graphicsUtils.py (original)

```
# graphicsUtils.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 sys
import math
import random
import string
import time
import types
import Tkinter

_Windows = sys.platform == 'win32'  # True if on Win95/98/NT

_root_window = None      # The root window for graphics output
_canvas = None      # The canvas which holds graphics
_canvas_xs = None      # Size of canvas object
_canvas_ys = None
_canvas_x = None      # Current position on canvas
_canvas_y = None
_canvas_col = None      # Current colour (set to black below)
_canvas_tsize = 12
_canvas_tserifs = 0

def formatColor(r, g, b):
  return '#%02x%02x%02x' % (int(r * 255), int(g * 255), int(b * 255))

def colorToVector(color):
  return map(lambda x: int(x, 16) / 256.0, [color[1:3], color[3:5], color[5:7]])

if _Windows:
    _canvas_tfonts = ['times new roman', 'lucida console']
else:
    _canvas_tfonts = ['times', 'lucidasans-24']
    pass # XXX need defaults here

def sleep(secs):
    global _root_window
    if _root_window == None:
        time.sleep(secs)
    else:
        _root_window.update_idletasks()
        _root_window.after(int(1000 * secs), _root_window.quit)
        _root_window.mainloop()

def begin_graphics(width=640, height=480, color=formatColor(0, 0, 0), title=None):

    global _root_window, _canvas, _canvas_x, _canvas_y, _canvas_xs, _canvas_ys,
_bg_color

    # Check for duplicate call
    if _root_window is not None:
        # Lose the window.
        _root_window.destroy()

    # Save the canvas size parameters
    _canvas_xs, _canvas_ys = width - 1, height - 1
    _canvas_x, _canvas_y = 0, _canvas_ys
    _bg_color = color
    # Create the root window
    _root_window = Tkinter.Tk()
    _root_window.protocol('WM_DELETE_WINDOW', _destroy_window)
    _root_window.title(title or 'Graphics Window')
    _root_window.resizable(0, 0)

    # Create the canvas object
    try:
      _canvas = Tkinter.Canvas(_root_window, width=width, height=height)
      _canvas.pack()
      draw_background()
      _canvas.update()
    except:
      _root_window = None
      raise

    # Bind to key-down and key-up events
    _root_window.bind( "<KeyPress>", _keypress )
    _root_window.bind( "<KeyRelease>", _keyrelease )
    _root_window.bind( "<FocusIn>", _clear_keys )
    _root_window.bind( "<FocusOut>", _clear_keys )
    _root_window.bind( "<Button-1>", _leftclick )
    _root_window.bind( "<Button-2>", _rightclick )
    _root_window.bind( "<Button-3>", _rightclick )
    _root_window.bind( "<Control-Button-1>", _ctrl_leftclick)
    _clear_keys()

_leftclick_loc = None
_rightclick_loc = None
_ctrl_leftclick_loc = None

def _leftclick(event):
  global _leftclick_loc
  _leftclick_loc = (event.x, event.y)

def _rightclick(event):
  global _rightclick_loc
  _rightclick_loc = (event.x, event.y)

def _ctrl_leftclick(event):
  global _ctrl_leftclick_loc
  _ctrl_leftclick_loc = (event.x, event.y)

def wait_for_click():
  while True:
    global _leftclick_loc
    global _rightclick_loc
    global _ctrl_leftclick_loc
    if _leftclick_loc != None:
      val = _leftclick_loc
      _leftclick_loc = None
      return val, 'left'
    if _rightclick_loc != None:
      val = _rightclick_loc
      _rightclick_loc = None
      return val, 'right'
    if _ctrl_leftclick_loc != None:
      val = _ctrl_leftclick_loc
      _ctrl_leftclick_loc = None
      return val, 'ctrl_left'
    sleep(0.05)

def draw_background():
    corners = [(0,0), (0, _canvas_ys), (_canvas_xs, _canvas_ys), (_canvas_xs, 0)]
