Decomposition

Chris Piech
CS106A, Stanford University

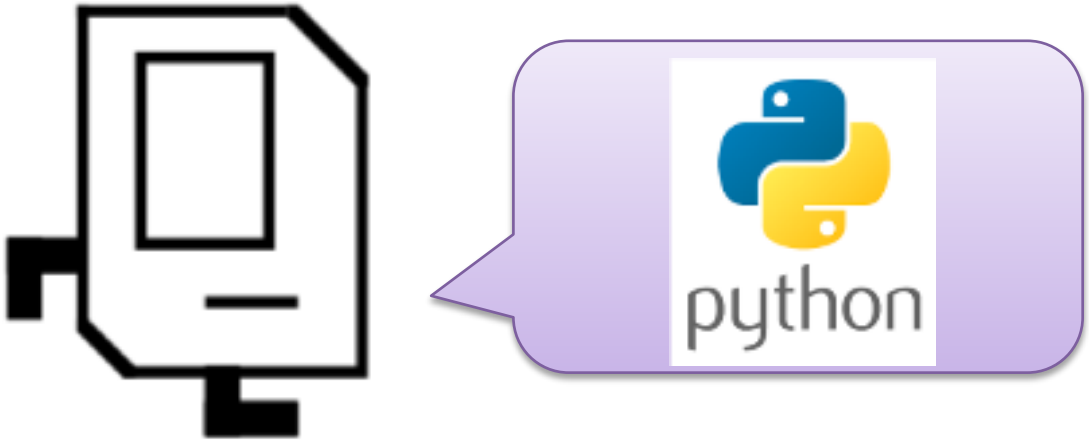
Today's Goal

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

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)
```

End review

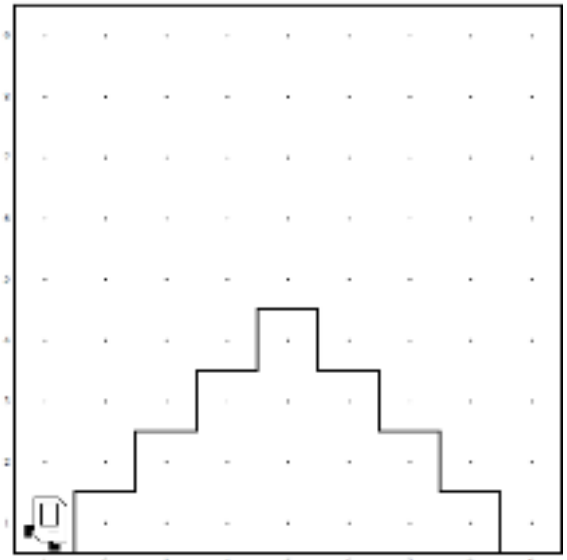
```
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
```

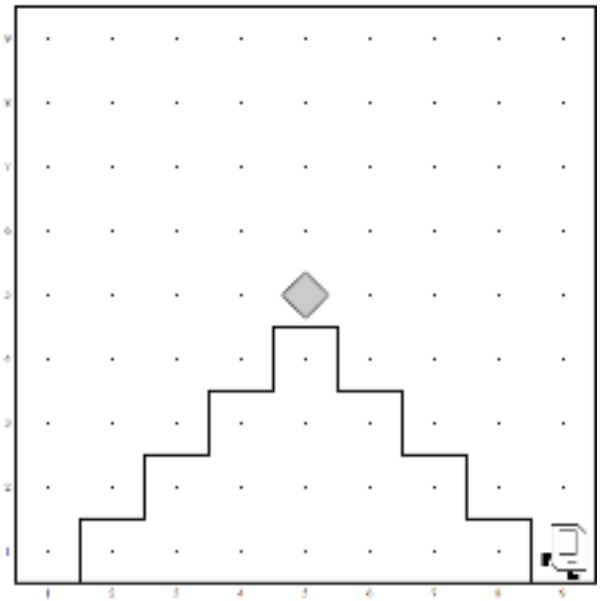
```
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
```

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

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

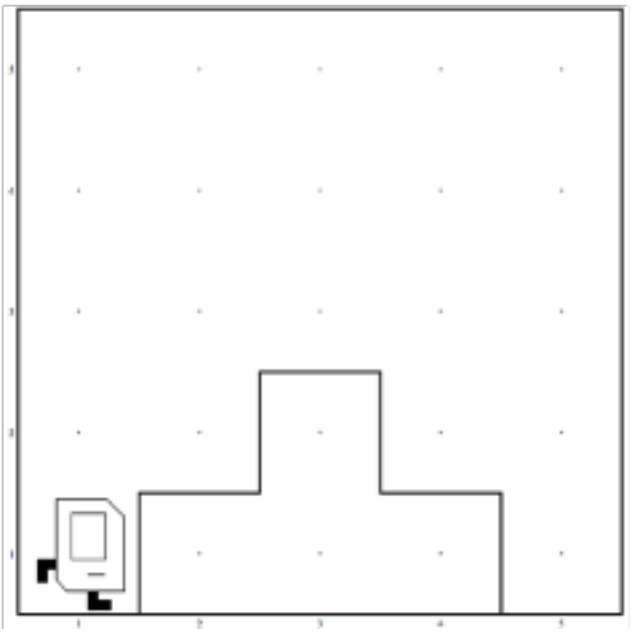
Mountain Karel

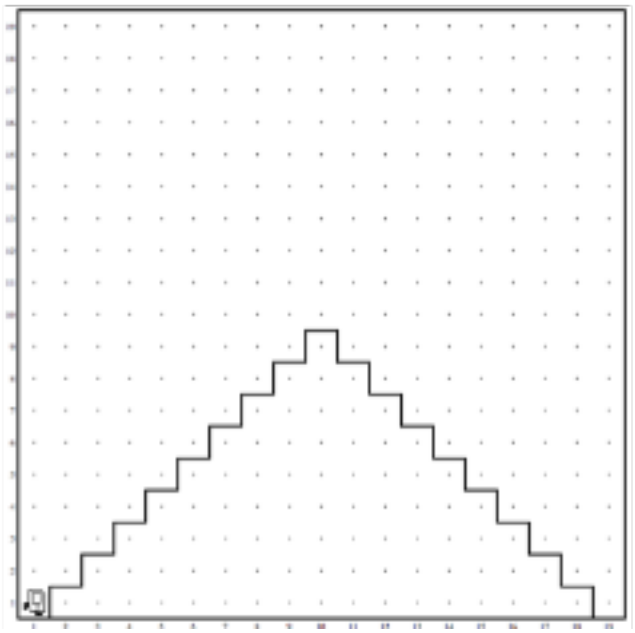


