```
int [][] tileA ={
  {0,0,0,0,0},
  {0,1,1,1,1},
  {0,1,1,1,1},
  {0,1,1,1,1},
  {0,1,1,1,1},
  };
int [][] tileB ={
  {1,1,1,1,1},
  {1,0,0,0,0},
  {1,0,0,0,0},
  {1,0,0,0,0},
  {1,0,0,0,0},
  };
boolean overBox = false; //updates when mouse hovering over specified locations
boolean leftClick = true;//updates whether click was right or left
boolean locked = true;   //updates when mouse is clicked over specified locations
int init = 16;           //initial size which code is built on (size of a single
                         //point in a tile)
int size = init*5;       //size of a tile (5x5 collection of points)
int next = size+init*2; //used for giving accurate spacing between (ui) tiles
                         //based on the initial size
int order = 0;           //holds current value for order (decides how
                         //grid is drawn)

void setup() {
  size(1160, 840);
  background(255);
  PFont f =
    createFont("Arial",init,true);
  textFont(f,init);
  fill(0);
  text("Recolour",size,size+init*3);
  text("Reorder",size*1.08,next*3+init);
}

void draw() {
  //detects mouse hovering over a ui tile
  mouseOver();

  //draws ui tile a
  startPattern(tileA);
  pushMatrix();
  //draws ui tile b
  translate(next, 0);
  startPattern(tileB);
  popMatrix();

  //draws ui order tile (changes how drawn depending on order value)
  if (order % 2 == 0) {
    checkOrder(tileA, tileB);
  } else {
    checkOrder(tileB, tileA);
  }

  //draws the grid
  myDraw();
}

void mousePressed() {
  if (mouseButton == LEFT) {
    leftClick = true;
  } else {
    leftClick = false;
  }
  //if hovering over a box, register a mouse click
```

```
67.   if(overBox) {
68.     locked = false;
69.   } else {
70.     locked = true;
71.   }
72. }
73.
74. void mouseOver() {
75.   if (mouseX >= init && mouseX < init+size &&
76.     mouseY >= init && mouseY < init+size) {
77.     //triggers when a point on tile A is clicked
78.     recolour(tileA, init, init);
79.   } else if (mouseX >= next+init && mouseX < next+init+size &&
80.     mouseY >= init && mouseY < init+size) {
81.     //triggers when a point on tile B is clicked
82.     recolour(tileB, next+init, init);
83.   } else if (mouseX >= next && mouseX < next+size &&
84.     mouseY >= next*2 && mouseY < next*2+size) {
85.     //triggers when the reorder tile is clicked
86.     reorder();
87.   } else {
88.     overBox = false;
89.   }
90. }
91.
92. void checkOrder(int [][] myData, int [][] nextData) {
93.   //used to create the reorder tile in the ui portion of the program
94.   //uses the changes that will be applied to the grid
95.   pushMatrix();
96.   translate(init*3.5, next*2);
97.   if (order <= 1) {
98.     //normal pattern (for A/B)
99.     startPattern(myData);
100.         } else if (order <= 5) {
101.           //patterns for alternating A/B's
102.           scale(0.5);
103.           translate(init, init);
104.           myPattern(myData);
105.           translate(size, 0);
106.           myPattern(nextData);
107.           translate(-size, size);
108.           if (order <= 3) {
109.             //pattern for the parallel alternation
110.             myPattern(myData);
111.             translate(size, 0);
112.             myPattern(nextData);
113.           } else if (order <= 5) {
114.             //pattern for the alternating lead tile on each line
115.             myPattern(nextData);
116.             translate(size, 0);
117.             myPattern(myData);
118.           }
119.         } else if (order <= 9) {
120.           //patterns for the reversing A/B tiles
121.           scale(0.5);
122.           translate(init, init);
123.           myPattern(myData);
124.           scale(-1, 1);
125.           translate(-next-size, 0);
126.           myPattern(myData);
127.           translate(0, size);
128.           if (order <= 7) {
129.             //flipping only A or B's
130.             myPattern(myData);
131.             scale(-1, 1);
132.             translate(-next-size, 0);
```

```
133.            myPattern(myData);
134.          } else {
135.            //flipping alternating A and B's
136.            myPattern(nextData);
137.            scale(-1, 1);
138.            translate(-next-size, 0);
139.            myPattern(nextData);
140.          }
141.        }
142.        popMatrix();
143.      }
144.
