problem set 3实验报告

16337183 孟衍璋

Problem 1

完成`RectangularRoom`和`Robot`两个类的构建。需要完成以下实现：

RectangularRoom

```
def __init__(self, width, height, dirt_amount):
    self.width = int(width)
    self.height = int(height)
    self.dirt_amount = dirt_amount
    self.tile = {}  # 定义一个字典，key为一个元组，代表瓷砖的位置；value为
dirt_amount
    for x in range(self.width):
        for y in range(self.height):
            self.tile[(x,y)] = dirt_amount
```

值得注意的是需要定义一个字典，来表示瓷砖的干净程度。key为一个元组，代表瓷砖的位置；value为dirt_amount。

```
def clean_tile_at_position(self, pos, capacity):
    # 将位置转化为整数
    x = math.floor(pos.get_x())
    y = math.floor(pos.get_y())
    # 如果经过清扫，污渍的数目大于等于0，则更新数值；如果小于0，那么置为0
    if self.tile[(x,y)] - capacity >= 0:
        self.tile[(x,y)] -= capacity
    else:
        self.tile[(x,y)] = 0
```

```
def is_tile_cleaned(self, m, n):
    if self.tile[(m,n)] == 0:
        return True
    else:
        return False
```

```
def get_num_cleaned_tiles(self):
    count = 0  # 干净的瓷砖的数目
    for x in range(self.width):
        for y in range(self.height):
            if self.is_tile_cleaned(x, y):
                count += 1
    return count
```

```
def is_position_in_room(self, pos):
    # 将位置转化为整数
    x = math.floor(pos.get_x())
    y = math.floor(pos.get_y())
    # 判断坐标是否在房间的范围中
    if x < self.width and x >= 0 and y < self.height and y >= 0:
        return True
    else:
        return False
```

```
def get_dirt_amount(self, m, n):
    return self.tile[(m,n)]
```

因为`RectangularRoom`是抽象类，不会被实例化，所以有些方法虽然定义了，但是不会实现，等到定义它的子类时才实现。

Robot

```
def __init__(self, room, speed, capacity):
    self.room = room
    self.speed = speed
    self.capacity = capacity
    self.position = room.get_random_position()  # 设置扫地机器人的初始位置
    self.direction = random.random() * 360  # 设置扫地机器人初始朝向
```

```
def get_robot_position(self):
    return self.position
```

```
def get_robot_direction(self):
    return self.direction
```

```
def set_robot_position(self, position):
    self.position = position
```

```
def set_robot_direction(self, direction):
    self.direction = direction
```

与`RectangularRoom`一样，`Robot`也是抽象类，所以有些方法需要等到定义子类的时候实现。

Problem 2

实现两个类：`EmptyRoom`与`FurnishedRoom`，它们都是继承自`RectangularRoom`类。

EmptyRoom

```
def get_num_tiles(self):
    return self.width * self.height
```

```
def is_position_valid(self, pos):
    if self.is_position_in_room(pos):
        return True
    else:
        return False
```

```
def get_random_position(self):
    # 设置初始位置，位置的范围限制在[0, width)与[0, height)中
    x = random.random() * self.width
    y = random.random() * self.height
    return Position(x,y)
```

FurnishedRoom

```
def is_tile_furnished(self, m, n):
    # 判断某块瓷砖是否有家具
    if (m,n) in self.furniture_tiles:
        return True
    else:
        return False
```

```
def is_position_furnished(self, pos):
    # 判断某个位置是否有家具
    x = math.floor(pos.get_x())
    y = math.floor(pos.get_y())
    return self.is_tile_furnished(x,y)
```

```
def is_position_valid(self, pos):
    # 使用之前的函数，如果位置在房间里且没有家具，返回True，反之返回False
    if self.is_position_in_room(pos) == 1 and
self.is_position_furnished(pos) == 0:
        return True
    else:
        return False
```

```
def get_num_tiles(self):
    # 返回可以到达的瓷砖的数目
    count = 0
    for x in range(self.width):
        for y in range(self.height):
            if self.is_position_valid(Position(x,y)):
                count += 1
    return count
```

```
def get_random_position(self):
    # 设置初始位置，位置的范围限制在[0, width)与[0, height)中，且相应位置没有家具
    while(1):
        x = random.random() * self.width
        y = random.random() * self.height
        if self.is_position_valid(Position(x,y)):
            return Position(x,y)
```

