Decomposition

CS106A, Stanford University

Happy Friday!

Today's Goal

1. Be able to approach a problem "top down" by using decomposition and stepwise refinement

Let’s review!

The Full Karel

Base Karel commnds:

```
move()
turn_left()
put_beeper()
pick_beeper()
```

Karel program structures:

```
# Comments can be included in any part
# of a program. They start with a #
# and include the rest of the line.

def main() :
    code to execute


declarations of other functions
```

Names of the conditions:

```
front_is_clear()        front_is_blocked()
beepers_present()       no_beepers_present()
beepers_in_bag()        no_beepers_in_bag()
left_is_clear()         left_is_blocked()
right_is_clear()        right_is_blocked()
facing_north()          not_facing_north()
facing_south()          not_facing_south()
facing_east()           not_facing_east()
facing_west()           not_facing_west()
```

Conditions:

```
if condition:
    code run if condition passes

if condition:
    code block for "yes"
else:
    code block for "no"
```

Loops:

```
for i in range( count):
    code to repeat

while condition:
    code to repeat
```

Function Declaration:

```
def name():
    code in the body of the function.
```

Extra Karel Commands:

```
paint_corner(COLOR_NAME)
corner_color_is(COLOR_NAME)
```

Revisiting SteepleChaseKarel.py

More on Programming Style

```
"""
File: SteepleChaseKarel.py
--------------------------
Karel runs a steeple chase that is 9 avenues long.
Hurdles are of arbitrary height and placement.
"""


def main():
    """
    To run a race that is 9 avenues long, we need
    to move forward or jump hurdles 8 times.
    """
    for i in range(8):
        if front_is_clear():
            move()
        else:
            jump_hurdle()


def jump_hurdle():
    """
    Pre-condition:  Facing East at bottom of hurdle
    Post-condition: Facing East at bottom in next avenue after hurdle
    """
    ascend_hurdle()
    move()
    descend_hurdle()
```

Comments for program and *every* function

<u>Decomposition principle</u>: Each function should solve <u>one</u> step of problem

Consistent indentation

Descriptive ***names*** (snake_case)

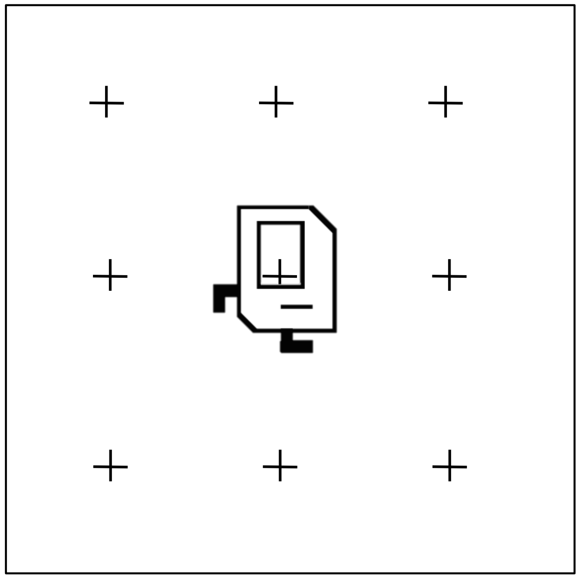
Short functions (usually 1-15 lines)

