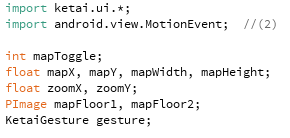
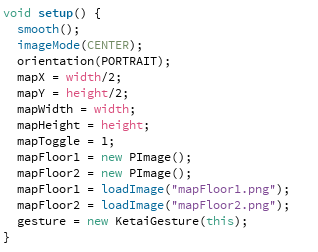
How to Create an Interactive Map of MODS using Processing

By Amarnath Maharaj, Kyle Martinez, and Bradley Zarek

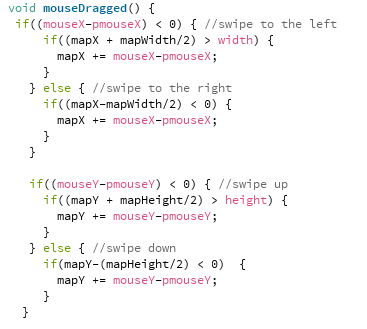
1. Import ketai.ui.\* and android.view.MotionEvent. Create the following variables. mapToggle is used to switch between the first and second floors. mapX, mapY, mapWidth, and mapHeight are used to store the position and size of the map. zoomX and zoomY are used to store the zoom ratio, and the images of the two floors are stored as PImages. In addition, a KetaiGesture object should be created.



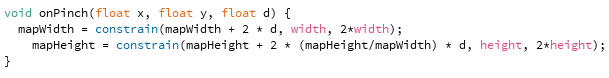
1. Create a setup() method. Under that method, set mapX equal to the width of the screen divided by two, and set mapY equal to the height divided by two. This centers the image. The mapWidth and mapHeight variables should equal the width and height to scale the image to the screen size. The images for the two maps are loaded. Make sure the image mode is set to center. Also, the orientation of the screen is set to portrait.



1. Create a mouseDragged() method. Under the mouseDragged() method, write one if statement to determine whether the mouse was dragged to the left or the right. If the mouseX-pmouseX is negative, the finger is moving to the left. Otherwise, it is moving to the right. For a swipe to the left, the next if statement makes sure that the right edge of the image is not past the right edge of the screen. If it has not reached that point, the difference between mouseX and pmouseX (which is negative) is added to the mapX integer. For a swipe to the right, a similar if statement makes sure that the left edge of the image is not past the left edge of the screen. If that is true, it adds mouseX-pmouseX, which is positive. Write another if statement for the y coordinate, using mouseY and pmouseY, mapY, mapHeight, and height instead.



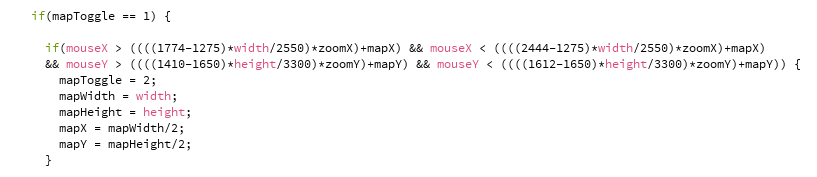
1. Create an onPinch method with arguments float x, float y, and float d. One statement in the method sets mapWidth equal to the mapWidth plus twice the distance d. It constrains that value in the range of width to twice the width so that you can’t zoom in or out too far. The same statement is used for mapHeight, but the value is multiplied by the height-to-width ratio. This keeps the image to scale when zooming.



