

```

1  __author__ = 'Sam Carton and Paul Resnick'
2
3  import pygame
4  import random
5  import math
6
7  debug = True
8
9
10 def as_cartesian(velocity,angle):
11     if angle is None:
12         return 0,0
13     else:
14         return velocity*math.cos(math.radians(angle)),velocity*math.sin(math.radians(
15             angle))
16
17 def sign(num):
18     if num >= 0:
19         return 1
20     else:
21         return -1
22
23 class GameObject(pygame.sprite.Sprite):
24
25     def __init__(self, img_file = None, initial_x = 0, initial_y = 0, game = None):
26         pygame.sprite.Sprite.__init__(self, img_file, initial_x, initial_y)
27         self.game = game
28
29         self.initial_x = initial_x
30         self.initial_y = initial_y
31
32         self.set_initial_position()
33
34     def set_initial_position(self):
35         # set_position method is inherited from Sprite class
36         self.set_position(self.initial_x,self.initial_y)
37         self.velocity = 0.0
38         self.angle = None
39
40     def move(self):
41         """
42         Move this game object one unit forward in the direction of its velocity.
43         :return:
44         """
45         x_vel,y_vel = as_cartesian(self.velocity, self.angle)
46         self.set_position(self.x + int(x_vel), self.y + int(y_vel))
47
48
49     def update(self,pressed_keys):
50         self.move()
51
52
53
54 class BallDeflector(GameObject):
55
56     def deflect_ball(self,ball,side_hit):
57         """
58         Deflect a ball that has collided with this object.

```

```

59     :param ball:
60     """
61
62     if side_hit == 'RIGHT' or side_hit == 'LEFT':
63         ball.angle = (180-ball.angle) % 360
64     elif side_hit == 'BOTTOM' or side_hit == 'TOP':
65         ball.angle = (- ball.angle) % 360
66
67     self.shunt(ball)
68
69     def shunt(self, ball):
70         # Shunt the ball in its new direction by enough so that it is no longer overlapping
71         with self.
72         # This avoids processing multiple collisions of self and ball before the ball "escapes"
73         while ball.colliding_with(self):
74             ball.move()
75             if (ball.x < 0) or (ball.y < 0):
76                 foobar
77
78 class EndLine(BallDeflector):
79
80     def deflect_ball(self, ball, side_hit):
81         print "hit an endline"
82         if side_hit == 'LEFT':
83             # ball approached from the left to right wall
84             self.game.reset()
85         elif side_hit == 'RIGHT':
86             # ball approached from the right
87             self.game.reset()
88         else:
89             # Shouldn't happen. Must have miscalculated which side was hit, since this is an
90             endline
91             raise Exception(side_hit)
92
93 class Ball(GameObject):
94
95     default_velocity = 6.0 #Number of pixels the ball should move per game cycle
96
97     def update(self, pressed_keys):
98         self.move()
99         if self.in_play:
100             for game_object in self.game.game_objects:
101                 side_hit = self.colliding_with(game_object)
102                 if side_hit:
103                     game_object.deflect_ball(self, side_hit)
104
105     def set_initial_position(self):
106         self.set_position(self.initial_x, self.initial_y)
107         self.velocity = self.default_velocity
108         self.angle = self.generate_random_starting_angle()
109         self.in_play = True
110
111     def generate_random_starting_angle(self):
112         """
113         Generate a random angle that isn't too close to straight up and down or straight side
114         to side
115         :return: an angle in degrees
116         """
117         angle = random.randint(15,75)+90*random.randint(0,3)

```

```

115     debug_print('Starting ball angle: ' + str(angle) + ' degrees')
116     return angle
117
118     def colliding_with(self, game_object):
119         """
120         self is a ball and game_object is some other game_object.
121         If their bounding boxes (the space they take up on screen) don't overlap,
122         return False.
123         If they do overlap, return one of 'LEFT', 'RIGHT', 'TOP', 'BOTTOM',
124         indicating which edge of game_object the ball has hit.
125
126         Note: this code is complicated, in part because of the geometric reasoning.
127         You don't have to understand how this method is implemented, but you will
128         need to understand what it does-- figure out which side of the game_object, if any,
129         the ball collided with first.
130         """
131
132         # x_distance is difference between rightmost object's left-side (x) and the other's right
side (x+width)
133         if (self.x < game_object.x):
134             left, right = self, game_object
135         else:
136             left, right = game_object, self
137         x_distance = right.x - (left.x + left.width)
138         # y_distance is difference between one object's bottom-side (y) and the other's top side
(y + height)
139         if (self.y < game_object.y):
140             bottom, top = self, game_object
141         else:
142             bottom, top = game_object, self
143         y_distance = top.y - (bottom.y + bottom.height)
144
145         if (x_distance > 0) or (y_distance > 0):
146             # no overlap
147             return False
148         else:
149             # figure out which side of game_object self hit
150             # first, special cases of horizontal or vertical approach angle
151             special_cases = {0: 'LEFT', 90: 'BOTTOM', 180: 'RIGHT', 270: 'TOP'}
152             if self.angle in special_cases:
153                 return special_cases[self.angle]
154             else:
155                 # Decide base on self's y position at the point where they intersected in the x-
dimension
156                 (x_vel, y_vel) = as_cartesian(self.velocity, self.angle)
157                 slope = y_vel / x_vel
158                 # go x_distance units either forward or back in x dimension; multiply by slope to
get
159                 # offset in y dimension
160                 y_at_x_collision = self.y - sign(y_vel)*math.fabs(x_distance * slope)
161                 if (self.angle < 90):
162                     # coming from below left, check if top of self was below game_object
163                     if y_at_x_collision + self.height < game_object.y:
164                         return 'BOTTOM'
165                 else:
166                     return 'LEFT'
167                 elif (self.angle < 180):
168                     # coming from below right, check if top of self was below game_object
169                     if y_at_x_collision + self.height < game_object.y:

