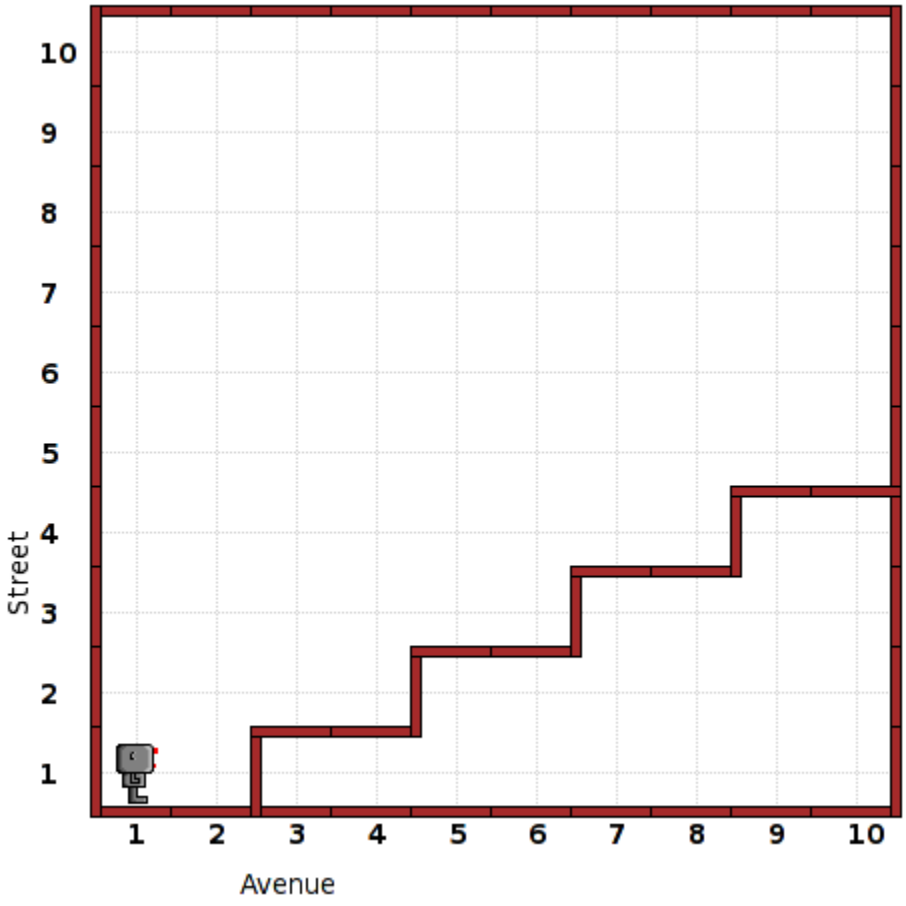
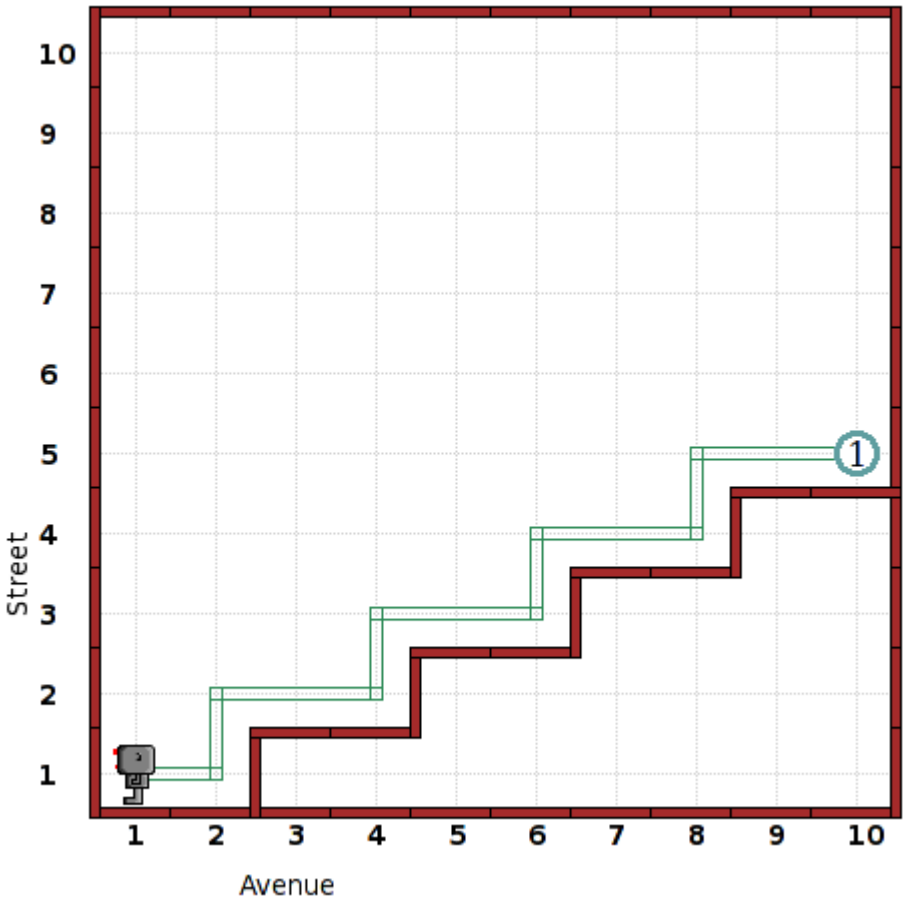
Newspaper IA1 and IB2a (fixed), or IA1, IB1a, IB1b, IB3, IIA2, IIIA1, IIIA2 (variable) adapted from RUR-PLE lessons

The task is to climb the stairs and leave a newspaper (beeper) at the top, then descend the stairs to leave. A fixed stair pattern allows the use of for loops. Varied stair patterns would necessitate the use of while loops and a variable to keep track of your position. It could be used as a program writing or prediction task.

Before:



After:



Fixed:

```
def turn_around():
    for i in range(2):
        turn_left()
for i in range(4):
    move()
    turn_left()
    move()
    turn_right()
    move()
move()
drop_beeper()
turn_around()
move()
for i in range(4):
    move()
    turn_left()
    move()
    turn_right()
    move()
turn_off()
```

Variable:

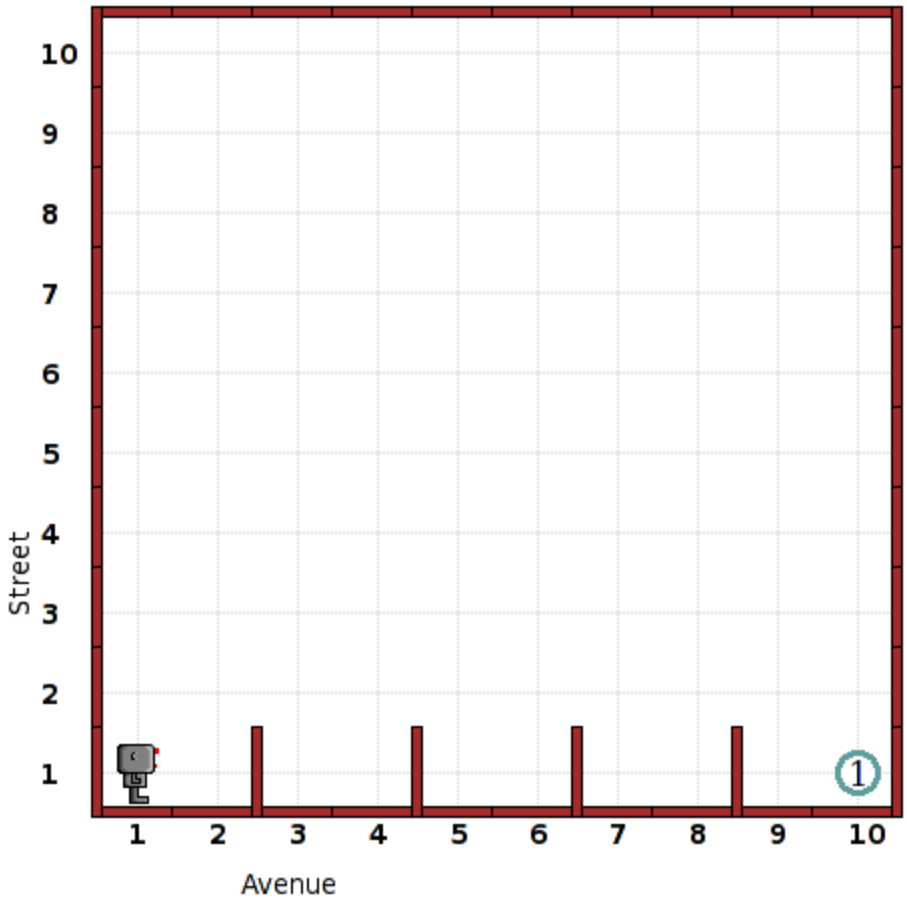
```
def turn_around():
    for i in range(2):
        turn_left()
pos = 1
```

```
while True:
    while front_is_clear():
        move()
        pos += 1
    if pos == 10:
        break
    turn_left()
    while not right_is_clear():
        move()
    turn_right()
drop_beeper()
turn_around()
while True:
    while not left_is_clear() and pos != 1:
        move()
        pos -= 1
    if pos == 1:
        break
    turn_left()
    while front_is_clear():
        move()
    turn_right()
turn_off()
```

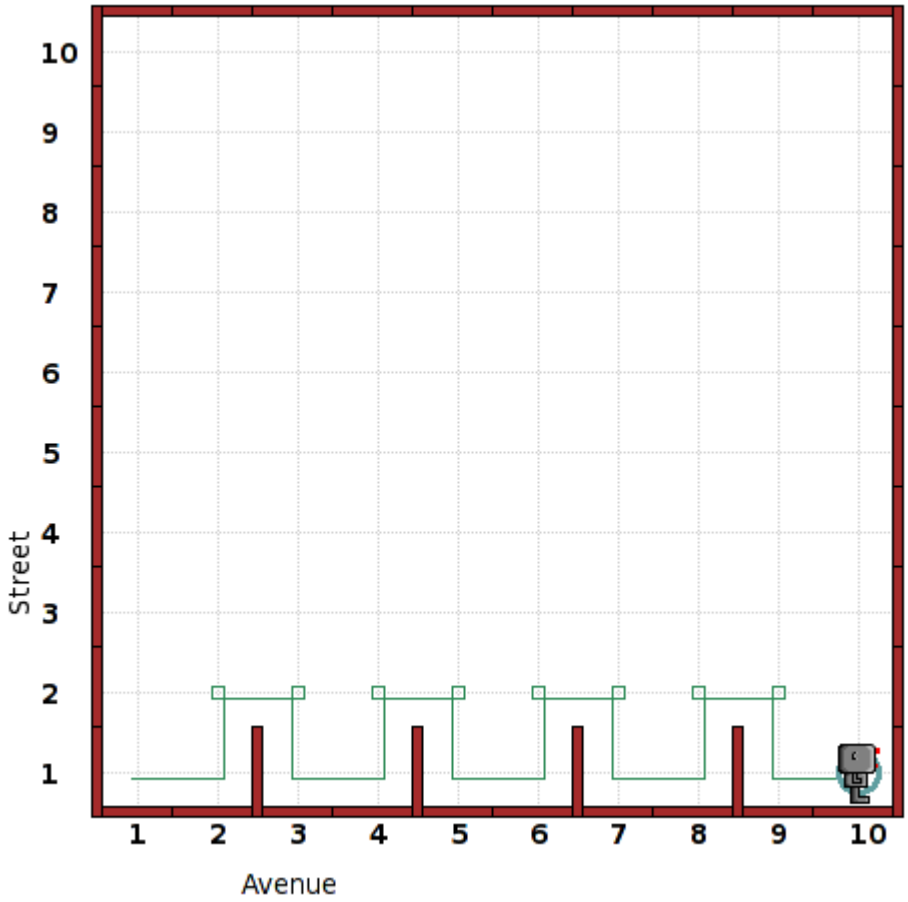
Hurdles IA1, IB1a, IB1b, IB3, IIIA2
adapted from RUR-PLE lessons

The task is to traverse the hurdles until the finish line (a beeper) is reached. A fixed pattern and beeper location would allow the use of for loops. Varying the spacing and height of the hurdles and location of the beeper necessitates the use of while loops. It could be used as a program writing or prediction task.

