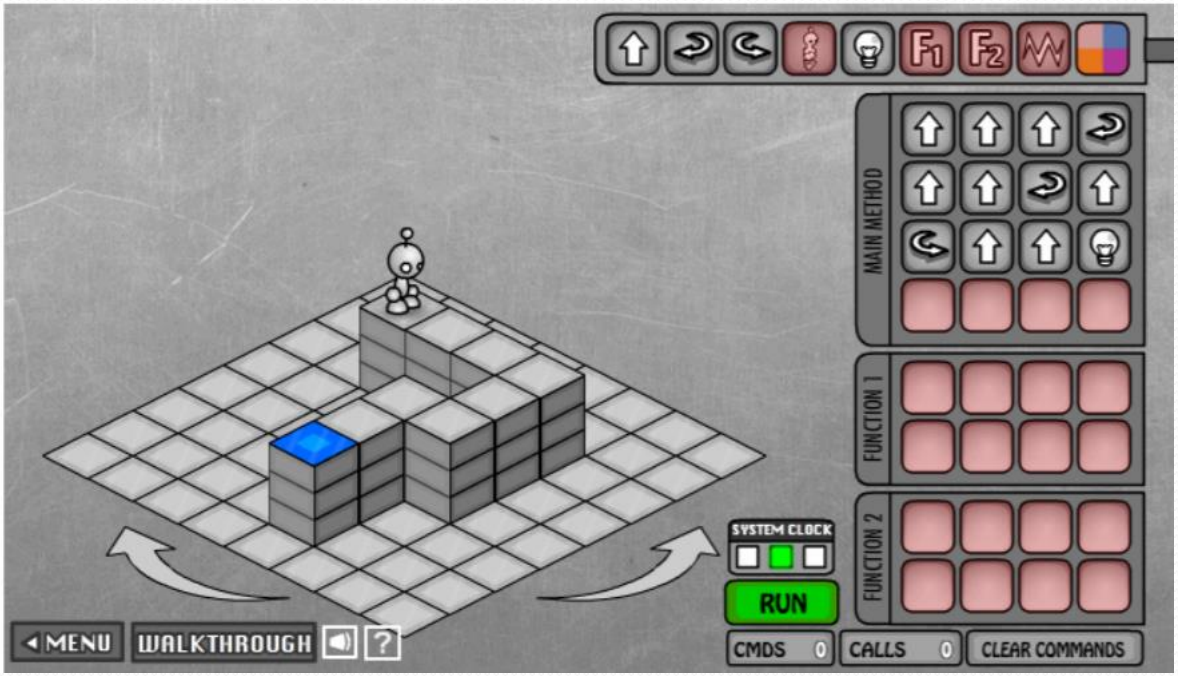
CS 115, Autumn 2020

Lecture 2: Light-Bot

Thank you to Marty Stepp, Stuart Reges and Larry Snyder for parts of these slides

Light-Bot

- What are you doing in Light-Bot?



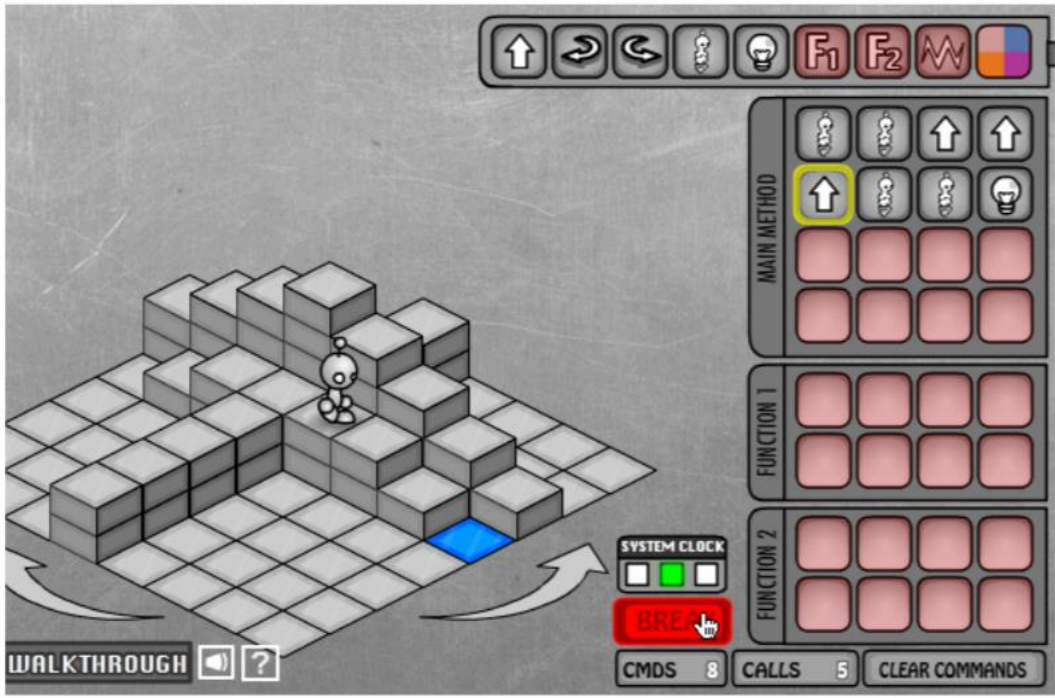
- Commanding a robot through a “blocks world”
- Programming is commanding an agent