Aside: Common Error

Lather,
Rinse,
Repeat

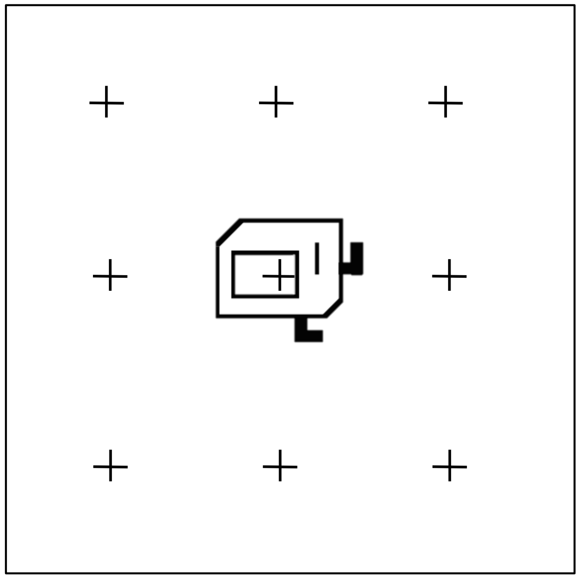
Infinite Loop

```
def turn_to_wall():
    while front_is_clear():
        turn_left()
```



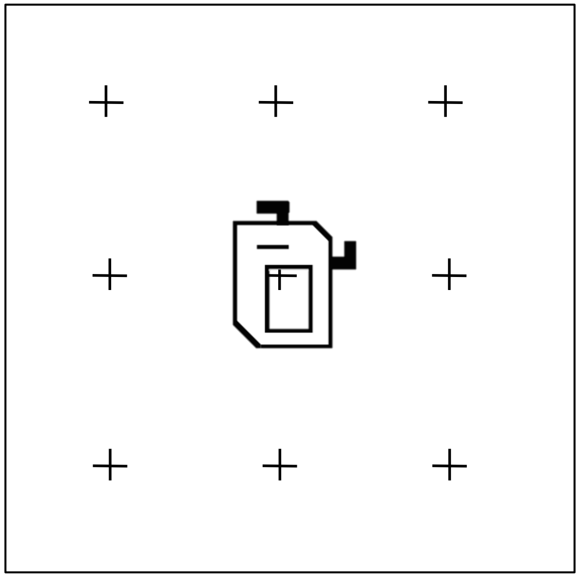
Infinite Loop

```
def turn_to_wall():
    while front_is_clear():
        turn_left()
```



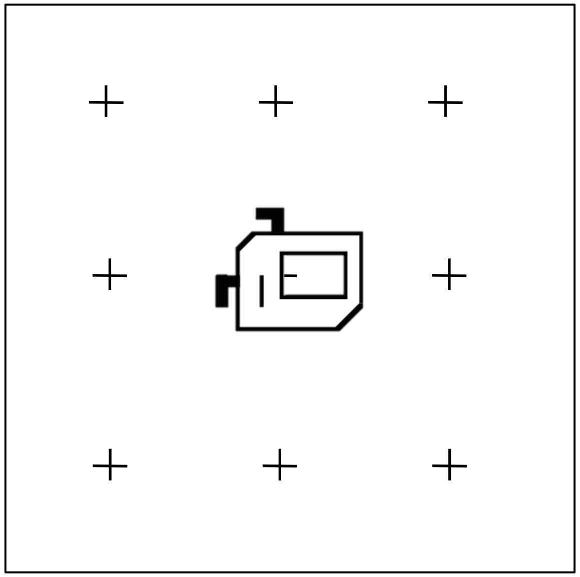
Infinite Loop

```
def turn_to_wall():
    while front_is_clear():
        turn_left()
```



Infinite Loop

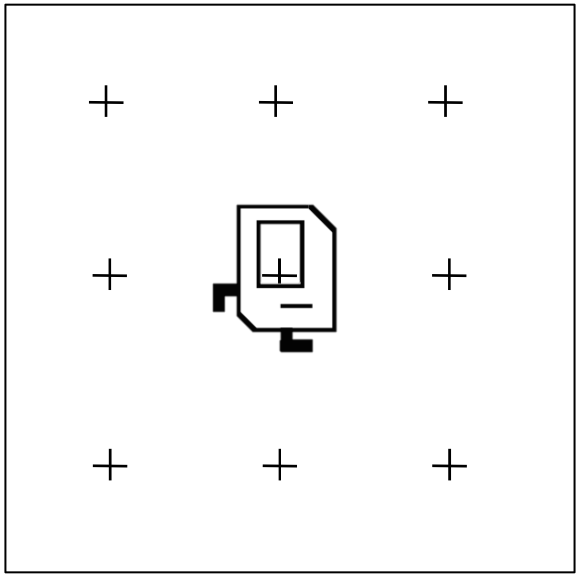
```
def turn_to_wall():
    while front_is_clear():
        turn_left()
```



Infinite Loop

```
def turn_to_wall():
    while front_is_clear():
        turn_left()
```

BUGGY!