```

```

170         return 'BOTTOM'
171     else:
172         return 'RIGHT'
173 elif self.angle < 270:
174     # coming from above right, check if bottom of self was above game_object
175     if y_at_x_collision > game_object.y + game_object.height:
176         return 'TOP'
177     else:
178         return 'RIGHT'
179 else:
180     # coming from above right, check if bottom of self was above game_object
181     if y_at_x_collision > game_object.y + game_object.height:
182         return 'TOP'
183     else:
184         return 'LEFT'
185
186 def deflect_ball(self, ball, side_hit):
187     # balls don't deflect other balls
188     pass
189
190 class Paddle (BallDeflector):
191
192     default_velocity = 4.0
193
194     def __init__(self, player = None, up_key = None, down_key = None, left_key =
None, right_key = None,
195                 name = None, img_file = None,
196                 initial_x = 0, initial_y = 0, game=None):
197         BallDeflector.__init__(self, img_file=img_file, initial_x=initial_x, initial_y=
initial_y, game=game)
198         self.player = player
199         self.up_key = up_key
200         self.down_key = down_key
201         self.left_key = left_key
202         self.right_key = right_key
203         self.name = name
204
205     def update(self, pressed_keys):
206
207         self.velocity = self.default_velocity
208         if self.up_key in pressed_keys and not self.down_key in pressed_keys:
209             self.angle = 90
210         elif self.down_key in pressed_keys and not self.up_key in pressed_keys:
211             self.angle = 270
212         elif self.left_key in pressed_keys and not self.right_key in pressed_keys:
213             self.angle = 180
214         elif self.right_key in pressed_keys and not self.left_key in pressed_keys:
215             self.angle = 0
216         else:
217             self.velocity = 0.0
218             self.angle = None
219
220         self.move()
221
222     def hit_position(self, ball):
223         """
224         Returns a number between 0 and 1, representing how far up the paddle the ball hit.
225         If it hit near the top, the number will be close to 1.
226         """

```

```

227
228     virtual_height = self.height + ball.height
229     y_dist = ball.y + ball.height - self.y
230     pct = y_dist / float(virtual_height)
231     return pct
232
233
234 class Game(object):
235     side_paddle_buffer = 50 # how far away from the side wall a paddle should start
236     aux_paddle_buffer = 550 # how far away a forward paddle should start
237     def __init__(self,
238         ball_img = None,
239         paddle_imgs=None,
240         wall_imgs = None,
241         width = 800,
242         height = 450,
243         game_window=None,
244         wall_width = 10,
245         paddle_width = 25,
246         brick_height = 40):
247
248         self.score = [0,0]
249         self.width = width
250         self.height = height
251         self.game_window = game_window
252         self.hit_count = 0
253
254         self.balls = [Ball(img_file= ball_img,
255             initial_x= self.width/2,
256             initial_y = self.height/2,
257             game=self)
258         ]
259         self.paddles = [
260             Paddle(player = 1,
261                 up_key=pyglet.window.key.W,
262                 down_key=pyglet.window.key.S,
263                 name = 'Player 1',
264                 img_file = paddle_imgs[0],
265                 initial_x= self.side_paddle_buffer + paddle_width/2,
266                 initial_y = self.height/2,
267                 game=self
268             ),
269             Paddle(player = 2,
270                 up_key=pyglet.window.key.U,
271                 down_key=pyglet.window.key.J,
272                 name='Player 2',
273                 img_file=paddle_imgs[1],
274                 initial_x = self.width-self.side_paddle_buffer - paddle_width/2,
275                 initial_y = self.height/2,
276                 game=self)
277         ]
278         self.walls = [
279             BallDeflector(initial_x = 0, #bottom
280                 initial_y = 0,
281                 img_file = wall_imgs[1],
282                 game = self),
283             BallDeflector(initial_x = 0, #top
284                 initial_y = self.height - wall_width,
285                 img_file = wall_imgs[1],
286                 game = self),