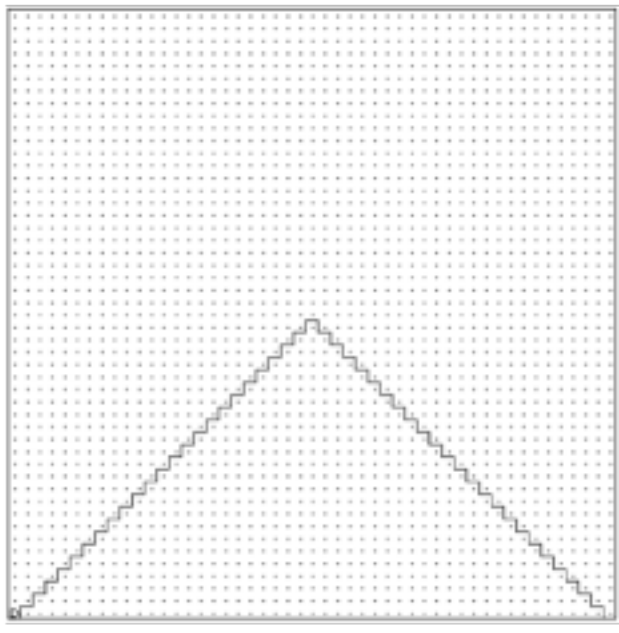


Muhammed ibn
Musa Al Kwarizmi

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

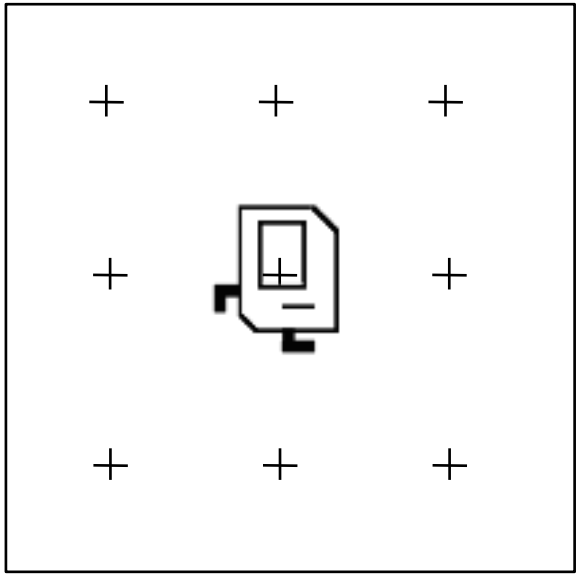
There are two types of programs.
One is so complex, there is nothing obvious wrong with it.
One is so clear, that this obviously nothing wrong with it.

Aside: Common Errors

Lather,
Rinse,
Repeat

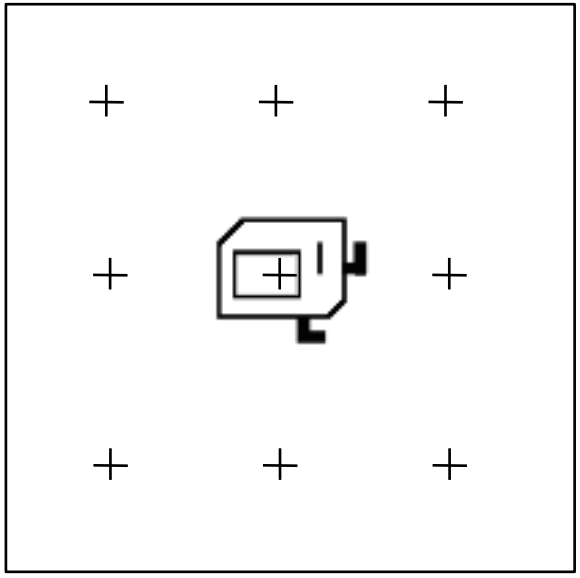
Infinite Loop

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



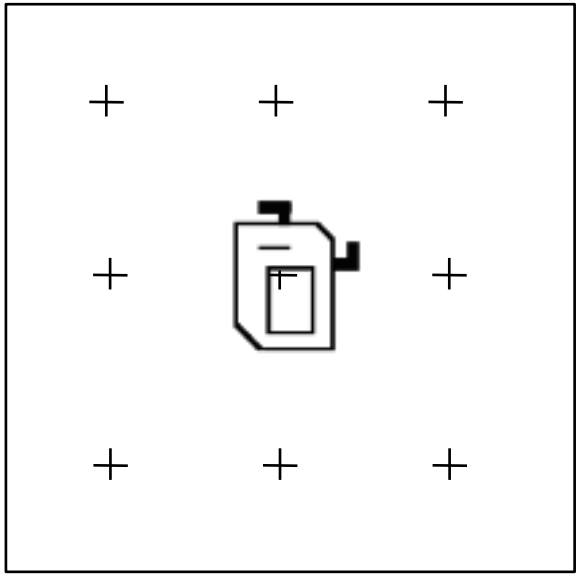
Infinite Loop

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



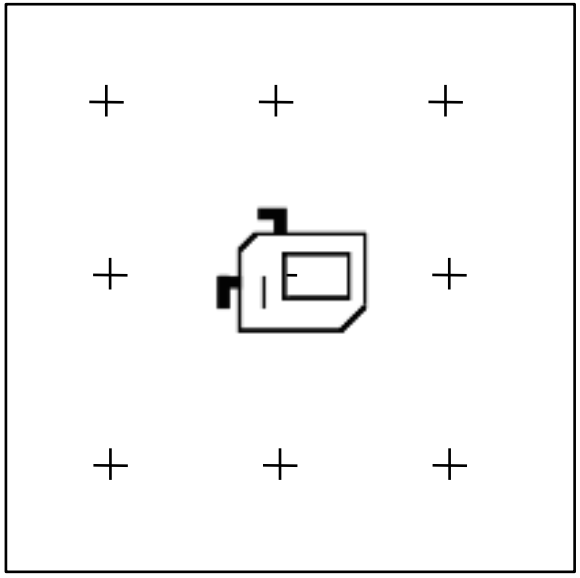
Infinite Loop

