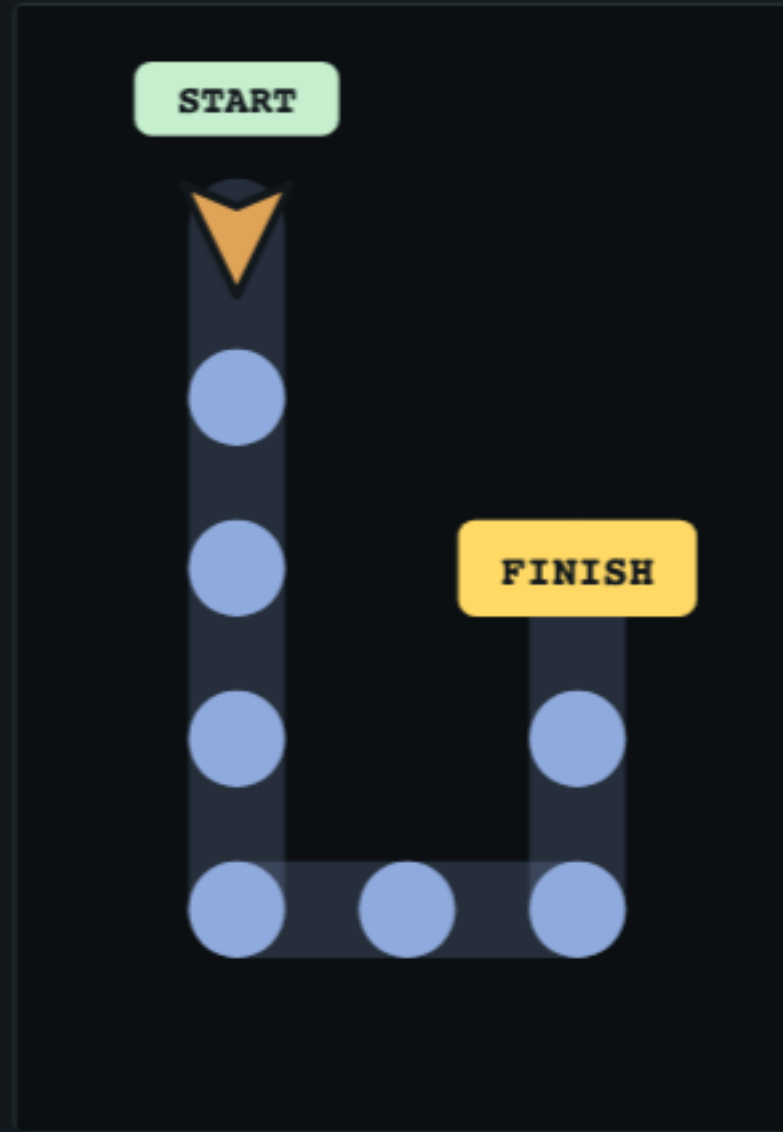


Where did the computers go?



a printed maze

seven cards

one turtle token

Today You Are The Turtle

- Your first proper program: on cards, in your own hands.
- No computer today. Same thinking as real programming, no keyboard.
- Seven levels. Work in pairs, at your own pace, then we check together.

» By the end you'll see that your cards already *are* code.

```
$ maze --pairs --cards■
```

The Turtle Maze

```
// program a maze with seven cards and a token. No screens needed.
```

— Dr. Hughes

How to Play

Programmer

Lays out the cards. **That is the program.**

Then hands off: never touches the token.



Runner

Moves the turtle token one card at a time,
exactly as printed.

Even straight into a dead-end.