What did you do this morning after you woke up?

What's Mozart Doing Now?

```
if mehran_teaching():
    not_funny()
    turn_left_in_grave()
```

```
while mehran_teaching():
    not_funny()
    turn_left_in_grave()
```

Pro Tips: Decomposing Functions

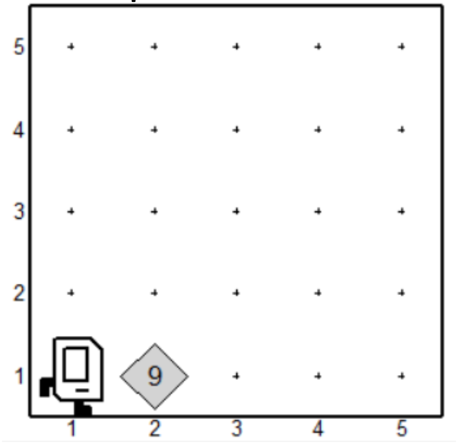
- A good function should do ”one conceptual thing”
- Function name should describe what it does
- Usually, functions are fairly short (e.g., 1-15 lines)
- Often, functions are reusable and easy to modify
- Each function should have a comment describing it

There are two types of programs.
One is so complex, there is nothing obvious wrong with it.
One is so clear, that this obviously nothing wrong with it.

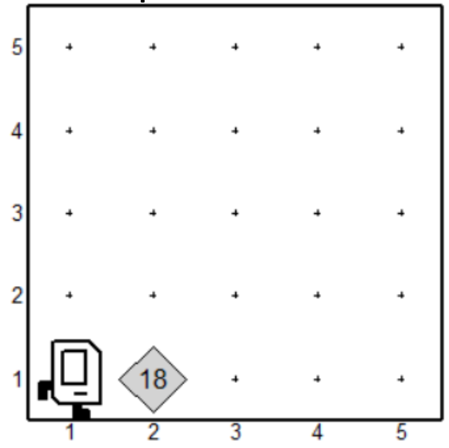
Karel Does Math: Doubling Beepers

- Write a program that has Karel double the number of beepers on the corner one avenue ahead of it
  - Karel starts at (1,1) facing East
  - There is a pile of 0 or more beepers on the corner one avenue ahead of Karel
  - Karel has infinite beepers in its bag
  - The world has empty corner on avenue after beeper pile