```
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?

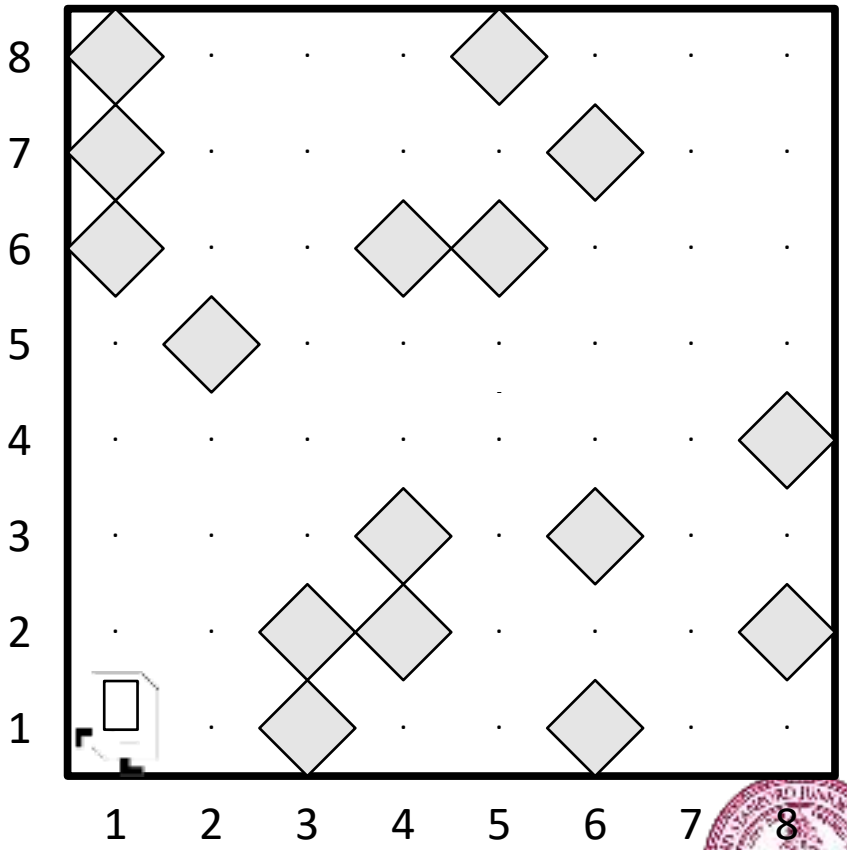
Does the "post condition" match?