```

```

286     EndLine(initial_x = 0, #left
287             initial_y = 0,
288             img_file = wall_imgs[0],
289             game = self),
290     EndLine(initial_x = self.width - wall_width, #right
291             initial_y = 0,
292             img_file = wall_imgs[0],
293             game = self),
294 ]
295 self.bricks = [] # Not used in this initial version
296 self.game_objects = self.walls + self.bricks + self.paddles + self.balls
297
298 def update(self, pressed_keys):
299     """
300     Update the game based on the current state of its game objects and the set of keys
301     currently
302     being pressed
303     :param pressed_keys: a set() object containing an int representing each key currently
304     being pressed
305     The matching between numbers and keys is defined by Pyglet. For example, pyglet.
306     window.key.W is
307     equal to 119
308     :return:
309     """
310     # debug_print('Updating game state with currently pressed keys : ' + str(
311     pressed_keys))
312     for game_object in self.game_objects:
313         game_object.update(pressed_keys)
314
315     def reset(self, pause=True):
316         # self.score = [0,0]
317         for game_object in self.game_objects:
318             game_object.set_initial_position()
319
320     self.hit_count = 0
321     debug_print('Game reset')
322     self.game_window.redraw()
323
324     if pause:
325         debug_print('Pausing. Hit P to unpause')
326         self.game_window.pause()
327
328     def draw(self):
329         for game_object in self.game_objects:
330             game_object.draw()
331
332     def increment_hit_count(self):
333         # this method will be used in an exercise in discussion section
334         self.hit_count += 1
335
336 class GameWindow(pyglet.window.Window):
337
338     def __init__(self, ball_img, paddle_imgs, wall_imgs,
339                 width = 800, height = 450, *args, **kwargs):
340         pyglet.window.Window.__init__(self, width=width, height=height, *args, **
341         kwargs)
342         self.paused = False

```

```

340     self.game = Game(ball_img,paddle_imgs, wall_imgs, width,height,self)
341     self.currently_pressed_keys = set() #At any given moment, this holds the keys that
are currently being pressed.
342     # This gets passed to Game.update() to help it decide how to move its various game
objects
343     self.score_label = pyglet.text.Label('Score: 0 - 0',
344         font_name='Times New Roman',
345         font_size=14,
346         x=width-75, y=height-25,
347         anchor_x='center', anchor_y='center')
348
349     # Decide how often we want to update the game, which involves
350     # first telling the game object to update itself and all its objects
351     # and then rendering the updated game using
352     self.fps = 20 #Number of frames per seconds
353
354     #This tells Pyglet to call Window.update() once every fps-th of a second
355     pyglet.clock.schedule_interval(self.update, 1.0/self.fps)
356     pyglet.clock.set_fps_limit(self.fps)
357
358     def on_key_press(self, symbol, modifiers):
359         """
360         This is an overwrite of pyglet.window.Window.on_key_press()
361         This gets called by the pyglet engine whenever a key is pressed. Whenever that
happens,
362         we want to add each key being pressed to the set of currently-pressed keys if it isn't
363         already in there
364         That's if the key pressed isn't 'Q' or 'Esc'. If it is, then just quit.
365         :param symbol: a single key identified as an int
366         :param modifiers: I don't know what this is. I am ignoring this.
367         :return:
368         """
369
370         if symbol == pyglet.window.key.Q or symbol == pyglet.window.key.ESCAPE:
371             debug_print('Exit key detected. Exiting game...')
372             pyglet.app.exit()
373         elif symbol == pyglet.window.key.R:
374             debug_print('Resetting...')
375             self.game.reset()
376         elif symbol == pyglet.window.key.P:
377             if not self.paused:
378                 self.pause()
379             else:
380                 self.unpause()
381         elif not symbol in self.currently_pressed_keys:
382             self.currently_pressed_keys.add(symbol)
383
384     def pause(self):
385         debug_print('Pausing')
386         pyglet.clock.unschedule(self.update)
387         self.paused = True
388
389     def unpause(self):
390         debug_print('Unpausing')
391         pyglet.clock.schedule_interval(self.update, 1.0/self.fps)
392         self.paused = False
393
394     def on_key_release(self, symbol, modifiers):
395         if symbol in self.currently_pressed_keys:

```

```

396         self.currently_pressed_keys.remove(symbol)
397
398     def update(self, *args, **kwargs):
399         self.game.update(self.currently_pressed_keys)
400         self.redraw()
401
402     def redraw(self):
403         self.clear()
404         self.game.draw()
405         self.score_label.draw()
406
407     def redraw_label(self):
408         self.score_label.text = 'Score: ' + str(self.game.score[0]) + ' - ' + str(self.game.
score[1])
409
410
411     def debug_print(string):
412         """
413         A little convenience function that prints the string if the global debug variable is True,
414         and otherwise does nothing
415         :param string:
416         :return:
417         """
418         if debug:
419             print string
420
421     def main():
422         debug_print("Initializing window...")
423         ball_img = pygame.image.load('ball.png')
424         # ball_img = pygame.image.load('vertical_wall.png')
425         paddle_imgs = [pygame.image.load('paddle1.png'),
426                        pygame.image.load('paddle2.png')]
427         wall_imgs = [pygame.image.load('vertical_wall.png'),
428                     pygame.image.load('horizontal_wall.png'),
429                     pygame.image.load('brick.png')]
430         window = GameWindow(ball_img, paddle_imgs, wall_imgs)
431         debug_print("Done initializing window! Initializing app...")
432
433         pygame.app.run()
434
435     main()
436
437

```