Welcome!

We'll get started shortly ...

Welcome to Section!

Week Two

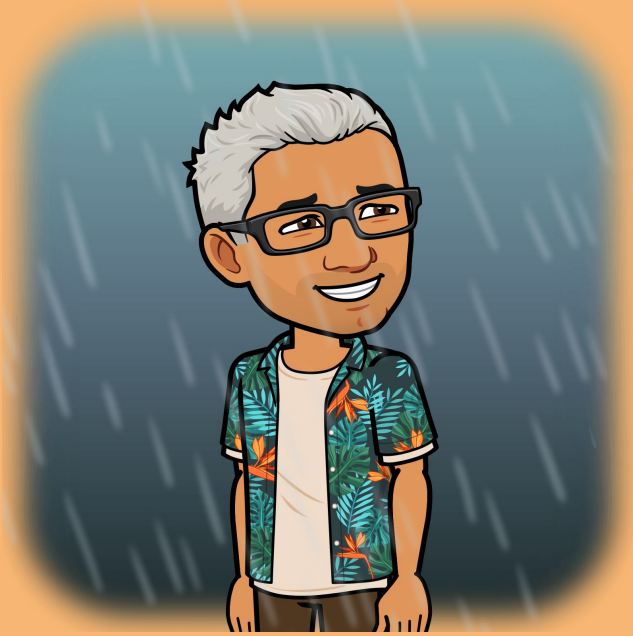
Surajit Bose

Agenda

- Introductions and Logistics
- Review of lecture concepts
  - Karel the robot
  - Control Flow: `for`, `while`, `if`, `if-else`
  - Functions
- Worked Example
- Problem: Hospital Karel

Introductions

A Little About Me



- ★ My name is Surajit (he/him), and I'll be your Section Leader for CS 49
- ★ I am retired from a tech career
- ★ In my free time I like to read novels and poetry
- ★ I enjoy Indian classical music

What About You?

Please let us know:

1. Your name, and if you would like to share them, your pronouns
2. Where you're tuning in from: locally from the Bay Area, or farther away?
3. What you'd like to get out of this class

Another question: Is there something I can do to help you feel comfortable during section? Please feel free to private message me in the chat if so!

Breaking the Ice

- Share your names one more time!!!
- Icebreaker Question:
  - What is your favorite home-cooked dish?
  - Who makes it?
- If no one wants to share first, the person who is geographically closest to Foothill shares first!
- After your turn, you get to "popcorn" to someone, i.e., choose who goes next

How to get hold of me / get help+

- The class forum or section forum, 24 hr turnaround
  - Feel free not only to ask, but also to answer questions there!
- Surajit's office hours:
  - Tuesdays 1p–2p on Zoom
  - By appointment on Zoom or on campus
- Lane's office hours
- Canvas inbox for Lane or Surajit
- Email bosesurajit@fhda.edu, 24 hr turnaround
- Online or in-person tutoring via the STEM center (Room 4213)
- The section GitHub repo has lecture and section slides and solutions

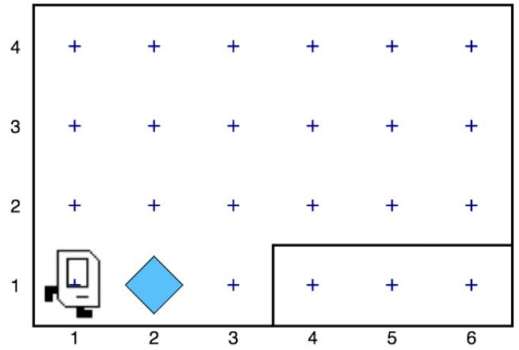
Lecture Review: Karel

Karel the Robot

- Karel is a robot who occupies a certain **world**
  - The world has certain elements like corners, beepers, and walls that Karel interacts with or navigates
- Karel and its world have certain **conditions**
  - Conditions are statements that can be either `True` or `False`
  - In our world, for example, one condition can be "it is raining"
  - Determining whether a condition is `True` or `False` is called **evaluating** the condition
- Karel can perform certain **commands** to navigate its world
- Let's look at Karel's world, conditions, and commands in the next few slides

Karel's World

Karel's world is defined by rows running horizontally (east-west) and columns running vertically (north-south). The intersection of a row and a column is called a corner. Karel can only be positioned on corners and must be facing one of the four standard compass directions (north, south, east, west). A sample Karel world is shown below. Here Karel is located at the corner of 1st row and 1st column, facing east.



