kadash12 / UF-CAP3027 Public

Code Issues Pull requests Actions Projects Security Insights

master

UF-CAP3027 / Li_Johnny_Project2 / Li_Johnny_Project2.pde

kadash12 Add files via upload History

1 contributor

525 lines (490 sloc) 12.4 KB

```
/*
 Johnny Li
 CAP3027
 Project 2: Random Walk Variant
 */

//Using the ControlP5 library.
import controlP5.*;
ControlP5 cp5;

//Global Variable
//UI Components
Button start1;
DropdownList ddl2;
Slider slider3;
Slider slider4;
Slider slider5;
Slider slider6;
CheckBox box78910;
Textfield text11;

//Set inital position
int x = 400;
int y = 350;
float hx;
float hy;

//Temp variable
int temp;
```

```
boolean starting=false;
int count=0;
//Hashmap
HashMap<PVector, Integer> map = new HashMap();

void setup() {
  cp5 = new ControlP5(this);
  size(800, 700);
  //Dark Grey
  background(0, 0, 170, 0);
  noStroke();
  //Light Grey
  fill(100, 100, 100);
  rect(0, 0, 200, 700);

  //Start Button
  start1 = cp5.addButton("Start")
    .setPosition(20, 20)
    .setColorBackground(0xff009600)
    .setSize(90, 30);

  //Square/Hexagon List
  ddl2 = cp5.addDropdownList("SQUARES")
    .setPosition(20, 60)
    .setItemHeight(40)
    .setBarHeight(35)
    //Different Shapes
    .addItem("SQUARES", 0)
    .addItem("Hexagons", 1)
    .setSize(150, 300)
    //Close at first
    .setOpen(false);

  //Max Slider
  slider3 = cp5.addSlider("Max")
    .setPosition(20, 230)
    .setRange(100, 50000)
    .setCaptionLabel(" ")
    .setSize(130, 25);
  //Move Label up ////////////////////////
  slider3.getCaptionLabel().setText("Maximum Steps");

  //Step Rate
  slider4 = cp5.addSlider("Rate")
    .setPosition(20, 280)
    .setRange(1, 1000)
    .setCaptionLabel(" ")
    .setSize(130, 25);
  //Move Label up ////////////////////////
  slider4.getCaptionLabel().setText("Step Rate");

  //Step Size
  slider5 = cp5.addSlider("Size")
    .setPosition(20, 360)
    .setRange(10, 30)
    .setCaptionLabel(" ")
    .setSize(110, 25);
  //Move Label up ////////////////////////
  slider5.getCaptionLabel().setText("Step Size");

  //Step Size
  slider6 = cp5.addSlider("Scale")
    .setPosition(20, 415)
    .setRange(1.0, 1.5)
    .setCaptionLabel(" ")
    .setSize(110, 25);
  //Move Label up ////////////////////////
  slider6.getCaptionLabel().setText("Step Scale");

  //Checkbox
  box78910=cp5.addCheckBox("box")
    .setPosition(20, 455)
    .addItem("CONSTRAIN STEPS", 0)
    .addItem("SIMULATE TERRAIN", 1)
    .addItem("USE STROKE", 2)
    .addItem("USE RANDOM SEED", 3)
    .setSize(30, 30);

  //Seed input
  text11 = cp5.addTextfield("SEED VALUE")
    .setPosition(135, 550)
    .setInputFilter(ControlP5.INTEGER)
    .setSize(55, 30);
}

RandomWalkBaseClass someObject = null;

public class RandomWalkBaseClass {
  //Gloabl Variables for the class
  int shapeType;
  int max;
  int rate;
  int size;
  double scale;
  int seedValue;
  boolean stroke;
  boolean seed;
  boolean constrain;
  boolean terrainColor;

  //Load values from UI
  public RandomWalkBaseClass() {
    shapeType = (int)ddl2.getValue();
    max = (int)slider3.getValue();
    size = (int)slider5.getValue();
    rate = (int)slider4.getValue();
    scale = (double)slider6.getValue();
    constrain = box78910.getState(0);
    terrainColor = box78910.getState(1);
    stroke = box78910.getState(2);
    seed = box78910.getState(3);
    //Get textbox value
    String value = text11.getText();
    //Check if null/empty
    if (value != null) {
      if (!value.equals("")) {
        //Get integer value only
        seedValue = (int)Integer.parseInt(value);
      }
    } else {
      //Null/empty case
      text11.setText("0");
      seedValue = 0;
    }
  }
}

//START button function
public void Start() {
  //Reset program
  clear();
  temp=0;
  x = 400;
  y = 350;
  hx = 400;
  hy = 350;
  count=0;
  map.clear();

  //Set start
  starting=true;

  //Dark Grey
  background(0, 0, 170, 0);
  noStroke();
  //Light Grey
  fill(100, 100, 100);
  rect(0, 0, 200, 700);

  //Run seed
  if (box78910.getState(3) && text11.getText()!= "") {
    randomSeed(Integer.parseInt(text11.getText()));
  }
}

public void draw() {
