Laboratory: The Robots Game

We build a version of the robots game, using ASCII art. You can learn more about the robots game on the web https://en.wikipedia.org/wiki/Chase_(video_game). The game was originally called chase, but I met it as a game on Unix systems, where it was also used to help people learn the commands for cursor movement in the VI editor.

The game is placed on a two-dimensional grid and is turn-based. The player controls an avatar that moves up and down, left and right and also diagonally to a neighboring position. A large number of robots chase the avatar, always moving towards it. Luckily for the avatar, robots do not pay any attention to anything else. If they collide with each other they leave a heap of metal behind. Similarly, if a robot runs into a heap, the robot is destroyed (and presumably, the heap gets bigger). The skilled player will navigate to allow robots to run into each other or into heaps until no more robots are left. Sometimes, the player uses a teleport that moves the avatar to a random location not previously occupied by a robot or a heap. Since the robots do not move immediately after a teleport, the teleport always leave the player in a safe spot. Some variants of the game therefore limit the number of teleports per round.

After the player has caused all the robots to die, a new round ensues. Often, the size of the playing field and or or the number of robots increases. One can maintain a score during the game that depends on the number of robots slain and the number of teleports.

Task 1: Create (or copy) the class Geometry in the module geometry.py. As before, the geometry class has two fields for height and width and a construction as well as a string dunder.

The following tasks are in a module called model.py.

Task 2: Create a function doubles that returns all the elements in a list that are doubles. The result does not contain double elements themselves. For example, `doubles([1,2,3,4,5,4,3,4,5,4])` returns `[3,4,5].`

Task 3: Create a class Location with fields x and y (the coordinates of the position) and a geometry. Because updates to a location need to take cognizance of the geometry, I thought it easier to include it. The string-dunder should just return the x and y coordinates inside a pair of parentheses.

Task 4: In the same file, create another class Direction. A direction has a code, one of 'N', 'S', 'W', 'E', 'NE', 'NW', 'SE', 'SW' with obvious meaning and a non-direction '0'. The construction dunder throws a ValueError exception if it is given a code other than those.

At this point, you classes should have implemented

```
class Location:
    def __init__(self, x, y, geometry):
        self.geometry = geometry
        self.x = x
        self.y = y
    def __eq__(self, other):

    def __str__(self):

    def generate_random_locality(geometry):
        return Location( … )

    def update(self, direction):


class Direction:
    codes = {'N', 'W', 'E', 'S', 'NW', 'NE', 'SW', 'SE', '0'}
    def __init__(self, code):
        if code in Direction.codes:
            self.code = code
        else:
            raise ValueError
    def __str__(self):
        return "Direction {}".format(self.code)
```

Task 5: We have to update locations given a direction. This update should not let the location wander of the playing field as defined by the location field geometry.

```
def update(self, direction):
        if direction.code == 'N':
            self.y = min(self.geometry.height-1, self.y+1)
```

Task 6: Since robots move towards an avatar, we need to calculate the direction of their move. We do so by comparing a current location with the location of the target.

```
def move_towards(mover, aim):
        """determines a robot location mover
           towards an avatar location aim
           robots now move diagonally
        """
        if mover.x < aim.x and mover.y<aim.y:
            return Direction('NE')
```

At this point, you should be able to run the following test code:

```
def test():
```

```
mygeo = geo.Geometry(height=20, width=30)
print("location tests")
l1 = Location(5, 3, mygeo)
l2 = Location(5, 3, mygeo)
l3 = Location(2, 4, mygeo)
print(l1, l2, l3)
print("Should be (5 3) (5 3) (2 4)")
print(l1 is l1, l1 is l2, l1 == l2, l1 is l3, l1 == l3)
print("Should be True False True False False")
print("generate random locality:")
for _ in range(3):
    print(Location.generate_random_locality(mygeo))
while True:
    try:
        direction = Direction(input('where to? '))
    except ValueError:
        break
    l1.update(direction)
    print(l1)
```