Problem 3

这一步需要实现一个标准的机器人类，继承自`Robot`类，并完善了它的属性。

```
def update_position_and_clean(self):
    # 首先计算以当前方向和速度前进，下一个时间周期所在的位置，若这个位置是合理的，则更新位置信息，且根据capacity的值来清洁当前位置
    if self.room.is_position_valid(self.position.get_new_position(self.direction, self.speed)):
        self.position = self.position.get_new_position(self.direction, self.speed)
        self.room.clean_tile_at_position(self.position, self.capacity)
        # 若计算出的位置不合理，则随机转一个方向
    else:
        self.direction = random.random() * 360
```

Problem 4

这一步需要实现一个可能带有错误的机器人类，继承自`Robot`类。

```
def update_position_and_clean(self):
    # 如果机器人出现故障，随机转一个方向
    if self.gets_faulty():
        self.direction = random.random() * 360
    # 首先计算以当前方向和速度前进，下一个时间周期所在的位置，若这个位置是合理的，则更新位置信息，且根据capacity的值来清洁当前位置
    elif self.room.is_position_valid(self.position.get_new_position(self.direction, self.speed)):
        self.position = self.position.get_new_position(self.direction, self.speed)
        self.room.clean_tile_at_position(self.position, self.capacity)
    # 若计算出的位置不合理，则随机转一个方向
    else:
        self.direction = random.random() * 360
```

Problem 5

这一步需要使用之前定义的类的方法来运行仿真，仿真的函数如下：

```
def run_simulation(num_robots, speed, capacity, width, height,
dirt_amount, min_coverage, num_trials, robot_type):
    # 可视化
    # anim = ps3_visualize.RobotVisualization(num_robots, width, height,
False, delay = 0.01)

    steps_list = []  # 存储每个尝试需要的步数
    for trial in range(num_trials):
        room = EmptyRoom(width, height, dirt_amount)  # 生成指定的需要清扫的
房间
        robots = []  # 将num_robots个扫地机器人加入清扫队伍
        for _ in range(num_robots):
            robots.append(robot_type(room, speed, capacity))
        time_steps = 0
        while float(room.get_num_cleaned_tiles() / room.get_num_tiles()) <
min_coverage:
            # anim.update(room, robots)
            # 每个机器人分别清扫一个步骤
            for robot in robots:
                robot.update_position_and_clean()
            time_steps += 1
        # anim.done()
        steps_list.append(time_steps)  # 存储这一次清扫需要的步数
    return float(sum(steps_list) / num_trials)
```