Light-Bot and programming

- When you are commanding (programming), you direct an agent (by instructions) to a goal
  - The **agent** is usually a computer, but it can be a person, or other device (animated robot)
  - The agent follows the commands (**instructions**), flawlessly, and mindlessly, doing only what it is asked
  - The program implements **human intent** – you are trying to get the robot to the Blue Tile goal – it's the point of your instructions

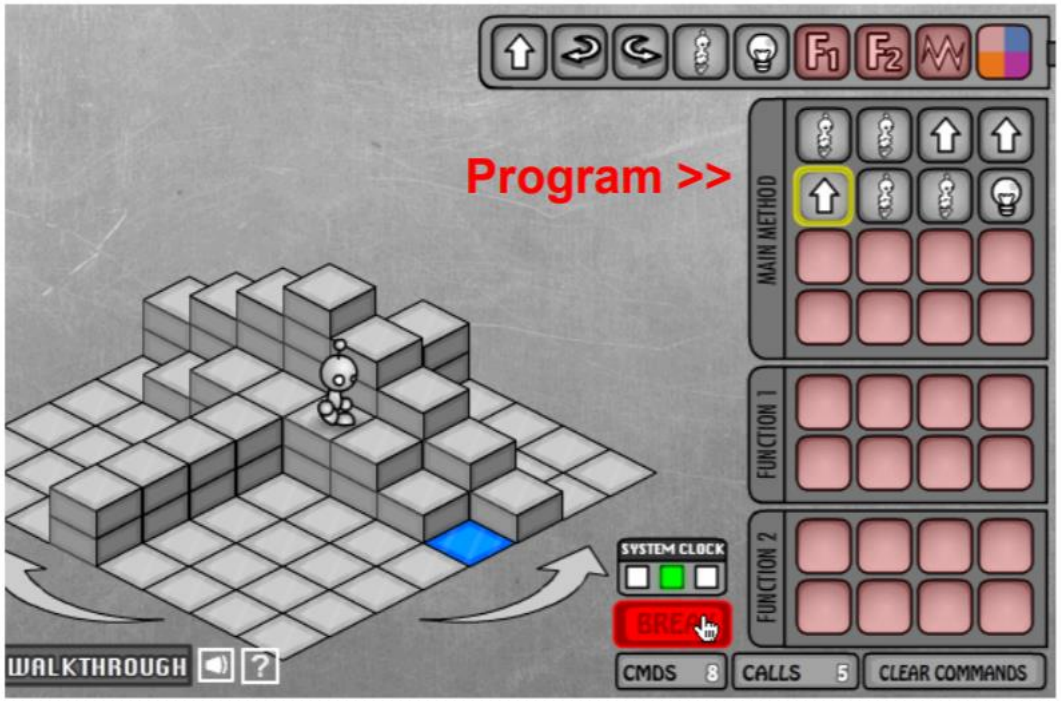
Sequencing

- Instructions are given in sequence (order matters)
- They are followed in sequence
  - **programming** : you giving instructions
  - **executing** or **running** : the agent following the instructions
  - **program counter** : marks the agent's place



