#http://www.codeskulptor.org/#user40\_7epauiPSob\_1.py

#http://www.codeskulptor.org/#user40\_uuggZG7LLZ\_10.py

# program template for Spaceship

import simplegui

import math

import random

# globals for user interfacegod

WIDTH = 800

HEIGHT = 600

score = 0

lives = 3

time = 0

started = False

#rock\_group = set([])

class ImageInfo:

def \_\_init\_\_(self, center, size, radius = 0, lifespan = None, animated = False):

self.center = center

self.size = size

self.radius = radius

if lifespan:

self.lifespan = lifespan

else:

self.lifespan = float('inf')

self.animated = animated

def set\_center(self, posx, posy):

self.center = [posx, posy]

def get\_center(self):

return self.center

def get\_size(self):

return self.size

def get\_radius(self):

return self.radius

def get\_lifespan(self):

return self.lifespan

def get\_animated(self):

return self.animated

# art assets created by Kim Lathrop, may be freely re-used in non-commercial projects, please credit Kim

# debris images - debris1\_brown.png, debris2\_brown.png, debris3\_brown.png, debris4\_brown.png

# debris1\_blue.png, debris2\_blue.png, debris3\_blue.png, debris4\_blue.png, debris\_blend.png

debris\_info = ImageInfo([320, 240], [640, 480])

debris\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/debris2\_blue.png")

# nebula images - nebula\_brown.png, nebula\_blue.png

nebula\_info = ImageInfo([400, 300], [800, 600])

nebula\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/nebula\_blue.f2014.png")

# splash image

splash\_info = ImageInfo([200, 150], [400, 300])

splash\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/splash.png")

# ship image

ship\_info = ImageInfo([45, 45], [90, 90], 35)

ship\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/double\_ship.png")

# missile image - shot1.png, shot2.png, shot3.png

missile\_info = ImageInfo([5,5], [10, 10], 3, 50)

missile\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/shot2.png")

# asteroid images - asteroid\_blue.png, asteroid\_brown.png, asteroid\_blend.png

asteroid\_info = ImageInfo([45, 45], [90, 90], 40)

asteroid\_image1 = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/asteroid\_blue.png")

asteroid\_image2 = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/asteroid\_brown.png")

asteroid\_image3 = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/asteroid\_blend.png")

# animated explosion - explosion\_orange.png, explosion\_blue.png, explosion\_blue2.png, explosion\_alpha.png

explosion\_info = ImageInfo([64, 64], [128, 128], 17, 24, True)

explosion\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/explosion\_alpha.png")

# sound assets purchased from sounddogs.com, please do not redistribute

soundtrack = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/sounddogs/soundtrack.mp3")

missile\_sound = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/sounddogs/missile.mp3")

missile\_sound.set\_volume(.5)

ship\_thrust\_sound = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/sounddogs/thrust.mp3")

explosion\_sound = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/sounddogs/explosion.mp3")

# helper functions to handle transformations

def angle\_to\_vector(ang):

return [math.cos(ang), math.sin(ang)]

def dist(p,q):

return math.sqrt((p[0] - q[0]) \*\* 2+(p[1] - q[1]) \*\* 2)

def group\_collide(group, other\_object):

''' Checks for a collision between a single object and a group'''

remove = set([])

for item in set(group):

if item.collide(other\_object):

remove.add(item)

for item in remove:

group.remove(item)

if len (remove) > 0:

return True

else:

return False

def group\_group\_collide(group1, group2):

collided = set([])

for item in set(group1):

if group\_collide(group2, item):

collided.add(item)

for item in collided:

group1.discard(item)

return collided

def restart():

global lives, score, rock\_group

lives = 3

score = 0

# #clear rocks

# remove = set(rock\_group)

# for rock in remove:

# rock\_group.remove(rock)

# Ship class

class Ship:

def \_\_init\_\_(self, pos, vel, angle, image, info):

self.pos = [pos[0],pos[1]]

self.vel = [vel[0],vel[1]]

self.thrust = False

self.angle = angle

self.angle\_vel = 0

self.image = image

self.image\_center = info.get\_center()

self.image\_size = info.get\_size()

self.radius = info.get\_radius()

self.set\_center = info.set\_center(135, 45)

def draw(self,canvas):

if self.thrust:

self.set\_center

canvas.draw\_image(self.image, (self.image\_center[0] + self.image\_size[0], self.image\_center[1]),

self.image\_size, self.pos, self.image\_size, self.angle)

else:

canvas.draw\_image(self.image, self.image\_center, self.image\_size,

self.pos, self.image\_size, self.angle)

def update(self):