Example Initial World



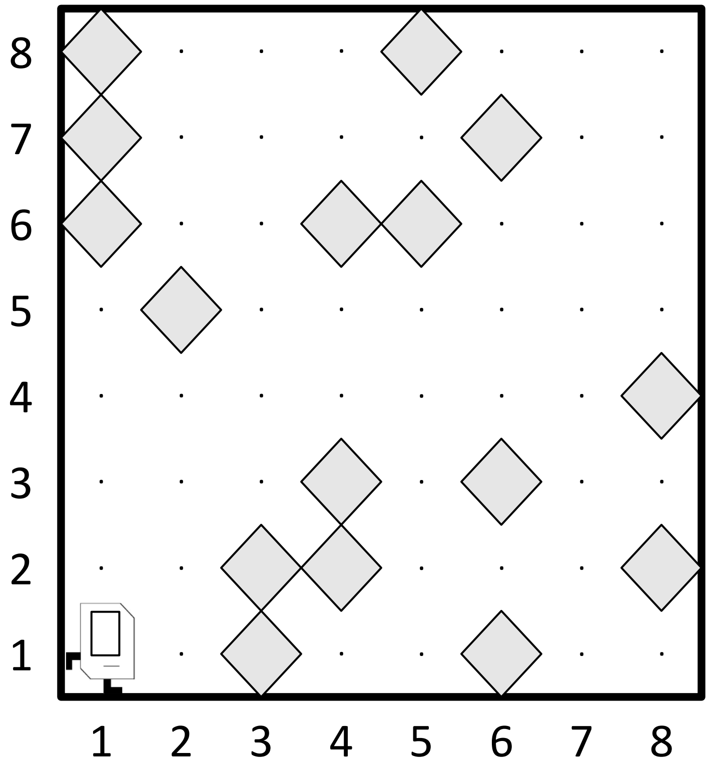
Example Final World



Let’s write DoubleBeepers.py together!

