高级编程技术实验报告

实验三：机器人

数据科学与计算机学院　17大数据与人工智能

17341015　陈鸿峥

一、 问题描述及求解思路

1. 问题一：实施房间与机器人基类

RectangularRoom基类：

- `__init__`：直接设置width和height即可，这里要注意尽管width和height题目中说了是整数，但是在问题六中却输入了一个浮点数，因此在最开始应该先对这两个值取整，以确保后面初始化的正确性。同时创建一个tiles列表，存储每个格的脏物数，每个格的坐标以左下角的坐标作为下标。如正方形格$(1,1),(2,1),(2,2),(1,2)$，存储在tiles[1][1]中。

- clean_tile_at_position：先用pos.get_x()和pos.get_y()取出坐标，注意要**用int进行取整**（由于格点标号以左下角为准，故不需要对取整的数做进一步处理）。由于capacity可以是负数，故应先对tiles[x][y]直接减去capacity后，才判断是否小于0，如果小于0，则将其值置为0。
- is_tile_cleaned：直接判断self.tiles[m][n]是否为0，如果为0则返回真（被清空的）。

- get_num_cleaned_tiles：利用is_tile_cleaned数被清理的块，并返回数目。
- is_position_in_room：判断x和y是否处在self.width和self.height之间，如果是则意味着该位置在房间中，返回真。
- get_dirt_amount：返回self.tiles[m][n]

Robot基类：

- `__init__`：除了设置room、speed和capacity外，还要设置self.direction为一个$[0,360)$之间的随机浮点数（random.random() * 360），self.position通过room.get_random_position()方法获得随机初始位置
- get和set方法：对应返回或设置即可

2. 问题二：实施空房间和家具房

EmptyRoom派生类：

- get_num_tiles：返回self.height * self.width

- is_position_valid：返回self.is_position_in_room(pos)
- get_random_position：先生成坐标随机数x和y，然后返回Position(x,y)

FurnishedRoom派生类：

- is_tile_furnished：用in语句判断(m,n)是否在self.furniture_tiles中，是则返回真
- is_position_furnished：类似于is_position_in_room，对x和y取整后，调用is_tile_furnished判断
- is_position_valid：调用is_position_in_room和is_position_furnished进行判断
- get_num_tiles：用总砖数减去放家具的砖数
- get_random_position：如下，不断循环直到产生合法位置为止

```
def get_random_position(self):
    while True: # consistently generate random tiles
        x = random.random() * self.width
        y = random.random() * self.height
        pos = Position(x,y)
        if self.is_position_valid(pos):
            return pos
```

3. 问题三：标准机器人及模拟时间步

只要实施update_position_and_clean即可。先获取旧的方向及坐标，然后获取新的位置，如果位置合法则前进并清理，否则更改方向（注意不要清理当前格，也不要移动到另一个格）。代码如下。

```
def update_position_and_clean(self):
    # This is a single time step simulation!
    pos = self.get_robot_position()
    direction = self.get_robot_direction()
    new_pos = pos.get_new_position(direction,self.speed)
    if self.room.is_position_valid(new_pos):
        self.set_robot_position(new_pos)
        self.room.clean_tile_at_position(new_pos,self.capacity)
    else:
        self.set_robot_direction(random.random() * 360)
```

测试结果见实验结果部分第3节图2。

4. 问题四：实施错误机器人

只需在判断self.room.is_position_valid(new_pos)的同时，判断是否not self.gets_faulty()。如果出错，则不进行清理，并且随机旋转一个方向；否则表现得与StandardRobot相同。测试

结果见实验结果部分第3节图3。

5. 问题五：创建模拟器

`run_simulation`实施方式如下：

- 先初始化`avg_t`变量
- 然后对于每一次尝试(trial)，都新创建空房间及机器人列表
- 模拟每一个时间步，每个时间步对每个机器人进行`update_position_and_clean`操作
- 通过获取清理和总砖数，判断覆盖率是否超过预期值，如果超过则停止
- 将当前时间步数目加入`avg_t`中，最后再统一除以尝试次数

代码如下，已将源文件注释删除。其中注释的部分为可视化代码，结果见图5。

```
def run_simulation(num_robots, speed, capacity, width, height, dirt_amount,
    ↪ min_coverage, num_trials, robot_type):
    ######
    # anim = ps3_visualize.RobotVisualization(num_robots,width,height,False,0.2)
    ######
    avg_t = 0
    for i in range(num_trials):
        t = 0
        room = EmptyRoom(width,height,dirt_amount)
        robots = [] # create robot list
        for j in range(num_robots):
            robots.append(robot_type(room,speed,capacity))
        while True: # simulate each time step
            t += 1
            for j in range(num_robots):
                robots[j].update_position_and_clean()
            ######
            # anim.update(room,robots)
            ######
            if room.get_num_cleaned_tiles() >= room.get_num_tiles() * min_coverage:
                break
        avg_t += t # summary
        ######
        # anim.done()
        ######
    return avg_t / num_trials
```

