# graphicsCrawlerDisplay.py (original)

```
# graphicsCrawlerDisplay.py
# -------------------------
# Licensing Information: Please do not distribute or publish solutions to this
# project. You are free to use and extend these projects for educational
# purposes. The Pacman AI projects were developed at UC Berkeley, primarily by
# John DeNero (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# For more info, see http://inst.eecs.berkeley.edu/~cs188/sp09/pacman.html

import Tkinter
import qlearningAgents
import time
import threading
import sys
import crawler
#import pendulum
import math
from math import pi as PI

robotType = 'crawler'

class Application:

    def sigmoid(self, x):
        return 1.0 / (1.0 + 2.0 ** (-x))

    def incrementSpeed(self, inc):
        self.tickTime *= inc
#         self.epsilon = min(1.0, self.epsilon)
#         self.epsilon = max(0.0,self.epsilon)
#         self.learner.setSpeed(self.epsilon)
        self.speed_label['text'] = 'Step Delay: %.5f' % (self.tickTime)

    def incrementEpsilon(self, inc):
        self.ep += inc
        self.epsilon = self.sigmoid(self.ep)
        self.learner.setEpsilon(self.epsilon)
        self.epsilon_label['text'] = 'Epsilon: %.3f' % (self.epsilon)

    def incrementGamma(self, inc):
        self.ga += inc
        self.gamma = self.sigmoid(self.ga)
        self.learner.setDiscount(self.gamma)
        self.gamma_label['text'] = 'Discount: %.3f' % (self.gamma)

    def incrementAlpha(self, inc):
        self.al += inc
        self.alpha = self.sigmoid(self.al)
        self.learner.setLearningRate(self.alpha)
        self.alpha_label['text'] = 'Learning Rate: %.3f' % (self.alpha)

    def __initGUI(self, win):
        ## Window ##
        self.win = win

        ## Initialize Frame ##
        win.grid()
        self.dec = -.5
        self.inc = .5
        self.tickTime = 0.1

        ## Epsilon Button + Label ##
        self.setupSpeedButtonAndLabel(win)

        self.setupEpsilonButtonAndLabel(win)
        ## Gamma Button + Label ##
        self.setUpGammaButtonAndLabel(win)

        ## Alpha Button + Label ##
        self.setupAlphaButtonAndLabel(win)

        ## Exit Button ##
        #self.exit_button = Tkinter.Button(win,text='Quit', command=self.exit)
        #self.exit_button.grid(row=0, column=9)

        ## Simulation Buttons ##
#        self.setupSimulationButtons(win)

         ## Canvas ##
        self.canvas = Tkinter.Canvas(root, height=200, width=1000)
        self.canvas.grid(row=2,columnspan=10)

    def setupAlphaButtonAndLabel(self, win):
        self.alpha_minus = Tkinter.Button(win,
        text="-",command=(lambda: self.incrementAlpha(self.dec)))
        self.alpha_minus.grid(row=1, column=3, padx=10)

        self.alpha = self.sigmoid(self.al)
        self.alpha_label = Tkinter.Label(win, text='Learning Rate: %.3f' %
(self.alpha))
        self.alpha_label.grid(row=1, column=4)

        self.alpha_plus = Tkinter.Button(win,
        text="+",command=(lambda: self.incrementAlpha(self.inc)))
        self.alpha_plus.grid(row=1, column=5, padx=10)

    def setUpGammaButtonAndLabel(self, win):
        self.gamma_minus = Tkinter.Button(win,
        text="-",command=(lambda: self.incrementGamma(self.dec)))
        self.gamma_minus.grid(row=1, column=0, padx=10)

        self.gamma = self.sigmoid(self.ga)
        self.gamma_label = Tkinter.Label(win, text='Discount: %.3f' % (self.gamma))
        self.gamma_label.grid(row=1, column=1)

        self.gamma_plus = Tkinter.Button(win,
        text="+",command=(lambda: self.incrementGamma(self.inc)))
        self.gamma_plus.grid(row=1, column=2, padx=10)

    def setupEpsilonButtonAndLabel(self, win):
        self.epsilon_minus = Tkinter.Button(win,
        text="-",command=(lambda: self.incrementEpsilon(self.dec)))
        self.epsilon_minus.grid(row=0, column=3)

        self.epsilon = self.sigmoid(self.ep)
        self.epsilon_label = Tkinter.Label(win, text='Epsilon: %.3f' %
(self.epsilon))
        self.epsilon_label.grid(row=0, column=4)

        self.epsilon_plus = Tkinter.Button(win,
        text="+",command=(lambda: self.incrementEpsilon(self.inc)))
        self.epsilon_plus.grid(row=0, column=5)

    def setupSpeedButtonAndLabel(self, win):
        self.speed_minus = Tkinter.Button(win,
        text="-",command=(lambda: self.incrementSpeed(.5)))
        self.speed_minus.grid(row=0, column=0)

        self.speed_label = Tkinter.Label(win, text='Step Delay: %.5f' %
(self.tickTime))
        self.speed_label.grid(row=0, column=1)

        self.speed_plus = Tkinter.Button(win,
        text="+",command=(lambda: self.incrementSpeed(2)))
        self.speed_plus.grid(row=0, column=2)








    def skip5kSteps(self):
        self.stepsToSkip = 5000

    def __init__(self, win):

        self.ep = 0
        self.ga = 2
        self.al = 2
        self.stepCount = 0
        ## Init Gui

        self.__initGUI(win)

        # Init environment
        if robotType == 'crawler':
            self.robot = crawler.CrawlingRobot(self.canvas)
            self.robotEnvironment = crawler.CrawlingRobotEnvironment(self.robot)
        elif robotType == 'pendulum':
            self.robot = pendulum.PendulumRobot(self.canvas)
            self.robotEnvironment = \
                pendulum.PendulumRobotEnvironment(self.robot)
        else:
            raise "Unknown RobotType"

        # Init Agent
        simulationFn = lambda agent: \
          simulation.SimulationEnvironment(self.robotEnvironment,agent)
        actionFn = lambda state: \
          self.robotEnvironment.getPossibleActions(state)
        self.learner = qlearningAgents.QLearningAgent(actionFn=actionFn)

        self.learner.setEpsilon(self.epsilon)
        self.learner.setLearningRate(self.alpha)
        self.learner.setDiscount(self.gamma)

        # Start GUI
        self.running = True
        self.stopped = False
        self.stepsToSkip = 0
        self.thread = threading.Thread(target=self.run)
        self.thread.start()


    def exit(self):
      self.running = False
      for i in range(5):
        if not self.stopped:
#          print "Waiting for thread to die..."
          time.sleep(0.1)
      self.win.destroy()
      sys.exit(0)

    def step(self):

        self.stepCount += 1

        state = self.robotEnvironment.getCurrentState()
        actions = self.robotEnvironment.getPossibleActions(state)
        if len(actions) == 0.0:
            self.robotEnvironment.reset()
            state = self.robotEnvironment.getCurrentState()
            actions = self.robotEnvironment.getPossibleActions(state)
            print 'Reset!'
        action = self.learner.getAction(state)
        if action == None:
            raise 'None action returned: Code Not Complete'
        nextState, reward = self.robotEnvironment.doAction(action)
        self.learner.observeTransition(state, action, nextState, reward)

    def animatePolicy(self):
        if robotType != 'pendulum':
            raise 'Only pendulum can animatePolicy'


        totWidth = self.canvas.winfo_reqwidth()
        totHeight = self.canvas.winfo_reqheight()

        length = 0.48 * min(totWidth, totHeight)
        x,y = totWidth-length-30, length+10



        angleMin, angleMax = self.robot.getMinAndMaxAngle()
        velMin, velMax = self.robot.getMinAndMaxAngleVelocity()

        if not 'animatePolicyBox' in dir(self):
            self.canvas.create_line(x,y,x+length,y)
            self.canvas.create_line(x+length,y,x+length,y-length)
            self.canvas.create_line(x+length,y-length,x,y-length)
            self.canvas.create_line(x,y-length,x,y)
            self.animatePolicyBox = 1
            self.canvas.create_text(x+length/2,y+10,text='angle')
            self.canvas.create_text(x-30,y-length/2,text='velocity')
            self.canvas.create_text(x-60,y-length/4,text='Blue = kickLeft')
            self.canvas.create_text(x-60,y-length/4+20,text='Red = kickRight')
            self.canvas.create_text(x-60,y-length/4+40,text='White = doNothing')



        angleDelta = (angleMax-angleMin) / 100
        velDelta = (velMax-velMin) / 100
        for i in range(100):
            angle = angleMin + i * angleDelta

            for j in range(100):
                vel = velMin + j * velDelta
                state = self.robotEnvironment.getState(angle,vel)
                max, argMax = None, None
                if not self.learner.seenState(state):
                    argMax = 'unseen'
                else:
                     for action in ('kickLeft','kickRight','doNothing'):
                         qVal = self.learner.getQValue(state, action)
                         if max == None or qVal > max:
                             max, argMax = qVal, action
                if argMax != 'unseen':
                    if argMax == 'kickLeft':
                        color = 'blue'
                    elif argMax == 'kickRight':
                        color = 'red'
                    elif argMax == 'doNothing':
                        color = 'white'
                    dx = length / 100.0
                    dy = length / 100.0
                    x0, y0 = x+i*dx, y-j*dy
                    self.canvas.create_rectangle(x0,y0,x0+dx,y0+dy,fill=color)
    def run(self):
        self.stepCount = 0
        self.learner.startEpisode()
        while True:
          minSleep = .01
          tm = max(minSleep, self.tickTime)
          time.sleep(tm)
          self.stepsToSkip = int(tm / self.tickTime) - 1

          if not self.running:
            self.stopped = True
            return
          for i in range(self.stepsToSkip):
              self.step()
          self.stepsToSkip = 0
          self.step()
#          self.robot.draw()
        self.learner.stopEpisode()

    def start(self):
        self.win.mainloop()




def run():
  global root
  root = Tkinter.Tk()
  root.title( 'Crawler GUI' )
  root.resizable( 0, 0 )

#  root.mainloop()


  app = Application(root)
  def update_gui():
    app.robot.draw(app.stepCount, app.tickTime)
    root.after(10, update_gui)
  update_gui()

  root.protocol( 'WM_DELETE_WINDOW', app.exit)
  app.start()
```