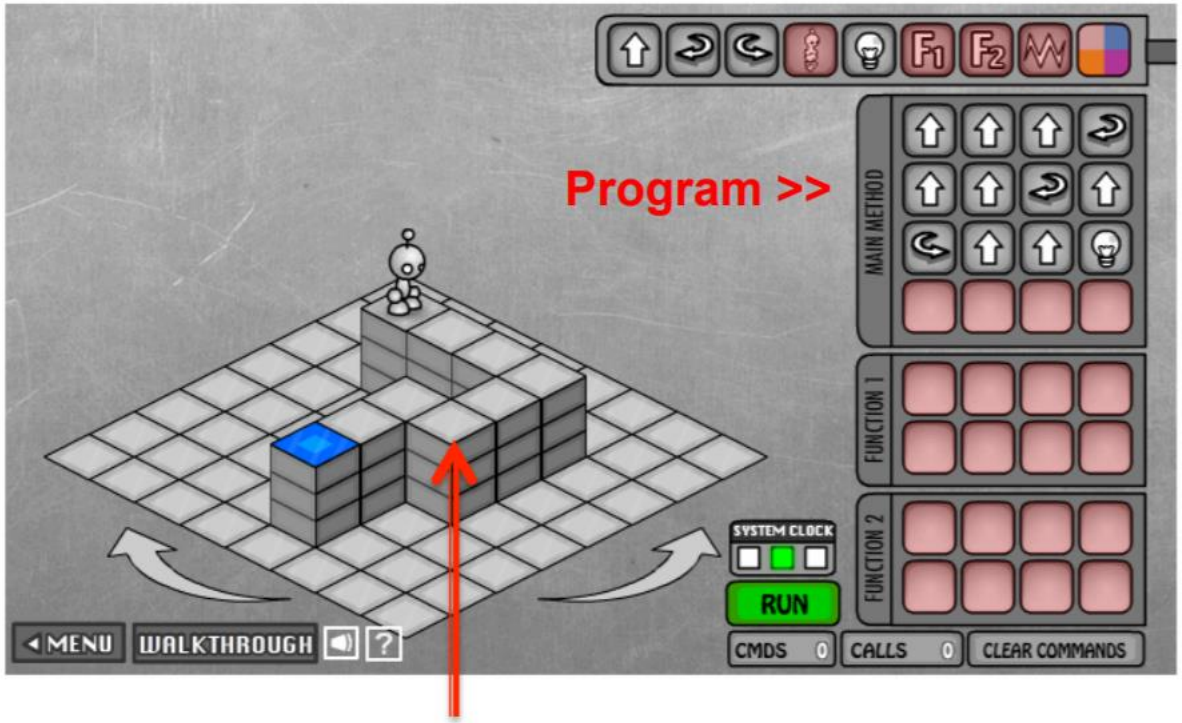
Order of Events

- The instructions are programmed ahead of time
- They are executed later, without programmer's intervention
  - Each instruction makes progress towards the goal
  - The instructions must be right and sufficient to achieve the goal



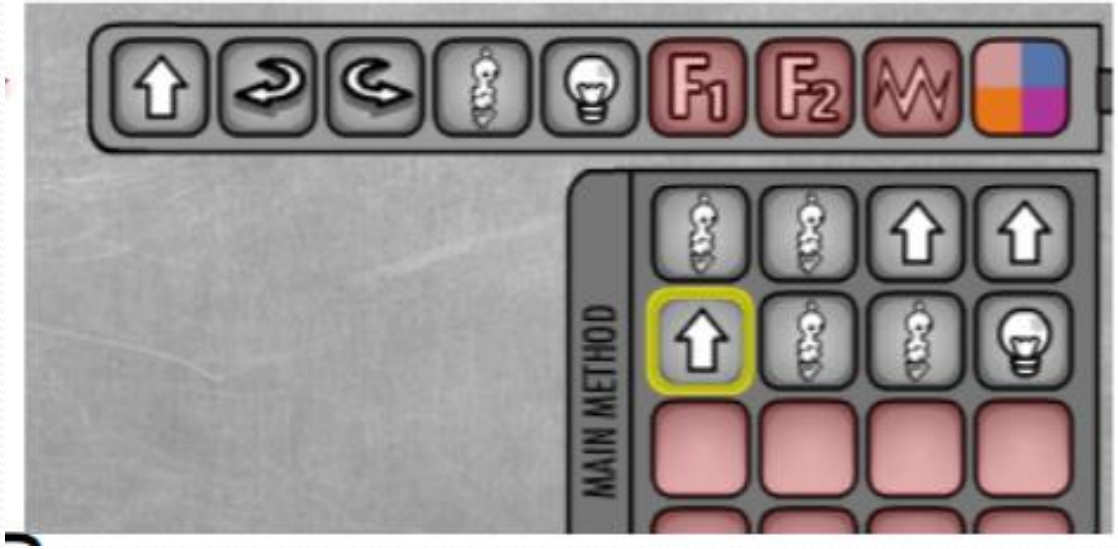
Point of View

- Programming requires you to take the agent's point of view



Limited Instructions

- The number and type of instructions is always limited – you need a solution using only them
  - Instructions
    - The agent can do only certain things
      - There is no JUMP_3
    - Executes the instructions one-at-a-time