Before:



After:

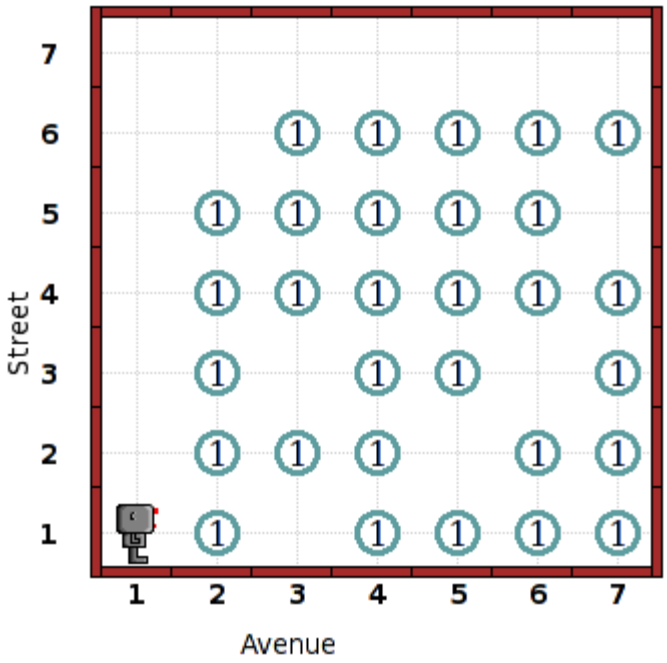



```
def turn_around():
    for i in range(2):
        turn_left()
def walk_to_wall():
    while front_is_clear() and not on_beeper():
        move()
while True:
    walk_to_wall()
    if on_beeper():
        break
    turn_left()
    while not right_is_clear():
        move()
    turn_right()
    move()
    turn_right()
    walk_to_wall()
    turn_left()
turn_off()
```

Harvest IA1, IA2, IB2a, IB3
adapted from RUR-PLE lessons

The task is to collect all of the beepers. The 6x6 grid is fixed, allowing the use of for loops, but the locations of beepers within the grid can vary, requiring an if statement. It could be used as a program writing task.

Before:

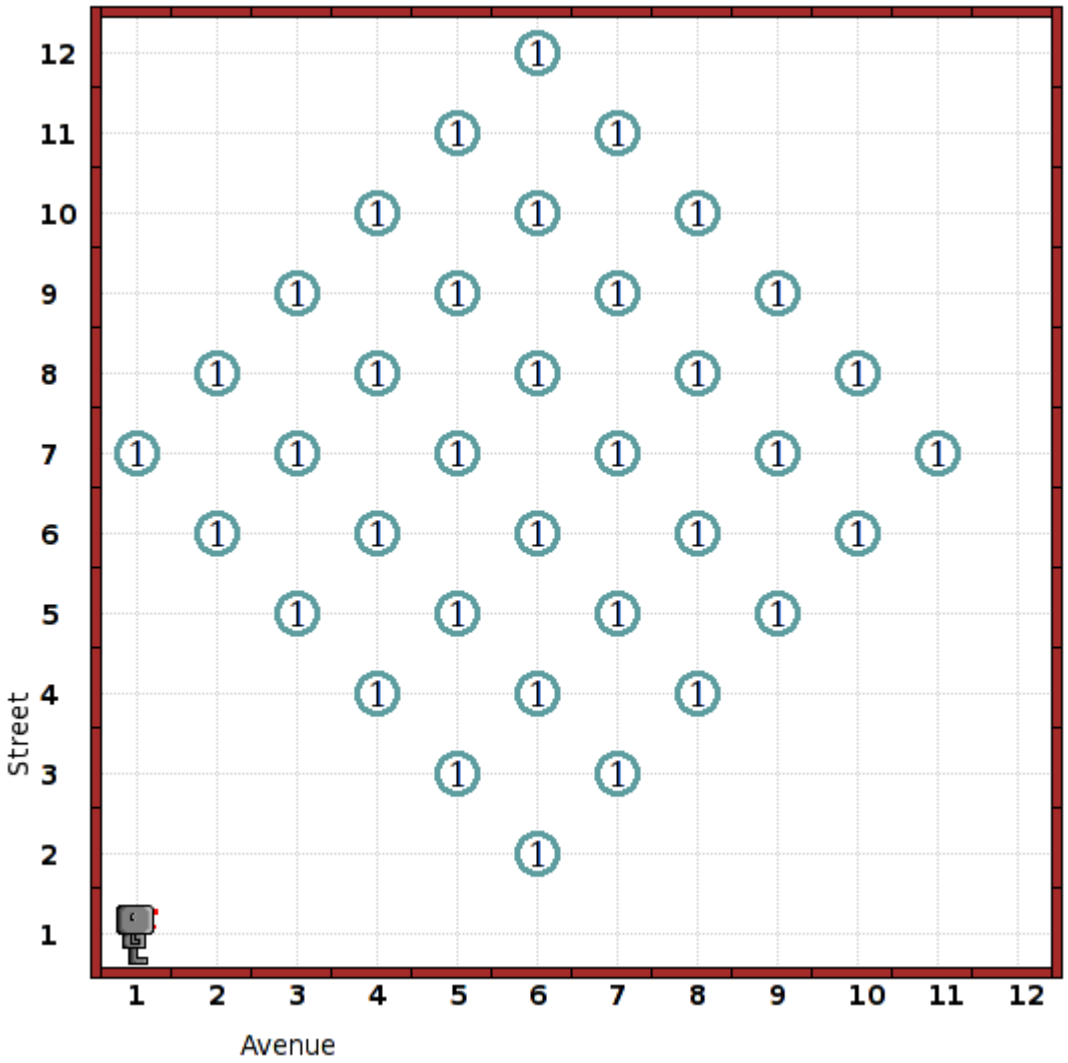



```
def turn_around():
    for i in range(2):
        turn_left()
move()
for i in range(6):
    for i in range(5):
        if on_beeper():
            grab_beeper()
        move()
    if on_beeper():
        grab_beeper()
    if front_is_clear():
        turn_right()
        move()
        turn_right()
    else:
        turn_left()
        move()
        turn_left()
turn_off()
```

Harvest 2 IB2a, IB3
adapted from RUR-PLE lessons

The task is to collect all of the beepers, moving diagonally. The fixed grid allows the use of for loops. It could be used as a program writing or prediction task.

Before:



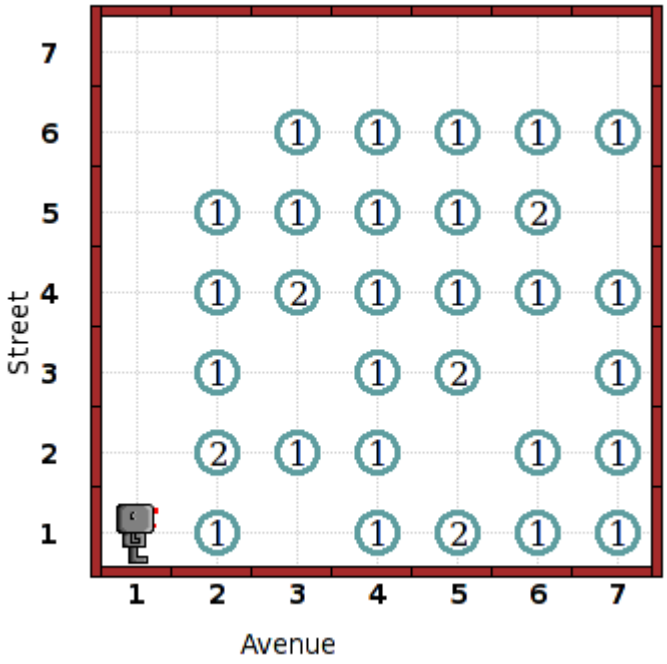

```
def turn_around():
    for i in range(2):
        turn_left()
def move_diagonal():
    turn_right()
    move()
    turn_left()
    move()
def harvest_row():
    for i in range(5):
        grab_beeper()
        move_diagonal()
    grab_beeper()
def harvest_two_rows():
    harvest_row()
```

```
    move()
    turn_left()
    move()
    turn_left()
    harvest_row()
for i in range(5):
    move()
turn_left()
move()
for i in range(2):
    harvest_two_rows()
    turn_right()
    move()
    turn_right()
    move()
harvest_two_rows()
turn_off()
```

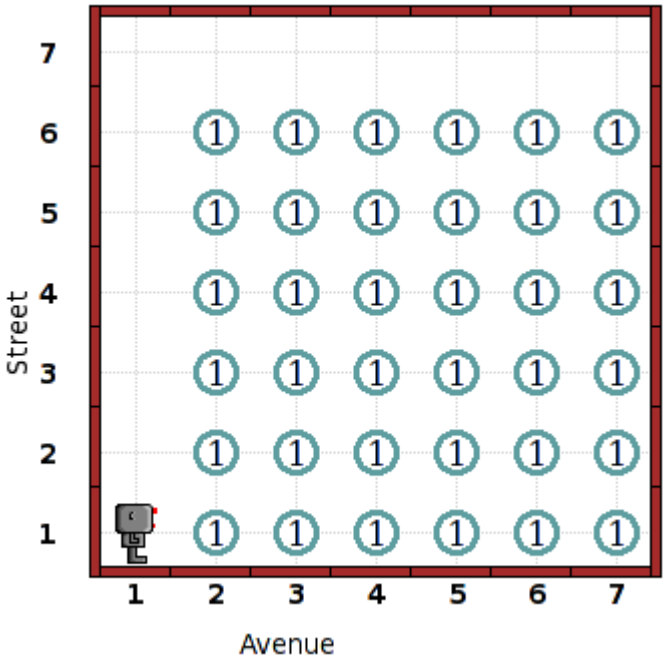
Harvest 3 IA1, IA2, IA3, IB2a, IB3, IIIA2
adapted from RUR-PLE lessons

The task is to even out the garden. In some spots, there is no plant, so one needs to be planted. In other spots there is a plant and a weed, so the weed needs to be picked. Since Reeborg can't detect how many beepers are in a location, locations that are correct need to be picked and replanted. The grid is fixed, allowing the use of for loops, but the number of beepers on each spot varies, requiring an if statement. It could be used as a program writing task.

Before:



After:

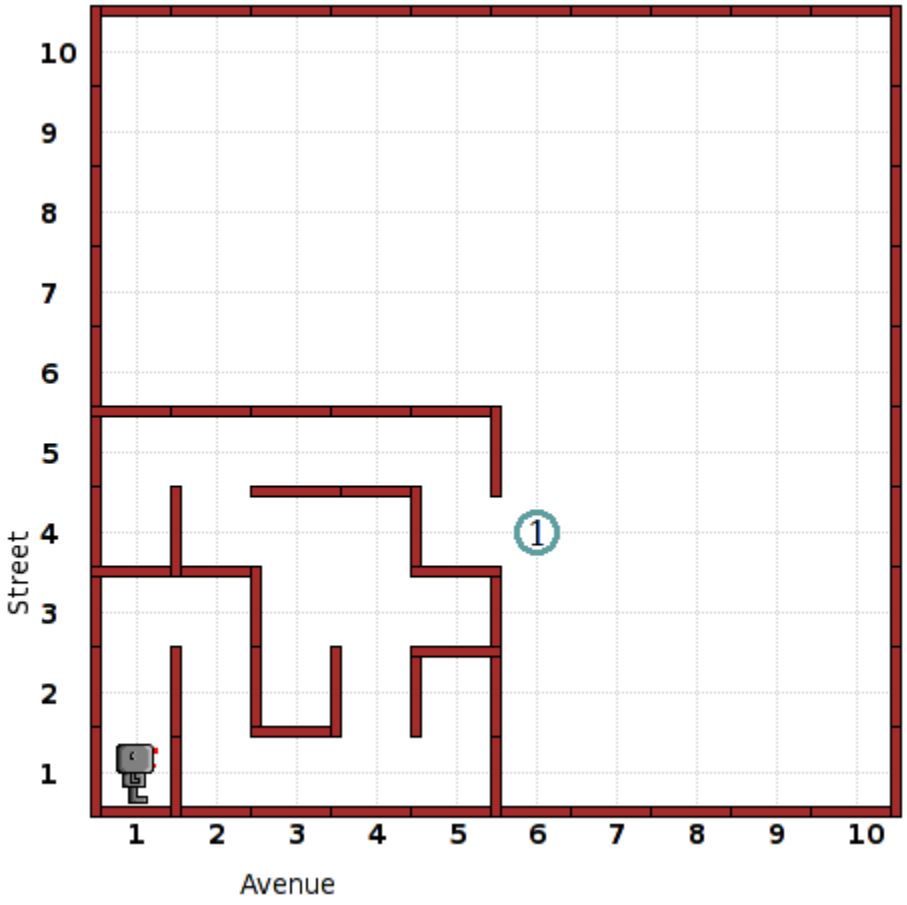


```
def turn_around():
    for i in range(2):
        turn_left()
def garden_spot():
    if not on_beeper():
        drop_beeper()
    else:
        grab_beeper()
        if not on_beeper():
            drop_beeper()
def garden_row():
    for i in range(5):
        garden_spot()
        move()
    garden_spot()
def move_to_next_row():
    if not front_is_clear():
        turn_left()
        move()
        turn_left()
    else:
        turn_right()
        move()
        turn_right()
move()
for i in range(5):
    garden_row()
    move_to_next_row()
garden_row()
move()
turn_left()
for i in range(5):
    move()
turn_left()
turn_off()
```

Maze IA2, IB1a, IIIA2
adapted from RUR-PLE lessons

Obviously the task is to exit the maze and find the beeper. The solution strategy is to follow either the left or right wall. The maze layout varies, necessitating the use of a while loop. It could be used as a program writing or prediction task.