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

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

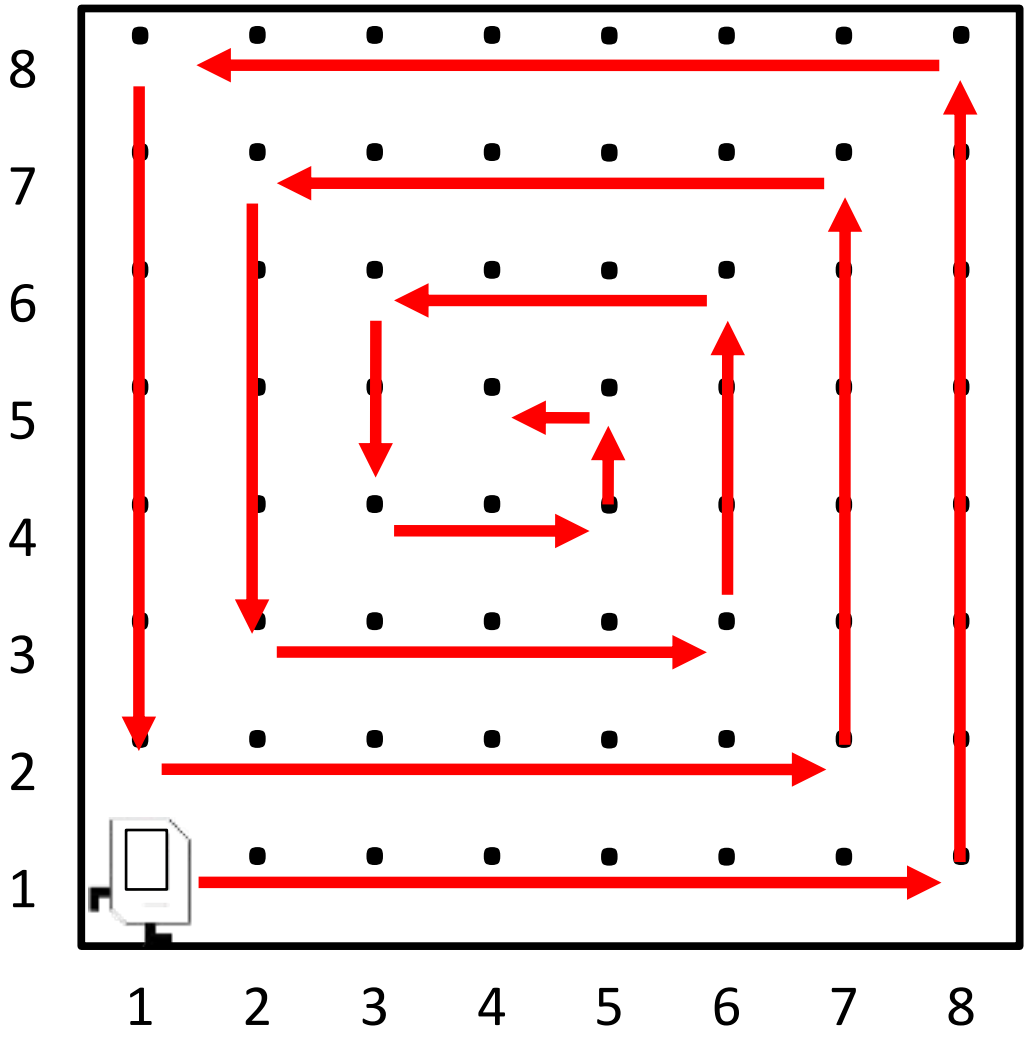
Rhoomba Karel

- 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.

