

# Decomposition

Chris Piech and Mehran Sahami  
CS106A, Stanford University

# Today's Goal

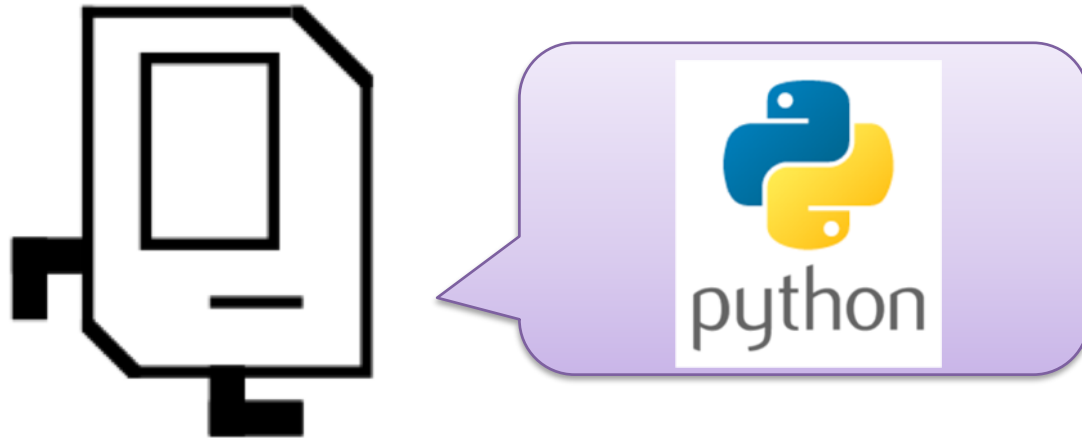
1. Be able to approach a problem “top down” by using decomposition (aka stepwise refinement)



First, a cool program

Quick review

# Karel the Robot



# Functions

```
def main():  
    go_to_moon()  
  
def go_to_moon():  
    build_spaceship()  
    # a few more steps  
  
def build_spaceship():  
    # todo  
    put_beeper()
```



# For Loops

```
def main():  
    # repeats the body 99 times  
    for i in range(99):  
        # the "body"  
        put_beeper()
```



# While Loops

```
def main():  
  
    # while condition holds runs body  
    # checks condition after body completes  
    while front_is_clear():  
        move()
```





# If Statement

```
def main():  
    # If the condition holds, runs body  
    if front_is_clear():  
        move()
```



# If / Else Statement

```
def main():  
  
    # If the condition holds,  
    if beepers_present():  
        # do this  
        pick_beeper()  
    else :  
        # otherwise, do this  
        put_beeper()
```



# The Full Karel

<b>Base Karel commands:</b> <code>move()</code> <code>turn_left()</code> <code>put_beeper()</code> <code>pick_beeper()</code>	<b>Conditions:</b> <i>if condition:</i> <i>code run if condition passes</i>  <i>if condition:</i> <i>code block for "yes"</i> <i>else:</i> <i>code block for "no"</i>
<b>Karel program structures:</b>  # Comments can be included in any part # of a program. They start with a # # and include the rest of the line.  <i>def main() :</i> <i>code to execute</i>  <i>declarations of other functions</i>	<b>Loops:</b>  <i>for i in range(count):</i> <i>code to repeat</i>  <i>while condition:</i> <i>code to repeat</i>
<b>Names of the conditions:</b> <code>front_is_clear()</code> <code>front_is_blocked()</code> <code>beepers_present()</code> <code>no_beepers_present()</code> <code>beepers_in_bag()</code> <code>no_beepers_in_bag()</code> <code>left_is_clear()</code> <code>left_is_blocked()</code> <code>right_is_clear()</code> <code>right_is_blocked()</code> <code>facing_north()</code> <code>not_facing_north()</code> <code>facing_south()</code> <code>not_facing_south()</code> <code>facing_east()</code> <code>not_facing_east()</code> <code>facing_west()</code> <code>not_facing_west()</code>	<b>Function Declaration:</b>  <i>def name():</i> <i>code in the body of the function.</i>  <b>Extra Karel Commands:</b>  <code>paint_corner(COLOR_NAME)</code> <code>corner_color_is(COLOR_NAME)</code>



End review

```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhoomba_karel()  
    if extra_time():  
        word_search_karel()
```

```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhoomba_karel()  
    if extra_time():  
        word_search_karel()
```

