CMPT 381 Assignment 3: MVC and 2D Graphics

Due: Sunday, Nov. 5, 11:59pm – **EXTENDED to Nov. 12, 11:59pm**

Overview

In this assignment you will build a visual editor in JavaFX that demonstrates your skills with immediate-mode graphics, interaction with graphics, model-view-controller architectures, and multiple views. The system is an interactive editor for creating and manipulating blocks in a visual workspace. Part 1 covers the basic system and architecture, Part 2 covers user interactions with the system and multiple views, and Part 3 covers a hotkey guide interaction technique.

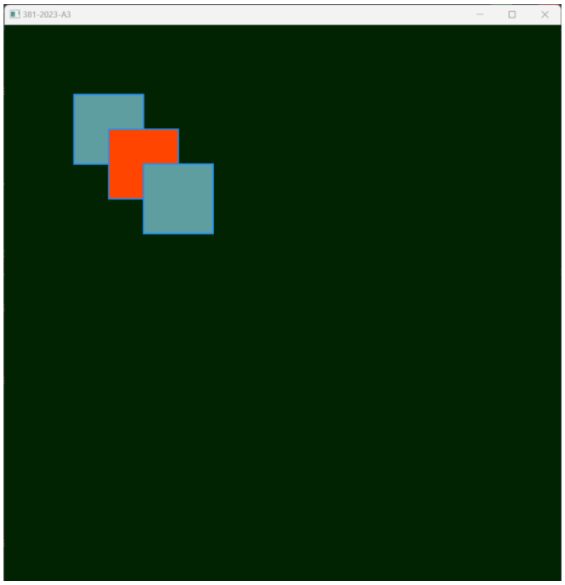
The editor allows users to execute commands using hotkeys: commands include creating objects, moving an object cursor through the visible objects, selecting objects, and manipulating selected objects (movement, deletion, alignment, distribution, and resizing).

You will develop this application incrementally through the four parts described below.

Part 1: MVC Basics

Using a full MVC architecture, develop a JavaFX application that shows visual objects that are contained in a model. Write the following classes (you may also write other classes as needed):

- Application class
  - EditorApp: the main application class
- Model classes
  - BoxModel: the model that stores all objects
  - Box: the objects that are stored in the model
- Interaction model classes
  - InteractionModel: the interaction model that stores all information related to the app's interaction state including the object cursor
- View classes
  - MainUI: a view that contains and lays out the graphical view (and will be used for an additional view in Part 2)
  - BoxView: a graphical view of the objects in the model
- Controller classes
  - AppController: the controller to handle all user events



Part 1 UI, with three objects created and the object cursor on the second object.

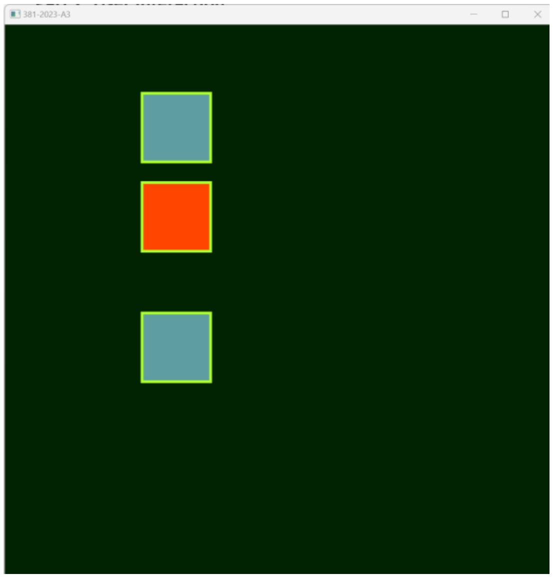
Interaction requirements:

- When the user presses Control-C the system creates a new object (the first object appears at 100,100, the second at 150,150, the third at 200,200, etc.)
- When the user presses Tab, the system will move an object cursor to the next object in the model's list; the object under the cursor is shown in orange.
- At system startup the object cursor is not on any object; the first time the user presses Tab, the cursor will move to the first object in the model
- The object cursor wraps around from the last object in the model back to the first

Software requirements:

- Your graphical view should be 800x800 pixels in size.
- Your system must use an MVC architecture with publish-subscribe communication between models and views (note that you will be upgrading the publish-subscribe mechanism in Part 3)
- Your controller should implement all methods for handling user events

Part 2: User Interaction



Part 2 UI, after creating three objects, moving them, selecting all with Control A, and aligning left with Control L.

Additional interaction requirements:

Implement additional commands that allow the user to select and manipulate objects in the system. Use a state machine approach in your controller, as described in lectures.

Commands to move the cursor (in addition to Tab from Part 1):

- Left/Right/Up/Down arrow: moves the cursor to the object that is immediately left/right/up/down of the current object in the horizontal dimension, based on the objects' left/top edges

Commands to set the selection:

- Control-S: selects the object underneath the object cursor (if it is not selected) or unselects it (if it is selected).
- Control-A: selects all objects (if not all are selected); if all are selected, unselect all objects

Commands to manipulate the selection:

- Control D: deletes the selected objects
- Control Left/Right/Up/Down arrow: moves the selected objects left/right/up/down by 15 pixels
- Control U: increase the width and height of the selected objects by 5 pixels each (no change to left/top)
- Control J: decrease the width and height of the selected objects by 5 pixels each
- Control L/R/T/B: align the left/right/top/bottom edges of the selected objects to the left/right/top/bottom edge of the object under the cursor (if there is one); if not, align to the edge of the first object in the selection
- Control H/V: distribute the selected objects evenly in the horizontal/vertical dimension, between the left/top edge of the leftmost/topmost selected object and the right/bottom edge of the rightmost/bottommost selected object