6. 问题六：运行模拟器

运行结果见图6，下面的叙述在代码源文件中也阐述了。

1. 从图中可以看出，`StandardRobot`的性能明显比`FaultyRobot`要高，在清理80%的20×20的

房间时，无论采用多少只机器人，StandardRobot耗费的时间始终比FaultyRobot要少。

2. 从图中可以看出，当纵横比增加时，两种机器人的运行时间都呈上升趋势（除了第一个点有轻微下降）。因为房间非常大(300×300)，故StandardRobot花费的时间远远低于FaultyRobot，由此说明保证机器人不出错是非常重要的。

二、 代码

代码实施及注释请见附件ps3.py。

三、 实验结果

实验运行结果如下面几幅图片所示。

注意由于给出的pyc文件是Python 3.5版本的，故需要在Anaconda中新创建环境进行管理。

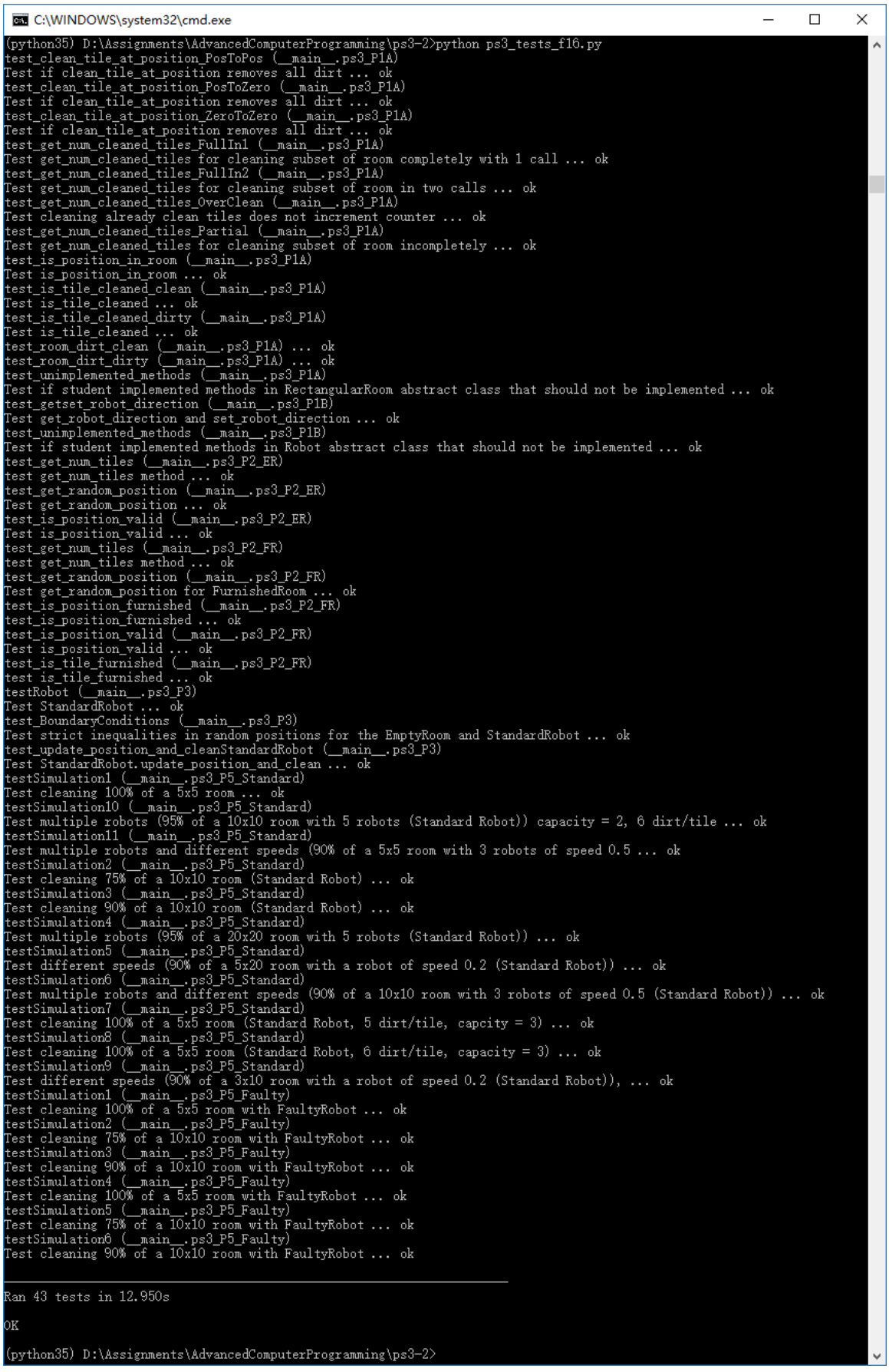

图 1: 运行ps3_tests_f16.py的结果，可以看出所有测试样例通过

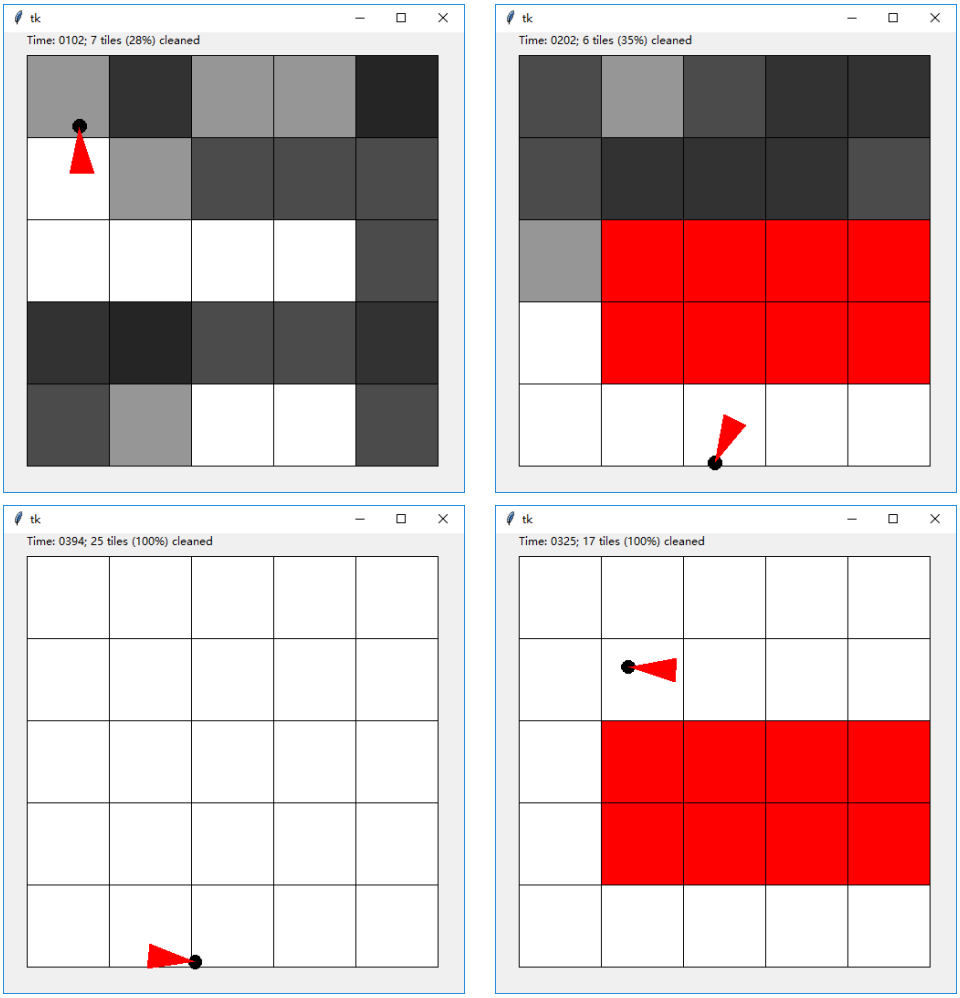

图 2: 问题三结果，左侧无家具，右侧有家具。这是运行期间截图，最终都能清理完