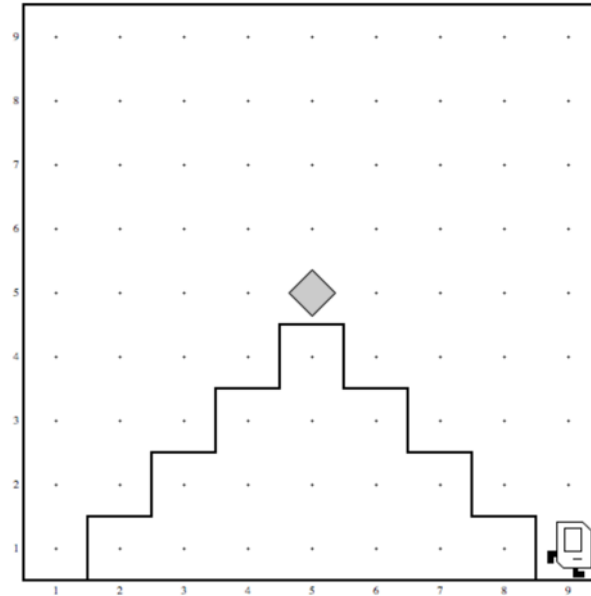
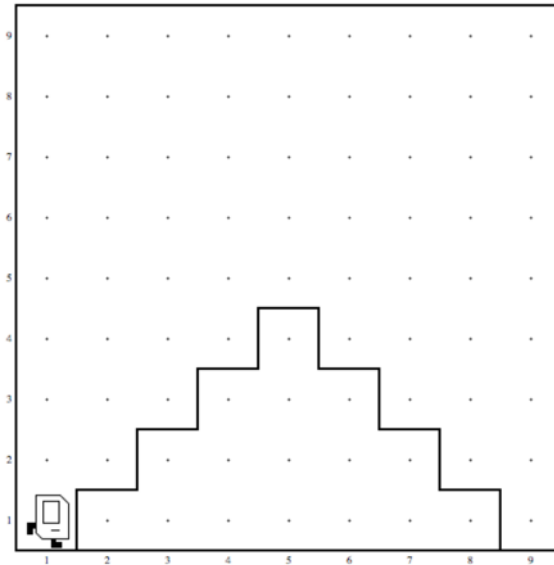


```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhoomba_karel()  
    if extra_time():  
        word_search_karel()
```



```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhomba_karel()  
    if extra_time():  
        word_search_karel()
```

# Mountain Karel



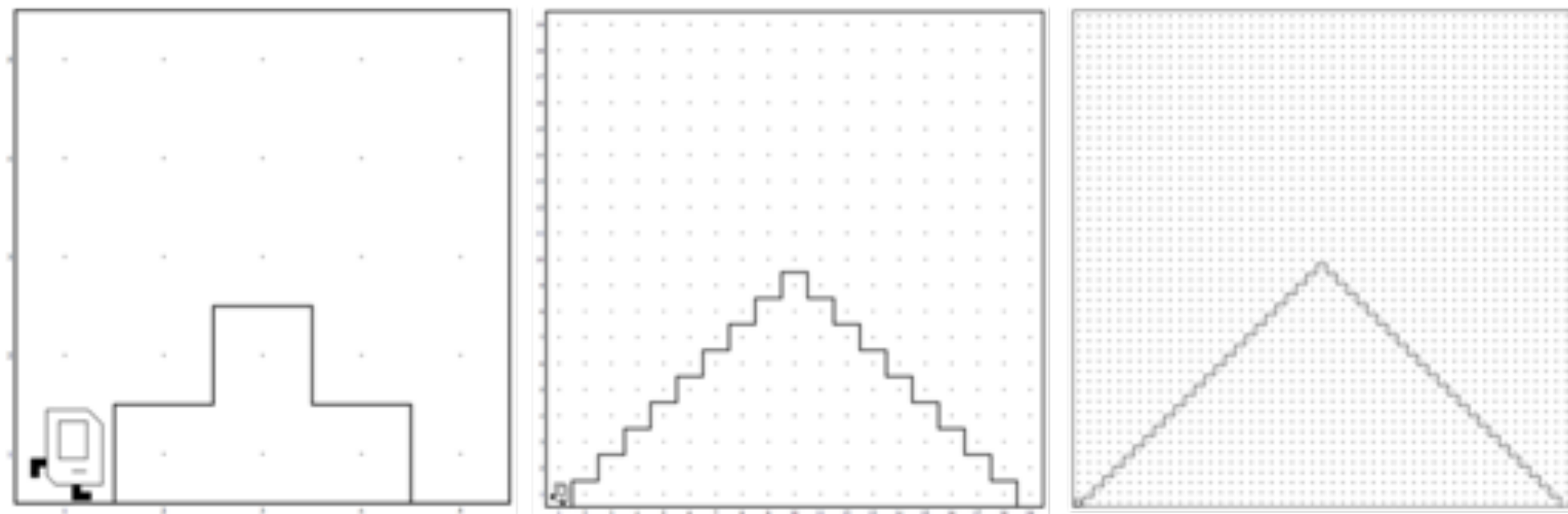
Muhammed ibn  
Musa Al Kwarizmi

Piech a

niversity



# Mountain Karel



DO  
YOUR  
THING



# Pro Tips



A good function should do “one conceptual thing”



Know what it does by looking at its name



Less than 10 lines, 3 levels of indentation



Reusable and easy to modify



Well commented

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.



