# Exemplar Graphics-5

# This is an example program that contains graphics using

# the pygame module, contains user-defined classes, and

# responds to multiple kinds of events.

# It contains these kinds of statements: expression, assignment,

# import, function definition, while, for, if, return, class

# definition

# It contains these kinds of expressions: identifier, literal,

# attribute reference, function call, binary operator, expression

# list

# It uses these types: str, int, float, bool, NoneType, function,

# module, tuple, pygame.Surface, pygame.Color, pygame.Rect, Game,

# Circle

import pygame, time, random

from pygame.locals import \*

# User-defined functions

def main():

surface = create\_window()

game = Game(surface)

game.play()

pygame.quit()

def create\_window():

# Open a window on the display and return its Surface

title = 'Jump'

size = (500, 400)

pygame.init()

surface = pygame.display.set\_mode(size, 0, 0)

pygame.display.set\_caption(title)

return surface

# User-defined classes

class Circle:

# An object in this class represents a colored circle.

def \_\_init\_\_(self, center, radius, color, surface):

# Initialize a Cirlcle.

# - self is the Circle to initialize

# - center is a list containing the x and y int

# coords of the center of the Circle

# - radius is the int pixel radius of the Circle

# - color is the pygame.Color of the Circle

# - surface is the window's pygame.Surface object

self.center = center

self.radius = radius

self.color = color

self.surface = surface

def draw(self):

# Draw the Circle.

# - self is the Circle to draw

pygame.draw.circle(self.surface, self.color, self.center, self.radius)

def jump(self, position):

# Move the circle to the mouse's position.

# - self is the Circle to move

self.center=position

if self.is\_outside():

self.color=pygame.Color("yellow")

else:

self.color=pygame.Color("blue")

def is\_outside(self):

#checks to see if any part of the circle is outside the surface

if (self.center[0]<self.radius) or (self.center[0]>self.surface.get\_width()-self.radius):

return True

if (self.center[1]<self.radius) or (self.center[1]>self.surface.get\_height()-self.radius):

return True

return False

def set\_color(self, color):

self.color=color

def get\_color(self):

# Return the color of the Circle.

# - self is the Circle

return self.color

def enlarge(self, increment):

# Enlarge the radius of the Circle.

# - self is the Circle

# - increment is the int number of pixels

# to add to the radius

self.radius = self.radius + increment

class Game:

# An object in this class represents a complete game.

def \_\_init\_\_(self, surface):

# Initialize a Game.

# - self is the Game to initialize

# - surface is the window's pygame.Surface object

self.surface = surface

self.bg\_color = pygame.Color('black')

self.pause\_time = 0.01 # smaller is faster game

self.close\_clicked = False

self.continue\_game = True

pos = [random.randint(50, surface.get\_width()-50), random.randint(50, surface.get\_height()-50)]

self.circle = Circle(pos, 50, pygame.Color('blue'), surface)

self.elapsed\_time = 0

self.radius\_increment = 5

def play(self):

# Play the game until the player presses the close box.

# - self is the Game that should be continued or not.

self.draw()

while not self.close\_clicked: # until player clicks close box

# play frame

self.handle\_event()

if self.continue\_game:

self.update()

self.decide\_continue()

self.draw()

time.sleep(self.pause\_time) # set game velocity by pausing

def handle\_event(self):

# Handle each user event by changing the game state

# appropriately.

# - self is the Game whose events will be handled

event = pygame.event.poll()

if event.type == QUIT:

self.close\_clicked = True

elif event.type == MOUSEBUTTONUP and self.continue\_game:

self.handle\_mouse\_up(event.pos)

def handle\_mouse\_up(self, position):

# Respond to the player releasing the mouse button by

# taking appropriate actions.

# - self is the Game where the mouse up occurred

self.circle.jump(position)

def draw(self):

# Draw all game objects.

# - self is the Game to draw

self.surface.fill(self.bg\_color)

self.circle.draw()

if not self.continue\_game:

# Perform appropriate game over actions

self.surface.fill(self.circle.get\_color())

pygame.display.update()

def update(self):

# Update the game objects.

# - self is the Game to update

self.elapsed\_time = pygame.time.get\_ticks() // 1000

def decide\_continue(self):

# Check and remember if the game should continue

# - self is the Game to check

pass

main()