Before:



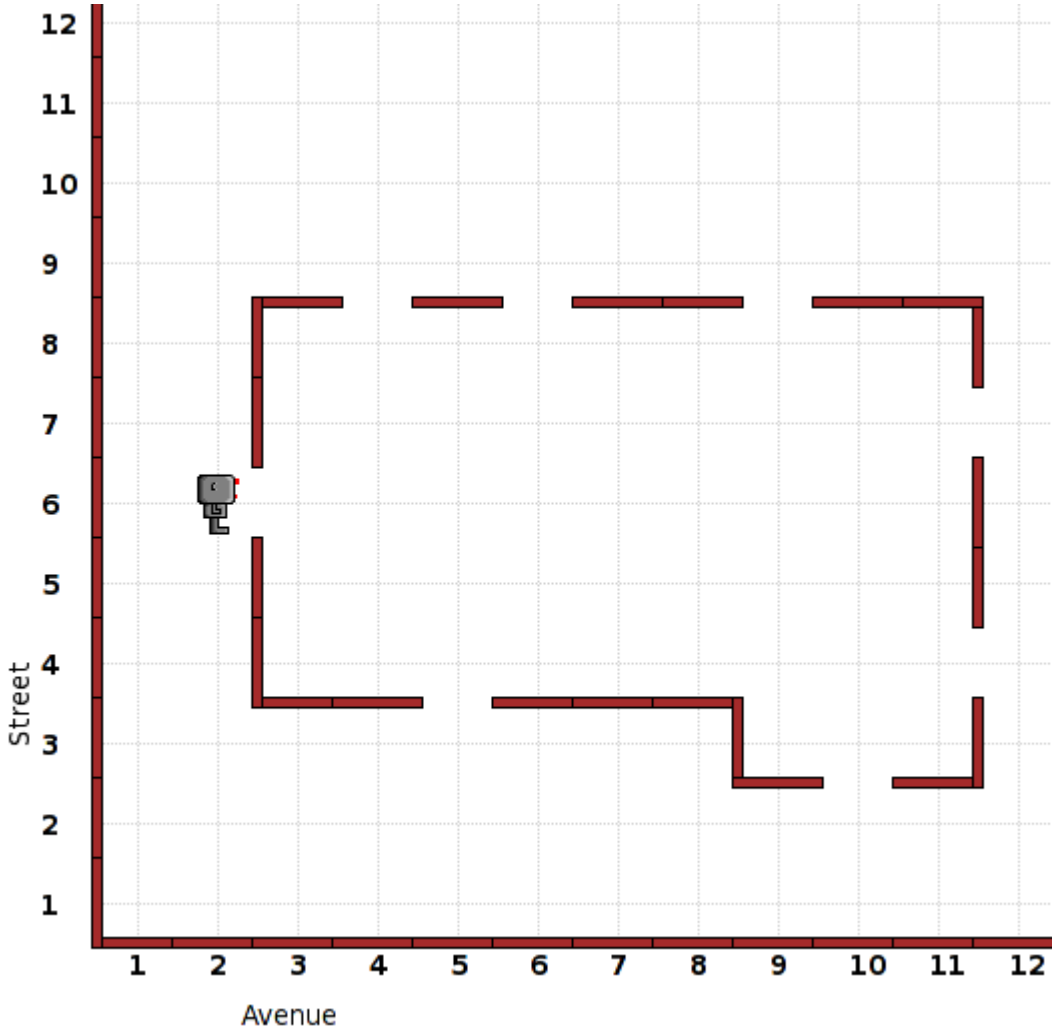
After:


```
def turn_around():
    for i in range(2):
        turn_left()
def traverse_wall():
    while front_is_clear() and not left_is_clear():
        move()
turn_left()
while not on_beeper():
    traverse_wall()
    if left_is_clear() and not on_beeper():
        turn_left()
        move()
    elif not right_is_clear():
        turn_around()
    else:
        turn_right()
turn_off()
```

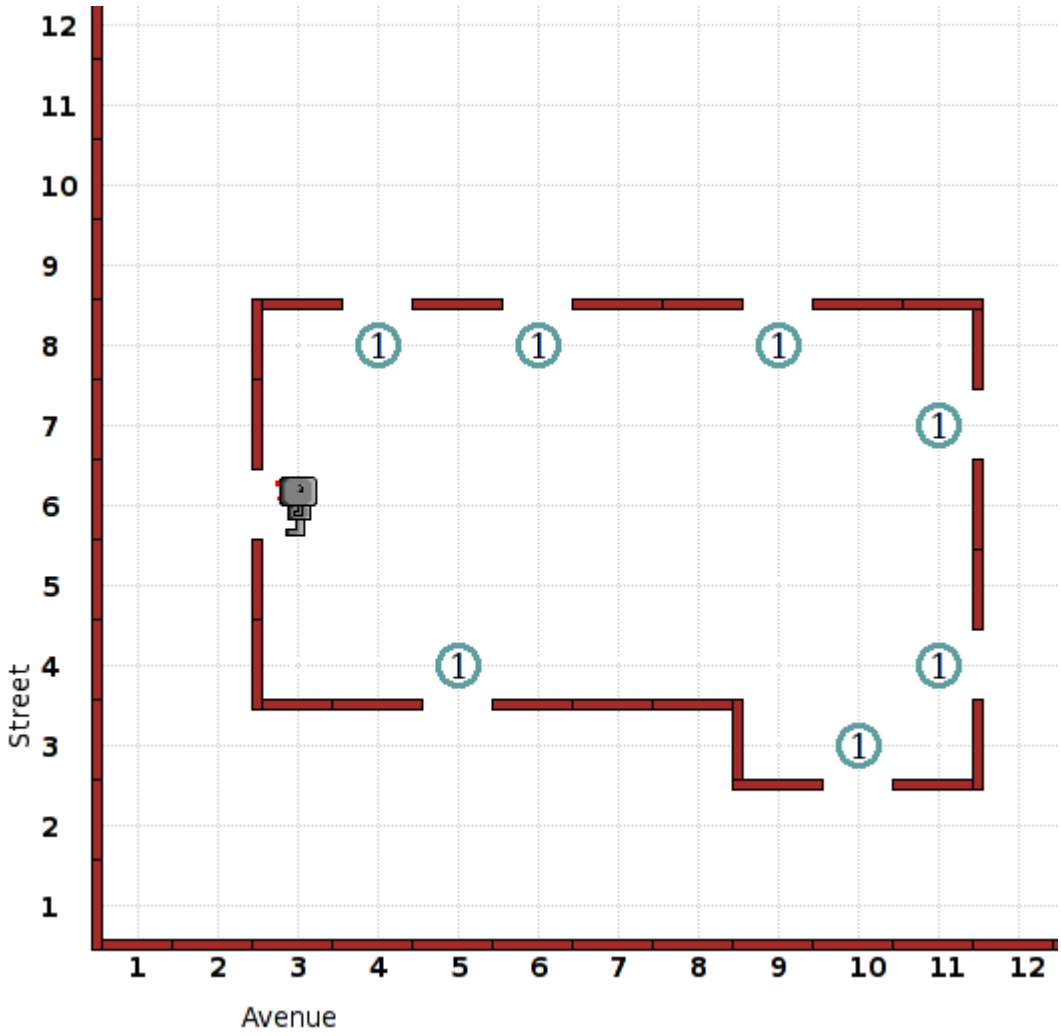
Rain IA1, IA2, IA3, IB1a, IIIA2
adapted from RUR-PLE lessons

The task is to enter the house and close all of the windows (by placing beepers in front of them) because it is raining. When the windows are closed, Reeborg will stand inside the house looking out the (open) front door, waiting for the rain to stop. The shape of the house varies. If the house is rectangular, it is a simpler task, because outside (convex) corners are tricky to deal with. It could be used as a program writing or prediction task.

Before:



After:



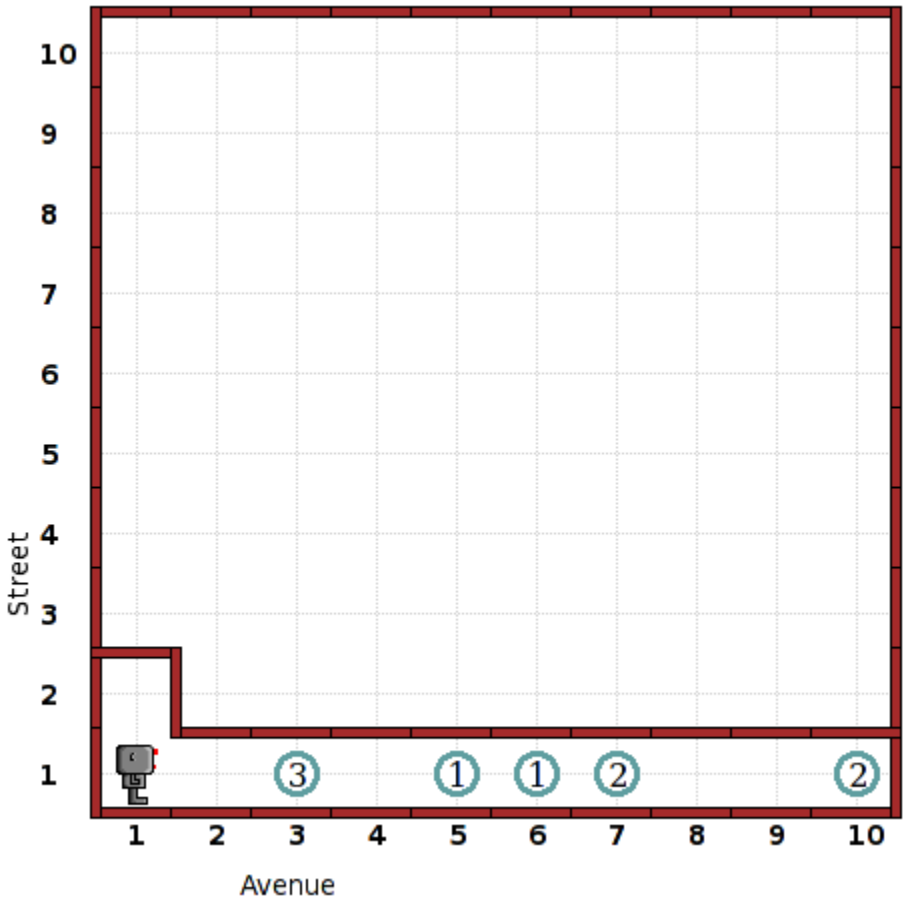

```
def turn_around():
    for i in range(2):
        turn_left()
def traverse_wall():
    while front_is_clear() and not left_is_clear():
        move()
move()
turn_left()
while not on_beeper():
    if left_is_clear():
        drop_beeper()
        if front_is_clear():
            move()
            if left_is_clear():
                turn_around()
                move()
                grab_beeper()
                turn_right()
                move()
        else:
            turn_right()
    elif not right_is_clear():
        turn_around()
    else:
```

```
        turn_right()
    traverse_wall()
grab_beeper()
turn_left()
turn_off()
```

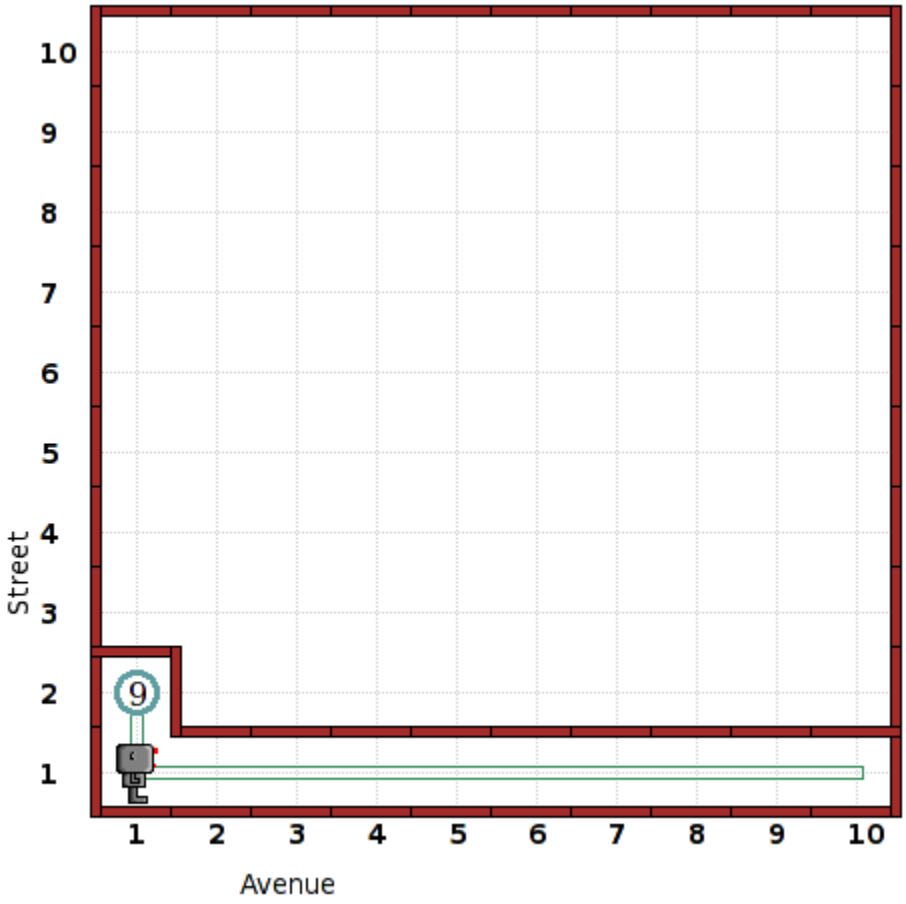
Trash IB1a, IB3
adapted from RUR-PLE lessons

The task is to collect all of the trash (beepers) and place them in the garbage can (above Reeborg's starting location), returning Reeborg to its starting position at the end. The locations and amount of beepers varies. As shown here, all of the trash is in one row. It requires a loop to pick up multiple beepers. The width of the driveway could also be varied to make the task more complicated. It could be used as a program writing or prediction task.