  //Check shape selection
  if (starting) {
    shaping();
  }
}

//Shaping
public void shaping() {
  if (shapeSele() == 1) {
    someObject = new SquareClass();
  } else {
    someObject = new HexagonClass();
  }
}

//Selection of shape
public int shapeSele() {
  double shapeType= ddl2.getValue();
  if ((int)shapeType == 1) {
    return 2;   // return 2 for hexagons,
  }
  return 1;   // return 1 for squares,
}
/////////////////////////////////////////////////////////////////////////////////////////
//Build square
public class SquareClass extends RandomWalkBaseClass {
  //Gloabl Variables for the class
  int step = (int)(size*scale);
  int boundx = 0; //x-axis

  //Constructor
  public SquareClass() {
    if (constrain) {
      boundx = 200;
    }
    if (!terrainColor) {
      fill(255, 0, 255);
    }
    Draw();
  }
  //Build square
  public void Draw() {
    //Go through all the iterations
    if (temp<max) {
      //Step rate per frame
      for (int i=0; i<rate; i++) {
        //Check if color checkbox is slected.
        if (stroke) {
          stroke(2);
        }
        //Switch Case of Direction
        switch(Update()) {
        case 0:
          //Move Up
          y=y+step;
          //Clump boundary of top of Y
          if (y>700) {
            y=y-step;
          }
          //Plot point
          else {
            col(temp, x, y);
            square(x, y, size);
            break;
          }
        case 1:
          //Move Down
          y=y-step;
          //Clump boundary of bottom of Y
          if (y<0) {
            y=y+step;
          }
          //Plot point
          else {
            col(temp, x, y);
            square(x, y, size);
            break;
          }
        case 2:
          //Move Left
          x=x-step;
          //Clump boundary of bottom of X
          if (x<boundx) {
            x=x+step;
          }
          //Plot point
          else {
            col(temp, x, y);
            square(x, y, size);
            break;
          }
        case 3:
          //Move Right
          x=x+step;
          //Clump boundary of top of X
          if (x>800) {
            x=x-step;
          }
          //Plot point
          else {
            col(temp, x, y);
            square(x, y, size);
            break;
          }
        }
        //increment
        temp++;
        //Done
        if (temp==max) {
          starting=false;
        }
      }
    }
  }

  public void col(int temp, int x, int y) {
    //Storing color count
    if (terrainColor) {
      //Vector for coloring
      PVector vector = new PVector(Math.round(x*100)/100.00, Math.round(y*100)/100.00);
      //Check if empty
      if (map.get(vector) == null) {
        map.put(vector, 1);
      } else {
        count = map.get(vector);
        //Store value
        map.put(vector, ++count);
      }

      //Coloring
      if (count < 4) {  //dirt
        fill(160, 126, 84);
      } else if (4<count && count< 7) {    //grass
        fill(143, 170, 64);
      } else if (7<count && count< 10) {    //rock
        fill(135, 135, 135);
      } else {  //snow
        fill(count*20, count*20, count*20);
      }
    }
  }
}


//Square random generator
public int Update() {
  //Generate a random number
  int walk =(int)random(4);
  //0=Up  1=Down  2=Left  3=Right
  return walk;
}
/////////////////////////////////////////////////////////////////////////////////////////
//Build Hexagon
public class HexagonClass extends RandomWalkBaseClass {
  //Gloabl Variables for the class
  float step = (float)(size*scale*sqrt(3));
  float stepxn = (float)(size*scale*sqrt(3)*cos(radians(-30)));
  float stepyn = (float)(size*scale*sqrt(3)*sin(radians(-30)));
  float stepxp = (float)(size*scale*sqrt(3)*cos(radians(30)));
  float stepyp = (float)(size*scale*sqrt(3)*sin(radians(30)));
  float stepxa = (float)(size*scale*sqrt(3)*cos(radians(150)));
  float stepya = (float)(size*scale*sqrt(3)*sin(radians(150)));
  float stepxan = (float)(size*scale*sqrt(3)*cos(radians(-150)));
  float stepyan = (float)(size*scale*sqrt(3)*sin(radians(-150)));
  int boundx = 0; //x-axis
  int count=0;

  //Constructor
