```
from pygame.locals import *
import pygame
import random
import copy

BLOCK_SIZE = 44
MEAN_FIRE_INTERVAL = 20
MEAN_FIRE_FIGHT_DURATION = 10
RED_LIGHT_DURATION = 10
NUMBER_OF_INTERSECTIONS = 2


class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y


class Player:
    speed = 1
    position = Point(1, 1)

    def moveRight(self):
        self.position.x = self.position.x + self.speed

    def moveLeft(self):
        self.position.x = self.position.x - self.speed

    def moveUp(self):
        self.position.y = self.position.y - self.speed

    def moveDown(self):
        self.position.y = self.position.y + self.speed


class Fire:
    def __init__(self, x, y, start, maze):
        self.position = Point(x, y)
        self.start = start
        self.severity = 1
        self.maze = copy.deepcopy(maze.maze)
        self.columns = maze.M
        self.rows = maze.N
```

```
        while True:
            if random.randrange(MEAN_FIRE_FIGHT_DURATION) == 0:
                break
            self.severity += 1
        self.floodfill(maze.M * maze.N)

    def floodfill(self, num_cells):
        self.maze[self.position.y][self.position.x] = -num_cells
        boundary_set = set(self.boundary(self.position))
        while boundary_set:
            p = boundary_set.pop()
            self.maze[p.y][p.x] = self.fill_value(p)
            boundary_set = boundary_set.union(set(self.boundary(p)))

    def boundary(self, position):
        return filter(lambda p: self.maze[position.y][position.x] + 1 < self.maze[p.y][p.x] <= 0,
                      self.base_neighborhood(position))

    def base_neighborhood(self, position):
        neighborhood = []
        neighborhood.append(Point(position.x - 1, position.y))
        neighborhood.append(Point(position.x + 1, position.y))
        neighborhood.append(Point(position.x, position.y - 1))
        neighborhood.append(Point(position.x, position.y + 1))
        return filter(lambda p: (0 <= p.x < self.columns) and (0 <= p.y < self.rows), neighborhood)

    def fill_value(self, position):
        best_neighbor = 1 + min(self.maze[p.y][p.x] for p in self.base_neighborhood(position))
        return min(best_neighbor, self.maze[position.y][position.x])

    def min_neighbor(self, position):
        return min(self.base_neighborhood(position), key=lambda p: self.maze[p.y][p.x])


class Intersection:
    def __init__(self, x, y):
        self.position = Point(x, y)


class Maze:
    def __init__(self):
        self.M = 10
        self.N = 8
```

```
        self.obstacles = []
        self.maze = [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                     [1, 0, 1, 1, 1, 1, 1, 1, 0, 1],
                     [1, 0, 1, 0, 0, 0, 0, 0, 0, 1],
                     [1, 0, 1, 0, 1, 1, 1, 1, 0, 1],
                     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], ]

    def draw(self, display_surf, obstacle_surf, intersection_surf):
        for x in range(0, self.M):
            for y in range(0, self.N):
                if self.maze[y][x] == 1:
                    display_surf.blit(obstacle_surf, (x * BLOCK_SIZE, y * BLOCK_SIZE))

                if self.maze[y][x] == 2:
                    display_surf.blit(intersection_surf, (x * BLOCK_SIZE, y * BLOCK_SIZE))

    def is_cell_clear(self, position):
        if self.maze[position.y][position.x] != 0:
            return False

        for obstacle in self.obstacles:
            if position.x == obstacle[1].x and position.y == obstacle[1].y:
                return False

        return True

    def is_cell_drivable(self, position):
        return self.maze[position.y][position.x] == 0

    def add_obstacle(self, type, position):
        self.obstacles.append((type, position))


class App:
    windowWidth = 800
    windowHeight = 600
    player = 0

    def __init__(self):
        # Step 3
```

```
        # This method starts setting up the application. It sets _running to true, it also calls the
constructor for the
        # Player object and the Maze object.
        # Notice that in Objected Oriented Programming (OOP), the object can refer to itself with
the keyword "self"
        self._running = True
        self._display_surf = None
        self._firetruck_surf = None
        self._obstacle_surf = None
        # You can look at the __init__ methods for the Player and Maze class if you want at this
point, I won't label
        # it with a step
        self.player = Player()
        self.maze = Maze()
        self.clock = -1
        self.fires = []
        self.fires_fought = []
        self.intersections = []
        self.create_intersections(NUMBER_OF_INTERSECTIONS)
        self.advance_clock()

    def on_init(self):
        pygame.init()
        self._display_surf = pygame.display.set_mode((self.windowWidth, self.windowHeight),
pygame.HWSURFACE)

        pygame.display.set_caption('Pygame pythonspot.com example')
        self._running = True
        # Load the image files
        self._firetruck_surf = pygame.image.load("firetruck.png").convert()
        self._obstacle_surf = pygame.image.load("block.png").convert()
        self._fire_surf = pygame.image.load("fire.png").convert()
        self._intersection_surf = pygame.image.load("intersection.png").convert()
        # Resize the images so they are the same size
        self._firetruck_surf = pygame.transform.scale(self._firetruck_surf, (40, 40))
        self._obstacle_surf = pygame.transform.scale(self._obstacle_surf, (40, 40))
        self._fire_surf = pygame.transform.scale(self._fire_surf, (40, 40))
        self._intersection_surf = pygame.transform.scale(self._intersection_surf, (40, 40))

    def on_event(self, event):
        if event.type == QUIT:
            self._running = False
```

```
    def on_loop(self):
        # Step 9
        # Right now we are not doing anything for on_loop, that is what "pass" means
        pass

    def on_render(self):
        # Step 11
        self._display_surf.fill((0, 0, 0))
        # This draws the obstacle/blocks
        self.maze.draw(self._display_surf, self._obstacle_surf, self._intersection_surf)
        # Note that here, it draws the image at self.player.x and self.player.y
        # This is how the position of the firetruck actually changes
        # blit -> means to draw
        for fire in self.fires:
            self._display_surf.blit(self._fire_surf, (fire.position.x * BLOCK_SIZE, fire.position.y *
BLOCK_SIZE))
        for intersection in self.intersections:
            self._display_surf.blit(self._intersection_surf,
                                    (intersection.position.x * BLOCK_SIZE, intersection.position.y *
BLOCK_SIZE))