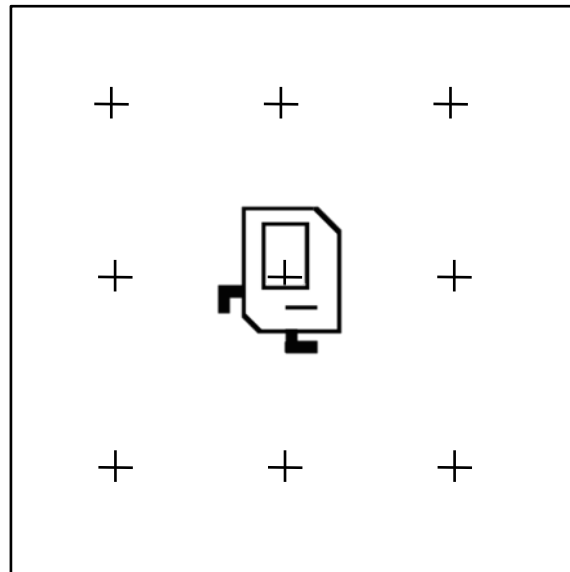
# Aside: Common Errors

Lather,  
Rinse,  
Repeat



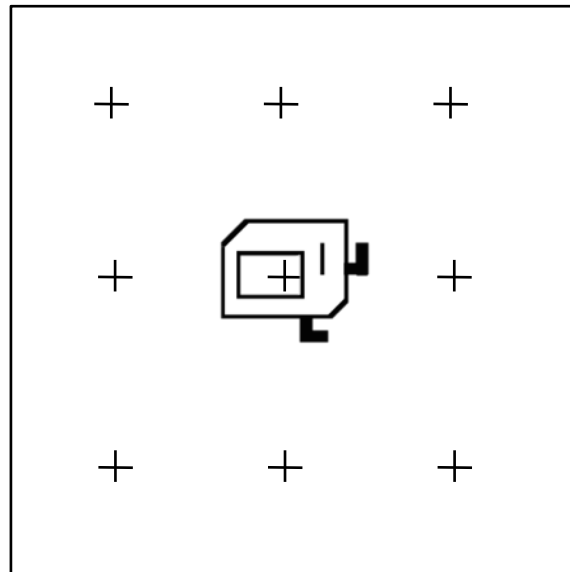
# Infinite Loop

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



# Infinite Loop

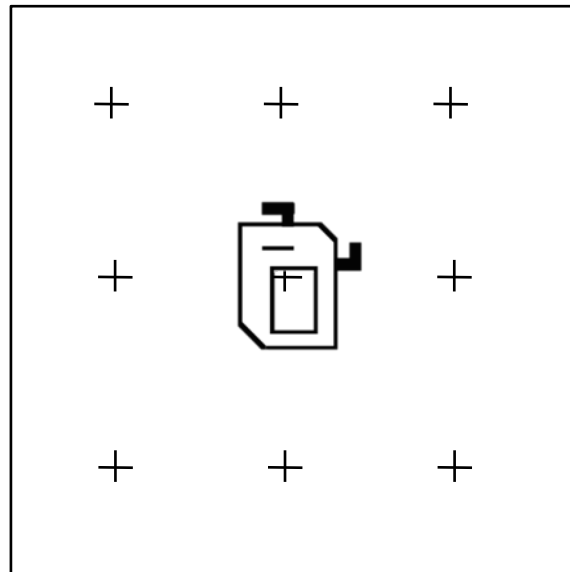
```
def turn_to_wall():  
    while left_is_clear():  
        turn_left()
```





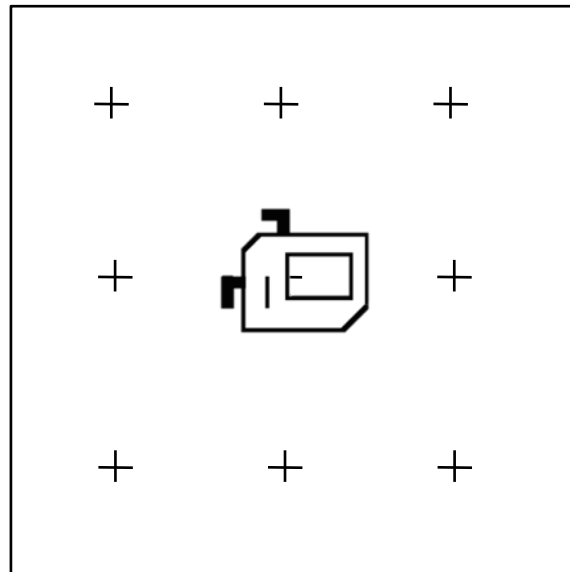
# Infinite Loop

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



# Infinite Loop

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



# Pre/Post that Don't Match

```
def jump_hurdles:
    for i range(8):
        if front_is_clear():
            move()
        else:
            jump_hurdle()
```

What do you  
assume here?

Does the “post  
condition” match?

