



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。

```

C:\WINDOWS\system32\cmd.exe
(python35) D:\Assignments\AdvancedComputerProgramming\ps3-2>python ps3_tests_f16.py
test_clean_tile_at_position_PosToPos (__main__.ps3_P1A)
Test if clean_tile_at_position removes all dirt ... ok
test_clean_tile_at_position_PosToZero (__main__.ps3_P1A)
Test if clean_tile_at_position removes all dirt ... ok
test_clean_tile_at_position_ZeroToZero (__main__.ps3_P1A)
Test if clean_tile_at_position removes all dirt ... ok
test_get_num_cleaned_tiles_FullIn1 (__main__.ps3_P1A)
Test get_num_cleaned_tiles for cleaning subset of room completely with 1 call ... ok
test_get_num_cleaned_tiles_FullIn2 (__main__.ps3_P1A)
Test get_num_cleaned_tiles for cleaning subset of room in two calls ... ok
test_get_num_cleaned_tiles_OverClean (__main__.ps3_P1A)
Test cleaning already clean tiles does not increment counter ... ok
test_get_num_cleaned_tiles_Partial (__main__.ps3_P1A)
Test get_num_cleaned_tiles for cleaning subset of room incompletely ... ok
test_is_position_in_room (__main__.ps3_P1A)
Test is_position_in_room ... ok
test_is_tile_cleaned_clean (__main__.ps3_P1A)
Test is_tile_cleaned ... ok
test_is_tile_cleaned_dirty (__main__.ps3_P1A)
Test is_tile_cleaned ... ok
test_room_dirt_clean (__main__.ps3_P1A) ... ok
test_room_dirt_dirty (__main__.ps3_P1A) ... ok
test_unimplemented_methods (__main__.ps3_P1A)
Test if student implemented methods in RectangularRoom abstract class that should not be implemented ... ok
test_getset_robot_direction (__main__.ps3_P1B)
Test get_robot_direction and set_robot_direction ... ok
test_unimplemented_methods (__main__.ps3_P1B)
Test if student implemented methods in Robot abstract class that should not be implemented ... ok
test_get_num_tiles (__main__.ps3_P2_ER)
test_get_num_tiles method ... ok
test_get_random_position (__main__.ps3_P2_ER)
Test get_random_position ... ok
test_is_position_valid (__main__.ps3_P2_ER)
Test is_position_valid ... ok
test_get_num_tiles (__main__.ps3_P2_FR)
test_get_num_tiles method ... ok
test_get_random_position (__main__.ps3_P2_FR)
Test get_random_position for FurnishedRoom ... ok
test_is_position_furnished (__main__.ps3_P2_FR)
test_is_position_furnished ... ok
test_is_position_valid (__main__.ps3_P2_FR)
Test is_position_valid ... ok
test_is_tile_furnished (__main__.ps3_P2_FR)
test_is_tile_furnished ... ok
testRobot (__main__.ps3_P3)
Test StandardRobot ... ok
test_BoundaryConditions (__main__.ps3_P3)
Test strict inequalities in random positions for the EmptyRoom and StandardRobot ... ok
test_update_position_and_cleanStandardRobot (__main__.ps3_P3)
Test StandardRobot.update_position_and_clean ... ok
testSimulation1 (__main__.ps3_P5_Standard)
Test cleaning 100% of a 5x5 room ... ok
testSimulation10 (__main__.ps3_P5_Standard)
Test multiple robots (95% of a 10x10 room with 5 robots (Standard Robot)) capacity = 2, 6 dirt/tile ... ok
testSimulation11 (__main__.ps3_P5_Standard)
Test multiple robots and different speeds (90% of a 5x5 room with 3 robots of speed 0.5 ... ok
testSimulation2 (__main__.ps3_P5_Standard)
Test cleaning 75% of a 10x10 room (Standard Robot) ... ok
testSimulation3 (__main__.ps3_P5_Standard)
Test cleaning 90% of a 10x10 room (Standard Robot) ... ok
testSimulation4 (__main__.ps3_P5_Standard)
Test multiple robots (95% of a 20x20 room with 5 robots (Standard Robot)) ... ok
testSimulation5 (__main__.ps3_P5_Standard)
Test different speeds (90% of a 5x20 room with a robot of speed 0.2 (Standard Robot)) ... ok
testSimulation6 (__main__.ps3_P5_Standard)
Test multiple robots and different speeds (90% of a 10x10 room with 3 robots of speed 0.5 (Standard Robot)) ... ok
testSimulation7 (__main__.ps3_P5_Standard)
Test cleaning 100% of a 5x5 room (Standard Robot, 5 dirt/tile, capacity = 3) ... ok
testSimulation8 (__main__.ps3_P5_Standard)
Test cleaning 100% of a 5x5 room (Standard Robot, 6 dirt/tile, capacity = 3) ... ok
testSimulation9 (__main__.ps3_P5_Standard)
Test different speeds (90% of a 3x10 room with a robot of speed 0.2 (Standard Robot)), ... ok
testSimulation1 (__main__.ps3_P5_Faulty)
Test cleaning 100% of a 5x5 room with FaultyRobot ... ok
testSimulation2 (__main__.ps3_P5_Faulty)
Test cleaning 75% of a 10x10 room with FaultyRobot ... ok
testSimulation3 (__main__.ps3_P5_Faulty)
Test cleaning 90% of a 10x10 room with FaultyRobot ... ok
testSimulation4 (__main__.ps3_P5_Faulty)
Test cleaning 100% of a 5x5 room with FaultyRobot ... ok
testSimulation5 (__main__.ps3_P5_Faulty)
Test cleaning 75% of a 10x10 room with FaultyRobot ... ok
testSimulation6 (__main__.ps3_P5_Faulty)
Test cleaning 90% of a 10x10 room with FaultyRobot ... ok

Ran 43 tests in 12.950s

OK

(python35) D:\Assignments\AdvancedComputerProgramming\ps3-2>

