greerviau / SnakeAI

Code Issues 6 Pull requests Actions Projects Wiki ···

master

SnakeAI / SnakeAI / SnakeAI.pde

greerviau [Redesigned neuralnet architecture to be scalable, now can designate n...](https://github.com/greerviau/SnakeAI) ··· History

1 contributor

Raw Blame

250 lines (227 sloc) 6.73 KB

```
final int SIZE = 20;
final int hidden_nodes = 16;
final int hidden_layers = 2;
final int fps = 100;  //15 is ideal for self play, increasing for AI does not directly increase sp

int highscore = 0;

float mutationRate = 0.05;
float defaultmutation = mutationRate;

boolean humanPlaying = false;  //false for AI, true to play yourself
boolean replayBest = true;  //shows only the best of each generation
boolean seeVision = false;  //see the snakes vision
boolean modelLoaded = false;

PFont font;

ArrayList<Integer> evolution;

Button graphButton;
Button loadButton;
Button saveButton;
Button increaseMut;
Button decreaseMut;

EvolutionGraph graph;

Snake snake;
Snake model;

Population pop;

public void settings() {
  size(1200,800);
}

void setup() {
  font = createFont("agencyfb-bold.ttf",32);
  evolution = new ArrayList<Integer>();
  graphButton = new Button(349,15,100,30,"Graph");
  loadButton = new Button(249,15,100,30,"Load");
  saveButton = new Button(149,15,100,30,"Save");
  increaseMut = new Button(340,85,20,20,"+");
  decreaseMut = new Button(365,85,20,20,"-");
  frameRate(fps);
  if(humanPlaying) {
    snake = new Snake();
  } else {
    pop = new Population(2000); //adjust size of population
  }
}

void draw() {
  background(0);
  noFill();
  stroke(255);

  line(400,0,400,height);
  rectMode(CORNER);
  rect(400 + SIZE,SIZE,width-400-40,height-40);
  textFont(font);
```

```
    if(humanPlaying) {
      snake.move();
      snake.show();
      fill(150);
      textSize(20);
      text("SCORE : "+snake.score,500,50);
      if(snake.dead) {
         snake = new Snake();
      }
    } else {
      if(!modelLoaded) {
        if(pop.done()) {
            highscore = pop.bestSnake.score;
            pop.calculateFitness();
            pop.naturalSelection();
        } else {
            pop.update();
            pop.show();
        }
        fill(150);
        textSize(25);
        textAlign(LEFT);
        text("GEN : "+pop.gen,120,60);
        //text("BEST FITNESS : "+pop.bestFitness,120,50);
        //text("MOVES LEFT : "+pop.bestSnake.lifeLeft,120,70);
        text("MUTATION RATE : "+mutationRate*100+"%",120,90);
        text("SCORE : "+pop.bestSnake.score,120,height-45);
        text("HIGHSCORE : "+highscore,120,height-15);
        increaseMut.show();
        decreaseMut.show();
      } else {
        model.look();
        model.think();
        model.move();
        model.show();
        model.brain.show(0,0,360,790,model.vision, model.decision);
        if(model.dead) {
          Snake newmodel = new Snake();
          newmodel.brain = model.brain.clone();
          model = newmodel;

       }
       textSize(25);
       fill(150);

       textAlign(LEFT);
       text("SCORE : "+model.score,120,height-45);
      }
      textAlign(LEFT);
      textSize(18);
      fill(255,0,0);
      text("RED < 0",120,height-75);
      fill(0,0,255);
      text("BLUE > 0",200,height-75);
      graphButton.show();
      loadButton.show();
      saveButton.show();
    }

  }

  void fileSelectedIn(File selection) {
    if (selection == null) {
      println("Window was closed or the user hit cancel.");
    } else {
      String path = selection.getAbsolutePath();
      Table modelTable = loadTable(path,"header");
      Matrix[] weights = new Matrix[modelTable.getColumnCount()-1];
      float[][] in = new float[hidden_nodes][25];
      for(int i=0; i< hidden_nodes; i++) {
        for(int j=0; j< 25; j++) {
          in[i][j] = modelTable.getFloat(j+i*25,"L0");
        }
      }
      weights[0] = new Matrix(in);

      for(int h=1; h<weights.length-1; h++) {
         float[][] hid = new float[hidden_nodes][hidden_nodes+1];
         for(int i=0; i< hidden_nodes; i++) {
            for(int j=0; j< hidden_nodes+1; j++) {
              hid[i][j] = modelTable.getFloat(j+i*(hidden_nodes+1),"L"+h);
            }
         }
         weights[h] = new Matrix(hid);
      }

      float[][] out = new float[4][hidden_nodes+1];
      for(int i=0; i< 4; i++) {
        for(int j=0; j< hidden_nodes+1; j++) {
          out[i][j] = modelTable.getFloat(j+i*(hidden_nodes+1),"L"+(weights.length-1));
        }
      }
      weights[weights.length-1] = new Matrix(out);


      evolution = new ArrayList<Integer>();
      int g = 0;
      int genscore = modelTable.getInt(g,"Graph");
      while(genscore != 0) {
         evolution.add(genscore);
         g++;
         genscore = modelTable.getInt(g,"Graph");
      }
      modelLoaded = true;
      humanPlaying = false;
      model = new Snake(weights.length-1);
      model.brain.load(weights);
    }
  }

  void fileSelectedOut(File selection) {
    if (selection == null) {
      println("Window was closed or the user hit cancel.");
    } else {
      String path = selection.getAbsolutePath();
      Table modelTable = new Table();
      Snake modelToSave = pop.bestSnake.clone();
      Matrix[] modelWeights = modelToSave.brain.pull();
      float[][] weights = new float[modelWeights.length][];
      for(int i=0; i<weights.length; i++) {
         weights[i] = modelWeights[i].toArray();
      }
      for(int i=0; i<weights.length; i++) {
         modelTable.addColumn("L"+i);
      }
      modelTable.addColumn("Graph");
      int maxLen = weights[0].length;
      for(int i=1; i<weights.length; i++) {
         if(weights[i].length > maxLen) {
            maxLen = weights[i].length;
         }
      }
      int g = 0;
      for(int i=0; i<maxLen; i++) {
         TableRow newRow = modelTable.addRow();
         for(int j=0; j<weights.length+1; j++) {
             if(j == weights.length) {
               if(g < evolution.size()) {
                  newRow.setInt("Graph",evolution.get(g));
                  g++;
               }
             } else if(i < weights[j].length) {

                newRow.setFloat("L"+j,weights[j][i]);
             }
         }
      }
      saveTable(modelTable, path);

    }
  }

  void mousePressed() {
     if(graphButton.collide(mouseX,mouseY)) {
         graph = new EvolutionGraph();
     }
     if(loadButton.collide(mouseX,mouseY)) {
         selectInput("Load Snake Model", "fileSelectedIn");
     }
     if(saveButton.collide(mouseX,mouseY)) {
         selectOutput("Save Snake Model", "fileSelectedOut");
     }
     if(increaseMut.collide(mouseX,mouseY)) {
        mutationRate *= 2;
        defaultmutation = mutationRate;
     }
     if(decreaseMut.collide(mouseX,mouseY)) {
        mutationRate /= 2;
        defaultmutation = mutationRate;
     }
  }


  void keyPressed() {
    if(humanPlaying) {
      if(key == CODED) {
         switch(keyCode) {
            case UP:
              snake.moveUp();
              break;
            case DOWN:
              snake.moveDown();
              break;
            case LEFT:
              snake.moveLeft();
              break;
            case RIGHT:
              snake.moveRight();
              break;
         }
      }

    }
  }
```