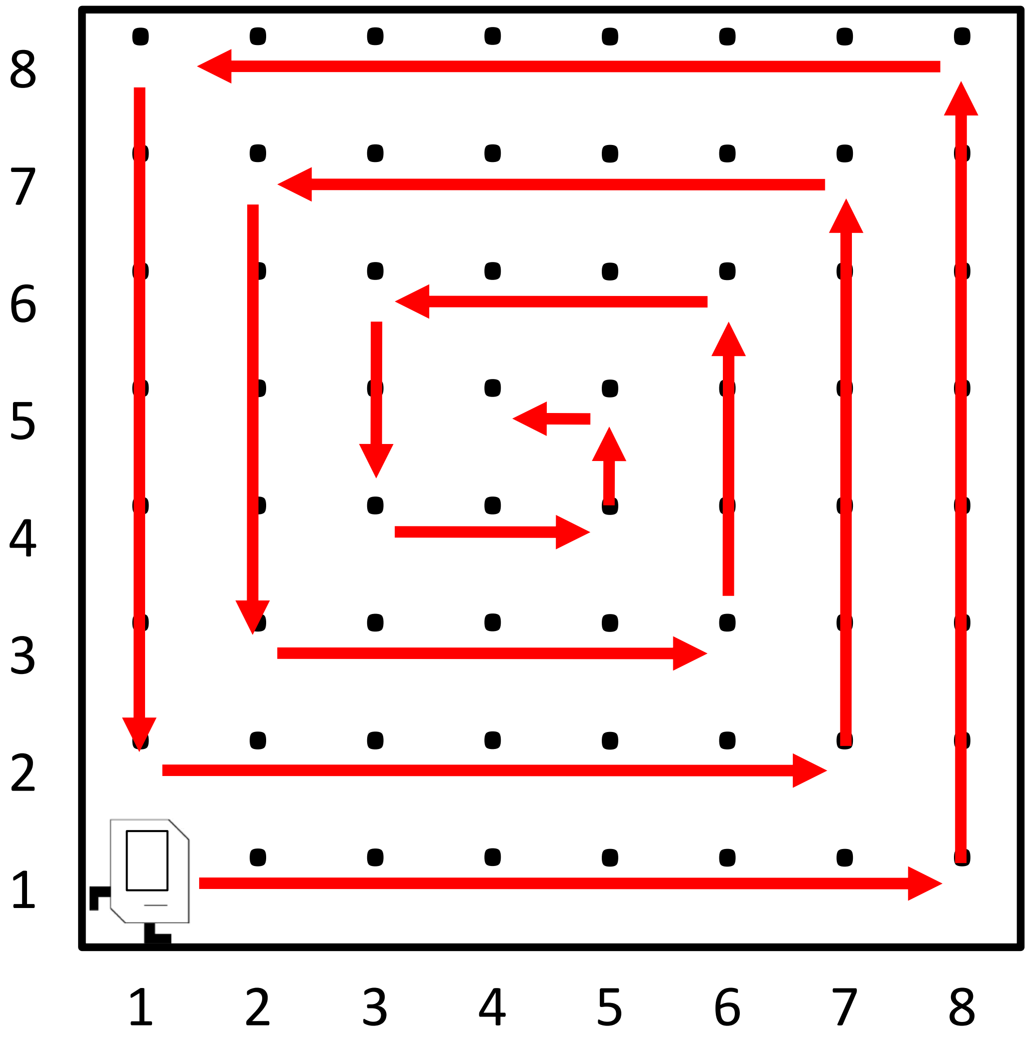
Karel the Room Cleaner

- Write a program that cleans up all beepers in the world
  - Karel starts at (1,1) facing East
  - The world is rectangular, and some squares contain (at most 1) beeper
  - There are no interior walls
  - When the program is done, the world should contain no beepers
  - Karel's ending location irrelevant
- What approach should we use?

