

kdash12 / UF-CAP3027 Public[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)

master ▾

...

UF-CAP3027 / Li_Johnny_Project2 / Li_Johnny_Project2.pde



kdash12 Add files via upload

[History](#)

1 contributor

525 lines (490 sloc) | 12.4 KB

...

```
1  /*
2   Johnny Li
3   CAP3027
4   Project 2: Random Walk Variant
5   */
6
7   //Using the ControlP5 library.
8   import controlP5.*;
9   ControlP5 cp5;
10
11  //Global Variable
12  //UI Components
13  Button start1;
14  DropDownList ddl2;
15  Slider slider3;
16  Slider slider4;
17  Slider slider5;
18  Slider slider6;
19  CheckBox box78910;
20  Textfield text11;
21
22  //Set initial position
23  int x = 400;
24  int y = 350;
25  float hx;
26  float hy;
27
28  //Temp variable
29  int temp;
```

```
30 boolean starting=false;
31 int count=0;
32 //HashMap
33 HashMap<PVector, Integer> map = new HashMap();
34
35 void setup() {
36   cp5 = new ControlP5(this);
37   size(800, 700);
38   //Dark Grey
39   background(0, 0, 170, 0);
40   noStroke();
41   //Light Grey
42   fill(100, 100, 100);
43   rect(0, 0, 200, 700);
44
45   //Start Button
46   start1 = cp5.addButton("Start")
47     .setPosition(20, 20)
48     .setColorBackground(0xff009600)
49     .setSize(90, 30);
50
51   //Square/Hexagon List
52   ddl2 = cp5.addDropDownList("SQUARES")
53     .setPosition(20, 60)
54     .setItemHeight(40)
55     .setBarHeight(35)
56     //Different Shapes
57     .addItem("SQUARES", 0)
58     .addItem("Hexagons", 1)
59     .setSize(150, 300)
60     //Close at first
61     .setOpen(false);
62
63   //Max Slider
64   slider3 = cp5.addSlider("Max")
65     .setPosition(20, 230)
66     .setRange(100, 50000)
67     .setCaptionLabel(" ")
68     .setSize(130, 25);
69   //Move Label up //////////////////////////////////
70   slider3.getCaptionLabel().setText("Maximum Steps");
71
72   //Step Rate
73   slider4 = cp5.addSlider("Rate")
74     .setPosition(20, 280)
75     .setRange(1, 1000)
76     .setCaptionLabel(" ")
77     .setSize(130, 25);
78   //Move Label up //////////////////////////////////
```

```
79 slider4.getCaptionLabel().setText("Step Rate");
80
81 //Step Size
82 slider5 = cp5.addSlider("Size")
83     .setPosition(20, 360)
84     .setRange(10, 30)
85     .setCaptionLabel(" ")
86     .setSize(110, 25);
87 //Move Label up //////////////////////////////////
88 slider5.getCaptionLabel().setText("Step Size");
89
90 //Step Size
91 slider6 = cp5.addSlider("Scale")
92     .setPosition(20, 415)
93     .setRange(1.0, 1.5)
94     .setCaptionLabel(" ")
95     .setSize(110, 25);
96 //Move Label up //////////////////////////////////
97 slider6.getCaptionLabel().setText("Step Scale");
98
99 //Checkbox
100 box78910=cp5.addCheckBox("box")
101     .setPosition(20, 455)
102     .addItem("CONSTRAIN STEPS", 0)
103     .addItem("SIMULATE TERRAIN", 1)
104     .addItem("USE STROKE", 2)
105     .addItem("USE RANDOM SEED", 3)
106     .setSize(30, 30);
107
108 //Seed input
109 text11 = cp5.addTextfield("SEED VALUE")
110     .setPosition(135, 550)
111     .setInputFilter(ControlP5.INTEGER)
112     .setSize(55, 30);
113 }
114
115 RandomWalkBaseClass someObject = null;
116
117 public class RandomWalkBaseClass {
118     //Gloabl Variables for the class
119     int shapeType;
120     int max;
121     int rate;
122     int size;
123     double scale;
124     int seedValue;
125     boolean stroke;
126     boolean seed;
127     boolean constrain;
```

```
128     boolean terrainColor;
129
130     //Load values from UI
131     public RandomWalkBaseClass() {
132         shapeType = (int)ddl2.getValue();
133         max = (int)slider3.getValue();
134         size = (int)slider5.getValue();
135         rate = (int)slider4.getValue();
136         scale = (double)slider6.getValue();
137         constrain = box78910.getState(0);
138         terrainColor = box78910.getState(1);
139         stroke = box78910.getState(2);
140         seed = box78910.getState(3);
141         //Get textbox value
142         String value = text11.getText();
143         //Check if null/empty
144         if (value != null) {
145             if (!value.equals("")) {
146                 //Get integer value only
147                 seedValue = (int)Integer.parseInt(value);
148             }
149         } else {
150             //Null/empty case
151             text11.setText("0");
152             seedValue = 0;
153         }
154     }
155 }
156
157 //START button function
158 public void Start() {
159     //Reset program
160     clear();
161     temp=0;
162     x = 400;
163     y = 350;
164     hx = 400;
165     hy = 350;
166     count=0;
167     map.clear();
168
169     //Set start
170     starting=true;
171
172     //Dark Grey
173     background(0, 0, 170, 0);
174     noStroke();
175     //Light Grey
176     fill(100, 100, 100);
```

```

177     rect(0, 0, 200, 700);
178
179     //Run seed
180     if (box78910.getState(3) && text11.getText() != "") {
181         randomSeed(Integer.parseInt(text11.getText()));
182     }
183 }
184
185 public void draw() {
186     //Check shape selection
187     if (starting) {
188         shaping();
189     }
190 }
191
192 //Shaping
193 public void shaping() {
194     if (shapeSele() == 1) {
195         someObject = new SquareClass();
196     } else {
197         someObject = new HexagonClass();
198     }
199 }
200
201 //Selection of shape
202 public int shapeSele() {
203     double shapeType= ddl2.getValue();
204     if ((int)shapeType == 1) {
205         return 2;    // return 2 for hexagons,
206     }
207     return 1;    // return 1 for squares,
208 }
209 ///////////////////////////////////////////////////
210 //Build square
211 public class SquareClass extends RandomWalkBaseClass {
212     //Gloabl Variables for the class
213     int step = (int)(size*scale);
214     int boundx = 0; //x-axis
215
216     //Constructor
217     public SquareClass() {
218         if (constrain) {
219             boundx = 200;
220         }
221         if (!terrainColor) {
222             fill(255, 0, 255);
223         }
224         Draw();
225     }