FRICTION = .014

self.pos[0] += self.vel[0]

self.pos[1] += self.vel[1]

self.angle += self.angle\_vel

forward = angle\_to\_vector(self.angle)

self.vel[0] \*= (1.0 - FRICTION) # slow ship down

self.vel[1] \*= (1.0 - FRICTION)

#Accelerate while thrusting

if self.thrust:

ACCEL\_CONSTANT = .21

self.vel[0] += forward[0] \* ACCEL\_CONSTANT

self.vel[1] += forward[1] \* ACCEL\_CONSTANT

#Screen edge wrap around

if self.pos[0] < 0:

self.pos[0] += WIDTH

if self.pos[0] > WIDTH:

self.pos[0] -= WIDTH

if self.pos[1] < 0:

self.pos[1] += HEIGHT

if self.pos[1] > HEIGHT:

self.pos[1] -= HEIGHT

def shoot(self):

MISSILE\_SPEED = 7

pos = [self.pos[0] + angle\_to\_vector(self.angle)[0] \* self.radius,

self.pos[1] + angle\_to\_vector(self.angle)[1] \* self.radius]

vel = [self.vel[0] + angle\_to\_vector(self.angle)[0] \* MISSILE\_SPEED,

self.vel[1] + angle\_to\_vector(self.angle)[1] \* MISSILE\_SPEED]

missile = Sprite(pos, vel, 0, 0, missile\_image, missile\_info, missile\_sound)

missile\_group.add(missile)

def get\_position(self):

return self.pos

def get\_radius(self):

return self.radius

def turn\_left(self):

angle\_constant = .11

self.angle\_vel = -1 \* angle\_constant

def turn\_right(self):

angle\_constant = .11

self.angle\_vel = angle\_constant

def zoom(self):

self.thrust = True

ship\_thrust\_sound.play()

def coast(self):

self.thrust = False

ship\_thrust\_sound.rewind()

# Sprite class

class Sprite:

def \_\_init\_\_(self, pos, vel, ang, ang\_vel, image, info, sound = None):

self.pos = [pos[0],pos[1]]

self.vel = [vel[0],vel[1]]

self.angle = ang

self.angle\_vel = ang\_vel

self.image = image

self.image\_center = info.get\_center()

self.image\_size = info.get\_size()

self.radius = info.get\_radius()

self.lifespan = info.get\_lifespan()

self.animated = info.get\_animated()

self.age = 0

if sound:

sound.rewind()

sound.play()

def draw(self, canvas):

canvas.draw\_image(self.image, self.image\_center, self.image\_size, self.pos,

self.image\_size, self.angle)

def get\_position(self):

return self.pos

def get\_radius(self):

return self.radius

def update(self):

self.pos[0] += self.vel[0]

self.pos[1] += self.vel[1]

self.angle += self.angle\_vel

#Screen edge wrap around

if self.pos[0] < 0:

self.pos[0] += WIDTH

if self.pos[0] > WIDTH:

self.pos[0] -= WIDTH

if self.pos[1] < 0:

self.pos[1] += HEIGHT

if self.pos[1] > HEIGHT:

self.pos[1] -= HEIGHT

#update lifespan

self.age += 1

if self.age >= self.lifespan:

return True

else:

return False

def collide(self, other\_object):

distance = dist(self.get\_position(), other\_object.get\_position())

if distance < self.get\_radius() + other\_object.get\_radius():

return True

else:

return False

def draw(canvas):

global time, lives, score

# animiate background

time += 1

wtime = (time / 4) % WIDTH

center = debris\_info.get\_center()

size = debris\_info.get\_size()

canvas.draw\_image(nebula\_image, nebula\_info.get\_center(), nebula\_info.get\_size(), [WIDTH / 2, HEIGHT / 2], [WIDTH, HEIGHT])

canvas.draw\_image(debris\_image, center, size, (wtime - WIDTH / 2, HEIGHT / 2), (WIDTH, HEIGHT))

canvas.draw\_image(debris\_image, center, size, (wtime + WIDTH / 2, HEIGHT / 2), (WIDTH, HEIGHT))