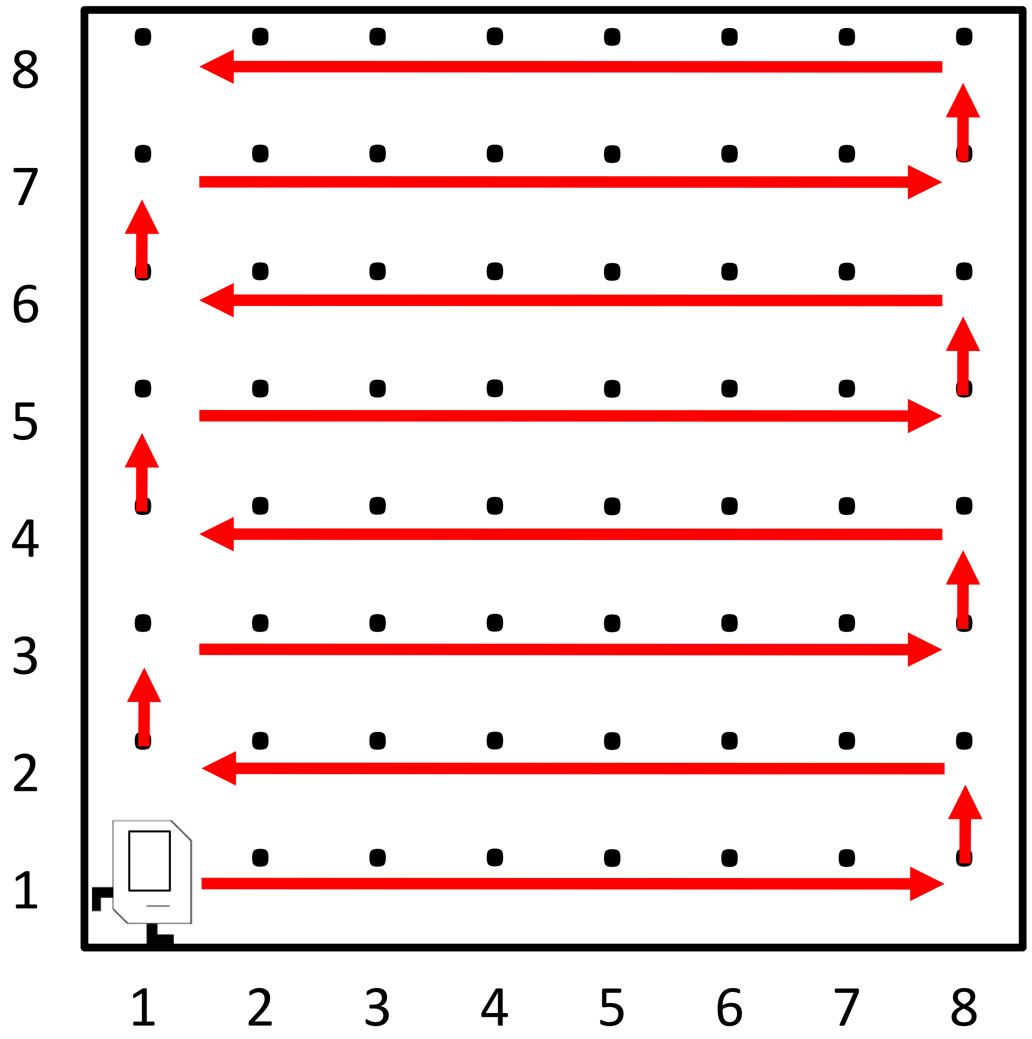


Muhammed ibn Musa Al Kwarizmi

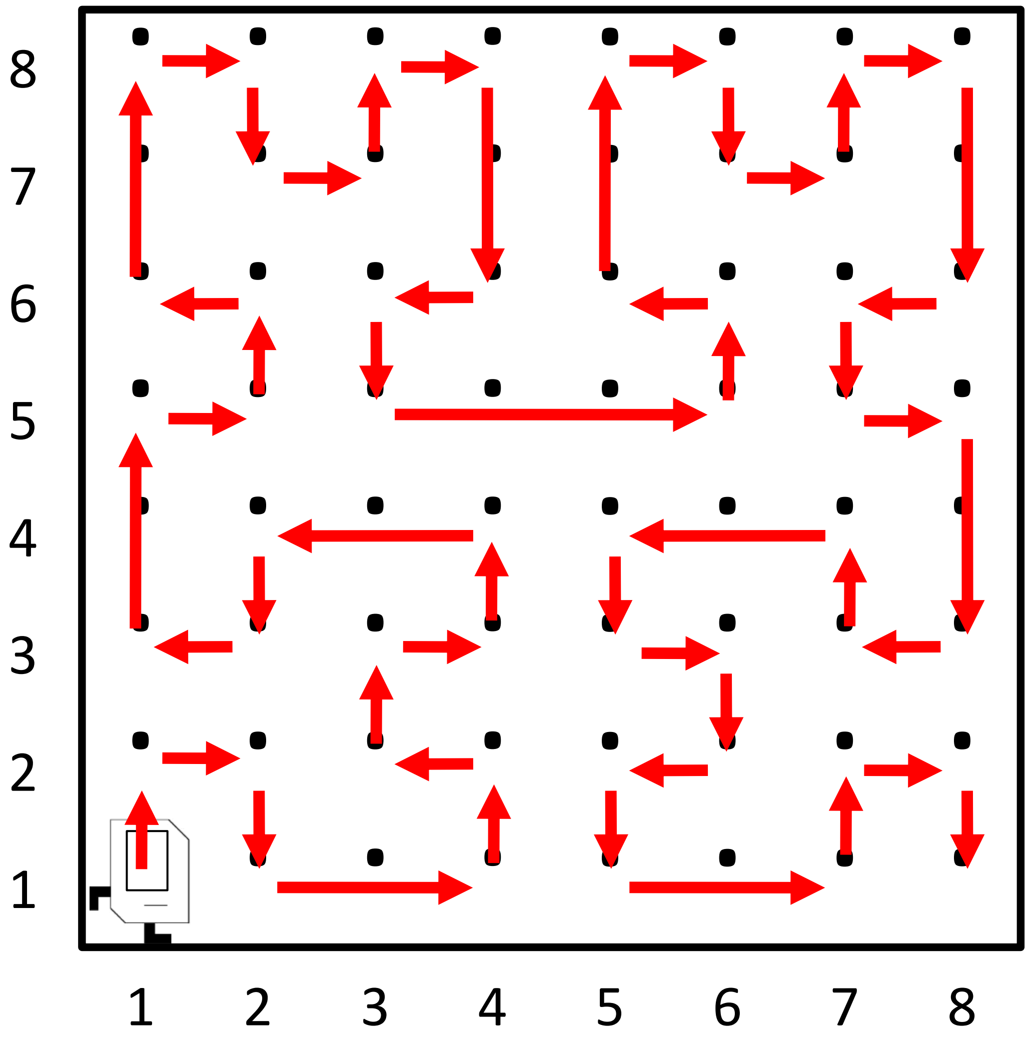
Possible Algorithm 1