Several other components of Karel's world can be seen in this example. The object in front of Karel is a beeper. As described in Rich Pattis's book, beepers are "plastic cones which emit a quiet beeping noise." Karel can only detect a beeper if it is on the same corner. The solid lines in the diagram are walls. Walls serve as barriers within Karel's world. Karel cannot walk through walls and must instead go around them. Karel's world is always bounded by walls along the edges, but the world may have different dimensions depending on the specific problem Karel needs to solve.

Karel Conditions

Test	Opposite	What it checks
`front_is_clear()`	`front_is_blocked()`	Is there a wall in front of Karel?
`beepers_present()`	`no_beepers_present()`	Are there beepers on this corner?
`left_is_clear()`	`left_is_blocked()`	Is there a wall to Karel's left?
`right_is_clear()`	`right_is_blocked()`	Is there a wall to Karel's right?
`beepers_in_bag()`	`no_beepers_in_bag()`	Does Karel have any beepers in its bag?
`facing_north()`	`not_facing_north()`	Is Karel facing north?
`facing_south()`	`not_facing_south()`	Is Karel facing south?
`facing_east()`	`not_facing_east()`	Is Karel facing east?
`facing_west()`	`not_facing_west()`	Is Karel facing west?

Karel Commands

Command	Description
`move()`	Asks Karel to move forward one block. Karel cannot respond to a `move()` command if there is a wall blocking its way.
`turn_left()`	Asks Karel to rotate 90 degrees to the left (counterclockwise).
`pick_beeper()`	Asks Karel to pick up one beeper from a corner and stores the beeper in its beeper bag, which can hold an infinite number of beepers. Karel cannot respond to a `pick_beeper()` command unless there is a beeper on the current corner.
`put_beeper()`	Asks Karel to take a beeper from its beeper bag and put it down on the current corner. Karel cannot respond to a `put_beeper()` command unless there are beepers in its beeper bag.

Lecture Review: Control Flow

Control Flow: Loops, Conditionals

`for` loop:

- Performs some block of code a specific number of times.

`while` loop:

- Repeatedly performs a block of code until a given condition is evaluated to `False`

`if` statement:

- Performs a block of code only when a condition is `True`, and only once.

`if-else` statement:

- Performs a block of code when a condition is `True`, or a different block when the condition is `False`. Either block is performed only once.

Python Loops

`for` Loop

- An example `for` loop that you may see and use with Karel:

```
def turn_right():
    for i in range(3):
        turn_left()
```

- **`i`** is a conventional name for the counter, from integer
- But you can call the counter anything you like, even **`x`** or **`bob`** or **`emma`**
- This loop is also called a *definite loop* because we know where it ends, when **`i`** reaches 3.

`while` Loop

- An example `while` loop that you may see and use with Karel:

```
def move_to_wall():
    while front_is_clear():
        move()
```

- This loop is also called an *indefinite loop* because it will run until the associated condition becomes `False`.

Python Conditionals

`if` Statement

- An example `if` statement that you may see and use with Karel:

```
def safe_move():
    if front_is_clear():
        move()
```

- An `if` statement runs code indented inside of it when the associated condition evaluates to `True`.

`if-else` Statement

- An `if-else` statement runs one of two blocks of code:
  - Either the code inside the `if` block when the associated condition evaluates to `True`,
  - Or the code inside the `else` block when the condition evaluates to `False`.

```
def safe_move_or_turn():
    if front_is_clear():
        move()
    else:
        turn_left()
```

Any Questions?