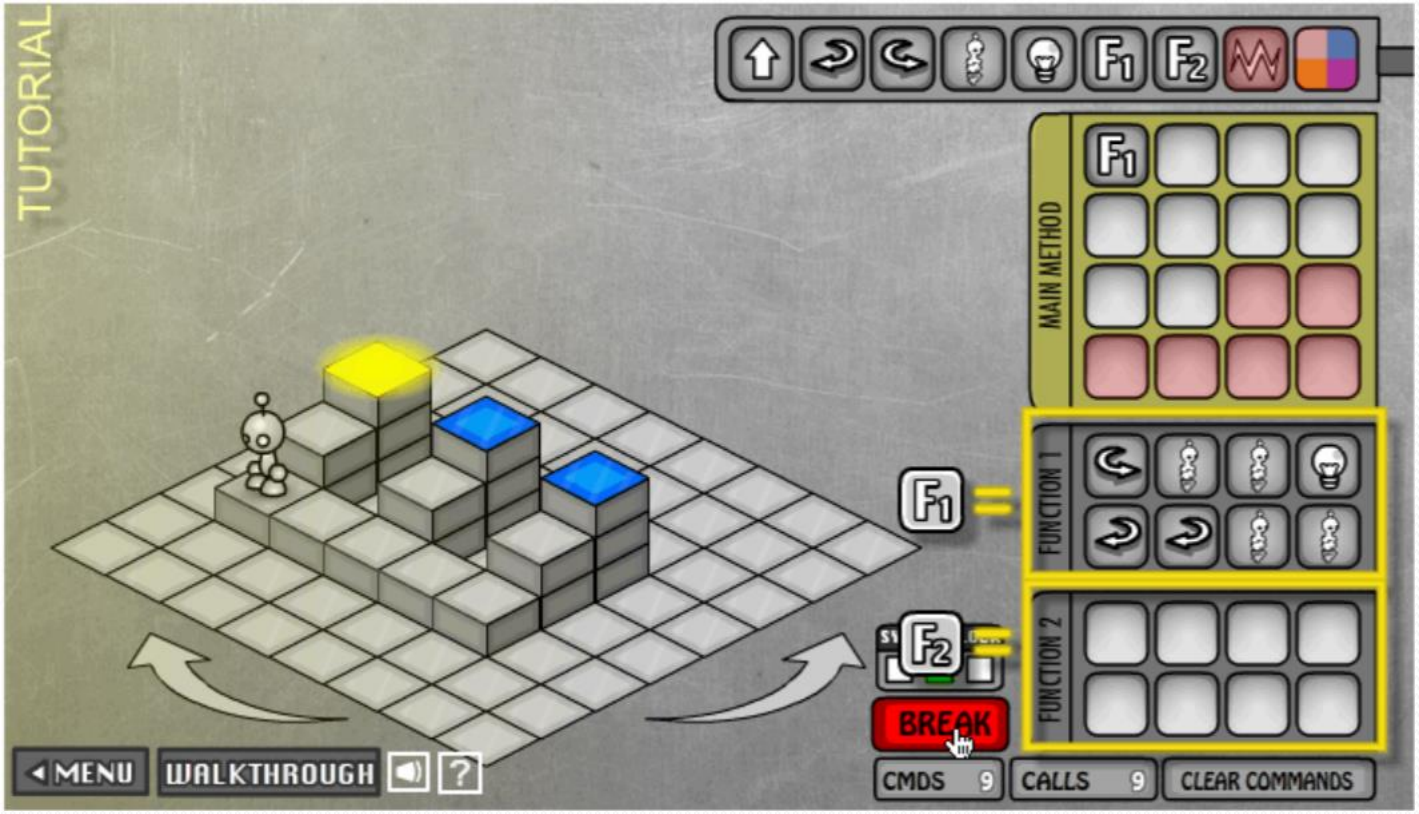


Limited Instructions

- In theory, a computer with just 6 instruction types could compute all known computations
  - Programming with 6 instructions would be tedious
    - No one would be a programmer no matter how much it paid
    - Apps as we know them would not exist
    - Programming was like this in the beginning
      - This is why they are called the "bad old days"
- Functions fix this!

Functions

- We make new instructions using functions!