运行结果：

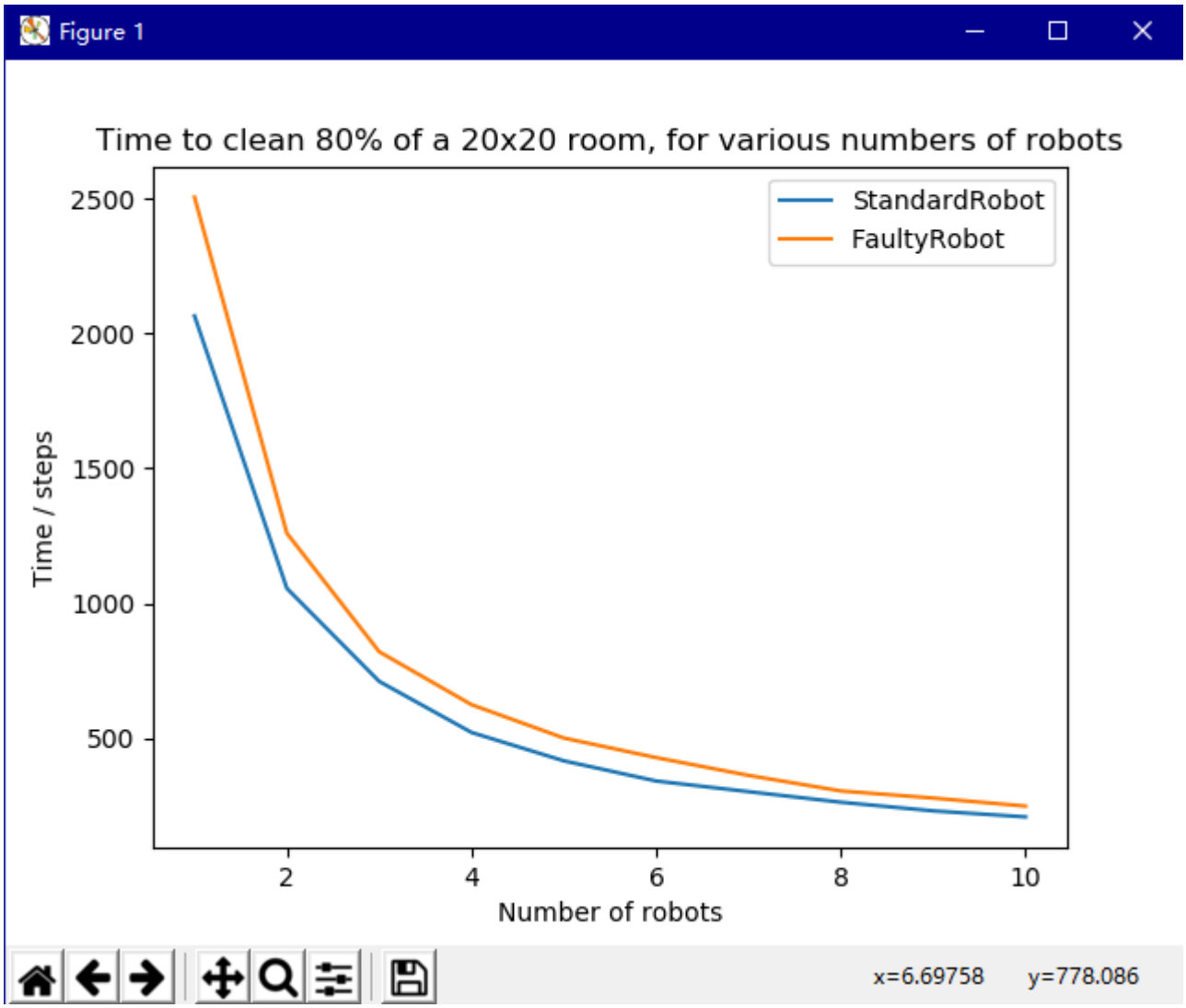
```
D:\Study\assignment\大三下\高级编程技术\assignment\6.0002\ps3>python3 ps3.py
avg time steps: 301.24
avg time steps: 570.32
avg time steps: 704.0
avg time steps: 1249.24
avg time steps: 416.88
```

Problem 6

首先测试清洁相同的房间，使用不同的机器人的效果：

```
show_plot_compare_strategies('Time to clean 80% of a 20x20 room, for various numbers of robots','Number of robots','Time / steps')
```