Before:



After:

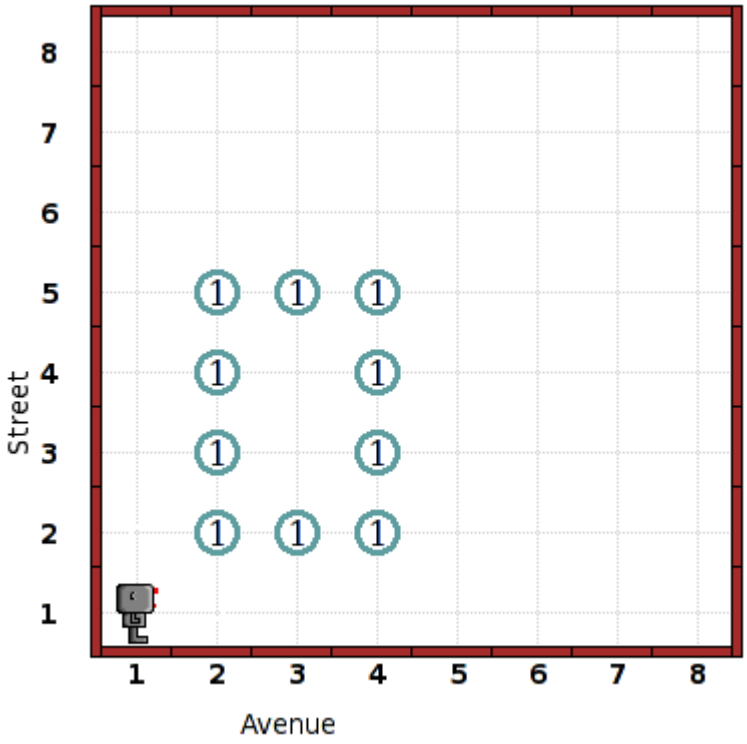



```
def turn_around():
    for i in range(2):
        turn_left()
while front_is_clear():
    move()
    while on_beeper():
        grab_beeper()
turn_around()
while front_is_clear():
    move()
turn_right()
move()
while carries_beepers():
    drop_beeper()
turn_around()
move()
turn_left()
turn_off()
```

Rectangle IB2a, IC, IIB1

One of many possible drawing tasks. It requires a subroutine that takes arguments to vary the length and width of the rectangle. A simpler version of this task would just draw a square. It could be used as a program writing or prediction task.

After:

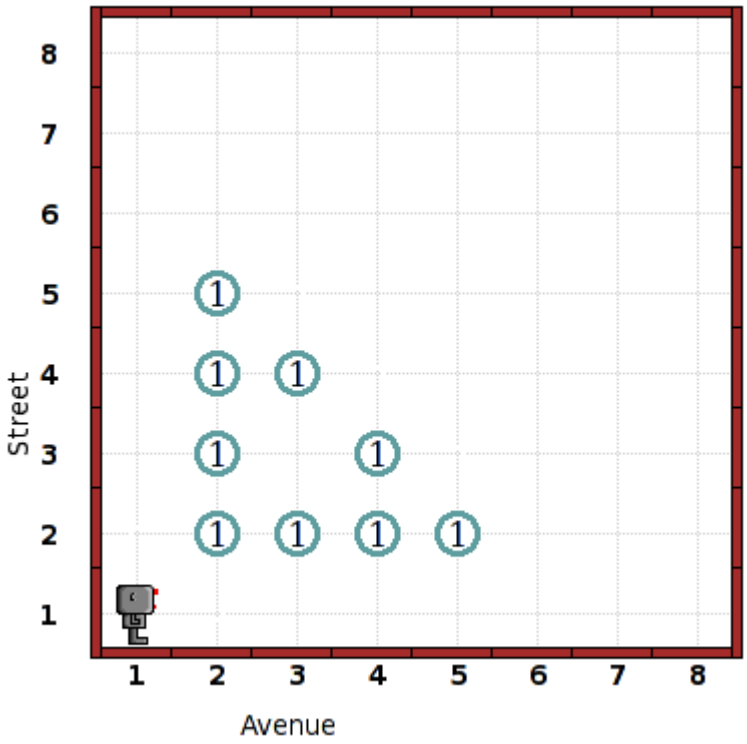



```
def turn_around():
    for i in range(2):
        turn_left()
def line(size):
    for i in range(size):
        drop_beeper()
        move()
def rectangle(length,width):
    size=[width,length]
    move()
    turn_left()
    move()
    turn_right()
    for i in range(4):
        line(size[i%2]-1)
        turn_left()
    turn_around()
    move()
    turn_left()
    move()
    turn_left()
def square(size):
    rectangle(size,size)
rectangle(4,3)
turn_off()
```

Triangle IB2a, IC, IIB1

Another drawing task where Reeborg would trace a triangle, dropping beepers along the way, and moving diagonally along the hypotenuse. It involves a subroutine with an argument to vary the number of beepers along each side of the triangle. It could be used as a program writing or prediction task.

After:



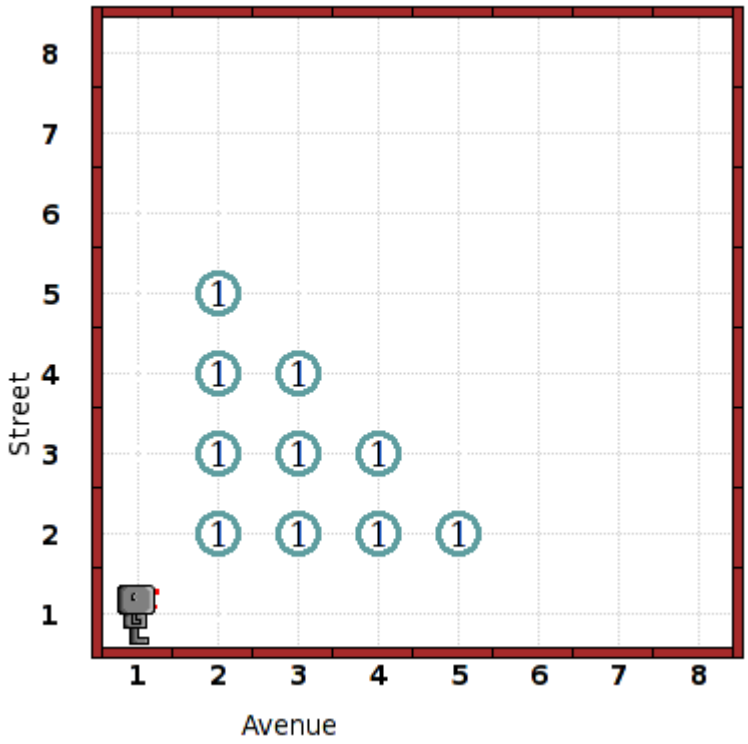

```
def turn_around():
    for i in range(2):
        turn_left()
def line(size):
    for i in range(size):
        drop_beeper()
        move()
def diagonal_line(size):
    for i in range(size):
        drop_beeper()
        move()
        turn_left()
        move()
        turn_right()
def triangle(size):
    move()
    turn_left()
    move()
    turn_right()
    line(size-1)
    turn_left()
    diagonal_line(size-1)
```

```
    turn_around()
    line(size-1)
    turn_right()
    move()
    turn_left()
    move()
    turn_left()
triangle(4)
turn_off()
```

Filled Triangle IA1, IA2, IB2a, IB2c, IC, IIB1, IIIA1, IIIB1, IIIB2, IIIB3

Drawing a filled triangle is a simple nested loop task if the robot returns to the left side before drawing every row. Drawing in a slightly more efficient zigzag fashion is slightly more complicated. The drawing is done in a subroutine with an argument to vary the size of the triangle. It could used as a program writing or prediction task.

After:



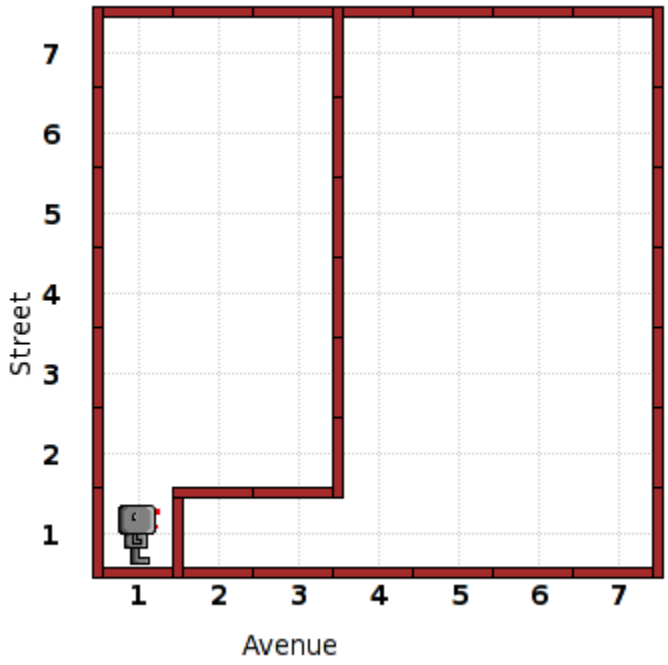

```
def turn_around():
    for i in range(2):
        turn_left()
def line(size):
    for i in range(size):
        move()
        drop_beeper()
def filled_triangle(size):
    for i in range(size+1):
        move()
    for i in range(size,0,-1):
```

```
        if (size-i)%2 == 0:
            turn_left()
            move()
            turn_left()
        else:
            move()
            turn_right()
            move()
            turn_right()
        line(i)
    if size%2 == 0:
        turn_around()
    move()
    turn_left()
    for i in range(size):
        move()
    turn_left()
filled_triangle(4)
turn_off()
```

Perimeter IA2, IB1a, IIA1, IIA2, IIIA2

The task is to measure the perimeter and print it at the end. The layout of the walls can vary. Getting it to work in all cases is trickier than it sounds. The program requires the use of a variable. As a prediction task, it would require the student to be able to enter a print statement as part of a prediction. It could also be used as a program writing task.

Before:



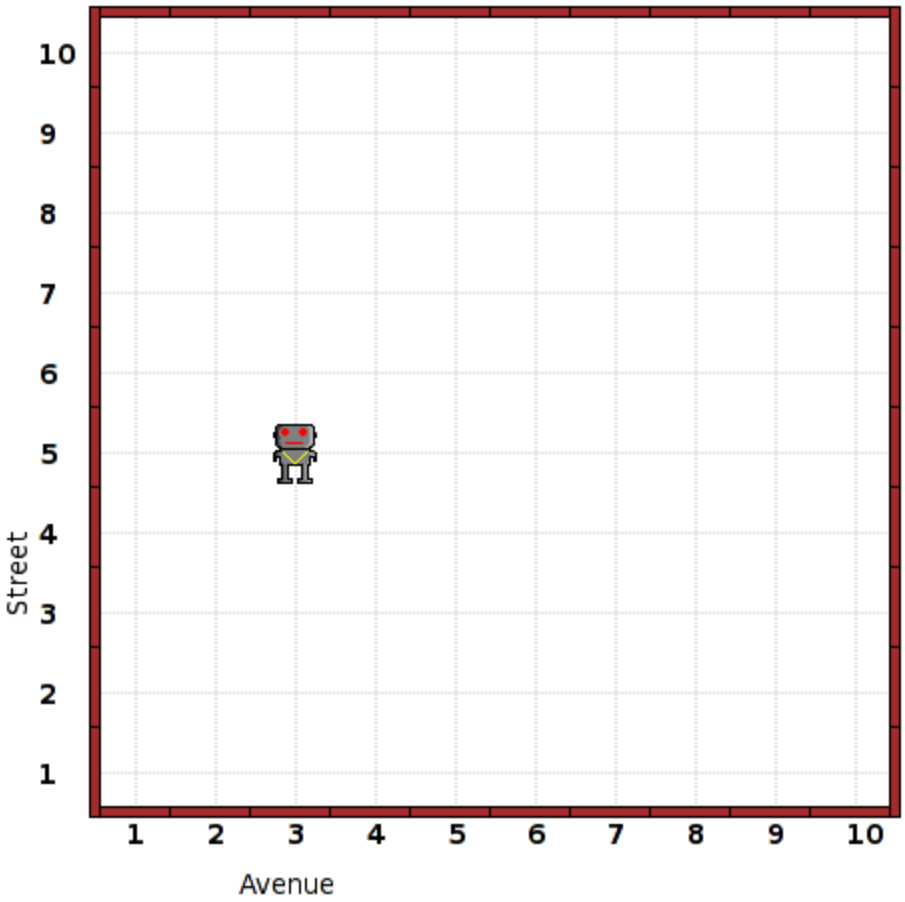

```
def turn_around():
    for i in range(2):
        turn_left()
def traverse_wall():
    i=0
    while front_is_clear() and not left_is_clear():
        move()
        i+=1
    return i
drop_beeper()
p=1
if left_is_clear():
    turn_left()
    move()
elif front_is_clear():
    move()
else:
    turn_left()
    p=4
while not on_beeper():
    p+=traverse_wall()
    if left_is_clear():
        turn_left()
        move()
    elif not right_is_clear():
        turn_around()
```

```
        p+=2
    else:
        turn_right()
        p+=1
grab_beeper()
while not facing_north():
    turn_left()
    p+=1
turn_right()
print "The perimeter is", p
turn_off()
```

