Laboratory 3: The Robots Game — Continued

Task 1: Create a class Avatar in model.py

```
class Avatar:
    def __init__(self, location):
        self.location = location
    def __str__(self):
        return "Avatar at {}".format(self.location)
    def move(self, direction):
        self.location.update(direction)
```

Task 2: Create a class Heap in model.py

```
class Heap:
    def __init__(self, location):
        self.location = location
    def __str__(self):
        return "Heap {}".format(self.location)
```

Task 3: Create a class Robot.

```
class Robot:
    def __init__(self, location):
        self.location = location
    def __str__(self):
        return "Robot at {}".format(self.location)
    def move(self, avatar):
        …
```

Task 4: Create a class Model. The interesting part is programming the update. You need to fill in the lacunae based on the comments. You can use list comprehension to create the list of the locations of robots and heaps. The function returns a code of 1 if there are no more robots left and a code of -1 if the avatar is on the same location as a robot or a heap and is therefore dead.

```
class Model:
    def __init__(self, geometry, nrrobots):
        self.geometry = geometry
        self.heaps = []
        self.avatar = Avatar(Location.generate_random_locality(geometry))
        self.robots = []
        while len(self.robots)<nrrobots:
            robot = Robot(Location.generate_random_locality(geometry))
            if (robot.location not in [rob.location for rob in self.robots] and
            robot.location != self.avatar.location):
                self.robots.append(robot)

    def __str__(self):
        return "{}\n{}\n{}".format(self.avatar,
                                   [str(robot) for robot in self.robots],
                                   [str(heap) for heap in self.heaps])

    def update(self, avatar_direction):
```

```
        if avatar_direction=='T':
            while True:
                self.avatar.location =
                     Location.generate_random_locality(self.geometry)
                if (self.avatar.location not in [r.location for r in self.robots] and
                    self.avatar.location not in [h.location for h in self.heaps]):
                    break
            return 0
        direction = Direction(avatar_direction)
        self.avatar.move(direction)
        for r in self.robots:
            r.move(self.avatar)
```

Task 5: Create a new file called control.py. In this file create a function run_level. You also need to adjust the file view.py from a previous lab. In this one, you need to have a function draw_screen which draws the model to the display.

```
def run_level(my_geometry, nr_robots):
    my_model = model.Model(my_geometry, nr_robots)
    result = 0
    while True:
        v = view.View(my_geometry)
        v.clear_screen()
        v.draw_screen(my_model)
        print(my_model.avatar.location, len(my_model.robots), "robots left")
        if result == -1:
            print('you lost')
            return -1
        if len(my_model.robots)==0:
            print('you won')
            return 1
        move = input('your move')
        try:
            result = my_model.update(move)
        except ValueError:
            print("I could not understand your command")
            continue
```

Task 6: Use this function above to create a game. Notice that we still use return values to indicate the outcome of a level. You should increment the number of robots and the size of the geometry every time you have a new level.