graphicsGridworldDisplay.py (original)

```
# graphicsGridworldDisplay.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 util
from graphicsUtils import *

class GraphicsGridworldDisplay:

  def __init__(self, gridworld, size=120, speed=1.0):
    self.gridworld = gridworld
    self.size = size
    self.speed = speed

  def start(self):
    setup(self.gridworld, size=self.size)

  def pause(self):
    wait_for_keys()

  def displayValues(self, agent, currentState = None, message = 'Agent Values'):
    values = util.Counter()
    policy = {}
    states = self.gridworld.getStates()
    for state in states:
      values[state] = agent.getValue(state)
      policy[state] = agent.getPolicy(state)
    drawValues(self.gridworld, values, policy, currentState, message)
    sleep(0.05 / self.speed)

  def displayNullValues(self, currentState = None, message = ''):
    values = util.Counter()
    #policy = {}
    states = self.gridworld.getStates()
    for state in states:
      values[state] = 0.0
      #policy[state] = agent.getPolicy(state)
    drawNullValues(self.gridworld, currentState,'')
    # drawValues(self.gridworld, values, policy, currentState, message)
    sleep(0.05 / self.speed)

  def displayQValues(self, agent, currentState = None, message = 'Agent Q-Values'):
    qValues = util.Counter()
    states = self.gridworld.getStates()
    for state in states:
      for action in self.gridworld.getPossibleActions(state):
        qValues[(state, action)] = agent.getQValue(state, action)
    drawQValues(self.gridworld, qValues, currentState, message)
    sleep(0.05 / self.speed)

BACKGROUND_COLOR = formatColor(0,0,0)
EDGE_COLOR = formatColor(1,1,1)
OBSTACLE_COLOR = formatColor(0.5,0.5,0.5)
TEXT_COLOR = formatColor(1,1,1)
MUTED_TEXT_COLOR = formatColor(0.7,0.7,0.7)
LOCATION_COLOR = formatColor(0,0,1)

WINDOW_SIZE = -1
GRID_SIZE = -1
GRID_HEIGHT = -1
```

```
MARGIN = -1

def setup(gridworld, title = "Gridworld Display", size = 120):
  global GRID_SIZE, MARGIN, SCREEN_WIDTH, SCREEN_HEIGHT, GRID_HEIGHT
  grid = gridworld.grid
  WINDOW_SIZE = size
  GRID_SIZE = size
  GRID_HEIGHT = grid.height
  MARGIN = GRID_SIZE * 0.75
  screen_width = (grid.width - 1) * GRID_SIZE + MARGIN * 2
  screen_height = (grid.height - 0.5) * GRID_SIZE + MARGIN * 2

  begin_graphics(screen_width,
                 screen_height,
                 BACKGROUND_COLOR, title=title)

def drawNullValues(gridworld, currentState = None, message = ''):
  grid = gridworld.grid
  blank()
  for x in range(grid.width):
    for y in range(grid.height):
      state = (x, y)
      gridType = grid[x][y]
      isExit = (str(gridType) != gridType)
      isCurrent = (currentState == state)
      if gridType == '#':
        drawSquare(x, y, 0, 0, 0, None, None, True, False, isCurrent)
      else:
        drawNullSquare(gridworld.grid, x, y, False, isExit, isCurrent)
  pos = to_screen(((grid.width - 1.0) / 2.0, - 0.8))
  text( pos, TEXT_COLOR, message, "Courier", -32, "bold", "c")


def drawValues(gridworld, values, policy, currentState = None, message = 'State
Values'):
  grid = gridworld.grid
  blank()
  valueList = [values[state] for state in gridworld.getStates()] + [0.0]
  minValue = min(valueList)
  maxValue = max(valueList)
  for x in range(grid.width):
    for y in range(grid.height):
      state = (x, y)
      gridType = grid[x][y]
      isExit = (str(gridType) != gridType)
      isCurrent = (currentState == state)
      if gridType == '#':
        drawSquare(x, y, 0, 0, 0, None, None, True, False, isCurrent)
      else:
        value = values[state]
        action = None
        if policy != None and state in policy:
          action = policy[state]
          actions = gridworld.getPossibleActions(state)
        if action not in actions and 'exit' in actions:
          action = 'exit'
        valString = '%.2f' % value
        drawSquare(x, y, value, minValue, maxValue, valString, action, False, isExit,
isCurrent)
  pos = to_screen(((grid.width - 1.0) / 2.0, - 0.8))