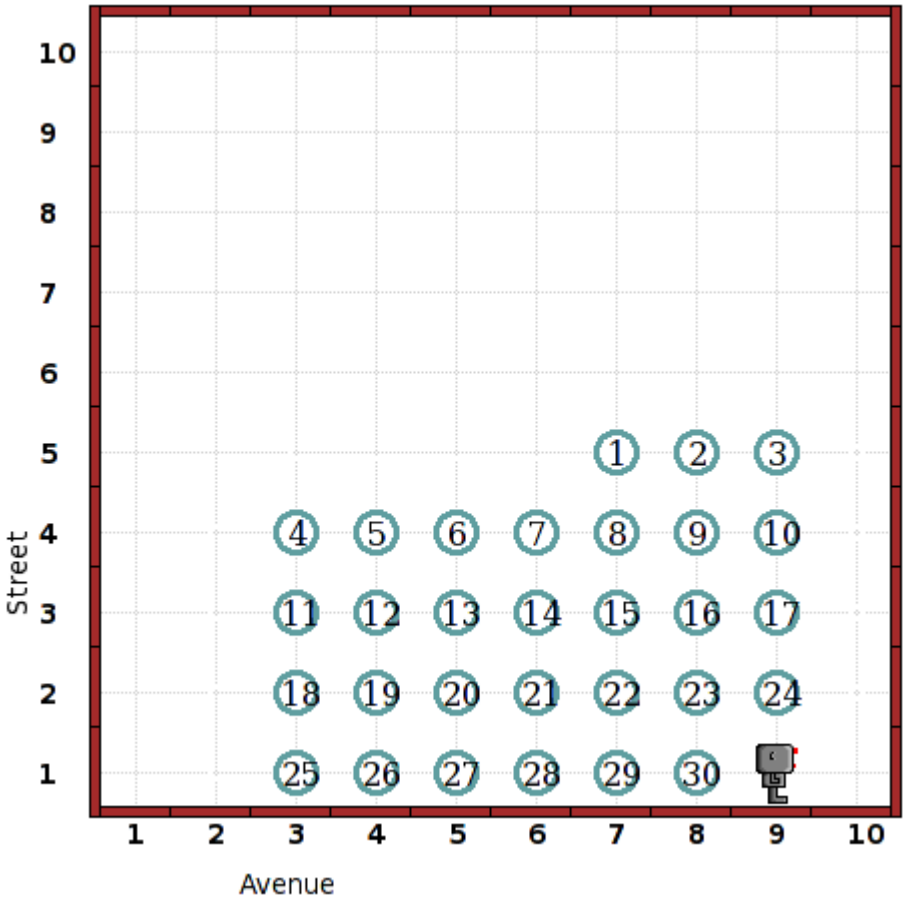
Calendar IA2, IB1a, IB2a, IB2c, IB3, IC, IIA1, IIA2, IIB1, IIIA1, IIIB1, IIIB2, IIIB3
adapted from a CMSC 201 assignment

The task is to draw a calendar in a subroutine with arguments (the number of days after Sunday that the month starts, the number of days in the month). The task is simpler if the robot returns to the left side before drawing a week. Drawing more efficiently in a zigzag fashion is trickier. It could be used as a program writing task.

Before:



After (4,30) (like April 2010):




```
def turn_around():
    for i in range(2):
        turn_left()
def week(first,days,dir=1):
    if dir == -1:
        start = first+days-1
        end = first-1
    else:
        start = first
        end = first+days
    for i in range(start,end,dir):
        for j in range(i):
            drop_beeper()
        move()
def calendar(first,days):
    turn_left()
    for i in range(first):
        move()
    week(1,7-first)
    start = 8-first
    dir=0
    while start <= days:
        if dir == 0:
            for i in range(2):
                turn_right()
                move()
            for i in range(start+6-days):
                move()
```

```
        else:
            for i in range(2):
                turn_left()
                move()
        week(start,min(days-start+1,7),dir*2-1)
        start += 7
        dir = (dir+1)%2
calendar(4,30)
turn_off()
```

Pattern IA2, IB1a, IB2a, IB2c, IB3, IC, IIB1, IIIA1, IIIA2, IIIB1
adapted from CMSC 201 exam review questions

This prediction task involving nested loops varies in the loop counts and if statement conditions.

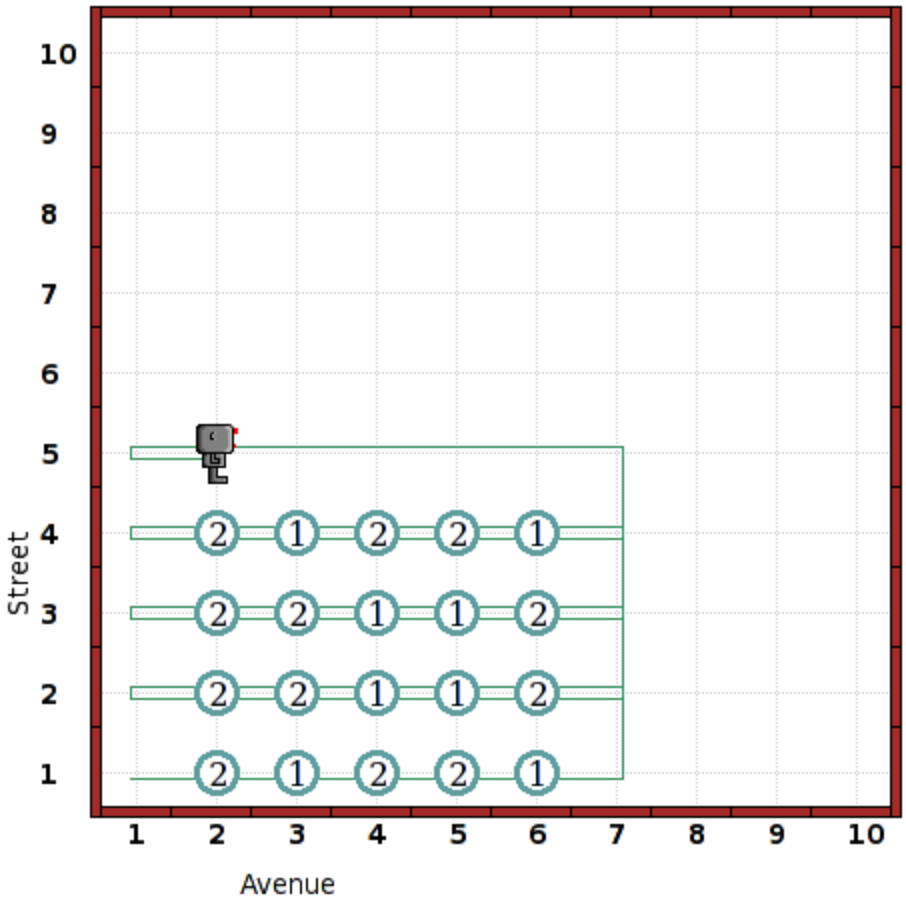
```
def turn_around():
    for i in range(2):
        turn_left()
def crlf():
    turn_left()
    move()
    turn_left()
    while front_is_clear():
        move()
    turn_around()
    move()
def drop_beepers(num):
    for i in range(num):
        drop_beeper()
    move()
move()
for i in range(4):
    for j in range(5):
        if i+1 == j or j+i == 4:
            drop_beepers(1)
        else:
            drop_beepers(2)
    crlf()
turn_off()
```

OR

```
def turn_around():
    for i in range(2):
        turn_left()
def crlf():
    turn_left()
    move()
    turn_left()
    while front_is_clear():
        move()
    turn_around()
    move()
```

```
def drop_beepers(num):
    for i in range(num):
        drop_beeper()
    move()
move()
for g in range(3):
    for h in range(3):
        if g == h+1 or g+h == 2:
            drop_beepers(1)
        else:
            drop_beepers(2)
    crlf()
turn_off()
```

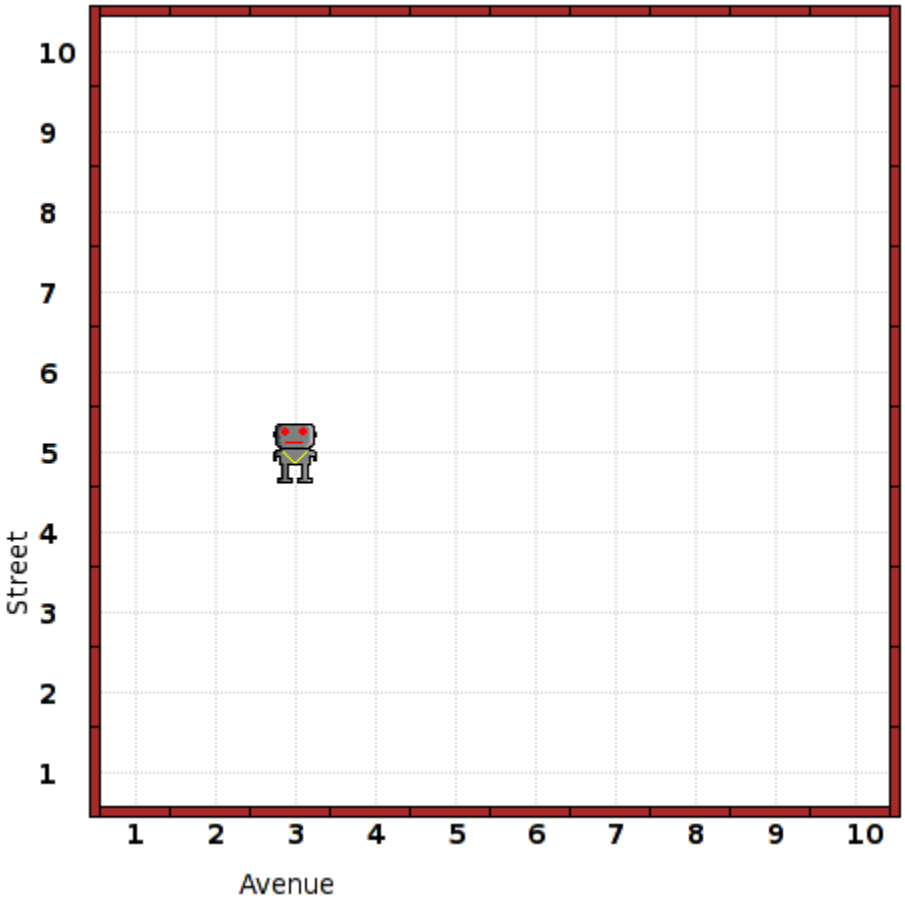
After (the first example):



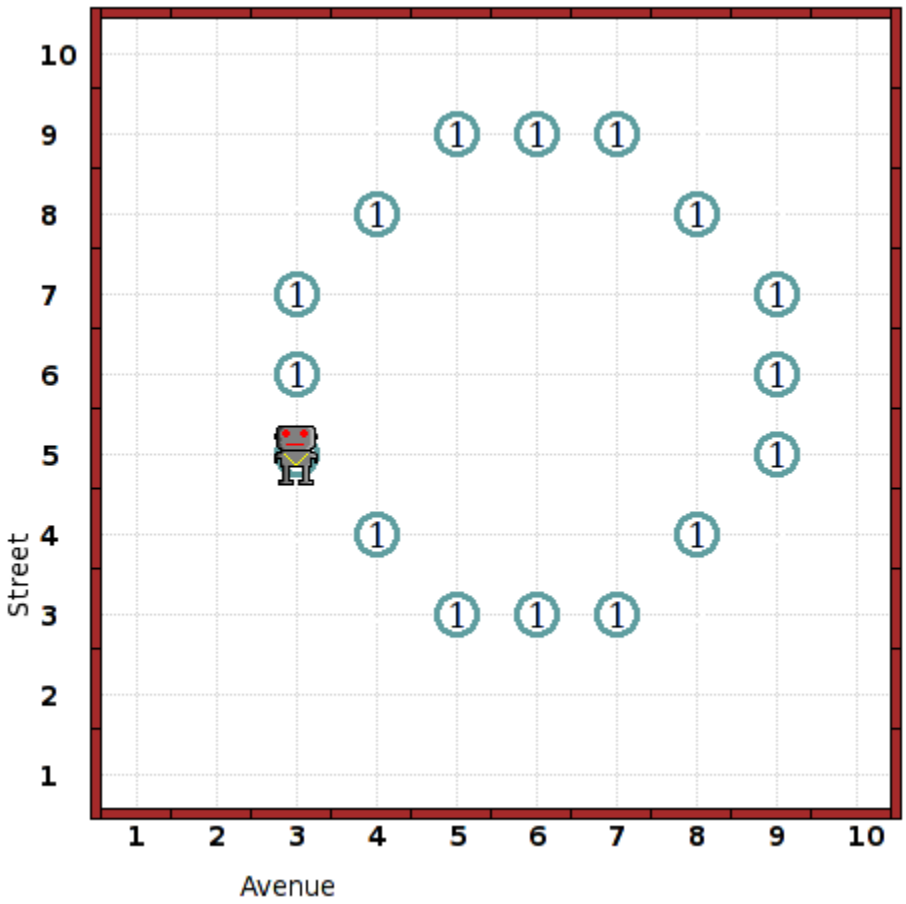
Octagon IB2a, IB3, IC, IIB1

Yet another drawing task, involving a subroutine with an argument for the length of each side of the octagon. It could be used as a program writing or prediction task.