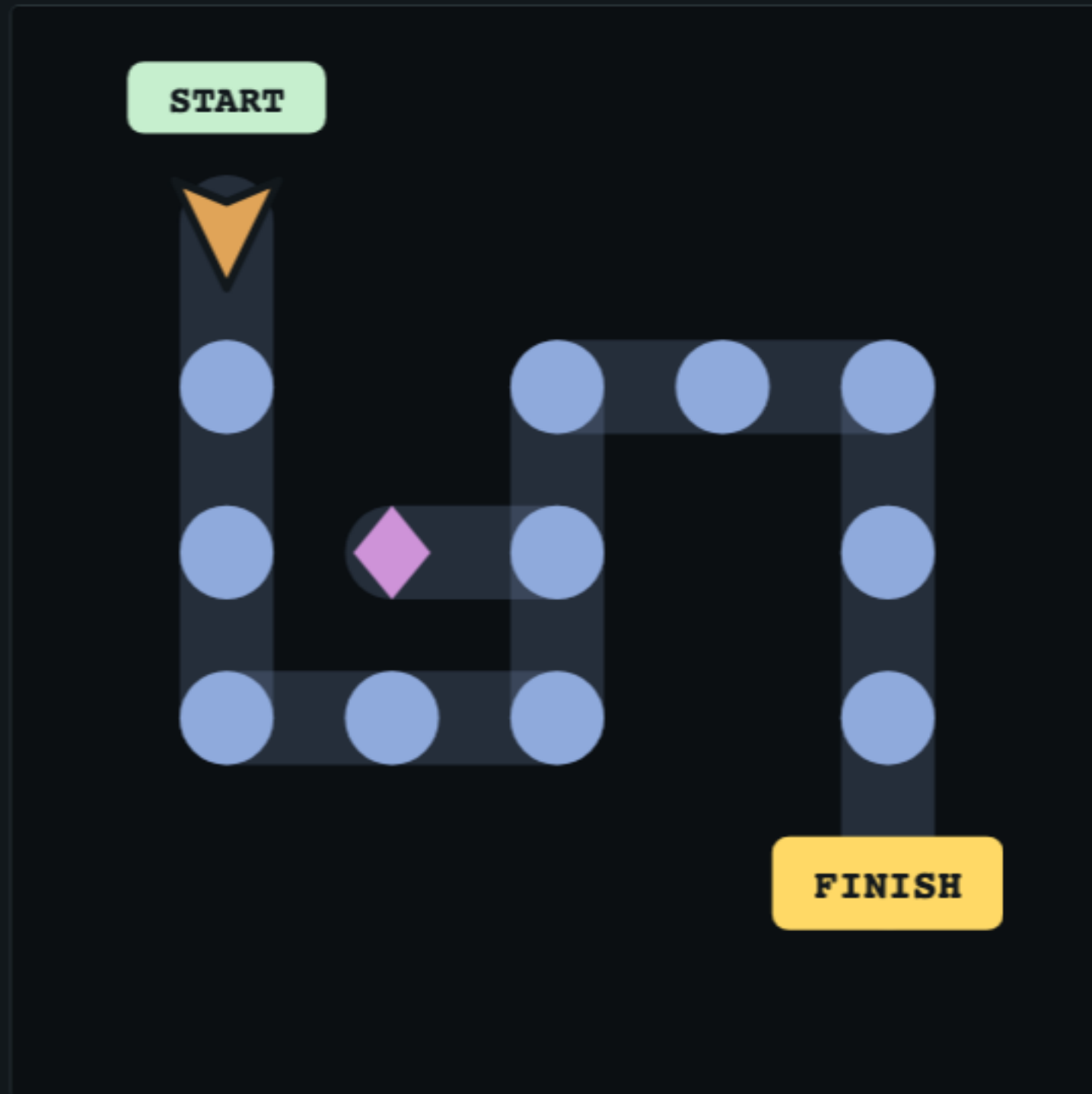
» Swap roles every level. Both of you do both jobs. Goal: token from **START** to **FINISH**.

Rules of the Pair

- The runner follows the cards **literally**. No improvising.
- A wrong turn is a **bug in the program**. Fix the cards, re-run from START.
- Get the teacher's **sign-off** before the next level.