```

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

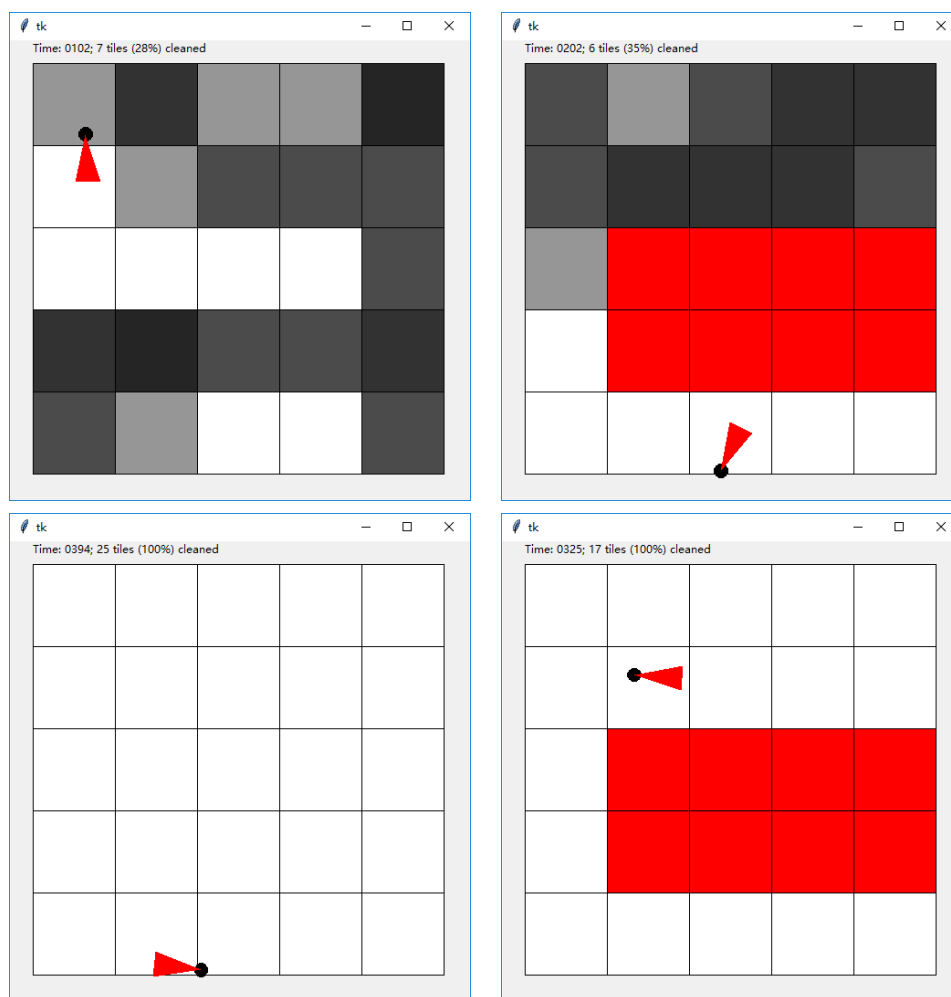


图 2: 问题三结果，左侧无家具，右侧有家具。这是运行期间截图，最终都能清理完

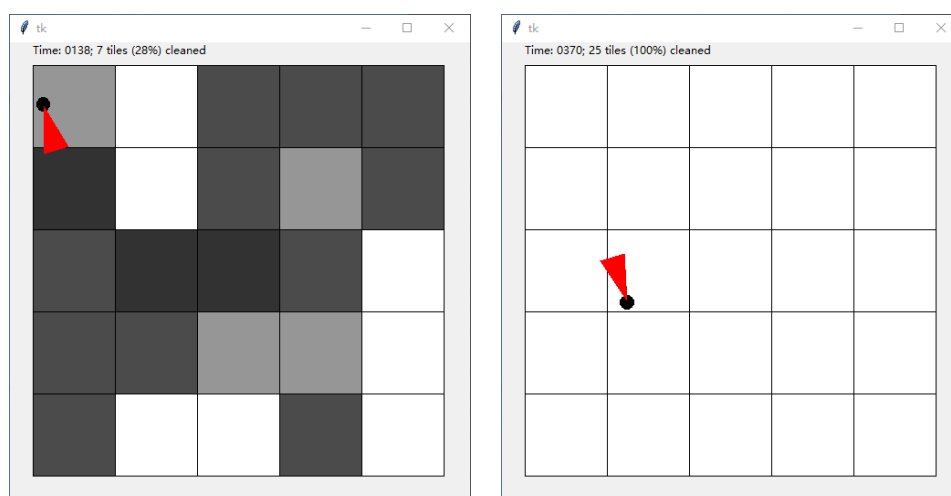


图 3: 问题四结果，虽然能够全部清扫完，但明显时间要长很多

```

C:\WINDOWS\system32\cmd.exe
(python35) D:\Assignments\AdvancedComputerProgramming\ps3-2>python ps3.py
avg time steps: 295.66