    polygon(corners, _bg_color, fillColor=_bg_color, filled=True, smoothed=False)

def _destroy_window(event=None):
    sys.exit(0)
#    global _root_window
#    _root_window.destroy()
#    _root_window = None
    #print "DESTROY"

def end_graphics():
    global _root_window, _canvas, _mouse_enabled
    try:
      try:
        sleep(1)
        if _root_window != None:
          _root_window.destroy()
      except SystemExit, e:
        print 'Ending graphics raised an exception:', e
    finally:
      _root_window = None
      _canvas = None
      _mouse_enabled = 0
      _clear_keys()

def clear_screen(background=None):
    global _canvas_x, _canvas_y
    _canvas.delete('all')
    draw_background()
    _canvas_x, _canvas_y = 0, _canvas_ys

def polygon(coords, outlineColor, fillColor=None, filled=1, smoothed=1, behind=0,
width=1):
  c = []
  for coord in coords:
    c.append(coord[0])
    c.append(coord[1])
  if fillColor == None: fillColor = outlineColor
  if filled == 0: fillColor = ""
  poly = _canvas.create_polygon(c, outline=outlineColor, fill=fillColor,
smooth=smoothed, width=width)
  if behind > 0:
    _canvas.tag_lower(poly, behind) # Higher should be more visible
  return poly

def square(pos, r, color, filled=1, behind=0):
  x, y = pos
  coords = [(x - r, y - r), (x + r, y - r), (x + r, y + r), (x - r, y + r)]
  return polygon(coords, color, color, filled, 0, behind=behind)

def circle(pos, r, outlineColor, fillColor, endpoints=None, style='pieslice',
width=2):
    x, y = pos
    x0, x1 = x - r - 1, x + r
    y0, y1 = y - r - 1, y + r
    if endpoints == None:
      e = [0, 359]
    else:
      e = list(endpoints)
    while e[0] > e[1]: e[1] = e[1] + 360

    return _canvas.create_arc(x0, y0, x1, y1, outline=outlineColor, fill=fillColor,
                              extent=e[1] - e[0], start=e[0], style=style,
width=width)

def image(pos, file="../../blueghost.gif"):
    x, y = pos
    # img = PhotoImage(file=file)
    return _canvas.create_image(x, y, image = Tkinter.PhotoImage(file=file), anchor =
Tkinter.NW)


def refresh():
      _canvas.update_idletasks()
```

```python
def moveCircle(id, pos, r, endpoints=None):
    global _canvas_x, _canvas_y

    x, y = pos
#    x0, x1 = x - r, x + r + 1
#    y0, y1 = y - r, y + r + 1
    x0, x1 = x - r - 1, x + r
    y0, y1 = y - r - 1, y + r
    if endpoints == None:
      e = [0, 359]
    else:
      e = list(endpoints)
    while e[0] > e[1]: e[1] = e[1] + 360

    edit(id, ('start', e[0]), ('extent', e[1] - e[0]))
    move_to(id, x0, y0)

def edit(id, *args):
    _canvas.itemconfigure(id, **dict(args))

def text(pos, color, contents, font='Helvetica', size=12, style='normal',
anchor="nw"):
    global _canvas_x, _canvas_y
    x, y = pos
    font = (font, str(size), style)
    return _canvas.create_text(x, y, fill=color, text=contents, font=font,
anchor=anchor)

def changeText(id, newText, font=None, size=12, style='normal'):
  _canvas.itemconfigure(id, text=newText)
  if font != None:
    _canvas.itemconfigure(id, font=(font, '-%d' % size, style))

def changeColor(id, newColor):
  _canvas.itemconfigure(id, fill=newColor)

def line(here, there, color=formatColor(0, 0, 0), width=2):
  x0, y0 = here[0], here[1]
  x1, y1 = there[0], there[1]
  return _canvas.create_line(x0, y0, x1, y1, fill=color, width=width)

##############################################################################
### Keypress handling ########################################################
##############################################################################

# We bind to key-down and key-up events.