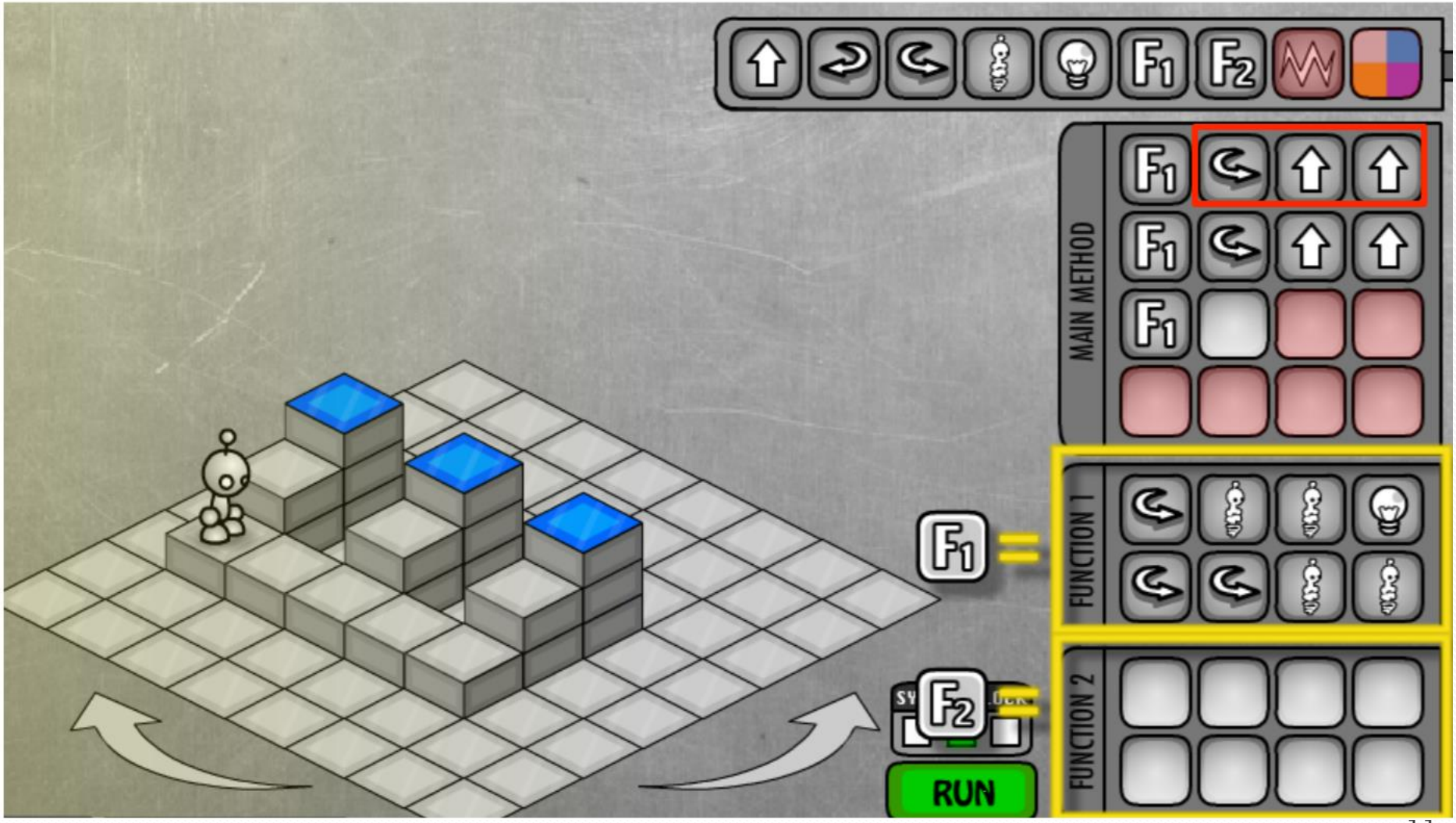
- F1( ) packages actions: E.G. “process a riser”

Where do functions come from?