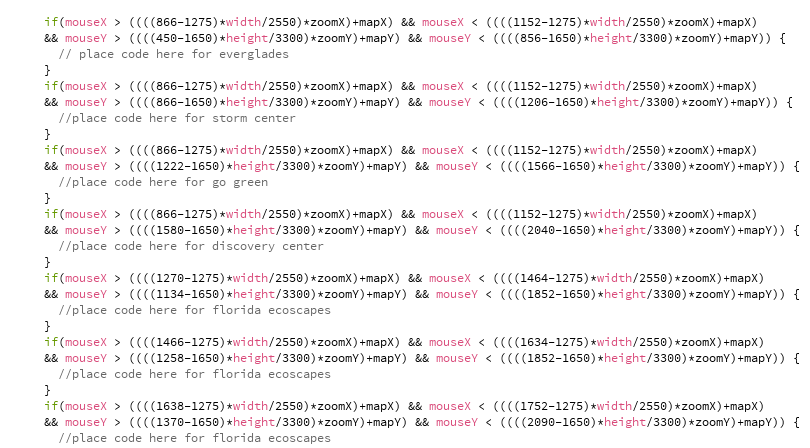
1. Create and onTap(float x, float y) method. Set zoomX equal to the ratio of the mapWidth to the screen width and set zoomY equal to the ratio of the mapHeight to the screen height.



1. Under the onTap method, write an if statement that checks if the mapToggle is on floor 1. Then create an if statement that determines if the switch floor button is clicked. The original map image is 2550 by 3300 pixels. The position of the floor number (from the top left) is between x=1774 (lower bound of x), x=2444 (upper bound of x), y=1410 (lower bound of y), and y=1612 (upper bound of y). For each x value, subtract 1275 (the pixel width divided by 2) from the x position to find the x distance from the center line. Then multiply it by the width divided by the pixel width to adjust the position to the screen width. Next, multiply that value by zoomX and add mapX. Write the same code for y using the y values. If mouseX and mouseY are greater than the lower bounds and less than the upper bounds, it will execute the code within the if statement. In this case, it changes the mapToggle value and resets the values for the image size and position.

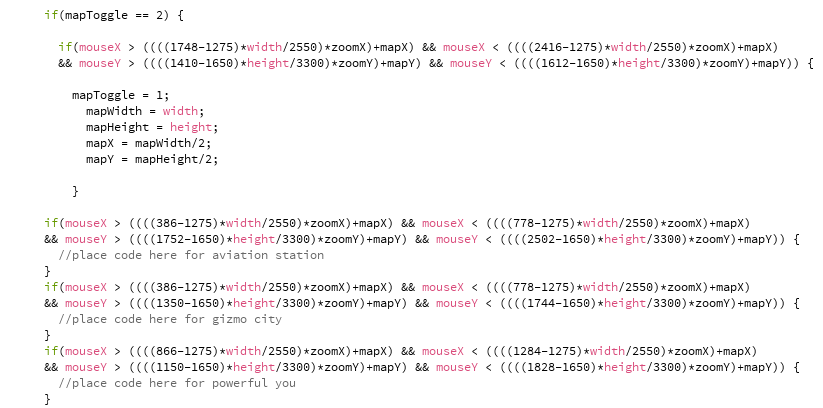


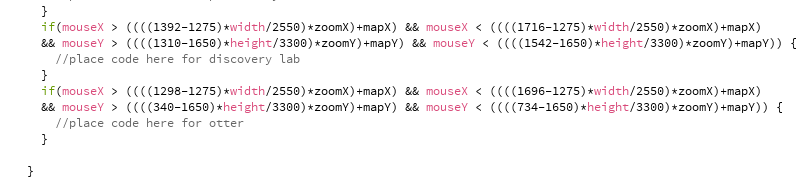
1. Write additional if statements for each location on the map using those x and y pixel coordinates.



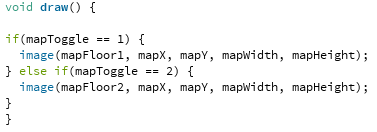


1. Write an if statement to check if the user is on the second floor, and create similar code.





1. Create a draw() method and write an if statement that switches between the first and second floors. If mapToggle equals 1, it displays the first floor map. If mapToggle equals 2, it displays the second floor map. mapX and mapY are used as the position of the map, and mapWidth and mapHeight are used as the width and height of the image.



1. Include the following code to ensure that mouseX and mouseY values are updated.

