



Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

greerviau / SnakeAI

<> Code

! Issues 6

Pull requests

Actions

Projects

Wiki

...

master ▼

...

SnakeAI / SnakeAI / Snake.pde



greerviau [Changed distance to food and body from float value \[0,1\] to just 1 if...](#) ...

History

1 contributor

Raw

Blame



329 lines (302 sloc) 8.75 KB

```
1  class Snake {
2
3      int score = 1;
4      int lifeLeft = 200; //amount of moves the snake can make before it dies
5      int lifetime = 0; //amount of time the snake has been alive
6      int xVel, yVel;
7      int foodIterate = 0; //iterator to run through the foodlist (used for replay)
8
9      float fitness = 0;
10
11     boolean dead = false;
12     boolean replay = false; //if this snake is a replay of best snake
```

```
13
14 float[] vision; //snakes vision
15 float[] decision; //snakes decision
16
17 PVector head;
18
19 ArrayList<PVector> body; //snakes body
20 ArrayList<Food> foodList; //list of food positions (used to replay the best snake)
21
22 Food food;
23 NeuralNet brain;
24
25 Snake() {
26     this(hidden_layers);
27 }
28
29 Snake(int layers) {
30     head = new PVector(800,height/2);
31     food = new Food();
32     body = new ArrayList<PVector>();
33     if(!humanPlaying) {
34         vision = new float[24];
35         decision = new float[4];
36         foodList = new ArrayList<Food>();
37         foodList.add(food.clone());
38         brain = new NeuralNet(24,hidden_nodes,4,layers);
39         body.add(new PVector(800,(height/2)+SIZE));
40         body.add(new PVector(800,(height/2)+(2*SIZE)));
41         score+=2;
42     }
43 }
44
45 Snake(ArrayList<Food> foods) { //this constructor passes in a list of food positions so that a
46     replay = true;
47     vision = new float[24];
48     decision = new float[4];
49     body = new ArrayList<PVector>();
50     foodList = new ArrayList<Food>(foods.size());
51     for(Food f: foods) { //clone all the food positions in the foodlist
52         foodList.add(f.clone());
53     }
54     food = foodList.get(foodIterate);
55     foodIterate++;
56     head = new PVector(800,height/2);
57
58     body.add(new PVector(800,(height/2)+SIZE));
59     body.add(new PVector(800,(height/2)+(2*SIZE)));
60     score+=2;
61 }
```



```
61
62 boolean bodyCollide(float x, float y) { //check if a position collides with the snakes body
63     for(int i = 0; i < body.size(); i++) {
64         if(x == body.get(i).x && y == body.get(i).y) {
65             return true;
66         }
67     }
68     return false;
69 }
70
71 boolean foodCollide(float x, float y) { //check if a position collides with the food
72     if(x == food.pos.x && y == food.pos.y) {
73         return true;
74     }
75     return false;
76 }
77
78 boolean wallCollide(float x, float y) { //check if a position collides with the wall
79     if(x >= width-(SIZE) || x < 400 + SIZE || y >= height-(SIZE) || y < SIZE) {
80         return true;
81     }
82     return false;
83 }
84
85 void show() { //show the snake
86     food.show();
87     fill(255);
88     stroke(0);
89     for(int i = 0; i < body.size(); i++) {
90         rect(body.get(i).x,body.get(i).y,SIZE,SIZE);
91     }
92     if(dead) {
93         fill(150);
94     } else {
95         fill(255);
96     }
97     rect(head.x,head.y,SIZE,SIZE);
98 }
99
100 void move() { //move the snake
101     if(!dead){
102         if(!humanPlaying && !modelLoaded) {
103             lifetime++;
104             lifeLeft--;
105         }
106         if(foodCollide(head.x,head.y)) {
107             eat();
108         }
109     }
110 }
```

```
109     shiftBody();
110     if(wallCollide(head.x,head.y)) {
111         dead = true;
112     } else if(bodyCollide(head.x,head.y)) {
113         dead = true;
114     } else if(lifeLeft <= 0 && !humanPlaying) {
115         dead = true;
116     }
117 }
118 }
119
120 void eat() { //eat food
121     int len = body.size()-1;
122     score++;
123     if(!humanPlaying && !modelLoaded) {
124         if(lifeLeft < 500) {
125             if(lifeLeft > 400) {
126                 lifeLeft = 500;
127             } else {
128                 lifeLeft+=100;
129             }
130         }
131     }
132     if(len >= 0) {
133         body.add(new PVector(body.get(len).x,body.get(len).y));
134     } else {
135         body.add(new PVector(head.x,head.y));
136     }
137     if(!replay) {
138         food = new Food();
139         while(bodyCollide(food.pos.x,food.pos.y)) {
140             food = new Food();
141         }
142         if(!humanPlaying) {
143             foodList.add(food);
144         }
145     } else { //if the snake is a replay, then we dont want to create new random foods, we want to
146         food = foodList.get(foodItterate);
147         foodItterate++;
148     }
149 }
150
151 void shiftBody() { //shift the body to follow the head
152     float tempx = head.x;
153
154     float tempy = head.y;
