```
    |0  |1  |2  |3  |4  |5  |6  |7  |8
1   # Toadie
2   # Code Angel
3
4   # Classes: MovingObject, Digger, Truck, Car, Turtle, Log, Pavement, Pad, Timer
5
6   import pygame
7
8   import toadie
9
10  # Define constants
11  BLOCK_SIZE = 32
12
13  PAVEMENT_LANE_1 = 13
14  CAR_LANE_1 = 12
15  DIGGER_LANE = 11
16  CAR_LANE_2 = 10
17  CAR_LANE_3 = 9
18  TRUCK LANE = 8
19  PAVEMENT_LANE_2 = 7
20  TURTLE_LANE_1 = 6
21  LOG_LANE_1 = 5
22  LOG_LANE_2 = 4
23  TURTLE_LANE_2 = 3
24  LOG_LANE_3 = 2
25  HOME_LANE = 1
26
27  TOAD_TIME = 45
28  MILLISECONDS = 1000
29
30
31  # MovingObject class, the base class for all road and river objects
32  class MovingObject:
33
34      # All moving objects have an image, rectangle and speed
35      def __init__(self, speed, location, object_image):
36          self.image = toadie.load_media('image', object_image)
37          self.rect = self.image.get_rect()
38          self.rect.x = location[0] * BLOCK_SIZE
39          self.rect.y = location[1] * BLOCK_SIZE
40          self.padding_height = (BLOCK_SIZE - self.image.get_height()) / 2
41
    |0  |1  |2  |3  |4  |5  |6  |7  |8
```

```
    |0  |1  |2  |3  |4  |5  |6  |7  |8
42          self.speed = speed
43
44      # All moving objects move using the same principle
45      def move(self, game_screen):
46
47          # Add speed on to x coordinate
48          self.rect.x = self.rect.x + self.speed
49
50          # If object goes off left of screen start again at right of screen
51          if self.rect.right < 0:
52              self.rect.x = toadie.SCREEN_WIDTH
53
54          # If object goes off right of screen start again at left of screen
55          if self.rect.left > toadie.SCREEN_WIDTH:
56              self.rect.x = 0 - self.rect.width
57
58          # Draw the object
59          game screen.blit(self.image, [self.rect.x, self.rect.y + self.padding height])
60
61
62  # Digger Class takes start_x as parameter, and speed is 2
63  class Digger(MovingObject):
64      def __init__(self, start_x):
65          location = [start_x, DIGGER_LANE]
66          MovingObject.__init__(self, 2, location, 'digger')
67
68
69  # Digger Class takes start_x as parameter, and speed is -3
70  class Truck(MovingObject):
71      def __init__(self, start_x):
72          location = [start_x, TRUCK_LANE]
73          MovingObject.__init__(self, -3, location, 'truck')
74
75
76  # RedCar Class takes start_x as parameter, and speed is -2
77  class RedCar(MovingObject):
78      def __init__(self, start_x):
79          location = [start_x, CAR_LANE_1]
80          MovingObject.__init__(self, -2, location, 'car_red')
81
82
    |0  |1  |2  |3  |4  |5  |6  |7  |8
```

```
    |0  |1  |2  |3  |4  |5  |6  |7  |8
# PurpleCar Class takes start_x as parameter, and speed is -3
class PurpleCar(MovingObject):
    def __init__(self, start_x):
        location = [start_x, CAR_LANE_2]
        MovingObject.__init__(self, -3, location, 'car_purple')


# PinkCar Class takes start x as parameter, and speed is -2
class PinkCar(MovingObject):
    def __init__(self, start_x):
        location = [start_x, CAR_LANE_3]
        MovingObject.__init__(self, -2, location, 'car_pink')


# Turtle Class takes start_x and size as parameter. Size of turtle chain is either 2 or 3
class Turtle(MovingObject):
    def __init__(self, start_x, size):
        if size == 3:
            MovingObject.__init__(self, -3, [start_x, TURTLE_LANE_1], 'turtle3')
        else:
            MovingObject.__init__(self, -3, [start_x, TURTLE_LANE_2], 'turtle2')


# Log Class takes start_x and size as parameter. Size of log chain is either 2, 3 or 4
class Log(MovingObject):