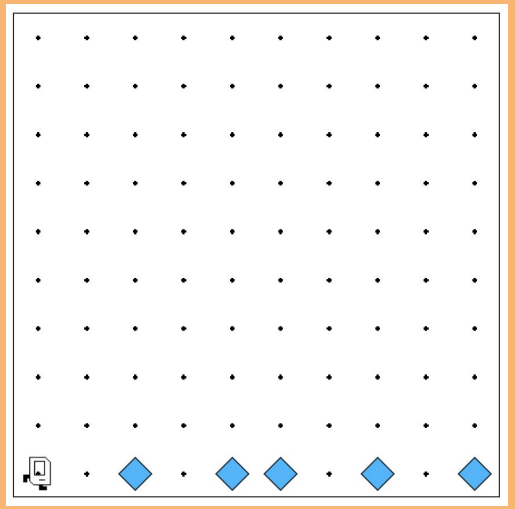
Lecture Review: Functions

Functions

- The process of breaking down a problem into smaller, self-contained building blocks is **decomposition**
- These smaller building blocks are **functions**
  - A function is a sequence of steps that achieves a specific outcome
  - Any set of steps that needs to be repeated could be made a function
  - So could any logically self-contained portion of the problem
- How to decompose a problem and write functions:
  - Identify the building blocks
  - Assemble blocks in `main()` to solve the big problem
  - Assume the building blocks are done (use the `pass` keyword)
  - Implement each building block!

Functions

Example: Karel needs to walk from the first corner `[1,1]` to the end of the row. Every time Karel lands on a beeper, it needs to spin 360º .



Big picture:

- Karel starts on `[1,1]`
- It moves until it is on a corner with a beeper.
- It spins, then moves forward again.
- This process continues until Karel reaches a wall.

What is the small building block that will be useful?

- What action does Karel not yet know how to do, but will need to do repeatedly?

Functions

```
spin()
```

Any Questions?

Worked Example

Using functions and control flow to make Karel spin when it is on a beeper

https://codeinplace.stanford.edu/foothill-cs49/ide/p/UWYFIAF12e6QZxdXoQg6

Solve the big problem

```
def main():
    while front_is_clear():
        move()
        if beepers_present():
            spin()

def spin():
    pass     # Placeholder
```

Implement the building blocks

```
def main():
    while front_is_clear():
        move()
        if beepers_present():
            spin()

def spin():
    for i in range(4):
        turn_left()
```

Let's Try It Out!

Did it work?

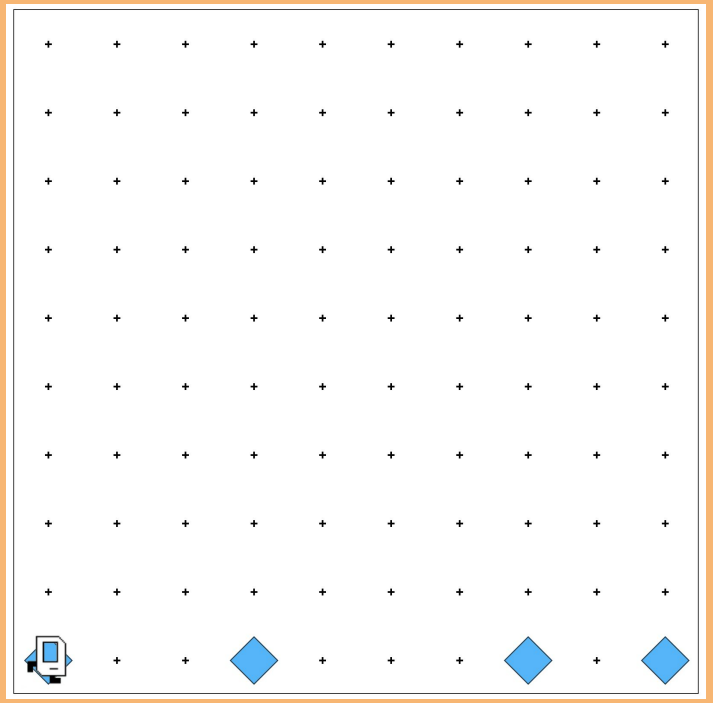
Test and refine the entire solution

```
def main():
    if beepers_present():      #  Fencepost problem: there
        spin()                 #  could be a beeper at [1,1]
    while front_is_clear():
        move()
        if beepers_present():
            spin()

def spin():
    for i in range(4):
        turn_left()
```

`spin()` in action

You can see Karel spinning as desired here. The implementation uses the code on the previous slide.



Any Questions?