155     head.x += xVel;
156     head.y += yVel;
157     float temp2x;
```

```
157     float temp2y;
158     for(int i = 0; i < body.size(); i++) {
159         temp2x = body.get(i).x;
160         temp2y = body.get(i).y;
161         body.get(i).x = tempx;
162         body.get(i).y = tempy;
163         tempx = temp2x;
164         tempy = temp2y;
165     }
166 }
167
168 Snake cloneForReplay() { //clone a version of the snake that will be used for a replay
169     Snake clone = new Snake(foodList);
170     clone.brain = brain.clone();
171     return clone;
172 }
173
174 Snake clone() { //clone the snake
175     Snake clone = new Snake(hidden_layers);
176     clone.brain = brain.clone();
177     return clone;
178 }
179
180 Snake crossover(Snake parent) { //crossover the snake with another snake
181     Snake child = new Snake(hidden_layers);
182     child.brain = brain.crossover(parent.brain);
183     return child;
184 }
185
186 void mutate() { //mutate the snakes brain
187     brain.mutate(mutationRate);
188 }
189
190 void calculateFitness() { //calculate the fitness of the snake
191     if(score < 10) {
192         fitness = floor(lifetime * lifetime) * pow(2,score);
193     } else {
194         fitness = floor(lifetime * lifetime);
195         fitness *= pow(2,10);
196         fitness *= (score-9);
197     }
198 }
199
200 void look() { //look in all 8 directions and check for food, body and wall
201
202     vision = new float[24];
203     float[] temp = lookInDirection(new PVector(-SIZE,0));
204     vision[0] = temp[0];
205     vision[1] = temp[1];
```

```
205 vision[2] = temp[2];
206 temp = lookInDirection(new PVector(-SIZE,-SIZE));
207 vision[3] = temp[0];
208 vision[4] = temp[1];
209 vision[5] = temp[2];
210 temp = lookInDirection(new PVector(0,-SIZE));
211 vision[6] = temp[0];
212 vision[7] = temp[1];
213 vision[8] = temp[2];
214 temp = lookInDirection(new PVector(SIZE,-SIZE));
215 vision[9] = temp[0];
216 vision[10] = temp[1];
217 vision[11] = temp[2];
218 temp = lookInDirection(new PVector(SIZE,0));
219 vision[12] = temp[0];
220 vision[13] = temp[1];
221 vision[14] = temp[2];
222 temp = lookInDirection(new PVector(SIZE,SIZE));
223 vision[15] = temp[0];
224 vision[16] = temp[1];
225 vision[17] = temp[2];
226 temp = lookInDirection(new PVector(0,SIZE));
227 vision[18] = temp[0];
228 vision[19] = temp[1];
229 vision[20] = temp[2];
230 temp = lookInDirection(new PVector(-SIZE,SIZE));
231 vision[21] = temp[0];
232 vision[22] = temp[1];
233 vision[23] = temp[2];
234 }
235
236 float[] lookInDirection(PVector direction) { //look in a direction and check for food, body and
237     float look[] = new float[3];
238     PVector pos = new PVector(head.x, head.y);
239     float distance = 0;
240     boolean foodFound = false;
241     boolean bodyFound = false;
242     pos.add(direction);
243     distance +=1;
244     while (!wallCollide(pos.x,pos.y)) {
245         if(!foodFound && foodCollide(pos.x,pos.y)) {
246             foodFound = true;
247             look[0] = 1;
248         }
249         if(!bodyFound && bodyCollide(pos.x,pos.y)) {
250             bodyFound = true;
251             look[1] = 1;
252         }
253     }
```

```
253     if(replay && seeVision) {
254         stroke(0,255,0);
255         point(pos.x,pos.y);
256         if(foodFound) {
257             noStroke();
258             fill(255,255,51);
259             ellipseMode(CENTER);
260             ellipse(pos.x,pos.y,5,5);
261         }
262         if(bodyFound) {
263             noStroke();
264             fill(102,0,102);
265             ellipseMode(CENTER);
266             ellipse(pos.x,pos.y,5,5);
267         }
268     }
269     pos.add(direction);
270     distance +=1;
271 }
272 if(replay && seeVision) {
273     noStroke();
274     fill(0,255,0);
275     ellipseMode(CENTER);
276     ellipse(pos.x,pos.y,5,5);
277 }
278 look[2] = 1/distance;
279 return look;
280 }
281
282 void think() { //think about what direction to move
283     decision = brain.output(vision);
284     int maxIndex = 0;
285     float max = 0;
286     for(int i = 0; i < decision.length; i++) {
287         if(decision[i] > max) {
288             max = decision[i];
289             maxIndex = i;
290         }
291     }
292
293     switch(maxIndex) {
294         case 0:
295             moveUp();
296             break;
297
298         case 1:
299             moveDown();
300             break;
```

```
301         moveLeft();
302         break;
303     case 3:
304         moveRight();
305         break;
306     }
307 }
308
309 void moveUp() {
310     if(yVel!=SIZE) {
311         xVel = 0; yVel = -SIZE;
312     }
313 }
314 void moveDown() {
315     if(yVel!=-SIZE) {
316         xVel = 0; yVel = SIZE;
317     }
318 }
319 void moveLeft() {
320     if(xVel!=SIZE) {
321         xVel = -SIZE; yVel = 0;
322     }
323 }
324 void moveRight() {
325     if(xVel!=-SIZE) {
326         xVel = SIZE; yVel = 0;
327     }
328 }
329 }
```