1 Introduction

The following exercises are updates of the Keyboard State Processing program. Put each project in its own directory and zip the entire set of directories into one zip file. Upload the zip file to the Homework #2 page of the MyClasses site for this class.

2 Exercises

1. The following exercise is an update of the Keyboard State Processing program example. Read through the Box class carefully and make sure that you understand all of the code. Make the following changes to the project.
   (a) Make the title bar display “Homework #2 — Program #1”.
   (b) In the keyboard state processing function of the UI class, remove all the features in the alt and shift modifier sections. Also remove the code from the alt modifier in the keyPressed function.
   (c) Add in the following if either alt key is pressed, B will turn the box solid blue, G will turn the box solid green, W will turn the box solid white, and M will reset the box vertices to the red/green/blue/white default vertex colors.
2. This exercise is an update the previous program. Copy all the files from the previous exercise to a new folder, hence new project. Make the following additions.
   (a) Make the title bar display “Homework #2 — Program #2”.
   (b) Add in the mouse functionality to highlight the box in red when the mouse is over the box and revert to its original colors when the mouse is not over the box. Here I would suggest to create a method inside the box class that will return a boolean of true if the mouse point is inside the box and false if the mouse point is not inside the box. You can then use this function to reset the color of the box.
   (c) Add in the mouse functionality to do a click and drag movement of the box. So if the mouse is over the box then a left click and drag will move the box along with the mouse. This is to be done smoothly with no “snapping” to the center of the box. So if the user has the mouse in the upper left corner of the box, does a click and drag, the mouse stays in the upper left corner of the box, it does not change the position to the center of the box.
3. This exercise is an update the previous program. Copy all the files from the previous exercise to a new folder, hence new project. Make the following additions.
   (a) Make the title bar display “Homework #2 — Program #3”.
   (b) The program will start with a blank screen.
   (c) If the control is down and the user right clicks the mouse on a place where there is no box then the program will place a box on the screen with center at the click position and a random width and height between 0.1 and 0.3 in length. The color of the box is to be a single color chosen at random from all possible colors.

       If the control is down and the user right clicks the mouse on an existing box then the box is removed.
   (d) If the mouse hovers over a box the box should be highlighted in red and then return to its original color when the mouse leaves the box.
   (e) If the user left clicks a box with the control key down, the current box will toggle a selected mode. Selected boxes will be yellow (RGB being $(1, 1, 0)$). The red hover will take precedence over the yellow.

(f) Control+A will select all of the boxes and Control+Q will deselect all of the boxes.

(g) If the control key is down and the user clicks outside of any of the boxes then all boxes are deselected.

(h) If the mouse is over the box then a left click and drag will move the box along with the mouse. This is to be done smoothly with no "snapping" to the center of the box. So if the user has the mouse in the upper left corner of the box, does a click and drag, the mouse stays in the upper left corner of the box, it does not change the position to the center of the box.

(i) The following options will apply to selected boxes only, but to all the selected boxes together.

- If no modifier keys are pressed:
  - Left: Moves the box to the left.
  - Right: Moves the box to the right.
  - Up: Moves the box up.
  - Down: Moves the box down.
- If the control key is down:
  - Left: Decreases the width of the box.
  - Right: Increases the width of the box.
  - Up: Decreases the height of the box.
  - Down: Increases the height of the box.
- If the alt key is down:
  - Up: Increases the height and width of the box.
  - Down: Decreases the height and width of the box.

(j) If the Shift key is down, and the mouse is over a selected box, then a left click and drag will move all of the selected boxes the box along with the mouse, but only the selected boxes. This is to be done smoothly with no "snapping" to the center of the box.

(k) Other program features:

- If no modifier keys are pressed:
  - Escape: Ends the program.
  - F1: Sets the mode to fill.
  - F2: Sets the mode to line.
  - F3: Sets the mode to point.
  - F4: Toggles between 60 FPS and full speed.
  - F8: Clears all of the boxes from the screen.
  - F12: Saves a screen shot of the graphics window to a png file.

Note that since you will potentially be deleting boxes from the program while it is running you should also create a destructor for the box class. You do not need to worry about clearing memory on the CPU side since Python will take care of that, but you need to clear the memory that you allocated on the graphics card. This is done with the glDeleteBuffers and the glDeleteVertexArrays commands like you used with the star in the last assignment.

```python
def __del__(self):
    try:
        glDeleteBuffers(1, self.ArrayBuffer)
        glDeleteBuffers(1, self.BoxEBO)
        glDeleteVertexArrays(1, self.BoxVAO)
    except Exception as err:
        for i in range(len(err.args)):
            print(err.args[i])
```

One interesting thing happened when I included this into the box class. If I ran the program and did not do any box deletions, i.e. a Control+Right-Click on an existing box, when the program ended I got the following exception on each Box object deletion.

```
No array-type handler for type <class 'numpy.uint32'>
```

On the other hand, if I deleted at least one box during the program run, there were no exceptions and the destructors worked fine. To me this looks like a numpy issue but it could easily be my code. Now this only happened when the program was closing and frankly at that point it really does not matter. One band-aid that stopped this issue was to create and delete a box in the GraphicsEngine constructor, i.e.

```python
box = Box()
del box
```

4. This exercise is an update the previous program. Copy all the files from the previous exercise to a new folder, hence new project. Make the following additions and changes.

   The only file that will need to be changed here is the UI. Although the Box class will have more functionality than is needed for this exercise you do not need to remove anything and use it as is. All the UI is to do for this program is create a random box on each frame while the program is running. The box center should be anywhere on the screen and the width and height should be random between 0.2 and 0.6. Once the number of boxes exceeds 200 remove the first box from the list so that there are always 200 boxes being drawn on the screen.

   What you have created is the "boxes" screensaver. When the first screensavers came out they were necessary to prevent burn-in on the CRT monitors of the time, this is not needed with our current screen technology. One of the first screensavers was the warp speed Star Trek screensaver that is now used as in technology jokes, but in the same package was this boxes screen saver, as well as a couple others.