        self._display_surf.blit(self._firetruck_surf,
                                (self.player.position.x * BLOCK_SIZE, self.player.position.y *
BLOCK_SIZE))
        pygame.display.flip()

    def on_cleanup(self):
        pygame.quit()

    def advance_clock(self, interval=1):
        for i in range(interval):
            self.clock += 1
            if random.randrange(MEAN_FIRE_INTERVAL) == 0:
                while True:
                    x = random.randrange(self.maze.M)
                    y = random.randrange(self.maze.N)
                    fire_point = Point(x, y)
                    if self.maze.is_cell_clear(fire_point):
                        self.fires.append(Fire(x, y, self.clock, self.maze))
                        self.maze.add_obstacle('Fire', fire_point)
                        break
        print("clock={}".format(self.clock))

    def create_intersections(self, num_intersections):
```

```
        for i in range(num_intersections):
            while True:
                x = random.randrange(self.maze.M)
                y = random.randrange(self.maze.N)
                intersection_point = Point(x, y)
                if self.maze.is_cell_clear(intersection_point):
                    self.intersections.append(Intersection(x, y))
                    self.maze.add_obstacle('Intersection', intersection_point)
                    break

    def on_execute(self):
        # Step 5
        # Try to initialize the game, if it fails, self.on_init() will return False so then self
        # running will be set to False. It was originally set to True in Step 3
        if self.on_init() == False:
            self._running = False

        # Step 6
        # This "while" loop will keep on spinning because self._running is true. Only when it is set
to False will
        # the loop stop
        while self._running:
            # Step 7
            # Gets the "event" - the event in this case would be a key being pressed
            # There could be multiple events at a time (like if you pressed a bunch of keys at the
same time)
            events = pygame.event.get()
            for event in events:
                # We are only processing the key press when the key goes "up", which means when it
is released.
                # We do this so that holding down the key doesn't register multiple times
                if event.type == pygame.KEYUP:
                    has_moved = False
                    direction = ""
                    # This checks the key that was pressed to see if it something that we care about.
                    # It can be the right, left,
                    # up, down, or escape keys
                    if event.key == K_RIGHT:
                        potential_position = Point(self.player.position.x + 1, self.player.position.y)
                        if self.maze.is_cell_drivable(potential_position):
                            self.player.moveRight()
                            has_moved = True
                            direction = "right"
```

```
                else:
                    print("Cell is not clear!")

            if event.key == K_LEFT:
                potential_position = Point(self.player.position.x - 1, self.player.position.y)
                if self.maze.is_cell_drivable(potential_position):
                    self.player.moveLeft()
                    has_moved = True
                    direction = "left"
                else:
                    print("Cell is not clear!")

            if event.key == K_UP:
                potential_position = Point(self.player.position.x, self.player.position.y - 1)
                if self.maze.is_cell_drivable(potential_position):
                    has_moved = True
                    self.player.moveUp()
                    direction = "up"
                else:
                    print("Cell is not clear!")

            if event.key == K_DOWN:
                potential_position = Point(self.player.position.x, self.player.position.y + 1)
                if self.maze.is_cell_drivable(potential_position):
                    self.player.moveDown()
                    has_moved = True
                    direction = "down"
                else:
                    print("Cell is not clear!")

            if event.key==K_SPACE:
                if self.fires:
                    worst_fire=max(self.fires, key=lambda f: f.severity)
                    best_direction=worst_fire.min_neighbor(self.player.position)
                    self.player.position=best_direction
                has_moved= True

            if has_moved:
                fire_fought = None
                for fire in self.fires:
                    if self.player.position.x == fire.position.x and self.player.position.y ==
fire.position.y:
                        fire_fought = fire
```

```
                        break

                in_intersection = False
                for intersection in self.intersections:
                    if self.player.position.x == intersection.position.x and self.player.position.y ==
intersection.position.y:
                        in_intersection = True

                if fire_fought:  # Fighting a fire
                    fire_fought.fought = self.clock
                    self.fires.remove(fire_fought)
                    self.fires_fought.append(fire_fought)
                    self.advance_clock(fire_fought.severity)
                elif in_intersection:
                    self.advance_clock(RED_LIGHT_DURATION)
                else:  # At empty cell
                    self.advance_clock()
                    print("You moved the truck %s to (%d,%d)"
                          % (direction, self.player.position.x, self.player.position.y))

            # Step 8
            # If the escape key is pressed, the game will stop
            if event.key == K_ESCAPE:
                self._running = False

        # This method can do whatever you want it to do at every loop, look at Step 9
        self.on_loop()

        # Step 10
        # Render means to draw the image on the screen, so everytime something happens the
screen will be re-drawn
        # again. Go to Step 11
        self.on_render()
    self.on_cleanup()


if __name__ == "__main__":
    # Step 1) Start the Pygame program

    # Step 2) This is called a Constructor - it creates an App object and calls it theApp
    # The constructor is called by App() - this calls the App Class' __init__ method.
    # Take a look at the method at Step 3. This just creates theApp but doesn't actually "start it"
    theApp = App()
```

```
# Step 4) This will actually start the app. Look at the .on_execute() method at Step 5.
theApp.on_execute()
```