```

```
226 //Build square
227 public void Draw() {
228     //Go through all the iterations
229     if (temp<max) {
230         //Step rate per frame
231         for (int i=0; i<rate; i++) {
232             //Check if color checkbox is slected.
233             if (stroke) {
234                 stroke(2);
235             }
236             //Switch Case of Direction
237             switch(Update()) {
238                 case 0:
239                     //Move Up
240                     y=y+step;
241                     //Clump boundary of top of Y
242                     if (y>700) {
243                         y=y-step;
244                     }
245                     //Plot point
246                     else {
247                         col(temp, x, y);
248                         square(x, y, size);
249                         break;
250                     }
251                 case 1:
252                     //Move Down
253                     y=y-step;
254                     //Clump boundary of bottom of Y
255                     if (y<0) {
256                         y=y+step;
257                     }
258                     //Plot point
259                     else {
260                         col(temp, x, y);
261                         square(x, y, size);
262                         break;
263                     }
264                 case 2:
265                     //Move Left
266                     x=x-step;
267                     //Clump boundary of bottom of X
268                     if (x<boundx) {
269                         x=x+step;
270                     }
271                     //Plot point
272                     else {
273                         col(temp, x, y);
274                         square(x, y, size);
```

```
275         break;
276     }
277     case 3:
278         //Move Right
279         x=x+step;
280         //Clump boundary of top of X
281         if (x>800) {
282             x=x-step;
283         }
284         //Plot point
285         else {
286             col(temp, x, y);
287             square(x, y, size);
288             break;
289         }
290     }
291     //increment
292     temp++;
293     //Done
294     if (temp==max) {
295         starting=false;
296     }
297 }
298 }
299 }
300
301 public void col(int temp, int x, int y) {
302     //Storing color count
303     if (terrainColor) {
304         //Vector for coloring
305         PVector vector = new PVector(Math.round(x*100)/100.00, Math.round(y*100)/100.00);
306         //Check if empty
307         if (map.get(vector) == null) {
308             map.put(vector, 1);
309         } else {
310             count = map.get(vector);
311             //Store value
312             map.put(vector, ++count);
313         }
314
315         //Coloring
316         if (count < 4) { //dirt
317             fill(160, 126, 84);
318         } else if (4<count && count< 7) { //grass
319             fill(143, 170, 64);
320         } else if (7<count && count< 10) { //rock
321             fill(135, 135, 135);
322         } else { //snow
323             fill(count*20, count*20, count*20);
```

```

324     }
325 }
326 }
327 }
328
329
330 //Square random generator
331 public int Update() {
332     //Generate a random number
333     int walk =(int)random(4);
334     //0=Up 1=Down 2=Left 3=Right
335     return walk;
336 }
337 ///////////////////////////////////////////////////
338 //Build Hexagon
339 public class HexagonClass extends RandomWalkBaseClass {
340     //Gloabl Variables for the class
341     float step = (float)(size*scale*sqrt(3));
342     float stepxn = (float)(size*scale*sqrt(3)*cos(radians(-30)));
343     float stepyn = (float)(size*scale*sqrt(3)*sin(radians(-30)));
344     float stepxp = (float)(size*scale*sqrt(3)*cos(radians(30)));
345     float stepyp = (float)(size*scale*sqrt(3)*sin(radians(30)));
346     float stepxa = (float)(size*scale*sqrt(3)*cos(radians(150)));
347     float stepya = (float)(size*scale*sqrt(3)*sin(radians(150)));
348     float stepxan = (float)(size*scale*sqrt(3)*cos(radians(-150)));
349     float stepyan = (float)(size*scale*sqrt(3)*sin(radians(-150)));
350     int boundx = 0; //x-axis
351     int count=0;
352
353     //Constructor
354     public HexagonClass() {
355         if (constrain) {
356             boundx = 200;
357         }
358         if (!terrainColor) {
359             fill(255, 0, 255);
360         }
361         Draw();
362     }
363
364     //Build Hexagon
365     public void Draw() {
366         //Go through all the iterations
367         if (temp<max) {
368             //Step rate per frame
369             for (int i=0; i<rate; i++) {
370                 //Check if color checkbox is slected.
371                 if (stroke) {
372                     stroke(2);