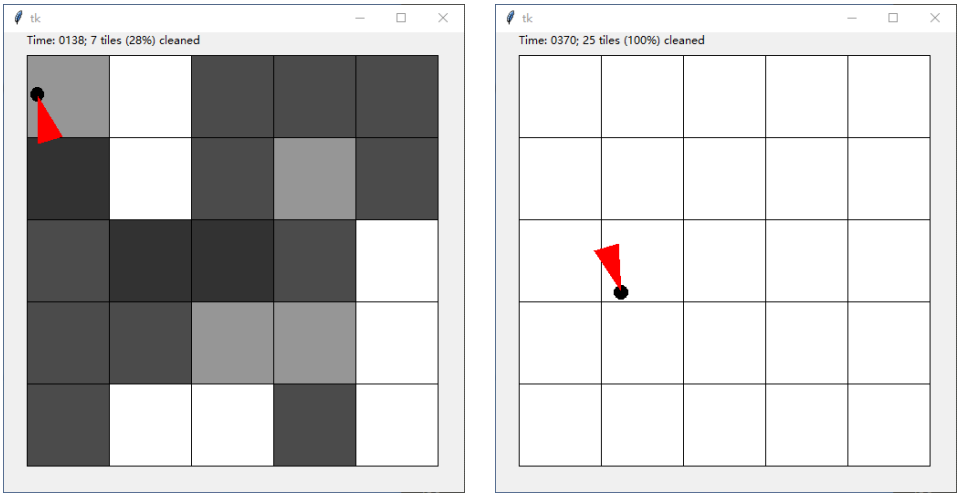

图 3: 问题四结果，虽然能够全部清扫完，但明显时间要长很多

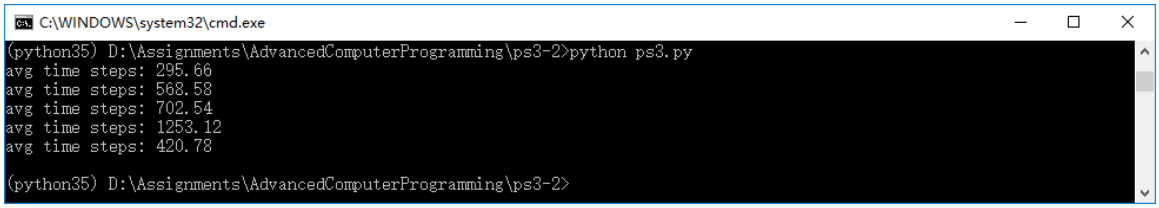

图 4: 问题五计数结果，明显看出多台机器人一起工作时清理时间最少

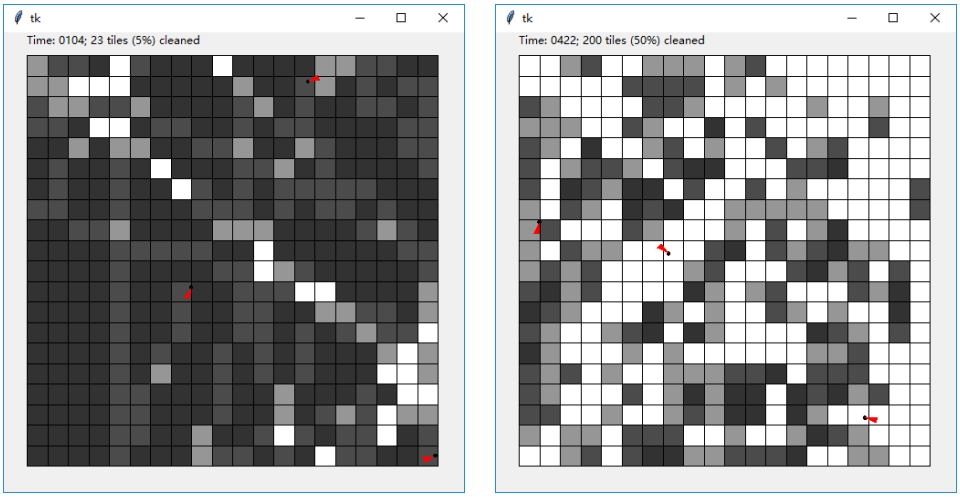

图 5: 问题五可视化结果，阈值为50%清理率

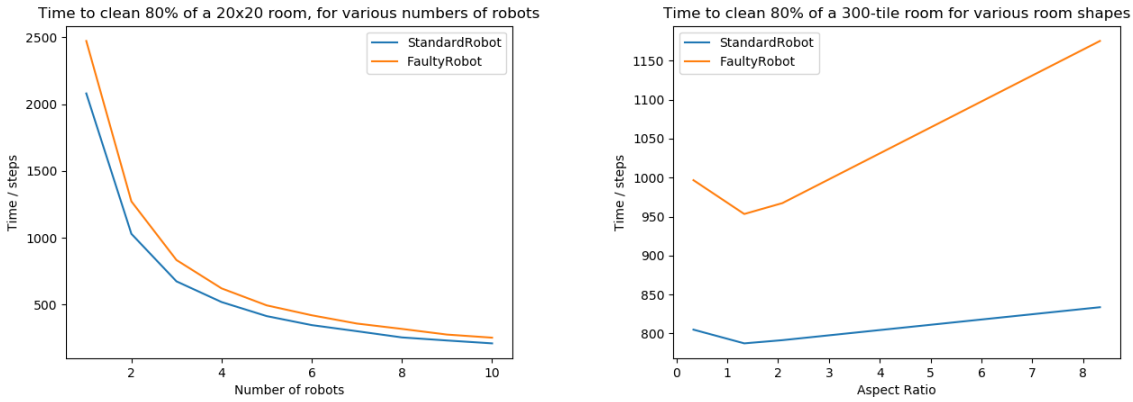

图 6: 问题六结果