145.      void reorder() {
146.        overBox = true;
147.        //if there has been a mouse click over the tile
148.        if(!locked) {
149.          if (leftClick) {
150.            //change to the next order in the cycle
151.            //cycle resets after 10th pattern
152.            if (order < 9) {
153.              order += 1;
154.            } else {
155.              order = 0;
156.            }
157.          } else {
158.            //change to the previous order in the cycle
159.            //cycle resets after first pattern
160.            if (order > 0) {
161.              order -= 1;
162.            } else {
163.              order = 9;
164.            }
165.          }
166.          //lock to prevent multiple runs,
167.          //unlocked when reclicked
168.          locked = true;
169.        }
170.      }
171.
172.      void recolour(int [][] myData, int minX, int minY) {
173.        overBox = true;
174.        //if there has been a mouse click over the tile
175.        if(!locked) {
176.          //get x and y on cursor and calculate
177.          //which point on the tile was clicked
178.          int x = (int) (long) ((mouseX-minX)/init);
179.          int y = (int) (long) ((mouseY-minY)/init);
180.          if (leftClick) {
181.            //change to the next colour in the cycle
182.            //for the clicked point - cycle resets after 7th colour
183.            if (myData[y][x] < 6) {
184.              myData[y][x] += 1;
185.            } else {
186.              myData[y][x] = 0;
187.            }
188.          } else {
189.            //change to the previous colour in the cycle
190.            //for the clicked point - cycle resets after first colour
191.            if (myData[y][x] > 0) {
192.              myData[y][x] -= 1;
193.            } else {
194.              myData[y][x] = 6;
195.            }
196.          }
197.          //lock to prevent multiple runs,
198.          //unlocked when reclicked
```

```
199.            locked = true;
200.          }
201.        }
202.
203.        void startPattern(int [][] myData) {
204.          //draws a smaller version of a single grid tile
205.          //with a blue square behind to highlight the tile
206.          stroke(0,0,255);
207.          fill(0,0,255);
208.          rect(init-1, init-1, size+2, size+2);
209.          stroke(195);
210.          myPattern(myData);
211.        }
212.
213.        void myPattern(int [][] myData) {
214.          //draw tile from array
215.          for (int i = 0; i < myData.length; i++) {
216.            for (int j = 0; j < myData.length; j++) {
217.              //change colour depending on array value
218.              switch (myData[j][i]) {
219.                case 1:
220.                  fill(127); break;
221.                case 2:
222.                  fill(255,0,0); break;
223.                case 3:
224.                  fill(255,255,0); break;
225.                case 4:
226.                  fill(0,255,0); break;
227.                case 5:
228.                  fill(0,0,255); break;
229.                case 6:
230.                  fill(0); break;
231.                default:
232.                  fill(255);
233.              }
234.              rect(init+(init*i), init+(init*j), init, init);
235.            }
236.          }
237.        }
238.
239.        void checkReversed(int [][] myData, int [][] nextData, int num) {
240.          if (order >= num) {
241.            //draw reversed tile
242.            scale(-1, 1);
243.            translate(-next, 0);
244.            myPattern(myData);
245.          } else {
246.            //draw unaltered opposing tile
247.            myPattern(nextData);
248.          }
249.        }
250.
251.        void myDraw() {
252.          //draw whole grid
253.          for (int i = 0; i < 10; i++) {
254.            for (int j = 0; j < 10; j++) {
255.              pushMatrix();
256.              translate(240+size*j, size*i);
257.              if (order <= 9) {
258.                //change lead tile depending on odd/even order value
259.                if (order % 2 == 0) {
260.                  myPattern(tileA);
261.                } else {
262.                  myPattern(tileB);
263.                }
264.                if (order >= 2) {
```

```
265.                //if order requires a different second tile,
266.                //this performs an extra loop cycle
267.                popMatrix();
268.                j++;
269.                pushMatrix();
270.                translate(240+size*j, size*i);
271.                //draw tiles flipped in y-axis
272.                if (order % 2 == 0) {
273.                  checkReversed(tileA, tileB, 6);
274.                } else {
275.                  checkReversed(tileB, tileA, 7);
276.                }
277.              }
278.            }
279.            popMatrix();
280.            //alters order value for specific orders
281.            //allows the program to perform more intricate patterns
282.            //with less code to run/check through in the loop
283.            if (order == 8) {
284.              order = 9;
285.            } else if (order == 9) {
286.              order = 8;
287.            }
288.          }
289.
290.          if (order == 4) {
291.            order = 5;
292.          } else if (order == 5) {
293.            order = 4;
294.          }
295.        }
296.      }
```