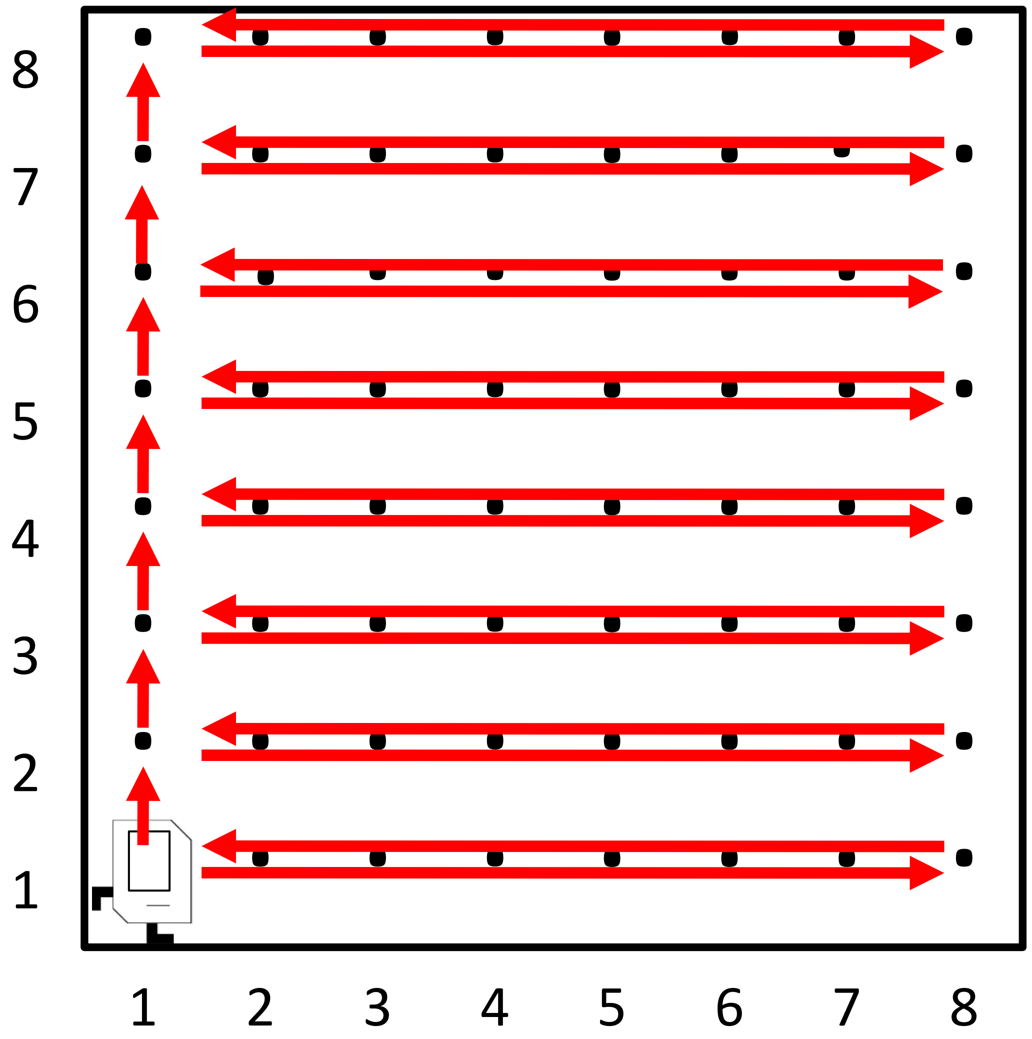
Possible Algorithm 2



Possible Algorithm 3



Possible Algorithm 4



CleanRoomKarel.py