    def __init__(self, start_x, size):
        if size == 1:
            MovingObject.__init__(self, 2, [start_x, LOG_LANE_1], 'log')
        elif size == 2:
            MovingObject.__init__(self, 3, [start_x, LOG_LANE_3], 'log2')
        elif size == 3:
            MovingObject.__init__(self, 4, [start_x, LOG_LANE_2], 'log3')


# Pavement class takes a location as parameter
class Pavement:
    def __init__(self, location):
        self.x = location[0]
        self.y = location[1]
        self.image = toadie.load_media('image', 'pavement')

    |0  |1  |2  |3  |4  |5  |6  |7  |8
```



```
    |0  |1  |2  |3  |4  |5  |6  |7  |8
124         self.rect = pygame.Rect(
125             self.x * BLOCK_SIZE,
126             self.y * BLOCK_SIZE,
127             BLOCK_SIZE,
128             BLOCK_SIZE
129         )
130
131     # The pavement draws itself in the correct location
132     def draw(self, game_screen):
133         game_screen.blit(self.image, [self.rect.x, self.rect.y])
134
135
136 # Pad Class takes the x coordinate as parameter
137 class Pad:
138     def __init__(self, x_coord):
139         self.x = x_coord
140         self.image = toadie.load_media('image', 'pad')
141         self.occupied_image = toadie.load_media('image', 'occupied_pad')
142         self.padding_width = (BLOCK_SIZE - self.image.get_width()) / 2 + self.image.get_width()
143         self.padding_height = (BLOCK_SIZE - self.image.get_height()) / 2
144
145         self.rect = pygame.Rect(
146             self.x * BLOCK_SIZE + self.padding_width,
147             HOME_LANE * BLOCK_SIZE + self.padding_height,
148             BLOCK_SIZE,
149             BLOCK_SIZE
150         )
151
152         self.occupied = False
153
154     # The pad draws itself in the correct location, with the image depicting either occupied or unoccupied pad
155     def draw(self, game_screen):
156         if self.occupied is False:
157             game_screen.blit(self.image, [self.rect.x, self.rect.y])
158         else:
159             game_screen.blit(self.occupied_image, [self.rect.x, self.rect.y])
160
161
162 # Timer Class
163 class Timer:
164
    |0  |1  |2  |3  |4  |5  |6  |7  |8
```



```
    |0  |1  |2  |3  |4  |5  |6  |7  |8
165     def __init__(self):
166
167         # The length of time the level runs for
168         self.duration = TOAD_TIME * MILLISECONDS
169
170         # The value in ticks that the timer started
171         self.start_time = pygame.time.get_ticks()
172
173         # The time left on the timer - starts as duration
174         self.time_remaining = self.duration
175
176     # Called from the main game loop, updates the time remaining on the timer
177     def update_time(self):
178         new_time = pygame.time.get_ticks()
179         elapsed_time = new_time - self.start_time
180         self.time_remaining = self.duration - elapsed_time
181         if self.time_remaining < 0:
182             self.time remaining = 0
183
184     # Tests if the timer is out of time, returnd True if out of time, False if not
185     def out_of_time(self):
186         if self.time_remaining <= 0:
187             no_time_left = True
188         else:
189             no_time_left = False
190
191         return no_time_left
192
193     # The number of seconds remaining on the timer
194     def get_seconds_left(self):
195         return int(self.time_remaining / MILLISECONDS)
196
197     # Reset the timer
198     def reset(self):
199         self.start_time = pygame.time.get_ticks()
200         self.time_remaining = self.duration
201
    |0  |1  |2  |3  |4  |5  |6  |7  |8
```