  public HexagonClass() {
    if (constrain) {
      boundx = 200;
    }
    if (!terrainColor) {
      fill(255, 0, 255);
    }
    Draw();
  }

  //Build Hexagon
  public void Draw() {
    //Go through all the iterations
    if (temp<max) {
      //Step rate per frame
      for (int i=0; i<rate; i++) {
        //Check if color checkbox is slected.
        if (stroke) {
          stroke(2);
        }
        //Switch Case of Direction
        switch(Update()) {
        case 0:
          //SE
          hy=hy+stepyn;
          hx=hx+stepxn;
          if (hy>800 || hy<0 || hx>800 || hx<boundx) {
            hy=hy-stepyn;
            hx=hx-stepxn;
            continue;
          }
          //Plot point
          else {
            col(temp, (int)hx, (int)hy);
            hexagons(hx, hy, size);
            break;
          }
        case 1:
          //Move Down
          hy=hy+step;
          //Clump boundary of bottom of Y
          if (hy>800 || hy<0|| hx>800 || hx<boundx) {
            hy=hy-step;
            continue;
          }
          //Plot point
          else {
            col(temp, (int)hx, (int)hy);
            hexagons(hx, hy, size);
            break;
          }
        case 2:
          //SW
          hx=hx+stepxan;
          hy=hy+stepyan;
          //Clump boundary of bottom of X
          if (hy>800 || hy<0|| hx>800 || hx<boundx) {
            hx=hx-stepxan;
            hy=hy-stepyan;
            continue;
          }
          //Plot point
          else {
            col(temp, (int)hx, (int)hy);
            hexagons(hx, hy, size);
            break;
          }
        case 3:
          //NW
          hx=hx+stepxa;
          hy=hy+stepya;
          //Clump boundary of top of X
          if (hy>800 || hy<0|| hx>800 || hx<boundx) {
            hx=hx-stepya;
            hy=hy-stepya;
            continue;
          }
          //Plot point
          else {
            col(temp, (int)hx, (int)hy);
            hexagons(hx, hy, size);
            break;
          }
        case 4:
          //N
          hy=hy-step;
          //Clump boundary of top of Y
          if (hy>800 || hy<0|| hx>800 || hx<boundx) {
            hy=hy+step;
            continue;
          }
          //Plot point
          else {
            col(temp, (int)hx, (int)hy);
            hexagons(hx, hy, size);
            break;
          }
        case 5:
          //NE
          hy=hy+stepyp;
          hx=hx+stepxp;
          //Clump boundary of bottom of Y
          if (hy>800 || hy<0|| hx>800 || hx<boundx) {
            hy=hy-stepyp;
            hx=hx-stepxp;
            continue;
          }
          //Plot point
          else {
            col(temp, (int)hx, (int)hy);
            hexagons(hx, hy, size);
            break;
          }
        }
        //Increment
        temp++;

        //Done
        if (temp==max) {
          starting=false;
        }
      }
    }
  }

  //Hexagon shape
  public void hexagons(float xx, float yy, int size) {
    beginShape();
    //generate sides
    for (float angle = 0; angle < 360; angle += 60) {
      float hhx = (xx + cos(radians(angle)) * size);
      float hhy = (yy + sin(radians(angle)) * size);
      vertex(hhx, hhy);
    }
    endShape(CLOSE);
  };

  //Square random generator
  public int Update() {
    //Generate a random number
    int walk =(int)random(6);
    //0=SE  1=S  2=SW  3=NW  4=N  5=NE
    return walk;
  }

  public void col(int temp, int x, int y) {
    //Storing color count
    if (terrainColor) {
      //Vector for coloring
      PVector vector = new PVector(Math.round(x*100)/100.00, Math.round(y*100)/100.00);
      //Check if empty
      if (map.get(vector) == null) {
        map.put(vector, 1);
      } else {
        count = map.get(vector);
        //Store value
        map.put(vector, ++count);
      }

      //Coloring
      if (count < 4) {  //dirt
        fill(160, 126, 84);
      } else if (4<count && count< 7) {    //grass
        fill(143, 170, 64);
      } else if (7<count && count< 10) {    //rock
        fill(135, 135, 135);
      } else {  //snow
        fill(count*20, count*20, count*20);
      }
    }
  }
}
```