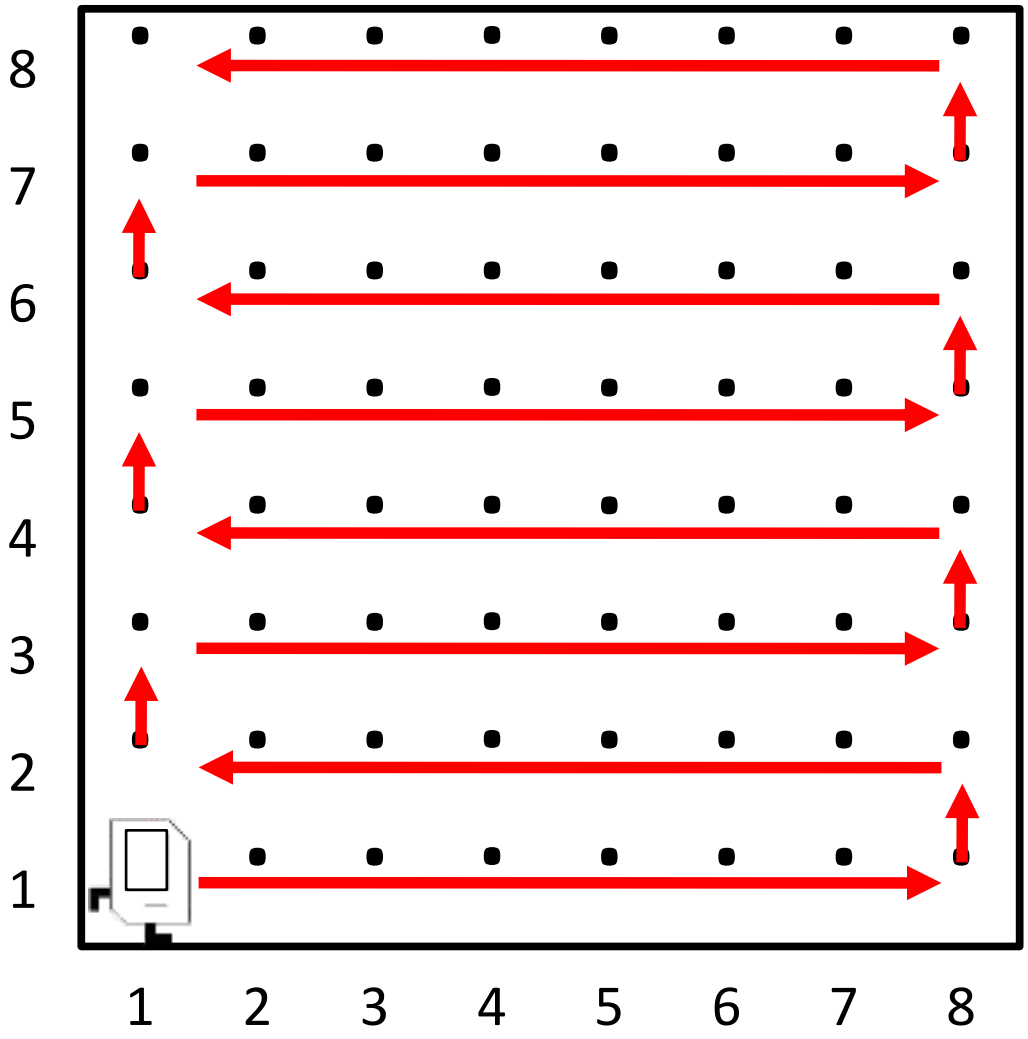


- How should we approach this tricky problem?