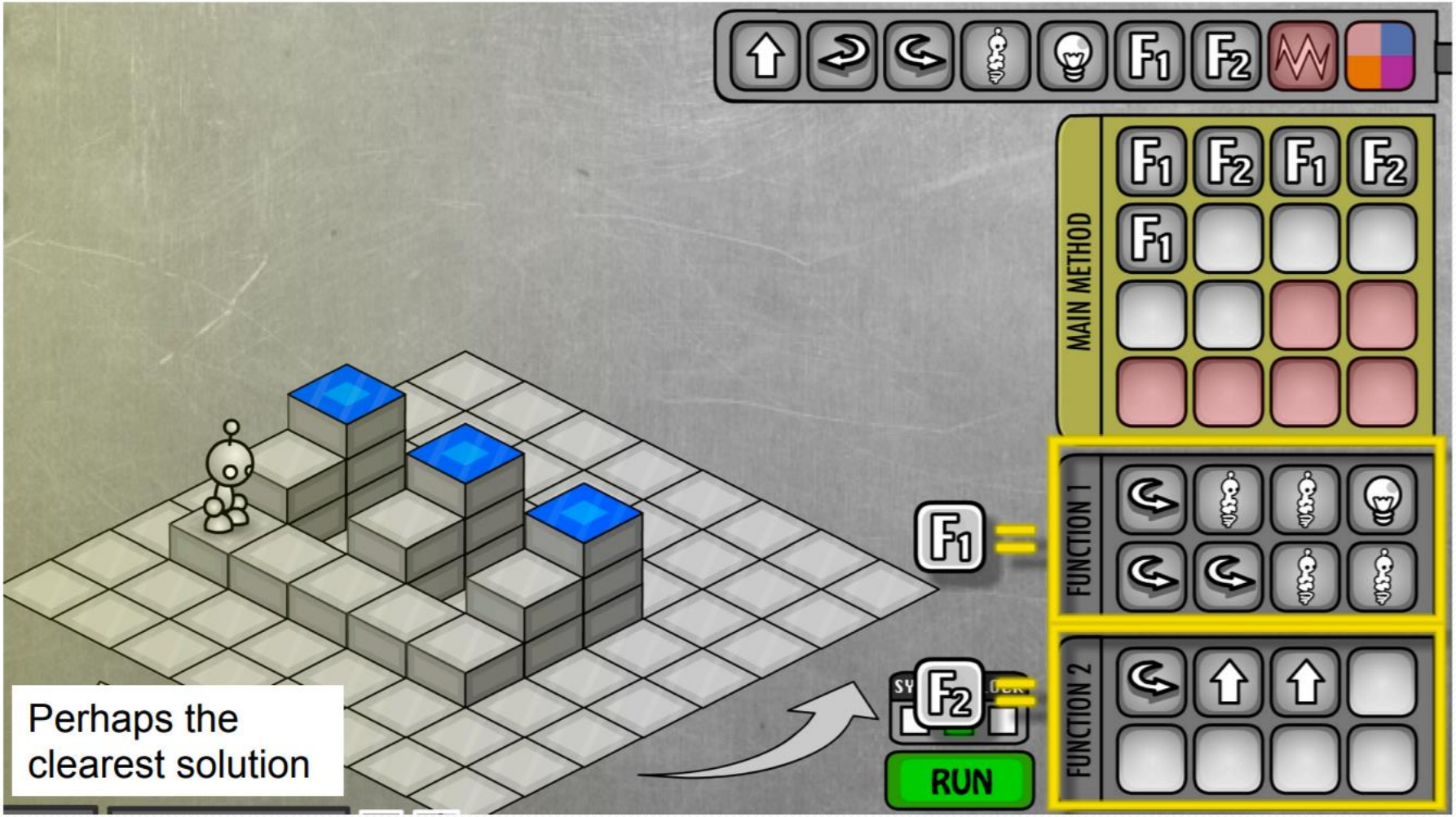
- The “process a riser” function was a sub-problem of the overall task – we just saw it was a useful operation to perform

- Spotting common patterns is also another place to find “work” that could be turned into functions

Move to next riser



Move to next riser



Functions

- Functions allow us to solve problems by solving parts, naming them (at least in our mind), and putting the part solutions together to solve the whole problem

Abstraction

No. 5 / No. 22
Mark Rothko

Abstraction

- Abstraction is the act of recognizing and then removing an idea or concept or process from a situation.

- Example:
  - **Saying**: "A fox saw some juicy grapes growing on a fence. He tried and tried to reach them, but failed. Finally, he walked away, saying 'They were probably sour'"

  - **Meaning**: one failing to get something they want, often claims in the end it's no good.