  text( pos, TEXT_COLOR, message, "Courier", -32, "bold", "c")

def drawQValues(gridworld, qValues, currentState = None, message = 'State-Action Q-
Values'):
  grid = gridworld.grid
  blank()
  stateCrossActions = [[(state, action) for action in
gridworld.getPossibleActions(state)] for state in gridworld.getStates()]
  qStates = reduce(lambda x,y: x+y, stateCrossActions, [])
```

```
  qValueList = [qValues[(state, action)] for state, action in qStates] + [0.0]
  minValue = min(qValueList)
  maxValue = max(qValueList)
  for x in range(grid.width):
    for y in range(grid.height):
      state = (x, y)
      gridType = grid[x][y]
      isExit = (str(gridType) != gridType)
      isCurrent = (currentState == state)
      actions = gridworld.getPossibleActions(state)
      if actions == None or len(actions) == 0:
        actions = [None]
      bestQ = max([qValues[(state, action)] for action in actions])
      bestActions = [action for action in actions if qValues[(state, action)] ==
bestQ]

      q = util.Counter()
      valStrings = {}
      for action in actions:
        v = qValues[(state, action)]
        q[action] += v
        valStrings[action] = '%.2f' % v
      if gridType == '#':
        drawSquare(x, y, 0, 0, 0, None, None, True, False, isCurrent)
      elif isExit:
        action = 'exit'
        value = q[action]
        valString = '%.2f' % value
        drawSquare(x, y, value, minValue, maxValue, valString, action, False, isExit,
isCurrent)
      else:
        drawSquareQ(x, y, q, minValue, maxValue, valStrings, actions, isCurrent)
  pos = to_screen(((grid.width - 1.0) / 2.0, - 0.8))
  text( pos, TEXT_COLOR, message, "Courier", 32, "bold", "c")

def blank():
  clear_screen()

def drawNullSquare(grid,x, y, isObstacle, isTerminal, isCurrent):

  square_color = getColor(0, -1, 1)

  if isObstacle:
    square_color = OBSTACLE_COLOR

  (screen_x, screen_y) = to_screen((x, y))
  square( (screen_x, screen_y),
                0.5* GRID_SIZE,
                color = square_color,
                filled = 1,
                width = 1)

  square( (screen_x, screen_y),
                0.5* GRID_SIZE,
                color = EDGE_COLOR,
                filled = 0,
                width = 3)

  if isTerminal and not isObstacle:
    square( (screen_x, screen_y),
                0.4* GRID_SIZE,
                color = EDGE_COLOR,
                filled = 0,
                width = 2)
    text( (screen_x, screen_y),
          TEXT_COLOR,
          str(grid[x][y]),
          "Courier", -24, "bold", "c")
```

```
  text_color = TEXT_COLOR

  if not isObstacle and isCurrent:
    circle( (screen_x, screen_y), 0.1*GRID_SIZE, LOCATION_COLOR,
fillColor=LOCATION_COLOR )

  # if not isObstacle:
  #   text( (screen_x, screen_y), text_color, valStr, "Courier", 24, "bold", "c")

def drawSquare(x, y, val, min, max, valStr, action, isObstacle, isTerminal,
isCurrent):

  square_color = getColor(val, min, max)

  if isObstacle:
    square_color = OBSTACLE_COLOR

  (screen_x, screen_y) = to_screen((x, y))
  square( (screen_x, screen_y),
                0.5* GRID_SIZE,
                color = square_color,
                filled = 1,
                width = 1)
  square( (screen_x, screen_y),
                0.5* GRID_SIZE,
                color = EDGE_COLOR,
                filled = 0,
                width = 3)
  if isTerminal and not isObstacle:
    square( (screen_x, screen_y),
                0.4* GRID_SIZE,
                color = EDGE_COLOR,
                filled = 1,
                width = 2)


  if action == 'north':
    polygon( [(screen_x, screen_y - 0.45*GRID_SIZE), (screen_x+0.05*GRID_SIZE,
screen_y-0.40*GRID_SIZE), (screen_x-0.05*GRID_SIZE, screen_y-0.40*GRID_SIZE)],
EDGE_COLOR, filled = 1, smoothed = False)
  if action == 'south':
    polygon( [(screen_x, screen_y + 0.45*GRID_SIZE), (screen_x+0.05*GRID_SIZE,
screen_y+0.40*GRID_SIZE), (screen_x-0.05*GRID_SIZE, screen_y+0.40*GRID_SIZE)],
EDGE_COLOR, filled = 1, smoothed = False)
  if action == 'west':