Before:



After:

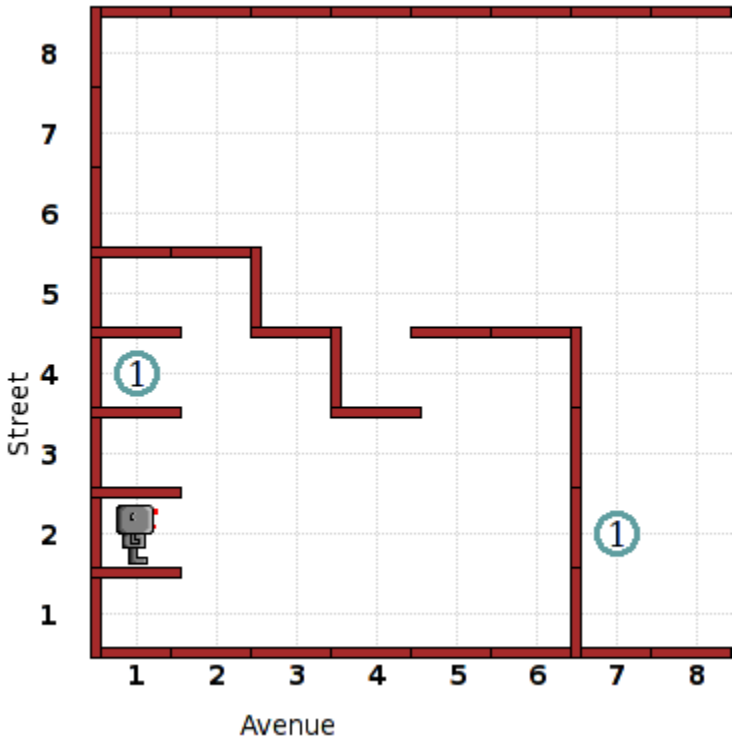



```
def turn_around():
    for i in range(2):
        turn_left()
def line(size):
    for i in range(size):
        drop_beeper()
        move()
def diagonal_line(size):
    for i in range(size):
        drop_beeper()
        move()
        turn_left()
        move()
        turn_right()
def octagon(size):
    for i in range(4):
        diagonal_line(size)
        turn_left()
        line(size)
octagon(2)
turn_off()
```

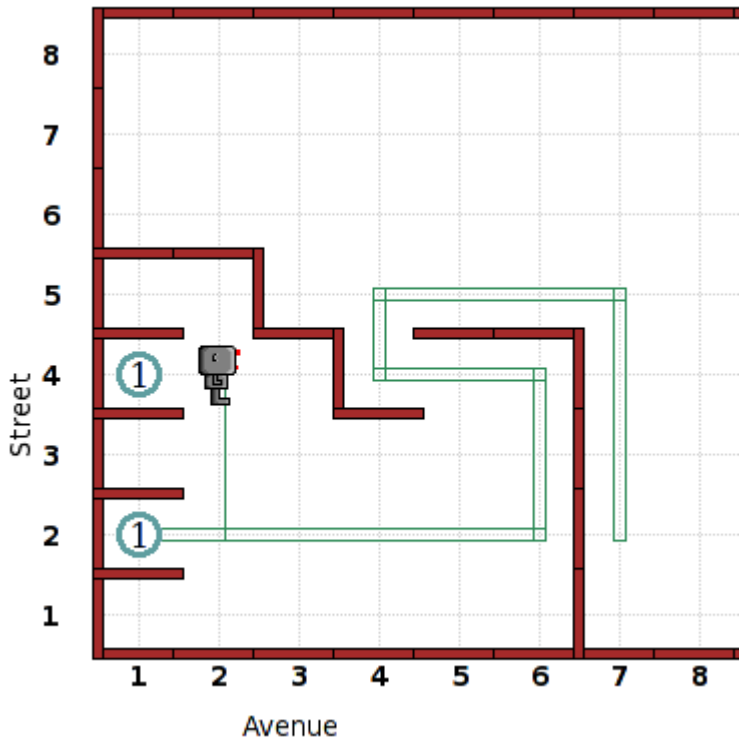
Carry my pads, rookie! IA2, IB1a, IB2a, IB2b, IB3, IIA1, IIA2, IIA3, IIIA1, IIIA2
adapted from Guido van Robot lessons and inspired by the Dallas Cowboys 2010 training camp

You are a robot named DEZ, a freshman on your school's robot football team. After practice, a senior robot named ROY asked you to carry his pads into the locker room. Unfortunately, you forgot and left them against the wall outside, so he stuffed you in his locker. You need to go back outside and retrieve his pads (represented by a beeper), bring them in and place them in his locker to prevent even worse hazing tomorrow. As if this isn't bad enough, you sustained a mild concussion during practice today, and the sensitivity to light is making it hard to see, so you are just feeling your way around. You should walk carefully until you hit a wall, and then feel your way around the wall until you find the pads. Additionally, you should remember the path you took to find them, so that you can just retrace your steps and don't have to open your eyes on the way back (you can't feel your way back because your hands will be carrying pads). This will require the use of a list. Finally, after you have put away the pads, go stand in front of your locker (the one two lockers away with the pads already in it) and face to the right so that you can talk to reporters. One last word of advice, don't yell at the reporters if they make a big deal about the pads incident. The exact layout of the locker room can vary, but the robot should have a clear shot at the right wall from his starting position. The entrance is designed so that you can't try to cheat and take a straight shot from the entrance to the locker.

Before:



After:




```
def turn_around():
    for i in range(2):
        turn_left()
def traverse_wall():
    i=0
    while front_is_clear() and not right_is_clear() and not on_beeper():
        move()
        i+=1
    return i
path=[]
p=0
while front_is_clear():
    move()
    p+=1
turn_left()
path.append((p,'L'))
p=traverse_wall()
while not on_beeper():
    if right_is_clear():
        turn_right()
        path.append((p,'R'))
        move()
        p=1
    elif not left_is_clear():
        turn_around()
        path.append((p,'A'))
        p=0
    else:
        turn_left()
        path.append((p,'L'))
        p=0
    p+=traverse_wall()
path.append((p,'A'))
```

```
path.reverse()
grab_beeper()
for steps,dir in path:
    if dir == 'L':
        turn_right()
    elif dir == 'R':
        turn_left()
    else:
        turn_around()
    for i in range(steps):
        move()
turn_around()
drop_beeper()
move()
turn_left()
for i in range(2):
    move()
turn_right()
turn_off()
```

Spaced Beepers IA1, IB2a, IB2c, IB3, IIIA1, IIIB1
adapted from CMSC 201 exam review questions

Very simple prediction tasks to test understanding of loops, loop indices, and if statements. Many variations are possible. Two examples are below.

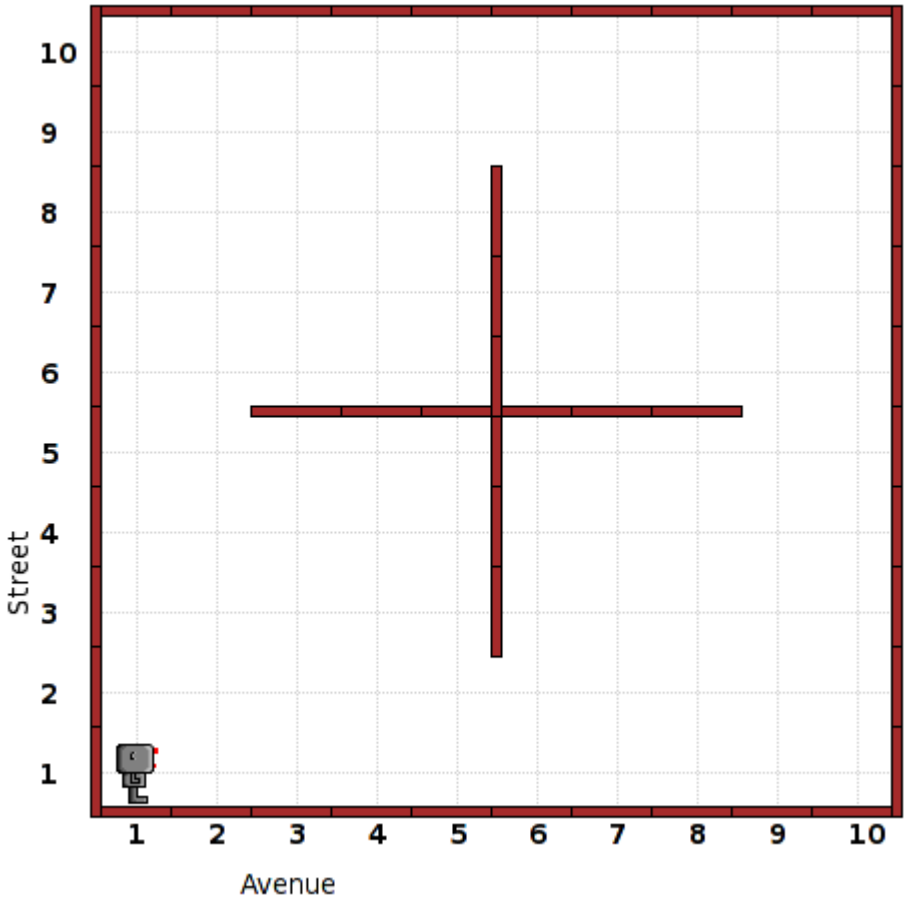
```
for i in range(10):
    if i%3 == 0:
        drop_beeper()
    move()
turn_off()
```

```
for i in range(11):
    if i%2 == 0:
        for j in range(i/2 + 1):
            drop_beeper()
    move()
turn_off()
```

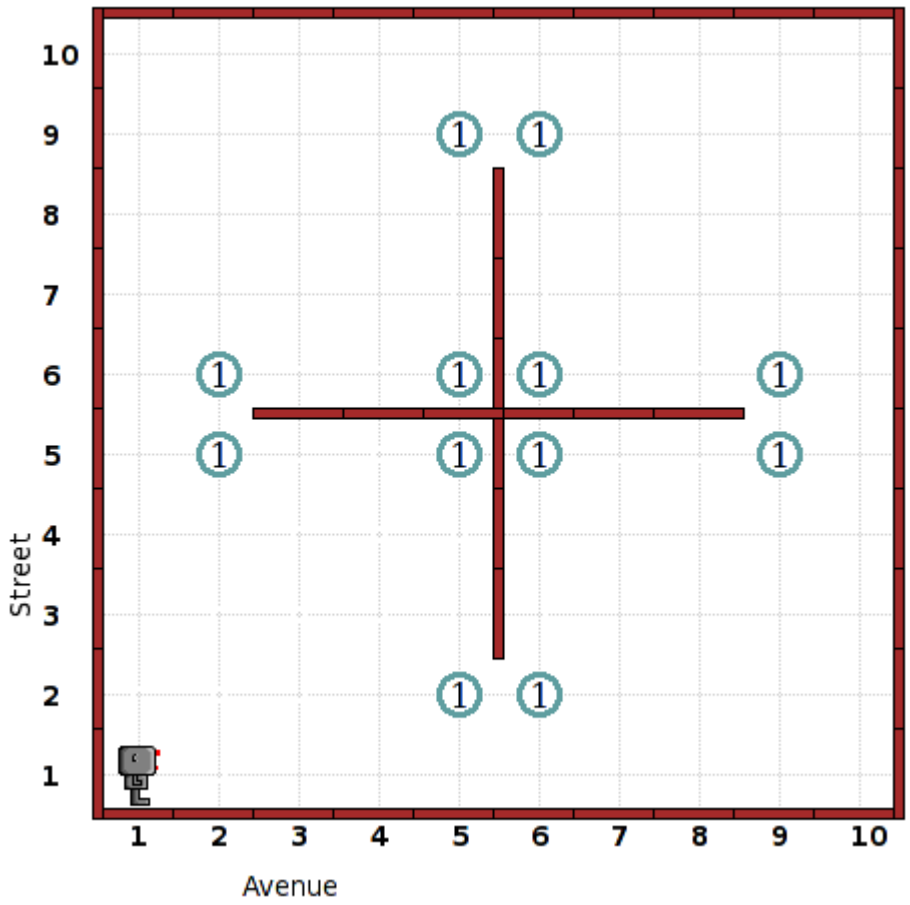
Planter IA2, IB1a, IIIA2
adapted from Karel J Robot exercises

The task is to walk to the walled structure in the middle of the room and place beepers around it at each end of every wall and return to the original position. Minor variations can be made in the shape of the structure (whether there are affects what type of loop can be used). The other possible variation is to place beepers on only inside or outside corners. It can be used as a program writing or prediction task.