# draw lives and score counters

canvas.draw\_text(("Lives: " + str(lives)), [WIDTH / 15, HEIGHT / 10], 24, "white")

canvas.draw\_text("Score: " + str(score), [WIDTH \* .83, HEIGHT / 10], 24, "white")

# draw ship and sprites

my\_ship.draw(canvas)

process\_sprite\_group(rock\_group, canvas)

process\_sprite\_group(missile\_group, canvas)

#a\_missile.draw(canvas)

# update ship and sprites

my\_ship.update()

#rock\_group.update()

#a\_missile.update()

#Check for collision with ship

if group\_collide(rock\_group, my\_ship):

lives -= 1

#Check for collision of missile with rock

if group\_group\_collide(rock\_group, missile\_group):

score += 1

#check for game over

if lives <= 0:

global started

started = False

#clear rocks

remove = set(rock\_group)

for rock in remove:

rock\_group.remove(rock)

# Draw splash screen if not started

if not started:

canvas.draw\_image(splash\_image, splash\_info.get\_center(),

splash\_info.get\_size(), [WIDTH / 2, HEIGHT / 2],

splash\_info.get\_size())

#handler for keyboard user controls:

def keydown(key):

inputs = {37: my\_ship.turn\_left, 39: my\_ship.turn\_right,

38: my\_ship.zoom, 32: my\_ship.shoot}

if key in inputs.keys():

inputs[key]()

def keyup(key):

if key == 37 or key == 39:

my\_ship.angle\_vel = 0

if key == 38:

my\_ship.coast()

def click(pos):

global started

center = [WIDTH / 2, HEIGHT / 2]

size = splash\_info.get\_size()

inwidth = (center[0] - size[0] / 2) < pos[0] < (center[0] + size[0] / 2)

inheight = (center[1] - size[1] / 2) < pos[1] < (center[1] + size[1] / 2)

if (not started) and inwidth and inheight:

started = True

soundtrack.rewind()

soundtrack.play()

restart()

# timer handler that spawns a rock

def rock\_spawner():

global rock\_group

if started:

if len(rock\_group) <= 11: #generates random rock data:

pos = [random.random() \* WIDTH, random.random() \* HEIGHT]

vel = [(random.random() -.5) \* 3, (random.random() - .5) \* 3]

ang = random.randrange(0, 7)

rot = random.choice([-.12, -.105, .09, -.07, .07, .09, .105, .12])

image = random.choice([asteroid\_image1, asteroid\_image2, asteroid\_image3])

#Check for safe spawn distance, cancel spawn otherwise

if dist(pos, my\_ship.get\_position()) > 200:

rock\_group.add (Sprite(pos, vel, ang, rot, image, asteroid\_info))

# handler for drawing sprite groups

def process\_sprite\_group(group, canvas):

remove = set([])

for sprite in group:

sprite.update()

sprite.draw(canvas)

if sprite.update(): #check to see if sprite past lifespan

remove.add(sprite)

for sprite in remove:

group.remove(sprite)

# initialize frame

frame = simplegui.create\_frame("Asteroids", WIDTH, HEIGHT)

# initialize ship and two sprites

my\_ship = Ship([WIDTH / 2, HEIGHT / 2], [0, 0], 0, ship\_image, ship\_info)

rock\_group = set([])

missile\_group = set([])

#Sprite([WIDTH / 3, HEIGHT / 3], [1, 1], 0, 0, asteroid\_image, asteroid\_info)

#a\_missile = Sprite([2 \* WIDTH / 3, 2 \* HEIGHT / 3], [-1,1], 0, 0, missile\_image, missile\_info, missile\_sound)

# register handlers

frame.set\_draw\_handler(draw)

frame.set\_keydown\_handler(keydown)

frame.set\_keyup\_handler(keyup)

frame.set\_mouseclick\_handler(click)

timer = simplegui.create\_timer(1000.0, rock\_spawner)

# get things rolling

timer.start()

frame.start()