运行结果：



根据以上结果，可以回答这个问题：

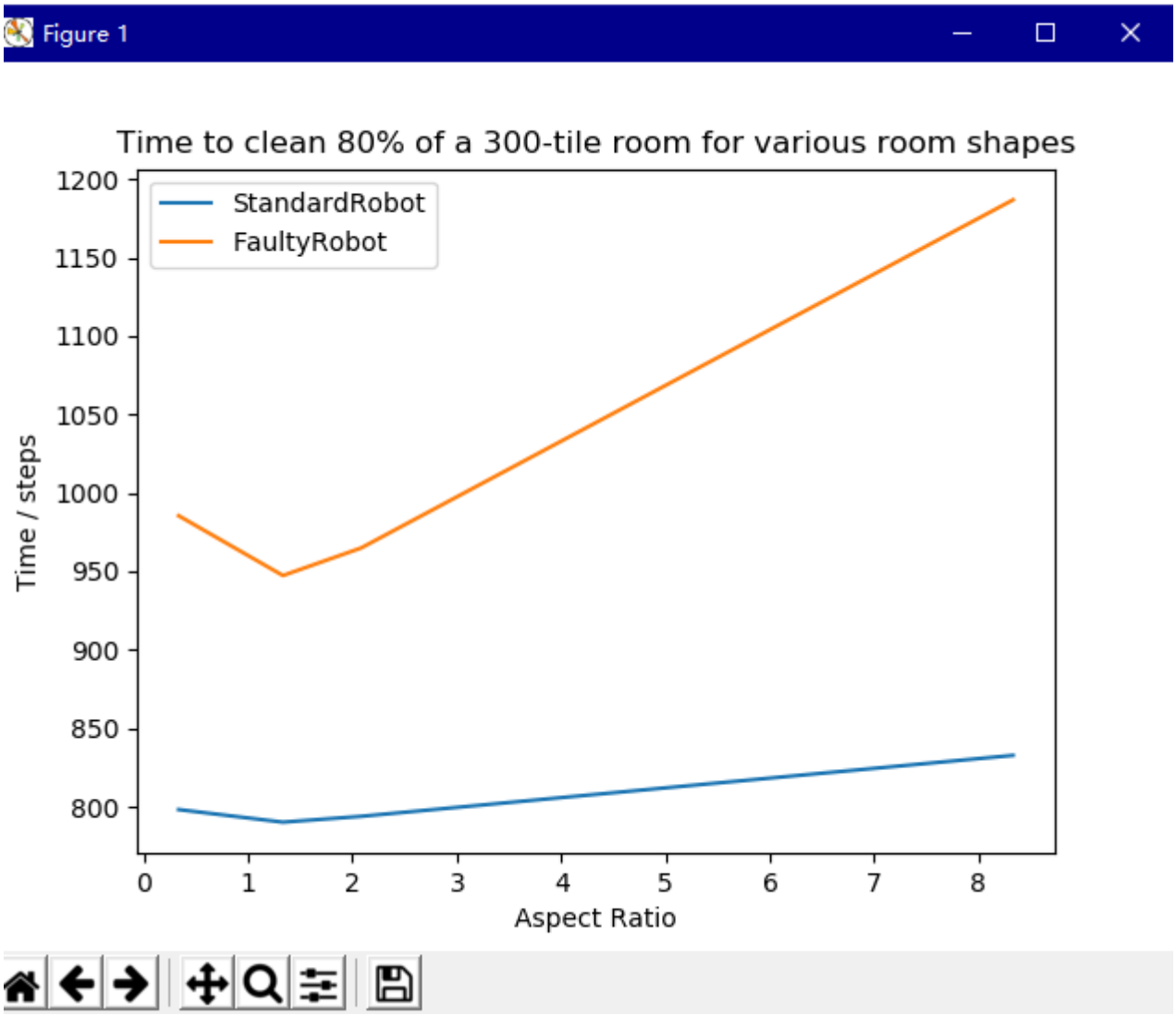
How does the performance of the two robot types compare when cleaning 80% of a 20x20 room?

两种不同的机器人类型分别清扫相同的房间，当机器人的数量增加时，他们所用的步数都有减少，且在机器人数目相同的情况下，FaultRobot的步数总多于StandardRobot。

然后再测试用同样的机器人清洁拥有指定瓷砖数的房间，但是长宽比不相同：

```
show_plot_room_shape('Time to clean 80% of a 300-tile room for various room shapes','Aspect Ratio', 'Time / steps')
```