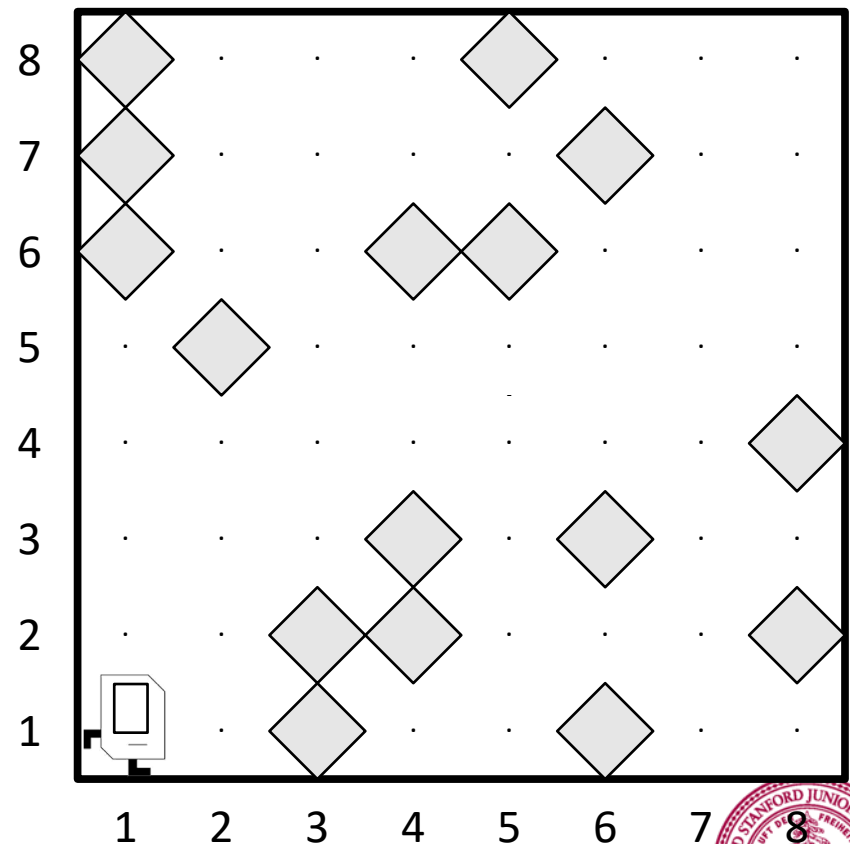


```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhomba_karel()  
    if extra_time():  
        word_search_karel()
```

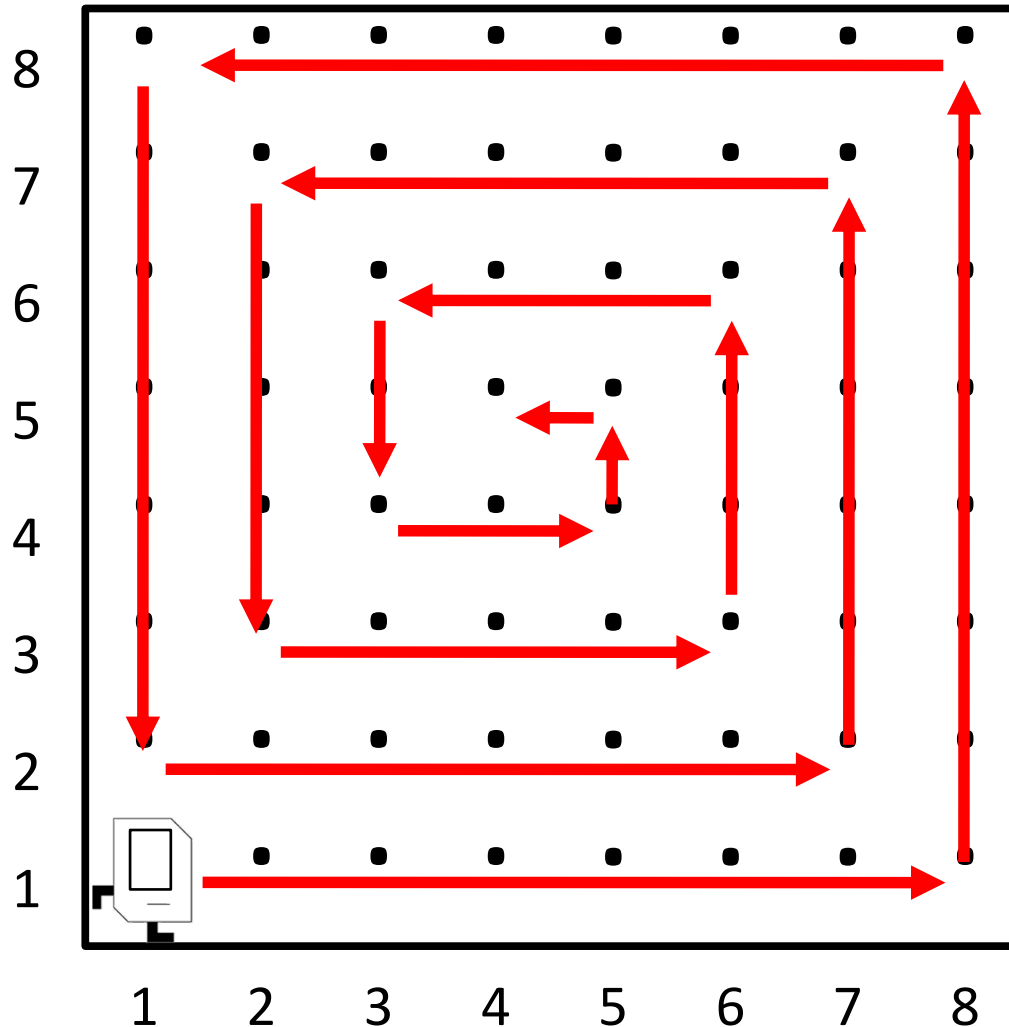
```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhomba_karel()  
    if extra_time():  
        word_search_karel()
```