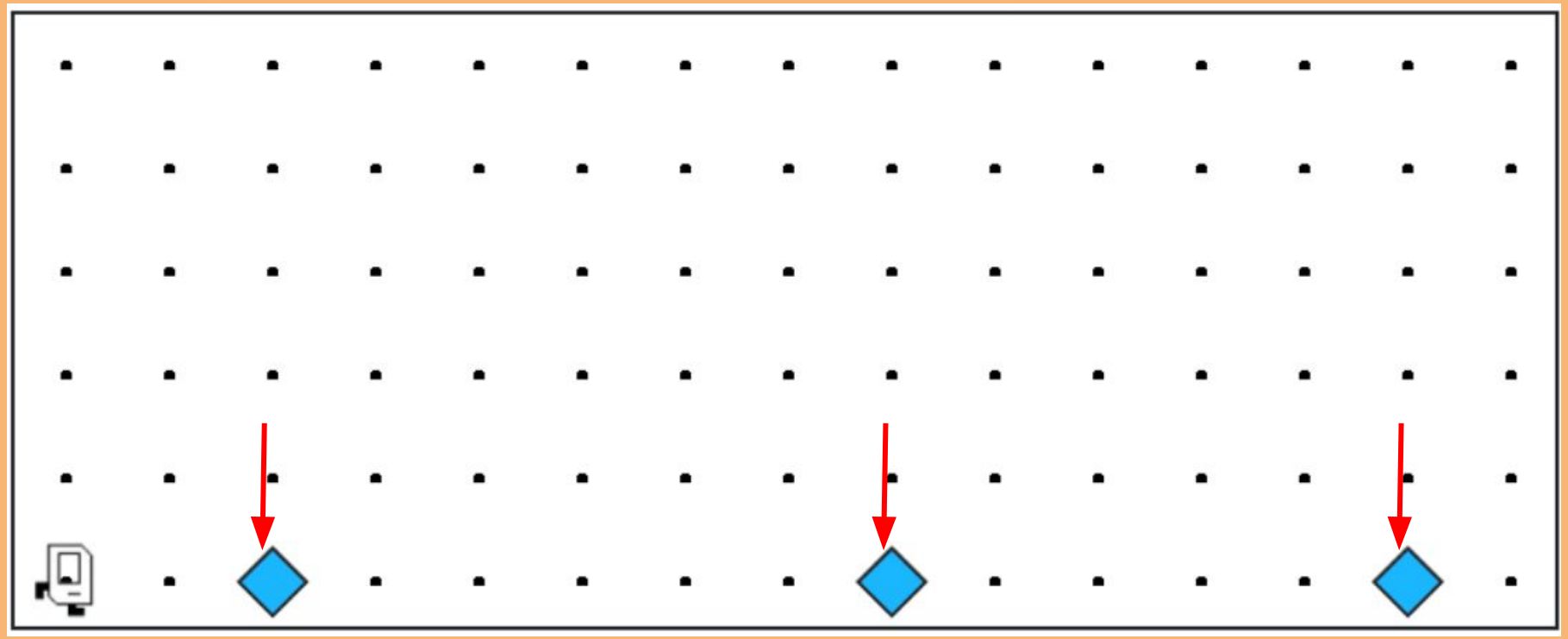
Section Problem: Hospital Karel

https://codeinplace.stanford.edu/foothill-cs49/ide/a/hospital

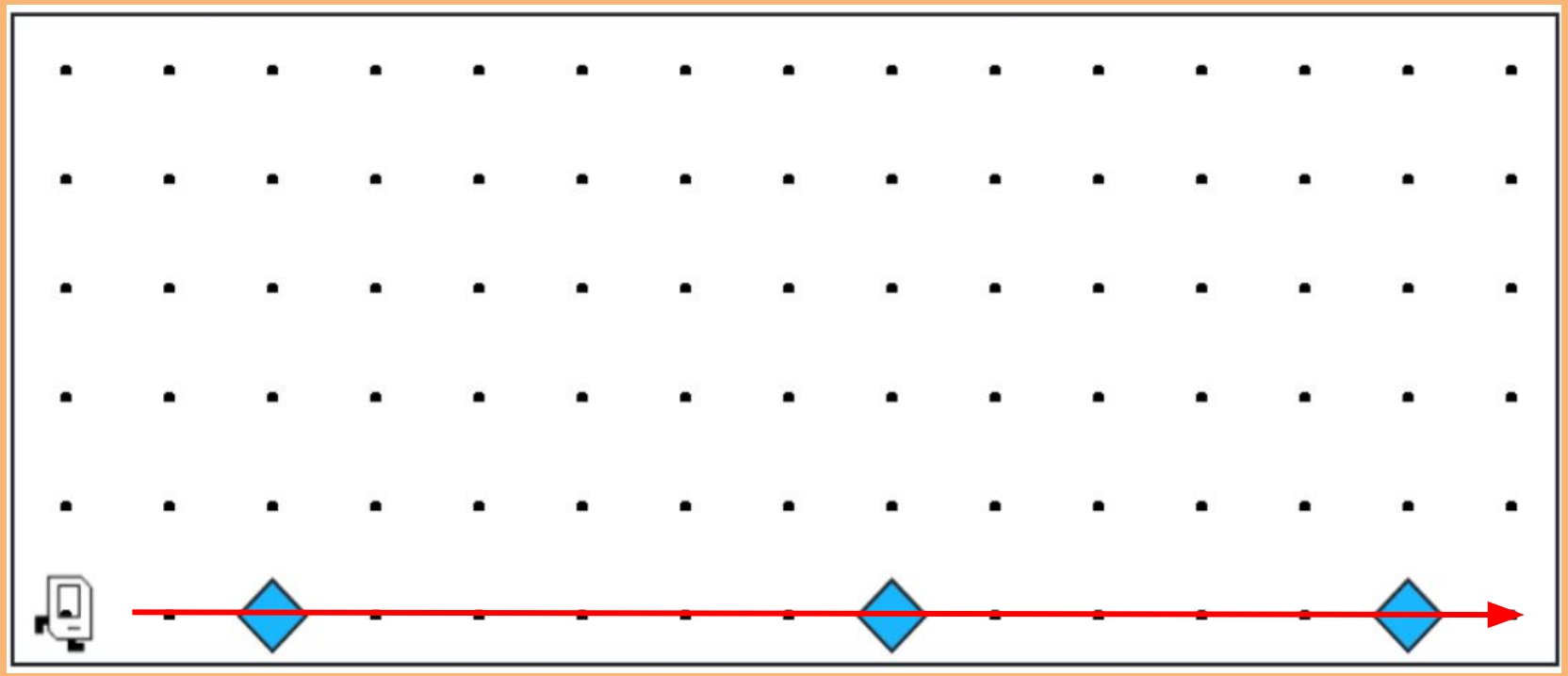
Setting Context

Countries around the world are dispatching