运行结果：

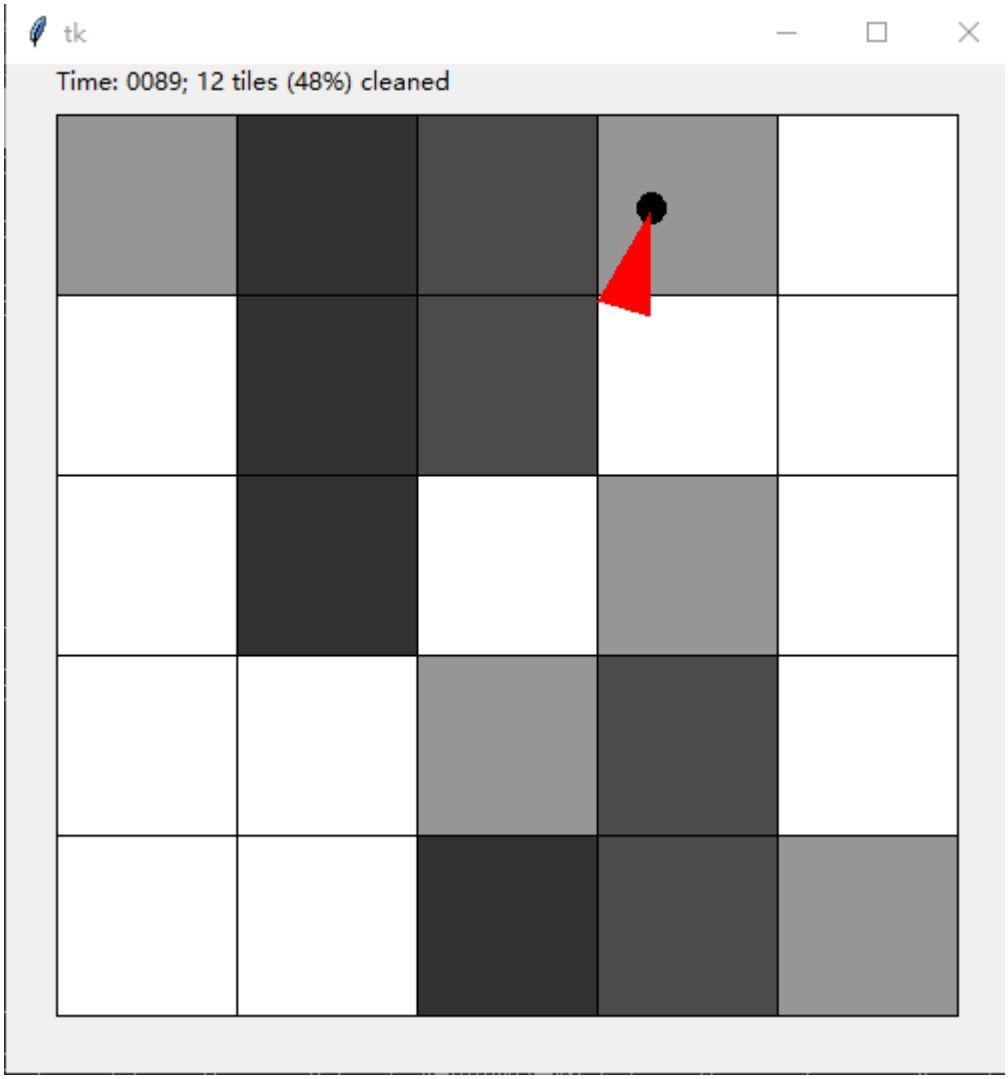


How does the performance of the two robot types compare when two of each robot cleans 80% of rooms with dimensions 10x30, 20x15, 25x12, and 50x6?

两种不同的机器人分别清扫固定瓷砖数的房间，但是长宽比例不同，当长宽比增加时，两种机器人运行的步数都是先减少再增加，长宽比大概为1.4时到达最低，但StandardRobot所用步数明显少于FaultRobot，且跟随长宽比增加，FaultRobot所用步数增长的幅度也更大。

Optional

使用提供的类来实现可视化，只需要在函数`run_simulation`里添加相应实现：


tk
Time: 0089; 12 tiles (48%) cleaned