avg time steps: 568.58
avg time steps: 702.54
avg time steps: 1253.12
avg time steps: 420.78
(python35) D:\Assignments\AdvancedComputerProgramming\ps3-2>

```

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

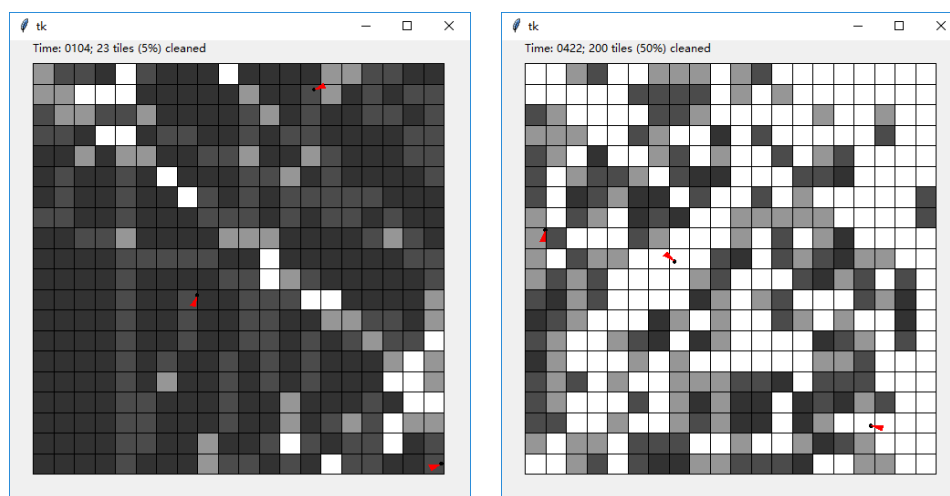


图 5: 问题五可视化结果, 阈值为50%清理率

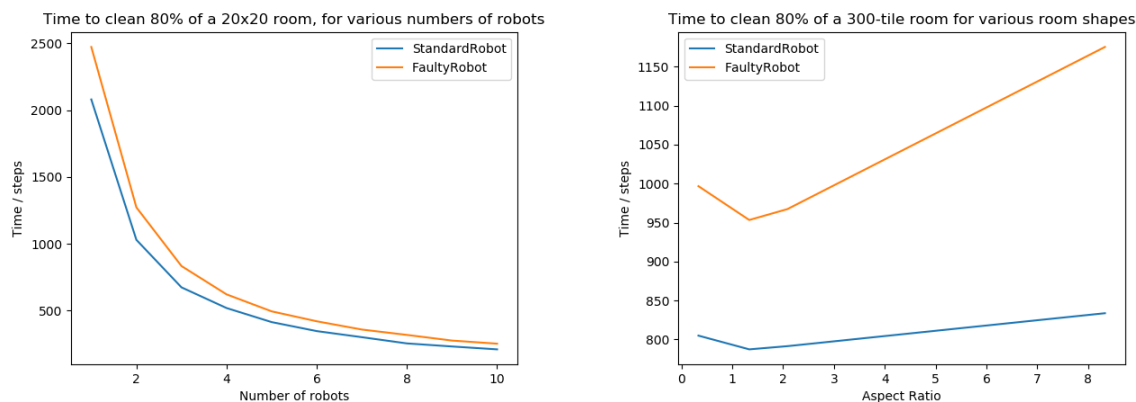


图 6: 问题六结果