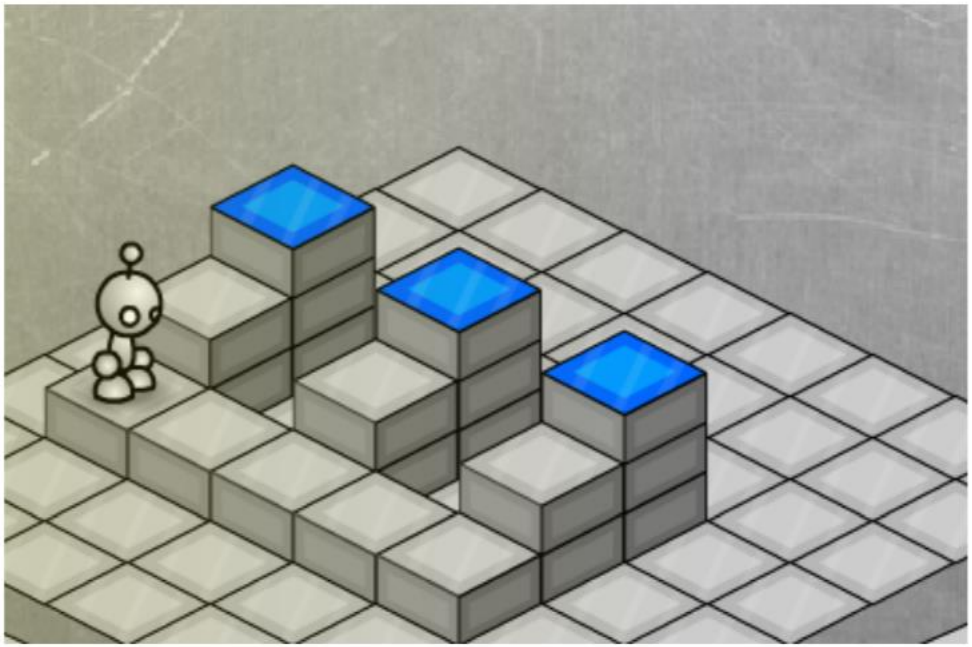
- Separate relevant from irrelevant

- Recast the idea in more general terms

Function becomes a concept

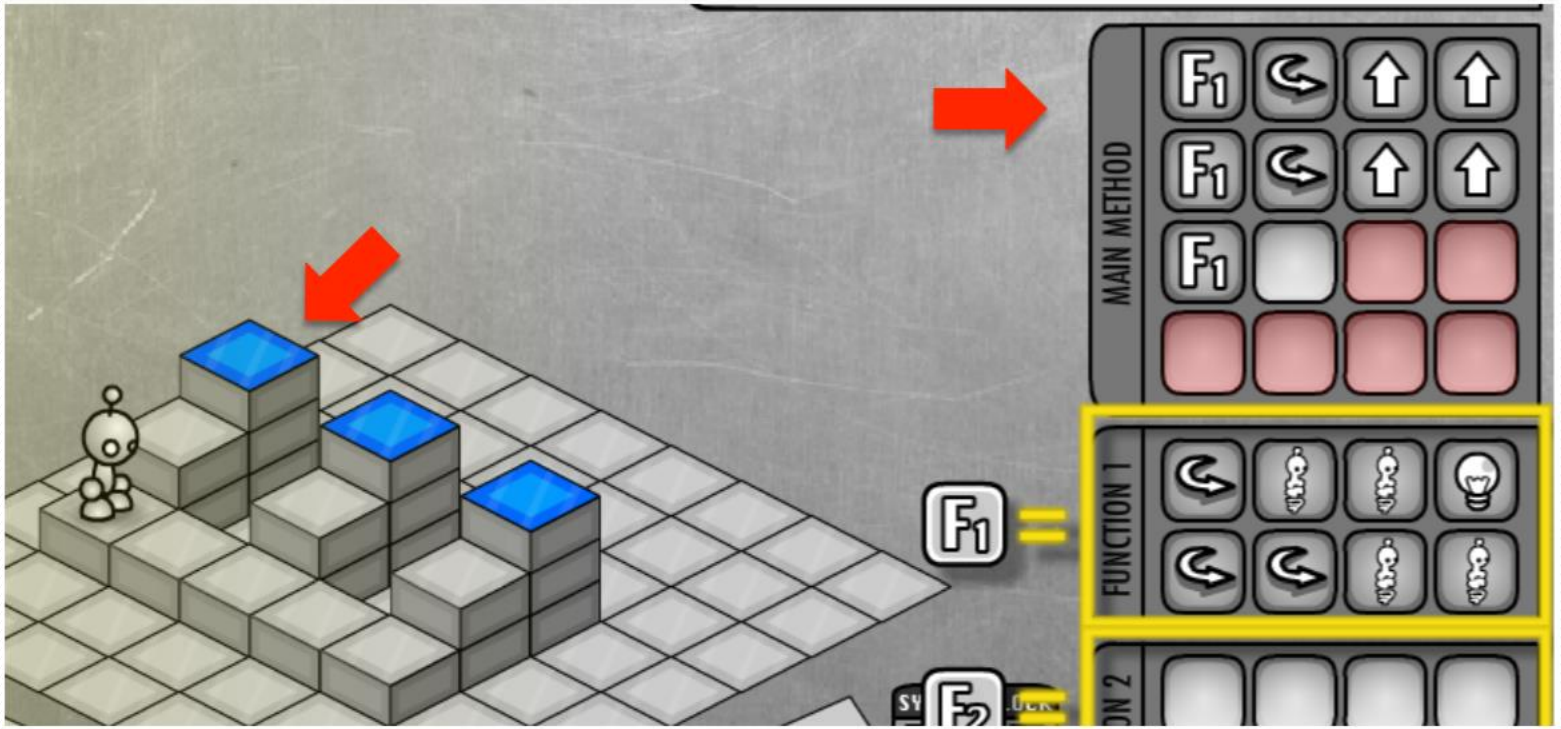
- Because we noticed "process a riser," as an action we needed to do (more than once) we think of the programming task as

Process a riser
Move to next riser
Process a riser
Move to next riser
Process a riser

