```
import os
import time
import random
import importlib

class Simulator(object):
    """Simulates agents in a dynamic smartcab environment.

    Uses PyGame to display GUI, if available.
    """

    colors = {
        'black'   : (  0,   0,   0),
        'white'   : (255, 255, 255),
        'red'     : (255,   0,   0),
        'green'   : (  0, 255,   0),
        'blue'    : (  0,   0, 255),
        'cyan'    : (  0, 200, 200),
        'magenta' : (200,   0, 200),
        'yellow'  : (255, 255,   0),
        'orange'  : (255, 128,   0)
    }

    def __init__(self, env, size=None, update_delay=1.0, display=True):
        self.env = env
        self.size = size if size is not None else ((self.env.grid_size[0] + 1) *
self.env.block_size, (self.env.grid_size[1] + 1) * self.env.block_size)
        self.width, self.height = self.size

        self.bg_color = self.colors['white']
        self.road_width = 5
        self.road_color = self.colors['black']

        self.quit = False
        self.start_time = None
        self.current_time = 0.0
        self.last_updated = 0.0
        self.update_delay = update_delay  # duration between each step (in secs)

        self.display = display
        if self.display:
            try:
                self.pygame = importlib.import_module('pygame')
                self.pygame.init()
                self.screen = self.pygame.display.set_mode(self.size)

                self.frame_delay = max(1, int(self.update_delay * 1000))  # delay between
```

```
GUI frames in ms (min: 1)
            self.agent_sprite_size = (32, 32)
            self.agent_circle_radius = 10  # radius of circle, when using simple
representation
            for agent in self.env.agent_states:
                agent._sprite =
self.pygame.transform.smoothscale(self.pygame.image.load(os.path.join("../images",
"car-{}.png".format(agent.color))), self.agent_sprite_size)
                agent._sprite_size = (agent._sprite.get_width(), agent._sprite.get_height())

            self.font = self.pygame.font.Font(None, 28)
            self.paused = False
        except ImportError as e:
            self.display = False
            print "Simulator.__init__(): Unable to import pygame; display disabled.\n{}:
{}".format(e.__class__.__name__, e)
        except Exception as e:
            self.display = False
            print "Simulator.__init__(): Error initializing GUI objects; display disabled.\n{}:
{}".format(e.__class__.__name__, e)

    def run(self, n_trials=1):
        self.quit = False
        for trial in xrange(n_trials):
            print "Simulator.run(): Trial {}".format(trial)  # [debug]
            self.env.reset()
            self.current_time = 0.0
            self.last_updated = 0.0
            self.start_time = time.time()
            while True:
                try:
                    # Update current time
                    self.current_time = time.time() - self.start_time
                    #print "Simulator.run(): current_time = {:.3f}".format(self.current_time)

                    # Handle GUI events
                    if self.display:
                        for event in self.pygame.event.get():
                            if event.type == self.pygame.QUIT:
                                self.quit = True
                            elif event.type == self.pygame.KEYDOWN:
                                if event.key == 27:  # Esc
                                    self.quit = True
                                elif event.unicode == u' ':
                                    self.paused = True

                        if self.paused:
```

```
                    self.pause()

                # Update environment
                if self.current_time - self.last_updated >= self.update_delay:
                    self.env.step()
                    self.last_updated = self.current_time

                # Render GUI and sleep
                if self.display:
                    self.render()
                    self.pygame.time.wait(self.frame_delay)
            except KeyboardInterrupt:
                self.quit = True
            finally:
                if self.quit or self.env.done:
                    break

        if self.quit:
            break

    def render(self):
        # Clear screen
        self.screen.fill(self.bg_color)

        # Draw elements
        # * Static elements
        for road in self.env.roads:
            self.pygame.draw.line(self.screen, self.road_color, (road[0][0] *
self.env.block_size, road[0][1] * self.env.block_size), (road[1][0] * self.env.block_size,
road[1][1] * self.env.block_size), self.road_width)

        for intersection, traffic_light in self.env.intersections.iteritems():
            self.pygame.draw.circle(self.screen, self.road_color, (intersection[0] *
self.env.block_size, intersection[1] * self.env.block_size), 10)
            if traffic_light.state:  # North-South is open
                self.pygame.draw.line(self.screen, self.colors['green'],
                    (intersection[0] * self.env.block_size, intersection[1] * self.env.block_size -
15),
                    (intersection[0] * self.env.block_size, intersection[1] * self.env.block_size +
15), self.road_width)
            else:  # East-West is open
                self.pygame.draw.line(self.screen, self.colors['green'],
                    (intersection[0] * self.env.block_size - 15, intersection[1] *
self.env.block_size),
                    (intersection[0] * self.env.block_size + 15, intersection[1] *
self.env.block_size), self.road_width)
```

```
        # * Dynamic elements
        for agent, state in self.env.agent_states.iteritems():
            # Compute precise agent location here (back from the intersection some)
            agent_offset = (2 * state['heading'][0] * self.agent_circle_radius, 2 *
state['heading'][1] * self.agent_circle_radius)
            agent_pos = (state['location'][0] * self.env.block_size - agent_offset[0],
state['location'][1] * self.env.block_size - agent_offset[1])
            agent_color = self.colors[agent.color]
            if hasattr(agent, '_sprite') and agent._sprite is not None:
                # Draw agent sprite (image), properly rotated
                rotated_sprite = agent._sprite if state['heading'] == (1, 0) else
self.pygame.transform.rotate(agent._sprite, 180 if state['heading'][0] == -1 else
state['heading'][1] * -90)
                self.screen.blit(rotated_sprite,
                    self.pygame.rect.Rect(agent_pos[0] - agent._sprite_size[0] / 2,
agent_pos[1] - agent._sprite_size[1] / 2,
                        agent._sprite_size[0], agent._sprite_size[1]))
            else:
                # Draw simple agent (circle with a short line segment poking out to indicate
heading)
                self.pygame.draw.circle(self.screen, agent_color, agent_pos,
self.agent_circle_radius)
                self.pygame.draw.line(self.screen, agent_color, agent_pos, state['location'],
self.road_width)
            if agent.get_next_waypoint() is not None:
                self.screen.blit(self.font.render(agent.get_next_waypoint(), True, agent_color,
self.bg_color), (agent_pos[0] + 10, agent_pos[1] + 10))
            if state['destination'] is not None:
                self.pygame.draw.circle(self.screen, agent_color, (state['destination'][0] *
self.env.block_size, state['destination'][1] * self.env.block_size), 6)
                self.pygame.draw.circle(self.screen, agent_color, (state['destination'][0] *
self.env.block_size, state['destination'][1] * self.env.block_size), 15, 2)

        # * Overlays
        text_y = 10
        for text in self.env.status_text.split('\n'):
            self.screen.blit(self.font.render(text, True, self.colors['red'], self.bg_color), (100,
text_y))
            text_y += 20

        # Flip buffers
        self.pygame.display.flip()

    def pause(self):
        abs_pause_time = time.time()
        pause_text = "[PAUSED] Press any key to continue..."
        self.screen.blit(self.font.render(pause_text, True, self.colors['cyan'], self.bg_color),
```

```
(100, self.height - 40))
        self.pygame.display.flip()
        print pause_text  # [debug]
        while self.paused:
            for event in self.pygame.event.get():
                if event.type == self.pygame.KEYDOWN:
                    self.paused = False
            self.pygame.time.wait(self.frame_delay)
        self.screen.blit(self.font.render(pause_text, True, self.bg_color, self.bg_color), (100,
self.height - 40))
        self.start_time += (time.time() - abs_pause_time)
```