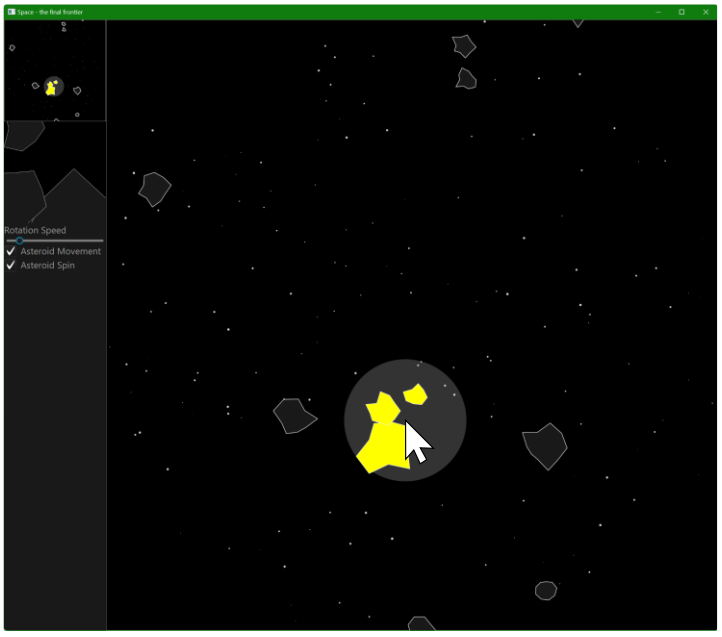
Add additional methods methods and code in the model to call contains() on each of the asteroids in the model, in the interaction model to keep track of the selection, and in the view to show selected asteroids with yellow fill colour. In addition, use the z-order method discussed in lectures to ensure that any asteroid the user clicks on is brought to the top of the view stack.

The second selection method will use the area-cursor approach discussed in lectures. Add code to your view class to capture mouse-wheel events (using the Node.setOnScroll() method), and send these events to a new method handleWheel() in the controller. The deltaY value in the event will either be positive or negative, and you can use this value to increase or decrease the radius of the area cursor (up to a maximum). The size of the area cursor should be stored in the interaction model, and the area cursor should be shown in the SpaceView.

To use the area cursor to select asteroids, add methods to the model and the Asteroid class to determine whether any of the control points of the asteroids are inside the area cursor – if any point is inside, the asteroid should be selected. Note that multiple asteroids can now be selected, so your code that keeps track of the selection may need to be adjusted to handle a list instead of a single object. The area-cursor selection technique does not need to bring asteroids to the top of the view stack (i.e., it does not need to use z-ordering).

The two selection operations are shown in the images below (point select at left, area-cursor selection at right).





Notes for the selection operations:

- Selected asteroids should have their x and y velocities set to zero so that they do not drift away from the cursor
- Point-based selection should work regardless of the size of the area cursor (e.g., if the area cursor is too small to intersect with any control points, the point-based method described above should still work).
- Selection should work equally well in the miniature SpaceView as in the main SpaceView

Last, add the ability to drag and fling the selected objects. Add code to your controller to allow the user to drag any asteroids that are selected (through either method). When the mouse is released after dragging one or more asteroids, the final dX and dY values for the drag become the asteroid's new x and y velocities.

Update your publish-subscribe mechanism to create additional channels so that views only receive notifications about things they need to know (e.g., the cursor view does not need to know about selection or the area cursor).

Additional Notes:

- Your system does not need to handle resizing for any part
- You may assume that the views will always be set to square dimensions

What to hand in (each student will hand in an assignment)

- Create a zip file of your IDEA project (File → Export → Project to Zip file…). Note that you do not need to hand in separate files for Part 1, 2, 3, and 4: if you have completed Part 2, you do not need to hand in anything for Part 1; if you have completed Part 4, you do not need to hand in anything for Part 1 or 2 or 3.
- Add a readme.txt file to the zip that indicates exactly what the marker needs to do to run your code, and explains which elements of the assignment are working or not.
- Systems for 381 should never require the marker to install external libraries, other than JavaFX.
- This is an individual assignment – each student will hand in separately
- Review the material in the course syllabus regarding academic honesty, and follow all guidelines when completing this assignment. If you have any questions, contact your instructor

Where to hand in

Hand in your zip file to the Assignment 4 link on the course Canvas site.

Evaluation

- Part 1: all MVC classes are implemented as specified, and objects are stored using normalized coordinates.
- Part 2: asteroid shapes are created as specified, and all animations (asteroid movement, asteroid spin, and world rotation) are correctly controlled and displayed using an animation timer.
- Part 3: the additional graphical views are implemented as specified, and show the correct parts of the model; the control panel operations correct affect the simulation as specified.
- Part 4: both types of selection are correctly implemented, and user interactions work correctly.
- Overall, your system should run correctly without errors, and your code should clearly demonstrate that you have satisfied the software requirements stated in the assignment description. (Document your code to assist the markers understand how you are satisfying the software requirements)

If parts of your system have only partial implementations (e.g., a feature does not work but has been partially developed), clearly indicate this in your readme.txt file. Code should be appropriately documented and tested (although documentation will not be explicitly marked).

In general, no late assignments will be allowed, and no extensions will be given, except for emergency or medical reasons. (If you wish to use your one-time free extension, you must contact the instructors at cmpt381@cs.usask.ca before the deadline).