hospital-building robots to make sure anyone who gets sick can be treated. They have decided to enlist Karel robots. Your job is to program those robots.

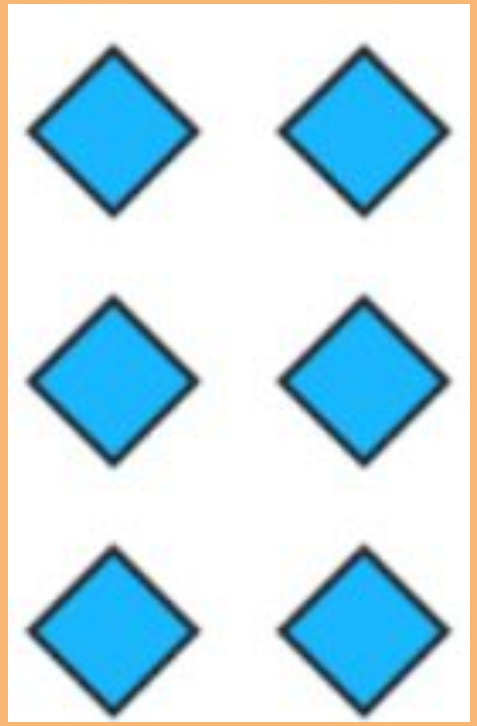
Each beeper in the figure represents a location