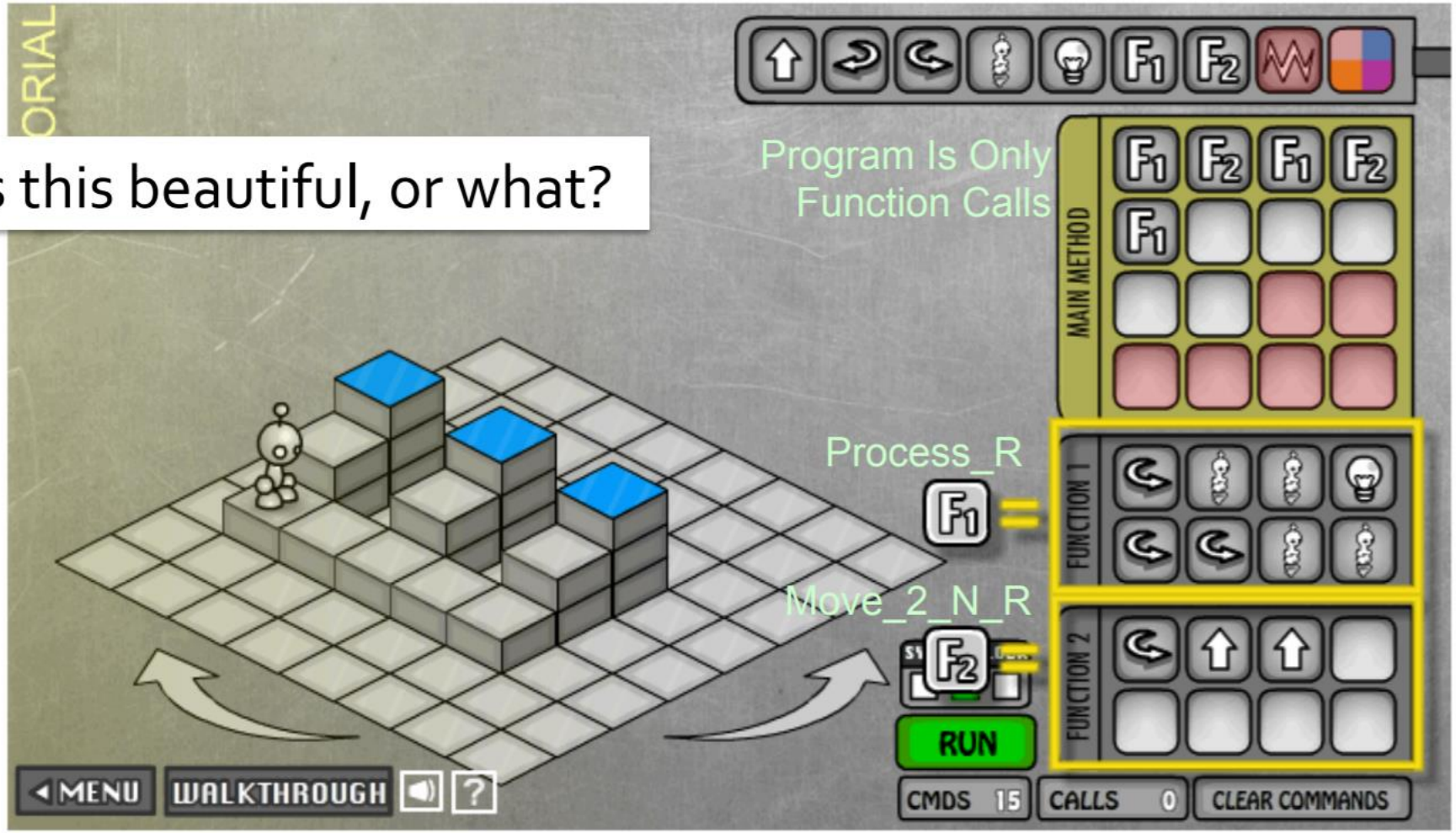


Noticing conceptual units

- We can "see" abstractions in the problem (riser picture) or in the solution (instruction pattern) … where we find them doesn't matter



Elegance


Is this beautiful, or what?
Program Is Only Function Calls
MAIN METHOD
Process_R
Move_2_N_R
FUNCTION 1
FUNCTION 2
RUN
MENU
WALKTHROUGH
CMDS 15
CALLS 0
CLEAR COMMANDS

Abstraction

- Collecting operations together and giving them a name is **functional abstraction**
  - The operations perform a **coherent** activity or action – they become a concept in our thinking
  - The operations accomplish a goal that is useful – and typically – is needed over and over again
  - Functions implement functional abstraction: 3 parts
    - A name
    - A definition (instruction sequence), frequently called a "body"
    - Parameters –stuff inside the parentheses, covered later

Abstraction is common

- Functional abstractions in which you are the agent, but someone taught you:
  - Parallel parking
  - Backstroke in swimming
- Functional abstractions you recognized and in which you are the agent
  - Doing a load of laundry
  - Making your favorite {sandwich, pizza, cookies, …}
- Others?

No one right abstraction

- We have abstracted “process a riser” and “move to the next riser” as components of a solution
- As concepts, they are packaged into functions
- Maybe you thought of this in a different way
- That is, there can be other “coherent” parts of a solution

How will abstraction help?