Before:



After:




```
def turn_around():
    for i in range(2):
        turn_left()
def move_diagonal():
    move()
    turn_left()
    move()
    turn_right()
def traverse_wall():
    while front_is_clear() and not left_is_clear():
        move()
while front_is_clear() and left_is_clear():
    move_diagonal()
while front_is_clear():
    move()
drop_beeper()
turn_right()
move()
traverse_wall()
while not on_beeper():
    drop_beeper()
    if left_is_clear():
        turn_left()
        move()
    else:
        turn_right()
    traverse_wall()
turn_around()
```

```
while front_is_clear() and left_is_clear():
    move_diagonal()
for i in range(2):
    while front_is_clear():
        move()
    turn_left()
turn_off()
```

Measurement IB1a, IB2a, IC, IIA2, IIB2
adapted from Karel J Robot instructor's guide

The first of two measurement tasks is to measure the distance to the closest wall in front of where the robot is initially facing and report the distance from the starting position either by printing or placing beepers. The second is to measure the area of the rectangular room that the robot is inside. Both a while and for loop are required, as well as variables. It could be used as a program writing task.

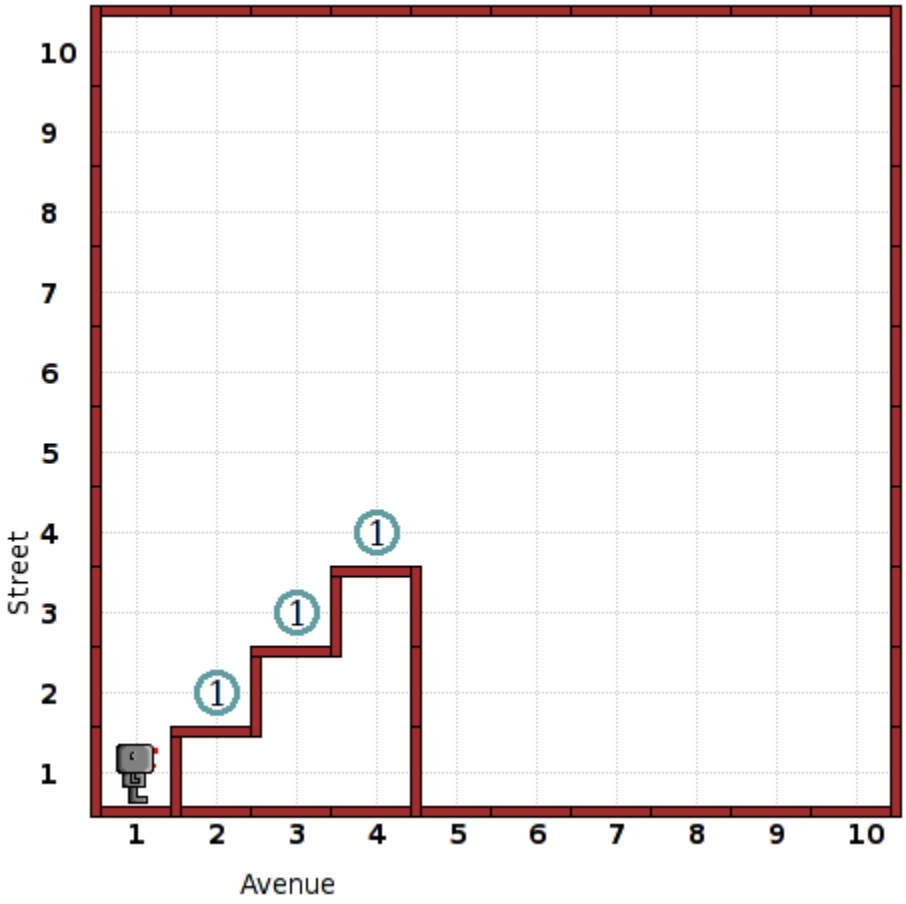
Area:

```
def turn_around():
    for i in range(2):
        turn_left()
def dist_to_wall():
    d=0
    while front_is_clear():
        move()
        d+=1
    turn_around()
    for i in range(d):
        move()
    turn_around()
    return d
a=dist_to_wall()
turn_around()
a+=dist_to_wall()
turn_left()
b=dist_to_wall()
turn_around()
b+=dist_to_wall()
turn_right()
print "The area is", (a+1)*(b+1)
turn_off()
```

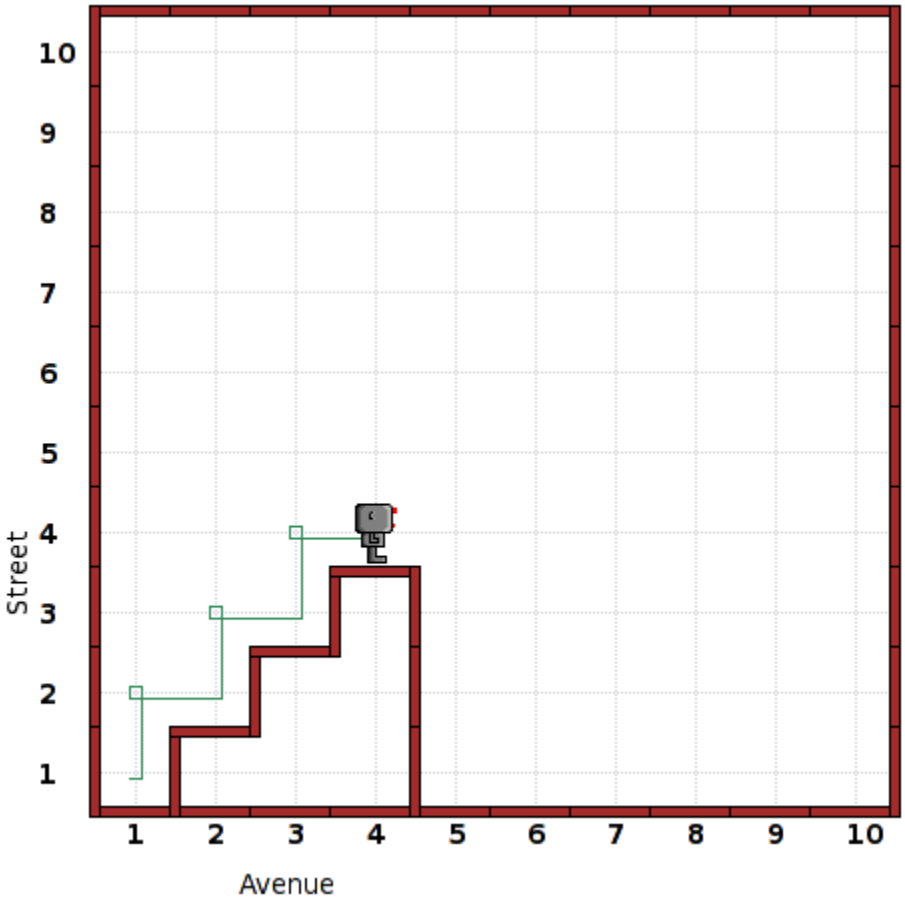
Stair Cleaning IB1a
adapted from Karel the Robot documentation

The task is to climb the stairs and collect the beepers, stopping on the top stair. With variation in the number of stairs, the implementation is a simple while loop. It could be used as a program writing task.

Before:



After:

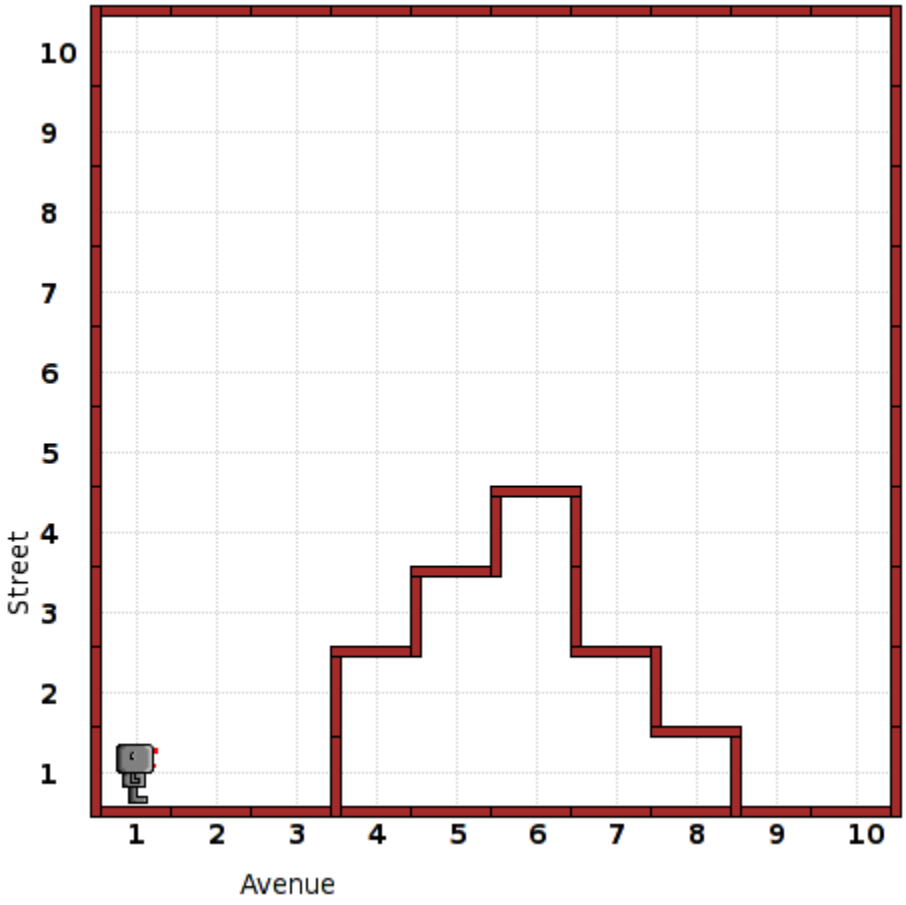



```
def turn_around():
    for i in range(2):
        turn_left()
while not front_is_clear():
    turn_left()
    move()
    turn_right()
    move()
    grab_beeper()
turn_off()
```

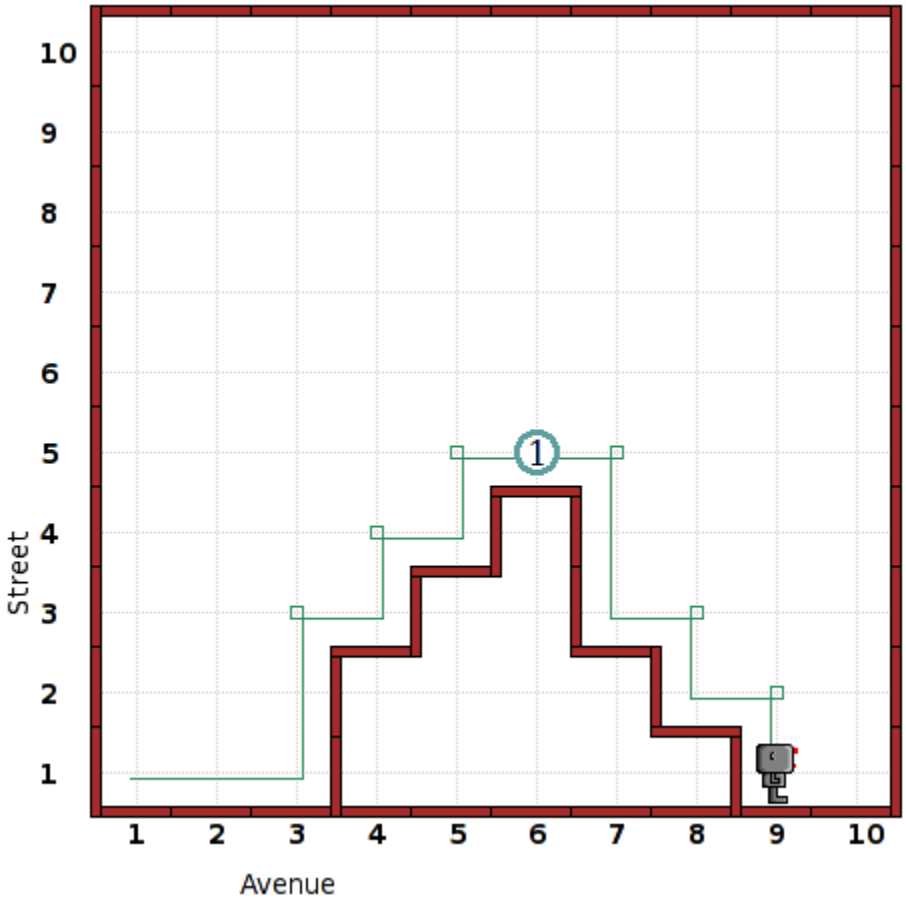
Mountain Climbing IB1a, IIA2, IIIA1, IIIA2
adapted from Karel J Robot exercises

The task is to climb the mountain and place a beeper on the top, then descend the other side of the mountain. The exact shape of the mountain can vary (but horizontal surfaces are always one wall), requiring the use of a while loop and variable. It could be used as a program writing or prediction task.