_keysdown = {}
_keyswaiting = {}
# This holds an unprocessed key release.  We delay key releases by up to
# one call to keys_pressed() to get round a problem with auto repeat.
_got_release = None

def _keypress(event):
    global _got_release
    #remap_arrows(event)
    _keysdown[event.keysym] = 1
    _keyswaiting[event.keysym] = 1
#    print event.char, event.keycode
    _got_release = None

def _keyrelease(event):
    global _got_release
    #remap_arrows(event)
    try:
      del _keysdown[event.keysym]
    except:
      pass
    _got_release = 1
```

```
def remap_arrows(event):
    # TURN ARROW PRESSES INTO LETTERS (SHOULD BE IN KEYBOARD AGENT)
    if event.char in ['a', 's', 'd', 'w']:
      return
    if event.keycode in [37, 101]: # LEFT ARROW (win / x)
      event.char = 'a'
    if event.keycode in [38, 99]: # UP ARROW
      event.char = 'w'
    if event.keycode in [39, 102]: # RIGHT ARROW
      event.char = 'd'
    if event.keycode in [40, 104]: # DOWN ARROW
      event.char = 's'

def _clear_keys(event=None):
    global _keysdown, _got_release, _keyswaiting
    _keysdown = {}
    _keyswaiting = {}
    _got_release = None

def keys_pressed(d_o_e=Tkinter.tkinter.dooneevent,
                 d_w=Tkinter.tkinter.DONT_WAIT):
    d_o_e(d_w)
    if _got_release:
      d_o_e(d_w)
    return _keysdown.keys()

def keys_waiting():
  global _keyswaiting
  keys = _keyswaiting.keys()
  _keyswaiting = {}
  return keys

# Block for a list of keys...

def wait_for_keys():
    keys = []
    while keys == []:
        keys = keys_pressed()
        sleep(0.05)
    return keys

def remove_from_screen(x,
                       d_o_e=Tkinter.tkinter.dooneevent,
                       d_w=Tkinter.tkinter.DONT_WAIT):
    _canvas.delete(x)
    d_o_e(d_w)

def _adjust_coords(coord_list, x, y):
    for i in range(0, len(coord_list), 2):
        coord_list[i] = coord_list[i] + x
        coord_list[i + 1] = coord_list[i + 1] + y
    return coord_list

def move_to(object, x, y=None,
            d_o_e=Tkinter.tkinter.dooneevent,
            d_w=Tkinter.tkinter.DONT_WAIT):
    if y is None:
        try: x, y = x
        except: raise  'incomprehensible coordinates'

    horiz = True
    newCoords = []
    current_x, current_y = _canvas.coords(object)[0:2] # first point
    for coord in  _canvas.coords(object):
      if horiz:
        inc = x - current_x
      else:
        inc = y - current_y
        horiz = not horiz

        newCoords.append(coord + inc)

      _canvas.coords(object, *newCoords)
      d_o_e(d_w)

def move_by(object, x, y=None,
            d_o_e=Tkinter.tkinter.dooneevent,
            d_w=Tkinter.tkinter.DONT_WAIT):
    if y is None:
        try: x, y = x
        except: raise Exception, 'incomprehensible coordinates'

    horiz = True
    newCoords = []
    for coord in  _canvas.coords(object):
      if horiz:
        inc = x
      else:
        inc = y
      horiz = not horiz

      newCoords.append(coord + inc)

    _canvas.coords(object, *newCoords)
    d_o_e(d_w)

def writePostscript(filename):
  "Writes the current canvas to a postscript file."
  psfile = file(filename, 'w')
  psfile.write(_canvas.postscript(pageanchor='sw',
                   y='0.c',
                   x='0.c'))
  psfile.close()

ghost_shape = [
    (0, - 0.5),
    (0.25, - 0.75),
    (0.5, - 0.5),
    (0.75, - 0.75),
    (0.75, 0.5),
    (0.5, 0.75),
    (- 0.5, 0.75),
    (- 0.75, 0.5),
    (- 0.75, - 0.75),
    (- 0.5, - 0.5),
    (- 0.25, - 0.75)
  ]

if __name__ == '__main__':
  begin_graphics()
  clear_screen()
  ghost_shape = [(x * 10 + 20, y * 10 + 20) for x, y in ghost_shape]
  g = polygon(ghost_shape, formatColor(1, 1, 1))
  move_to(g, (50, 50))
  circle((150, 150), 20, formatColor(0.7, 0.3, 0.0), endpoints=[15, - 15])
  sleep(2)
```