```



```
373     }
374     //Switch Case of Direction
375     switch(Update()) {
376     case 0:
377         //SE
378         hy=hy+stepyn;
379         hx=hx+stepxn;
380         if (hy>800 || hy<0 || hx>800 || hx<boundx) {
381             hy=hy-stepyn;
382             hx=hx-stepxn;
383             continue;
384         }
385         //Plot point
386         else {
387             col(temp, (int)hx, (int)hy);
388             hexagons(hx, hy, size);
389             break;
390         }
391     case 1:
392         //Move Down
393         hy=hy+step;
394         //Clump boundary of bottom of Y
395         if (hy>800 || hy<0 || hx>800 || hx<boundx) {
396             hy=hy-step;
397             continue;
398         }
399         //Plot point
400         else {
401             col(temp, (int)hx, (int)hy);
402             hexagons(hx, hy, size);
403             break;
404         }
405     case 2:
406         //SW
407         hx=hx+stepxan;
408         hy=hy+stepyan;
409         //Clump boundary of bottom of X
410         if (hy>800 || hy<0 || hx>800 || hx<boundx) {
411             hx=hx-stepxan;
412             hy=hy-stepyan;
413             continue;
414         }
415         //Plot point
416         else {
417             col(temp, (int)hx, (int)hy);
418             hexagons(hx, hy, size);
419             break;
420         }
421     case 3:
```

```
422     //NW
423     hx=hx+stepxa;
424     hy=hy+stepya;
425     //Clump boundary of top of X
426     if (hy>800 || hy<0 || hx>800 || hx<boundx) {
427         hx=hx-stepya;
428         hy=hy-stepya;
429         continue;
430     }
431     //Plot point
432     else {
433         col(temp, (int)hx, (int)hy);
434         hexagons(hx, hy, size);
435         break;
436     }
437 case 4:
438     //N
439     hy=hy-step;
440     //Clump boundary of top of Y
441     if (hy>800 || hy<0 || hx>800 || hx<boundx) {
442         hy=hy+step;
443         continue;
444     }
445     //Plot point
446     else {
447         col(temp, (int)hx, (int)hy);
448         hexagons(hx, hy, size);
449         break;
450     }
451 case 5:
452     //NE
453     hy=hy+stepyp;
454     hx=hx+stepxp;
455     //Clump boundary of bottom of Y
456     if (hy>800 || hy<0 || hx>800 || hx<boundx) {
457         hy=hy-stepyp;
458         hx=hx-stepxp;
459         continue;
460     }
461     //Plot point
462     else {
463         col(temp, (int)hx, (int)hy);
464         hexagons(hx, hy, size);
465         break;
466     }
467 }
468 //Increment
469 temp++;
470
```

```
471     //Done
472     if (temp==max) {
473         starting=false;
474     }
475 }
476 }
477 }
478
479 //Hexagon shape
480 public void hexagons(float xx, float yy, int size) {
481     beginShape();
482     //generate sides
483     for (float angle = 0; angle < 360; angle += 60) {
484         float hhx = (xx + cos(radians(angle)) * size);
485         float hhy = (yy + sin(radians(angle)) * size);
486         vertex(hhx, hhy);
487     }
488     endShape(CLOSE);
489 };
490
491 //Square random generator
492 public int Update() {
493     //Generate a random number
494     int walk =(int)random(6);
495     //0=SE 1=S 2=SW 3=NW 4=N 5=NE
496     return walk;
497 }
498
499 public void col(int temp, int x, int y) {
500     //Storing color count
501     if (terrainColor) {
502         //Vector for coloring
503         PVector vector = new PVector(Math.round(x*100)/100.00, Math.round(y*100)/100.00);
504         //Check if empty
505         if (map.get(vector) == null) {
506             map.put(vector, 1);
507         } else {
508             count = map.get(vector);
509             //Store value
510             map.put(vector, ++count);
511         }
512
513         //Coloring
514         if (count < 4) { //dirt
515             fill(160, 126, 84);
516         } else if (4<count && count< 7) { //grass
517             fill(143, 170, 64);
518         } else if (7<count && count< 10) { //rock
519             fill(135, 135, 135);
```

```
520     } else { //snow
521         fill(count*20, count*20, count*20);
522     }
523 }
524 }
525 }
```