Before:



After:

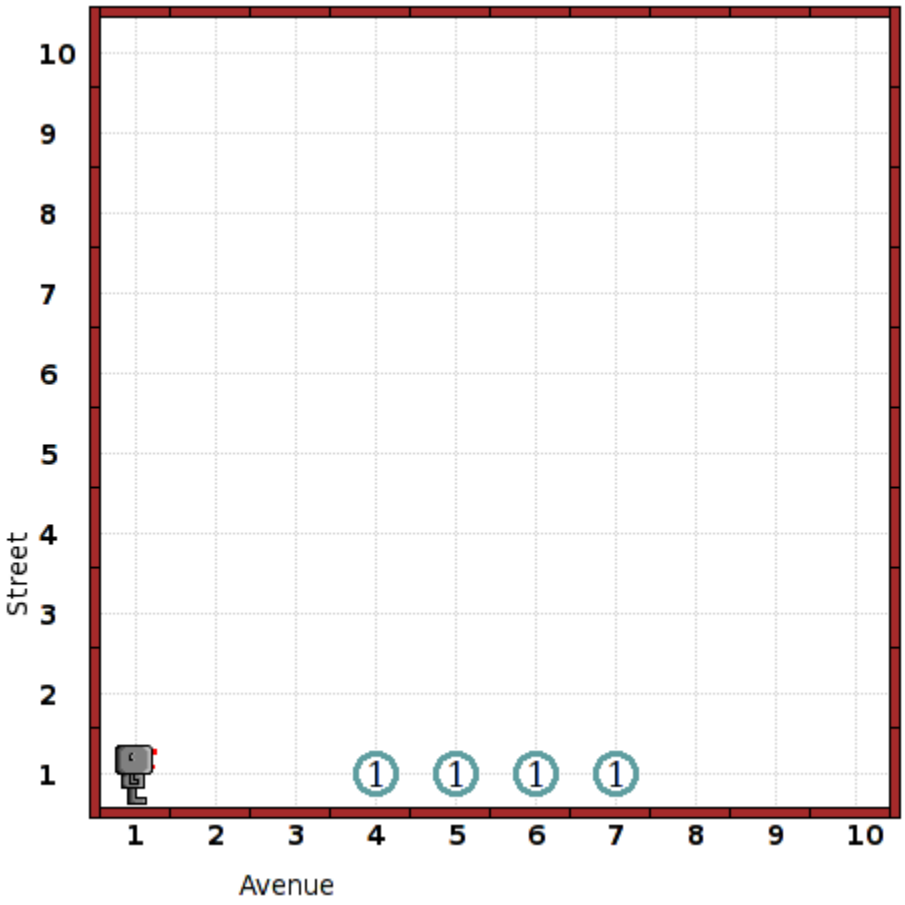



```
def turn_around():
    for i in range(2):
        turn_left()
while front_is_clear():
    move()
h=0
while not front_is_clear():
    turn_left()
    while not right_is_clear():
        move()
        h += 1
    turn_right()
    move()
drop_beeper()
while h > 0:
    move()
    turn_right()
    while front_is_clear():
        move()
        h -= 1
    turn_left()
turn_off()
```

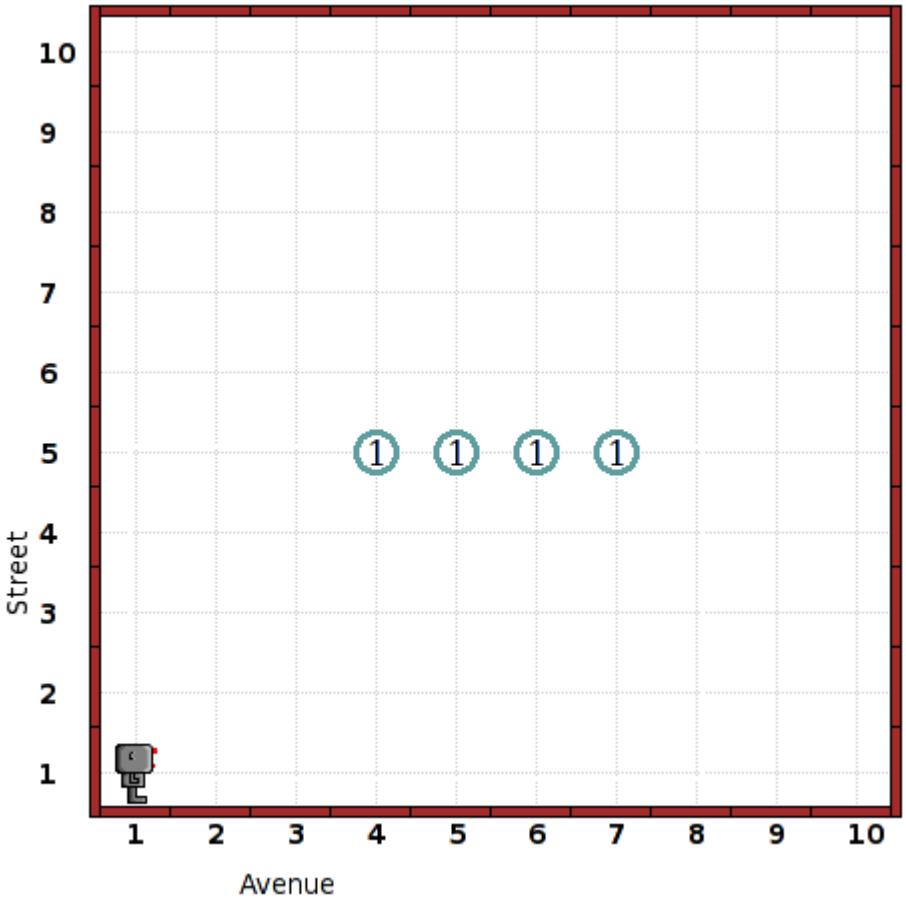
Beeper Mover IB1a, IB3, IB2a, IIA2, IIIA2
adapted from Karel J Robot exercises

The task is to move a line of beepers north as many spaces as there are beepers. It requires a while loop, variable, and for loop. It could be used as a program writing or prediction task.

Before:



After:

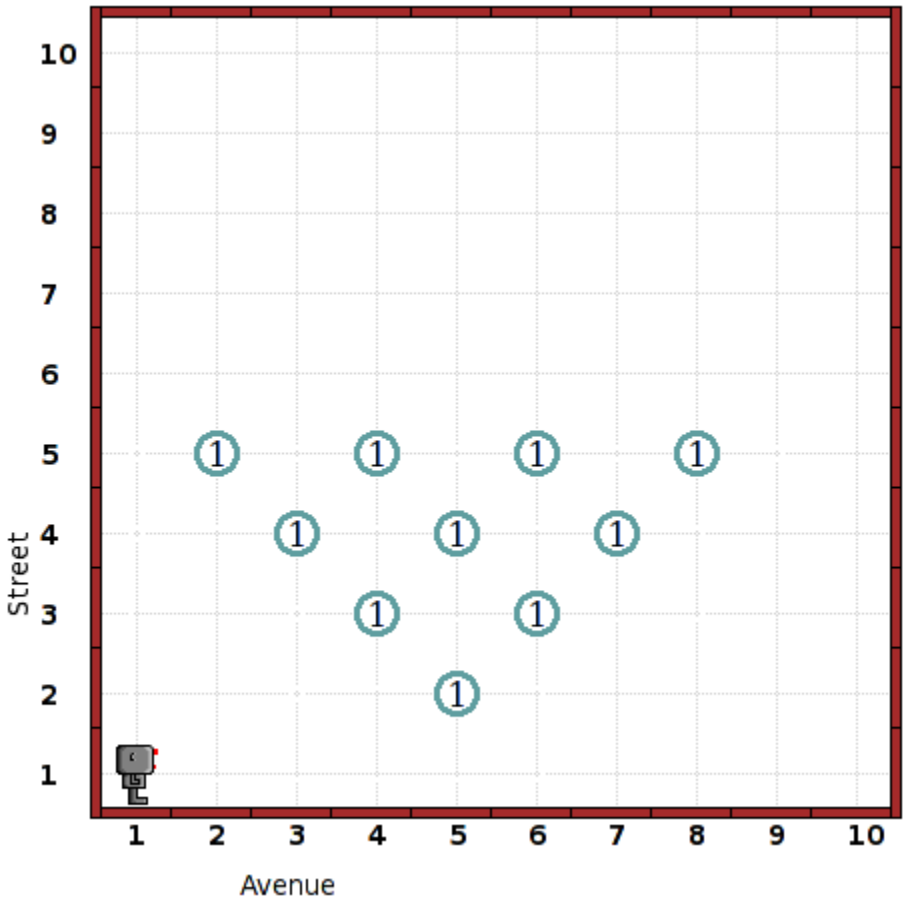



```
def turn_around():
    for i in range(2):
        turn_left()
while not on_beeper():
    move()
b=0
while on_beeper():
    grab_beeper()
    move()
    b+=1
turn_left()
for i in range(b):
    move()
turn_left()
for i in range(b):
    move()
    drop_beeper()
for i in range(2):
    while front_is_clear():
        move()
    turn_left()
turn_off()
```

Bowling Pins IA1, IA2, IB2a, IB2c, IC, IIB1, IIIB1
adapted from Karel J Robot exercises

Another drawing task with for loops. It varies with a subroutine that takes an argument for the number of rows to draw. It can be used as a program writing or prediction task.

After:



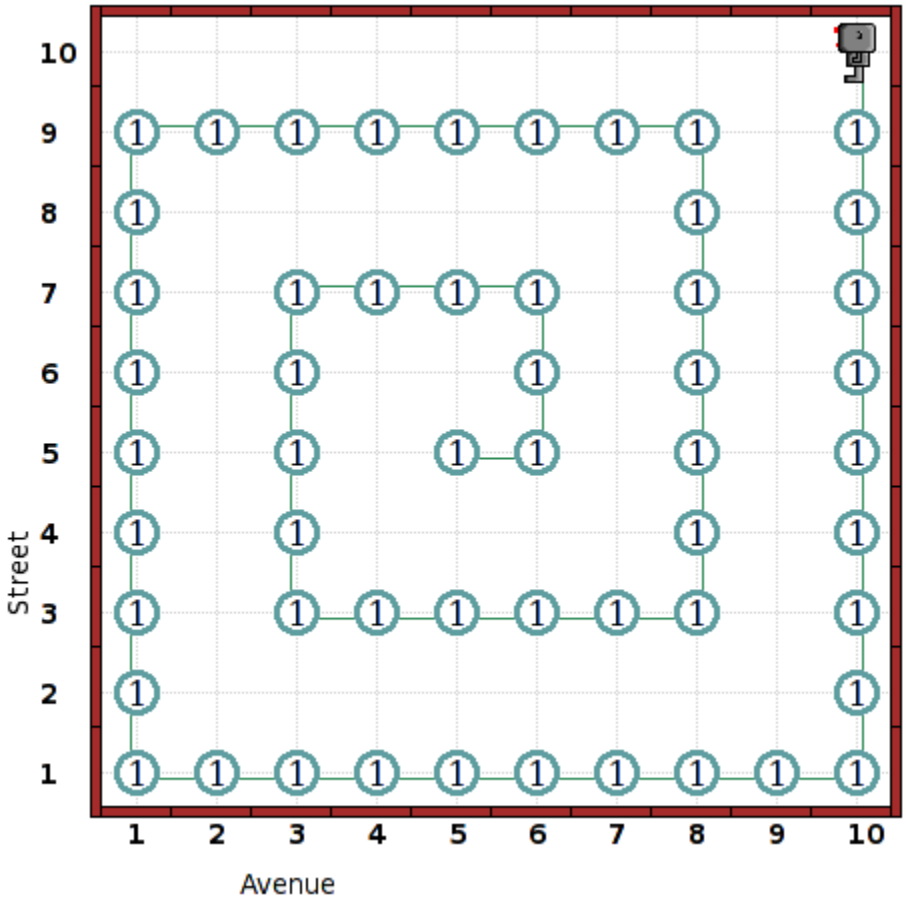

```
def turn_around():
    for i in range(2):
        turn_left()
def line(size):
    for i in range(size):
        move()
        drop_beeper()
        move()
def bowling_pins(size):
    for i in range(4):
        move()
    for i in range(size):
        move()
        if i%2 == 0:
            turn_left()
            move()
            turn_left()
        else:
            turn_right()
            move()
```

```
            turn_right()
        line(i+1)
    if size%2 == 0:
        turn_around()
    while front_is_clear():
        move()
    turn_left()
    for i in range(size):
        move()
    turn_left()
bowling_pins(4)
turn_off()
```

Spiral IB1a, IB3, IIA1, IIA2, IIIA1, IIIA2
adapted from Karel J Robot exercises

The task is, starting at corner 5,5 facing east, to draw a spiral until running into a wall and then turn off. It uses while loops and variables and can be used as a program writing or prediction task.

After:




```
def turn_around():
    for i in range(2):
        turn_left()
i=1
j=1
while j == i:
    j=1
    while j <= i and front_is_clear():
        drop_beeper()
        move()
        j+=1
    turn_left()
    i+=1
turn_off()
```