# Rhoomba Karel

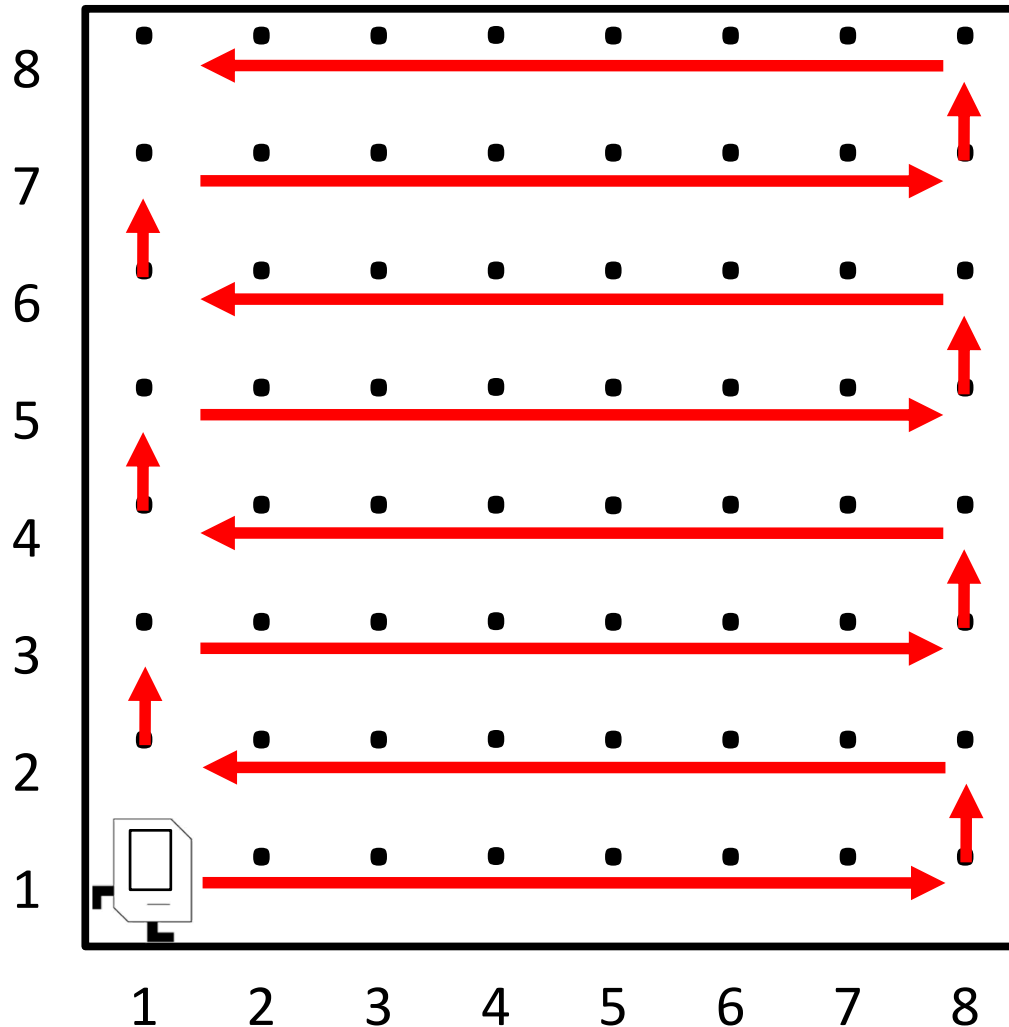
- Write a **Roomba** Karel that sweeps the entire world of all beepers.
  - Karel starts at (1,1) facing East.
  - The world is rectangular, and some squares contain beepers.
  - There are no interior walls.
  - When the program is done, the world should contain 0 beepers.
  - Karel's ending location does not matter.
- How should we approach this tricky problem?