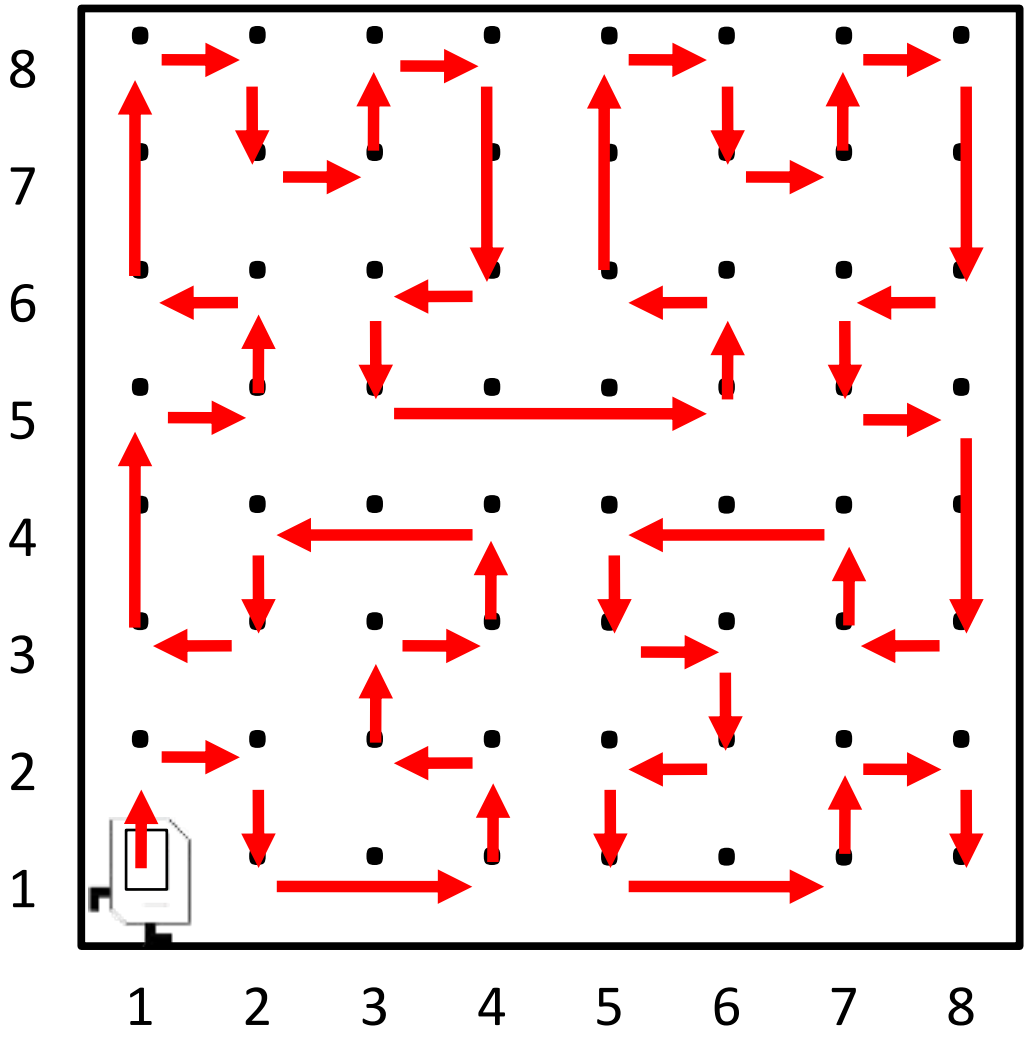
Possible Algorithm 1



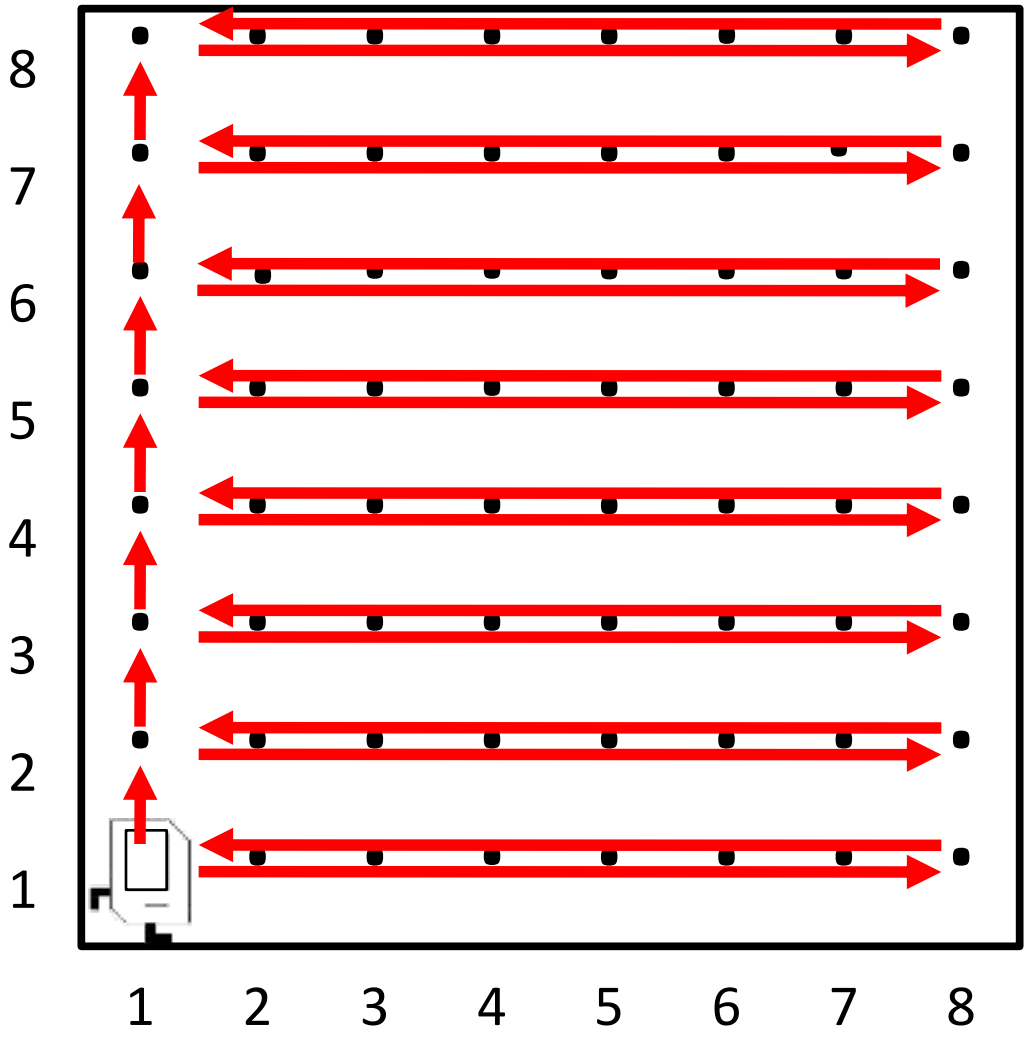
Possible Algorithm 2



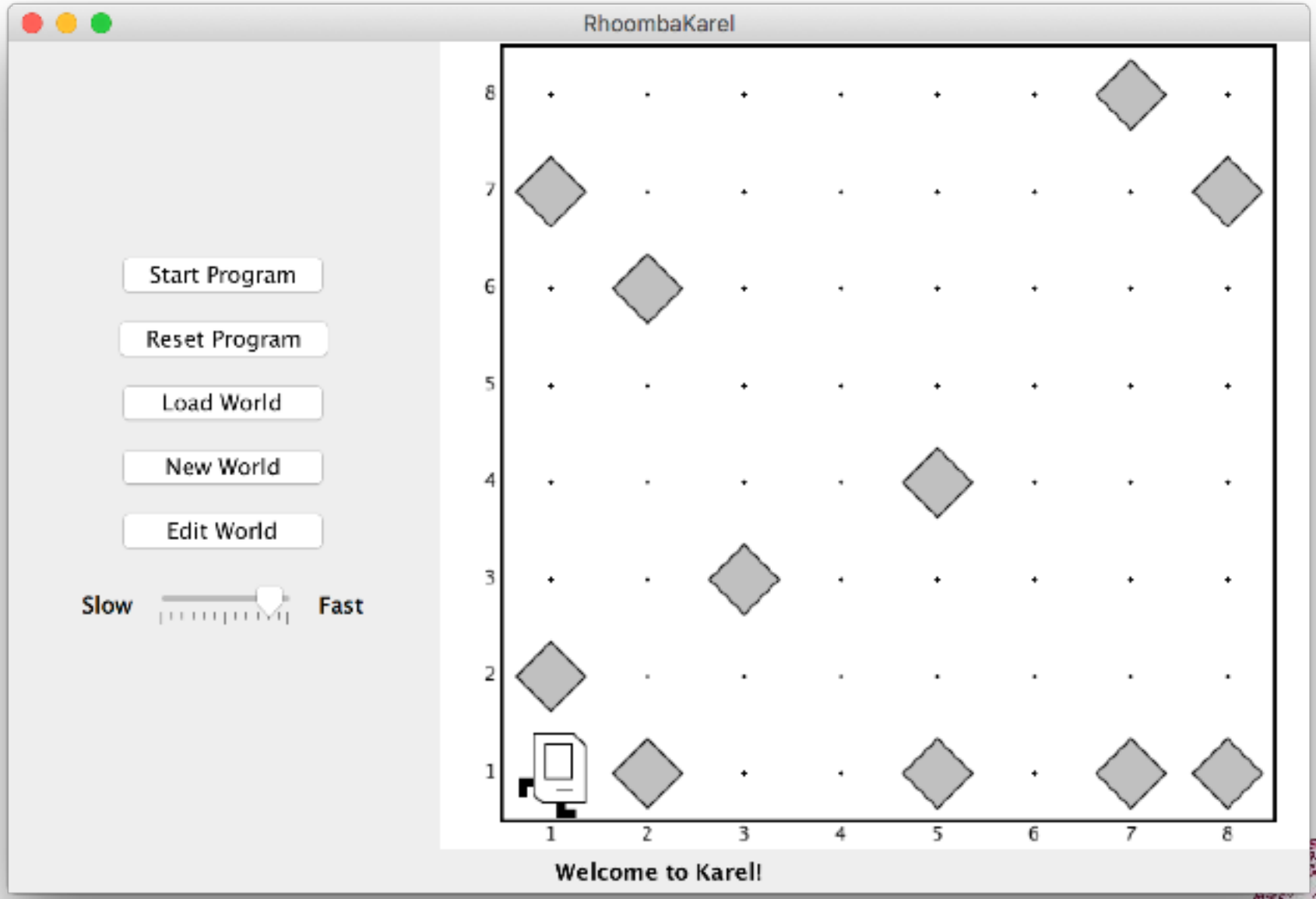