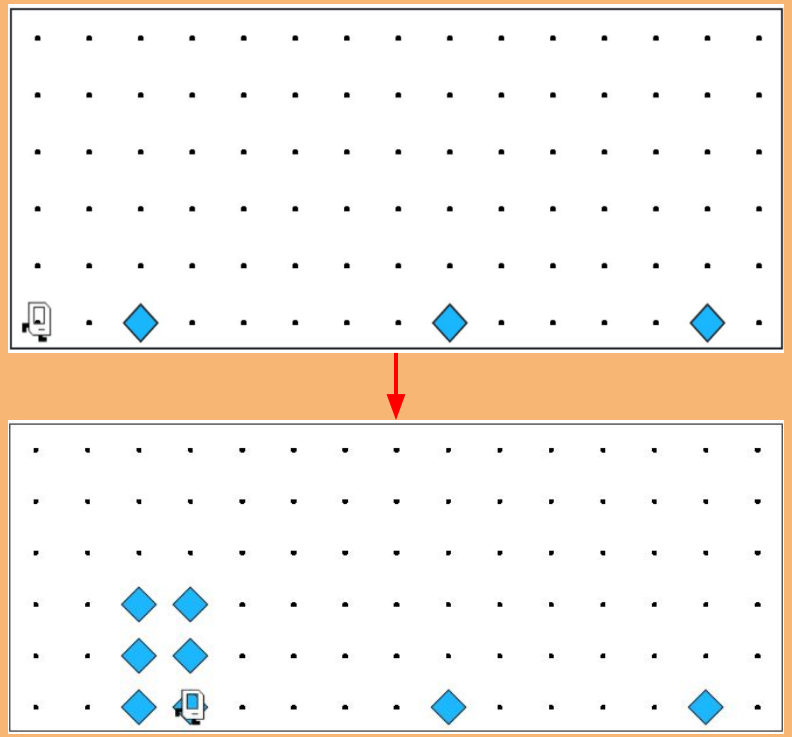
Karel must walk the row and build a hospital at each location



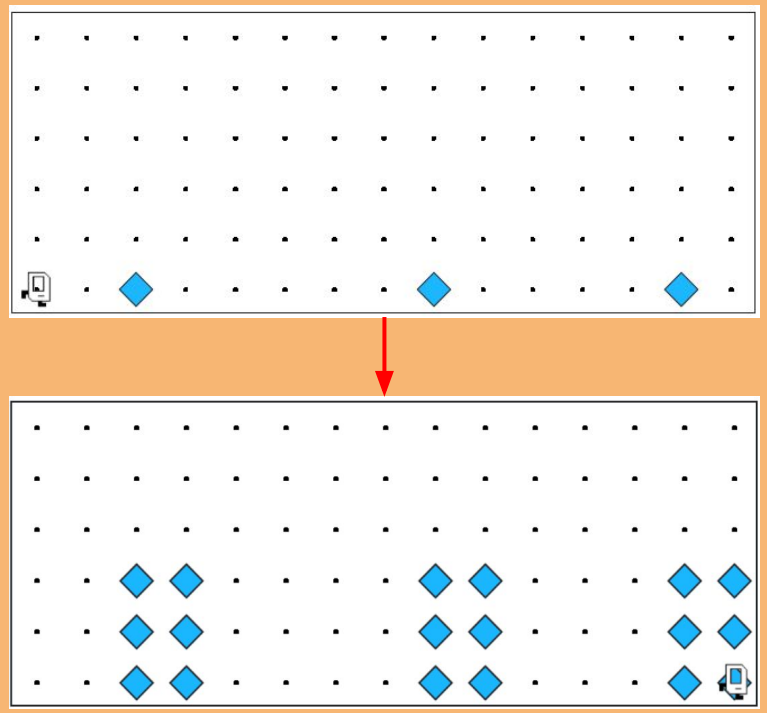
Hospitals look like this: a 3x2 rectangle of beepers!



Here is the state after Karel has built the first hospital



At the end of the run, Karel should be at the end of the row having created a set of hospitals. For the initial conditions shown, the result would look like this:



Notes to Keep in Mind

- Karel starts facing east at `[1, 1]` with an infinite number of beepers in its beeper bag
- The beepers indicating the positions at which hospitals should be built will be spaced so that there is room to build the hospitals without overlapping or hitting walls
- There will be no supplies left on the last column
- Karel should not run into a wall if it builds a hospital that extends into that final corner
- Remember to decompose the problem: identify the building blocks!

Questions Before We Begin?

Let's get to work!