# Possible Algorithm 1

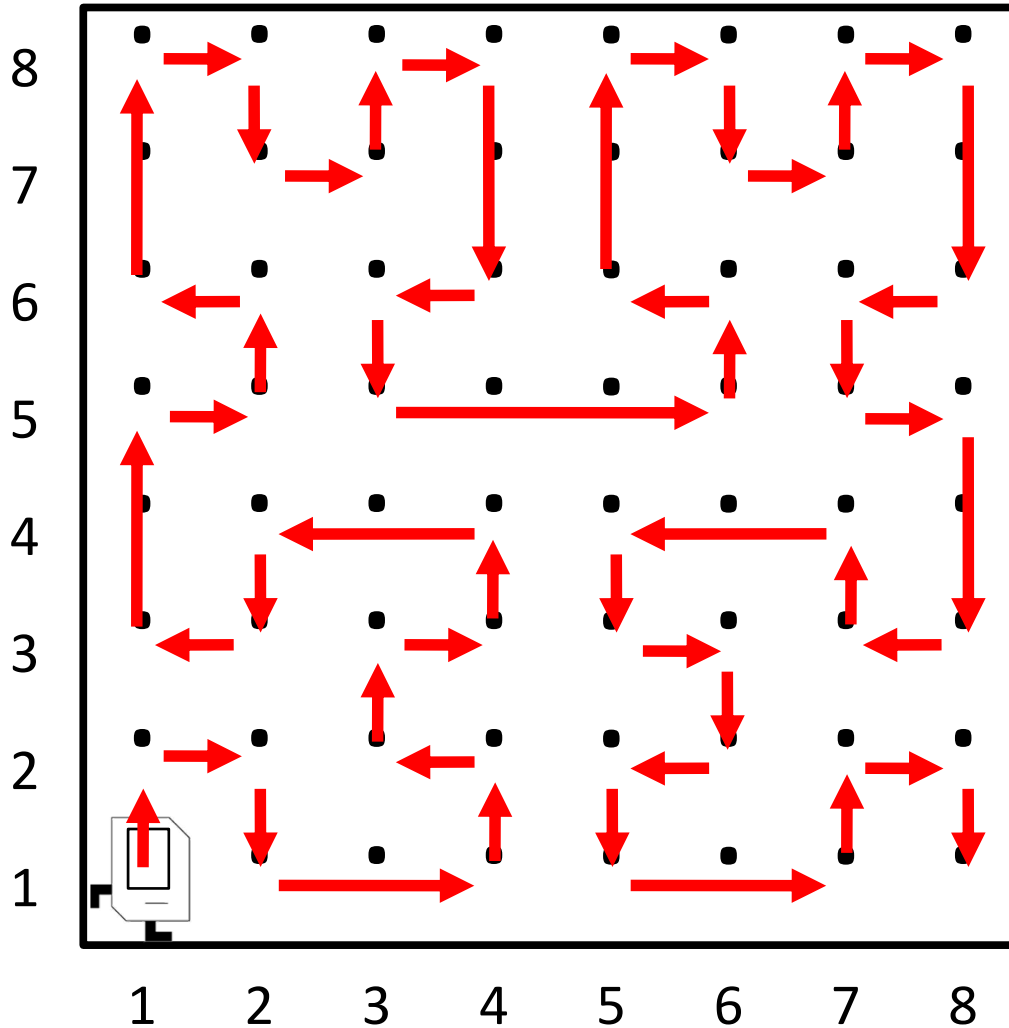


# Possible Algorithm 2

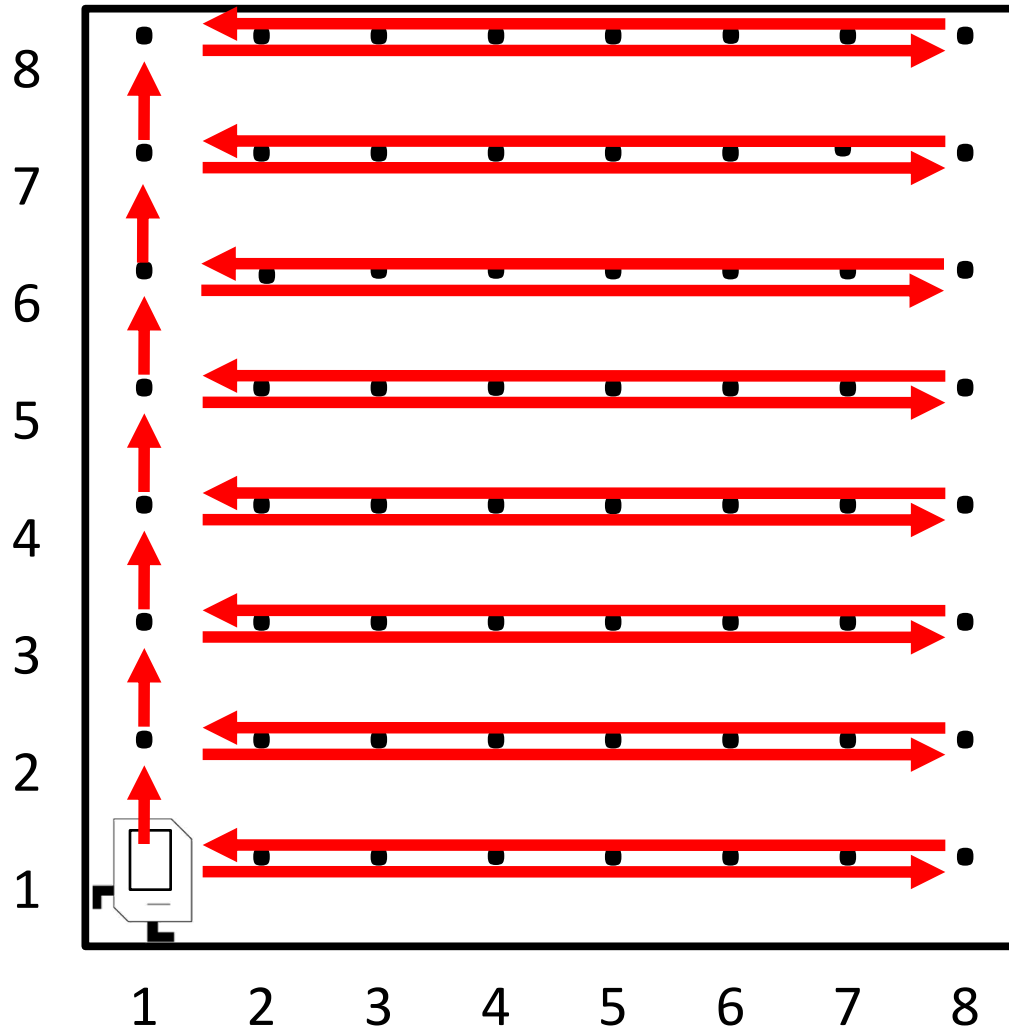




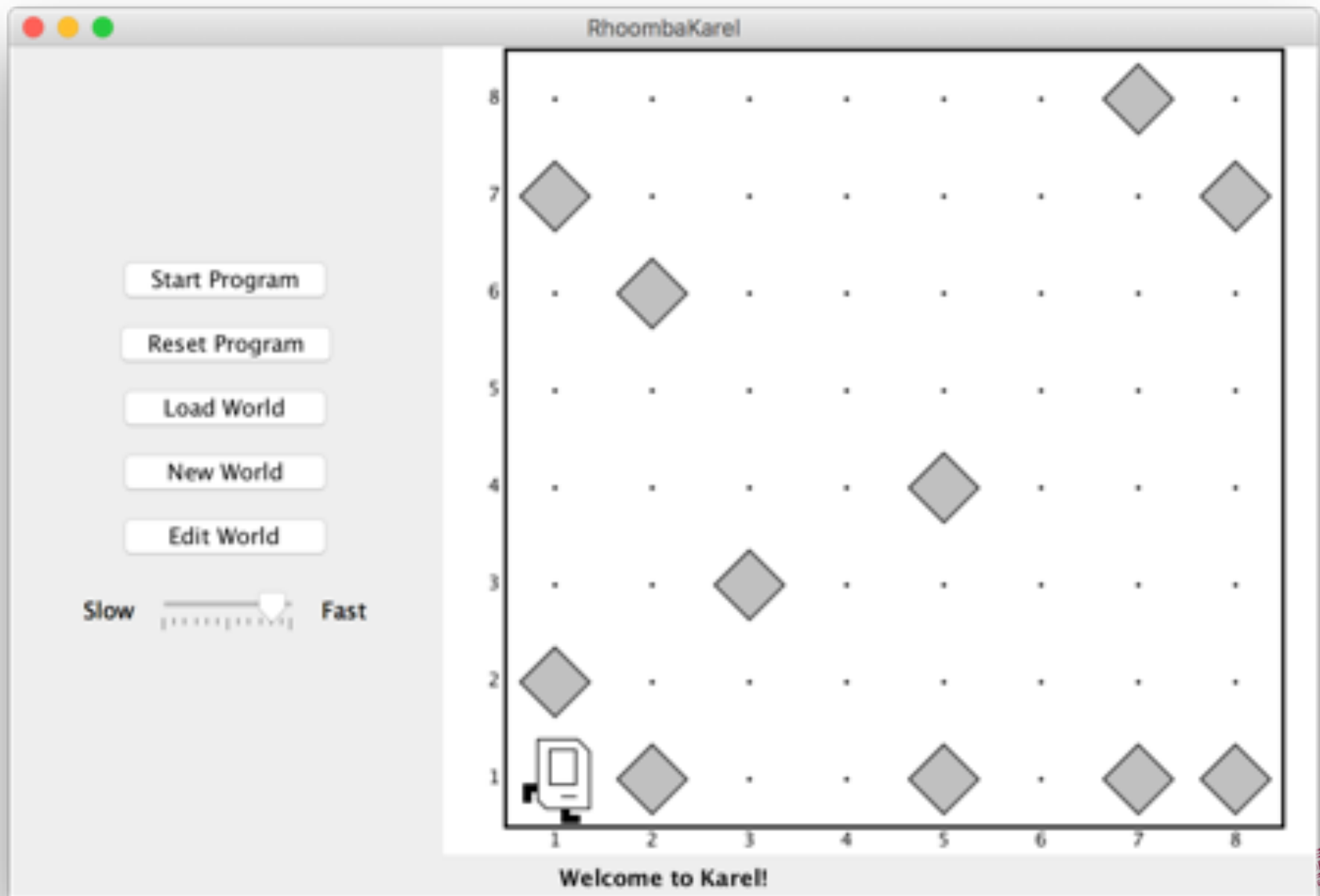
# Possible Algorithm 3



# Possible Algorithm 4



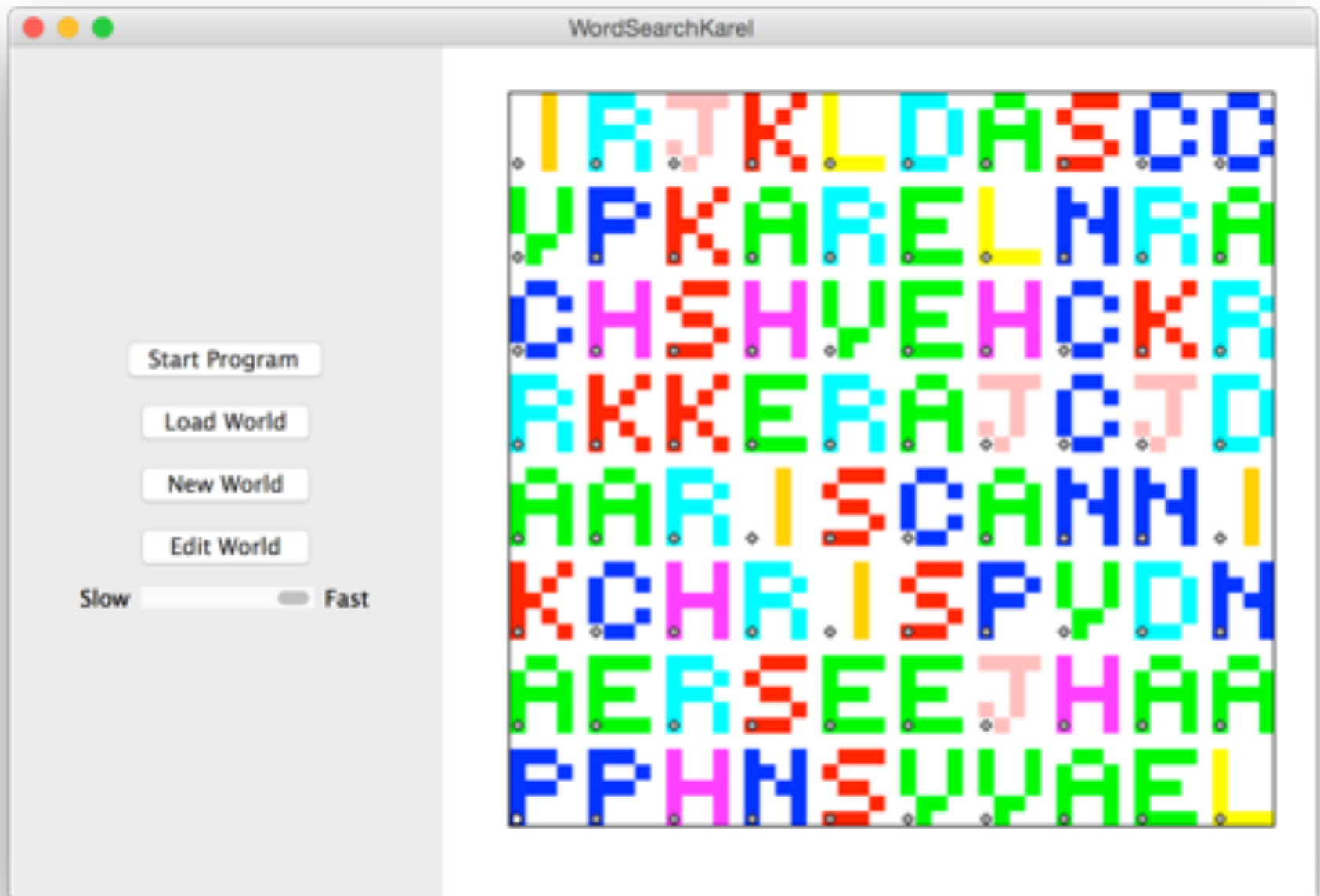
# Rhomba Karel



```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhomba_karel()  
    if extra_time():  
        word_search_karel()
```

```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhoomba_karel()  
    if extra_time():  
        word_search_karel()
```

```
def friday():  
    # heres our plan  
    decomposition()  
    mountain_karel()  
    rhoomba_karel()  
    if extra_time():  
        word_search_karel()
```



Happy Friday