Possible Algorithm 3



Possible Algorithm 4



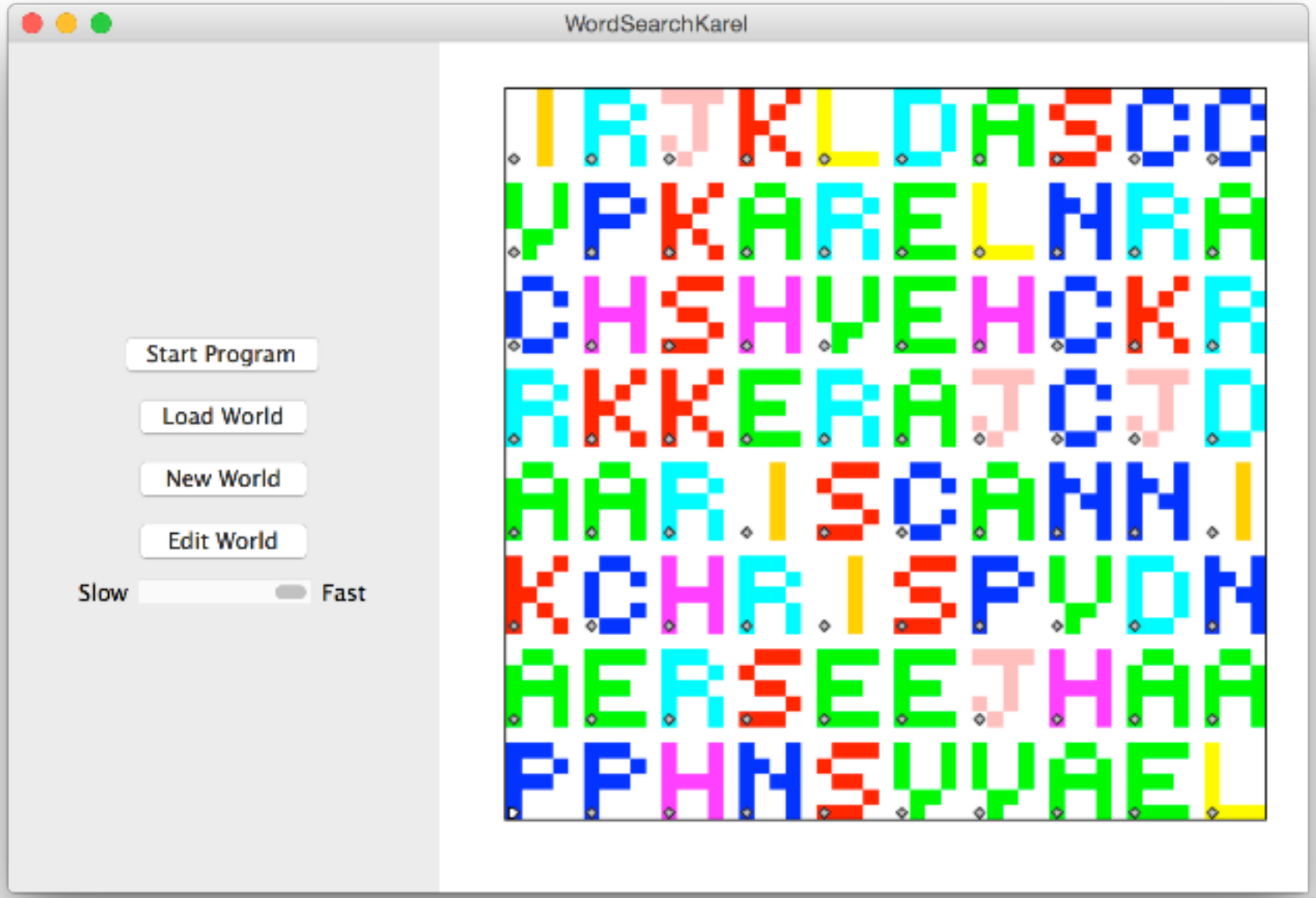
Rhoomba Karel



```
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()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
```


WordSearchKarel
Start Program
Load World
New World
Edit World
Slow
Fast

Happy Friday