    polygon( [(screen_x-0.45*GRID_SIZE, screen_y), (screen_x-0.4*GRID_SIZE,
screen_y+0.05*GRID_SIZE), (screen_x-0.4*GRID_SIZE, screen_y-0.05*GRID_SIZE)],
EDGE_COLOR, filled = 1, smoothed = False)
  if action == 'east':
    polygon( [(screen_x+0.45*GRID_SIZE, screen_y), (screen_x+0.4*GRID_SIZE,
screen_y+0.05*GRID_SIZE), (screen_x+0.4*GRID_SIZE, screen_y-0.05*GRID_SIZE)],
EDGE_COLOR, filled = 1, smoothed = False)


  text_color = TEXT_COLOR

  if not isObstacle and isCurrent:
    circle( (screen_x, screen_y), 0.1*GRID_SIZE, outlineColor=LOCATION_COLOR,
fillColor=LOCATION_COLOR )

  if not isObstacle:
    text( (screen_x, screen_y), text_color, valStr, "Courier", -30, "bold", "c")


def drawSquareQ(x, y, qVals, minVal, maxVal, valStrs, bestActions, isCurrent):

  (screen_x, screen_y) = to_screen((x, y))
```

```
  center = (screen_x, screen_y)
  nw = (screen_x-0.5*GRID_SIZE, screen_y-0.5*GRID_SIZE)
  ne = (screen_x+0.5*GRID_SIZE, screen_y-0.5*GRID_SIZE)
  se = (screen_x+0.5*GRID_SIZE, screen_y+0.5*GRID_SIZE)
  sw = (screen_x-0.5*GRID_SIZE, screen_y+0.5*GRID_SIZE)
  n = (screen_x, screen_y-0.5*GRID_SIZE+5)
  s = (screen_x, screen_y+0.5*GRID_SIZE-5)
  w = (screen_x-0.5*GRID_SIZE+5, screen_y)
  e = (screen_x+0.5*GRID_SIZE-5, screen_y)

  actions = qVals.keys()
  for action in actions:

    wedge_color = getColor(qVals[action], minVal, maxVal)

    if action == 'north':
      polygon( (center, nw, ne), wedge_color, filled = 1, smoothed = False)
      #text(n, text_color, valStr, "Courier", 8, "bold", "n")
    if action == 'south':
      polygon( (center, sw, se), wedge_color, filled = 1, smoothed = False)
      #text(s, text_color, valStr, "Courier", 8, "bold", "s")
    if action == 'east':
      polygon( (center, ne, se), wedge_color, filled = 1, smoothed = False)
      #text(e, text_color, valStr, "Courier", 8, "bold", "e")
    if action == 'west':
      polygon( (center, nw, sw), wedge_color, filled = 1, smoothed = False)
      #text(w, text_color, valStr, "Courier", 8, "bold", "w")

  square( (screen_x, screen_y),
                  0.5* GRID_SIZE,
                  color = EDGE_COLOR,
                  filled = 0,
                  width = 3)
  line(ne, sw, color = EDGE_COLOR)
  line(nw, se, color = EDGE_COLOR)

  if isCurrent:
    circle( (screen_x, screen_y), 0.1*GRID_SIZE, LOCATION_COLOR,
fillColor=LOCATION_COLOR )

  for action in actions:
    text_color = TEXT_COLOR
    if qVals[action] < max(qVals.values()): text_color = MUTED_TEXT_COLOR
    valStr = ""
    if action in valStrs:
      valStr = valStrs[action]
    h = -20
    if action == 'north':
      #polygon( (center, nw, ne), wedge_color, filled = 1, smooth = 0)
      text(n, text_color, valStr, "Courier", h, "bold", "n")
    if action == 'south':
      #polygon( (center, sw, se), wedge_color, filled = 1, smooth = 0)
      text(s, text_color, valStr, "Courier", h, "bold", "s")
    if action == 'east':
      #polygon( (center, ne, se), wedge_color, filled = 1, smooth = 0)
      text(e, text_color, valStr, "Courier", h, "bold", "e")
    if action == 'west':
      #polygon( (center, nw, sw), wedge_color, filled = 1, smooth = 0)
      text(w, text_color, valStr, "Courier", h, "bold", "w")


def getColor(val, minVal, max):
  r, g = 0.0, 0.0
  if val < 0 and minVal < 0:
    r = val * 0.65 / minVal
  if val > 0 and max > 0:
    g = val * 0.65 / max
  return formatColor(r,g,0.0)
```

```
def square(pos, size, color, filled, width):
  x, y = pos
  dx, dy = size, size
  return polygon([(x - dx, y - dy), (x - dx, y + dy), (x + dx, y + dy), (x + dx, y -
dy)], outlineColor=color, fillColor=color, filled=filled, width=width,
smoothed=False)


def to_screen(point):
  ( gamex, gamey ) = point
  x = gamex*GRID_SIZE + MARGIN
  y = (GRID_HEIGHT - gamey - 1)*GRID_SIZE + MARGIN
  return ( x, y )

def to_grid(point):
  (x, y) = point
  x = int ((y - MARGIN + GRID_SIZE * 0.5) / GRID_SIZE)
  y = int ((x - MARGIN + GRID_SIZE * 0.5) / GRID_SIZE)
  print point, "-->", (x, y)
  return (x, y)
```