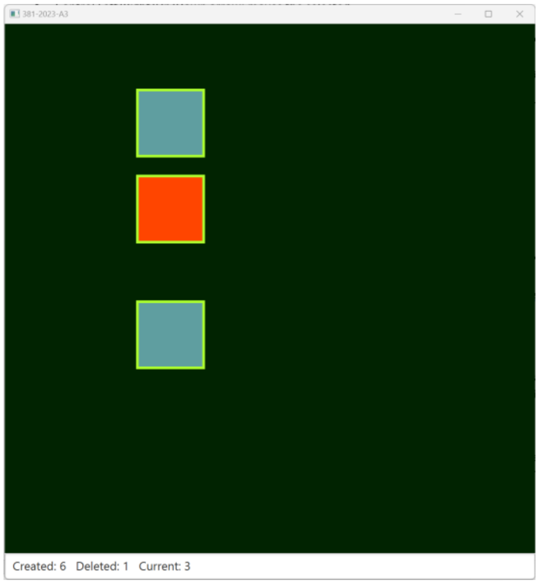
Additional software requirements:

- All of the added commands should be handled by your controller class
- Selected objects are shown in the view with a 5-pixel-wide yellow border

Part 3: PublishSubscribe Notifier and Second View

Implement a separate publish-subscribe manager, as discussed in lectures, to handle all inter-MVC communication in your system:

- Create class PublishSubscribe that maintains a list of subscribers for each of several channels
- Channels have a string key (e.g., "create"), and the class maintains a map from the key to a list of subscribers
- Create a Subscriber interface that will be the type of all items stored in the subscriber lists
- The PublishSubscribe class should allow publishers to create a channel (with a specific string key) and publish to a channel
- Subscribers should have a method to receive a notification (with the channel key passed as a parameter).
- Your publishers should create channels with keys "create", "delete", "update", and "selection".



Created: 6 Deleted: 1 Current: 3

Part 3 UI showing status bar view at bottom.

Create a second view that shows a status bar at the bottom of the window:

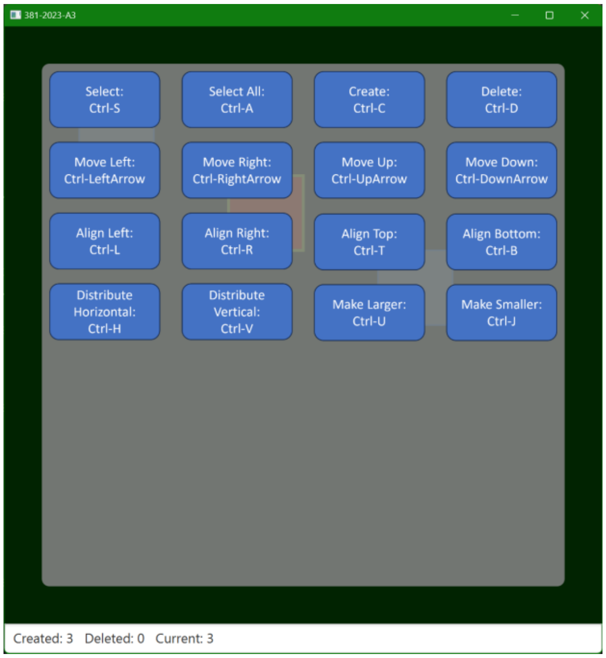
- The status bar should be encapsulated in a class StatusBarView
- The status bar view should subscribe to “create” and “delete” events; when it receives these notifications, it updates the status bar to show the number of objects that have been created since the system started, the number that have been deleted, and the current total number on screen.

Part 4: Hotkey Guide

Since all of the interaction with the system uses hotkeys, the user may forget which hotkey is associated with which command. To assist them, extend your system to show a pop-up hotkey guide as shown in the image below.

Additional interaction requirements:

- If the user presses and holds Control for longer than 1000ms, the guide appears; the user can then press the correct hotkey to execute the command.
- If the user wants to check their memory about a hotkey that they think they remember, they can press the full hotkey (e.g., Control-S) but hold both keys for 1000ms to show the guide; to execute the hotkey, the user must release the hotkey (e.g., the S) but not the Control key, and then press a hotkey (they can execute any hotkey with the second press).
- If the user releases Control before the hotkey, the command executes as normal (but the hotkey must be pressed before the Control key is released)
- If the user presses and releases a complete hotkey before the 1000ms timeout, the hotkey executes normally
- Cursor manipulation commands are not shown in the guide.



Created: 3 Deleted: 0 Current: 3

Part 4 UI showing the hotkey guide

Additional software requirements:

- Extend your controller state machine to implement the new interaction technique. Note that you must model the interaction using states rather than using other mechanisms (e.g., booleans or nested if/then statements)
- Note that you will need to execute hotkeys on key released, rather than on key pressed
- The guide is shown in the view as a transparent grey rectangle with the guide image shown on top. Place the image in the ‘resources’ folder of your IntelliJ project

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 3 link on the course Canvas site.

Evaluation

- Part 1: all classes are implemented as specified, all required interactions are working, and all software requirements are implemented as specified
- Part 2: all commands are working as specified, and all are handled by the controller’s state machine
- Part 3: the publish-subscribe classes are implemented and used as specified; the new view is implemented as specified, and all views use the publish-subscribe notifier
- Part 4: the hotkey guide technique is implemented as specified
- 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).