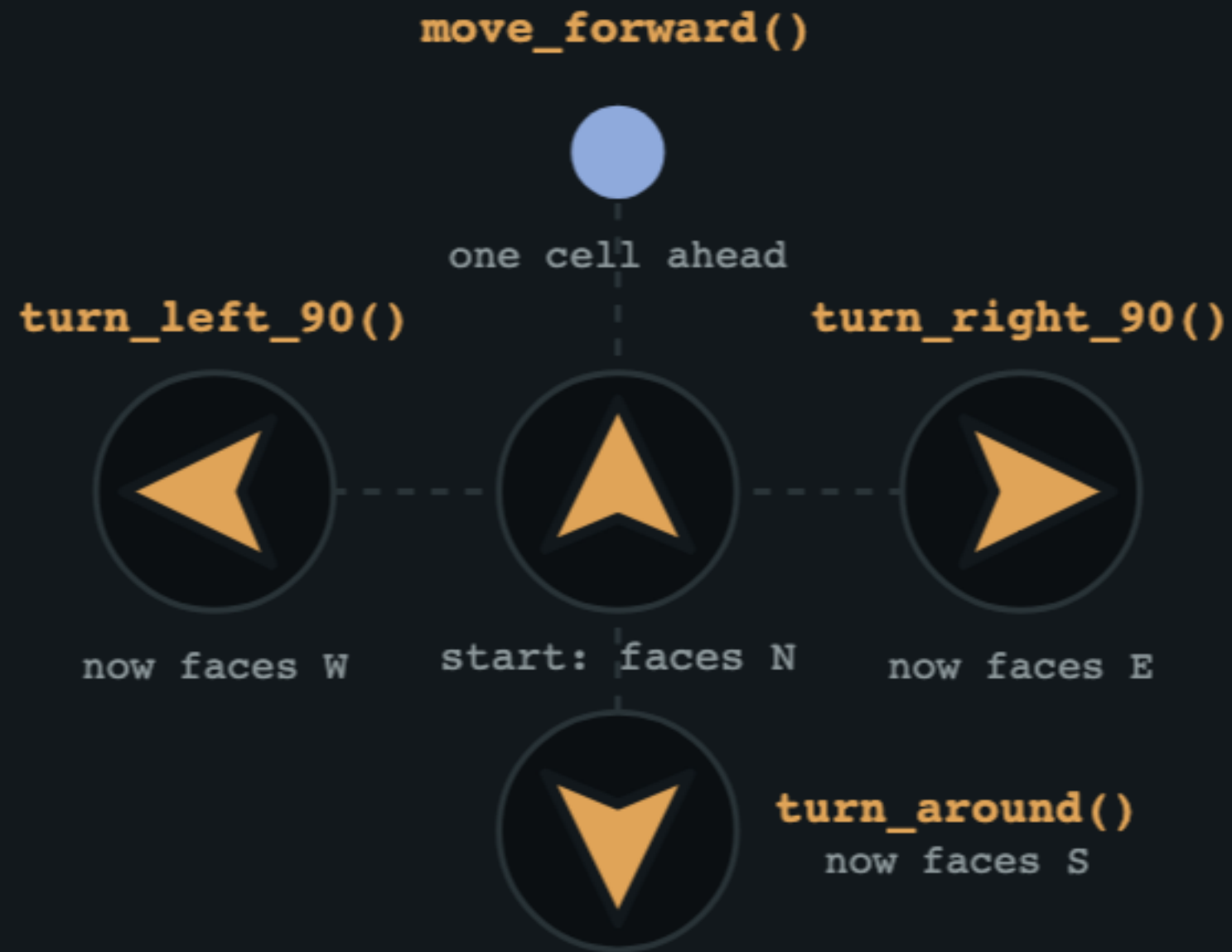
Running a buggy program teaches you more than reading a perfect one.

Reading the Maze



- ▲ **Turtle token.** The arrow points the way it is facing.
- **START** and ■ **FINISH**
- **Node.** One hop to the next node = one `move_forward()`.
- ◆ **Diamond.** Optional bonus. Never needed.
- ▶ **One-way.** Enter the corridor only the way the arrow points.
- ⊘ **Dead-end.** Go in and the only way out is `turn_around()`.

Which Way Am I Facing?



- Turns **pivot on the spot**. You never change cell.
- `turn_left_90()` and `turn_right_90()` turn a quarter.
- `turn_around()` turns a half.
- `move_forward()` goes one cell the way the arrow points.

Seven Cards. The Whole Language.

move_forward()

MOVE

turn_left_90()

LEFT

turn_right_90()

RIGHT

turn_around()

ABOUT

for _ in range(_):

body cards inside

LOOP

def ____():

DEFINE

____()

CALL

Every card is reusable, as many times as you like. These seven are the only ones that exist.

 Worksheet · page 1 · Part 3 – card set

 Cut out: command cards

The Loop Card

```
for _ in range(4):
```

```
    move_forward()
```

=

```
move_forward()
```

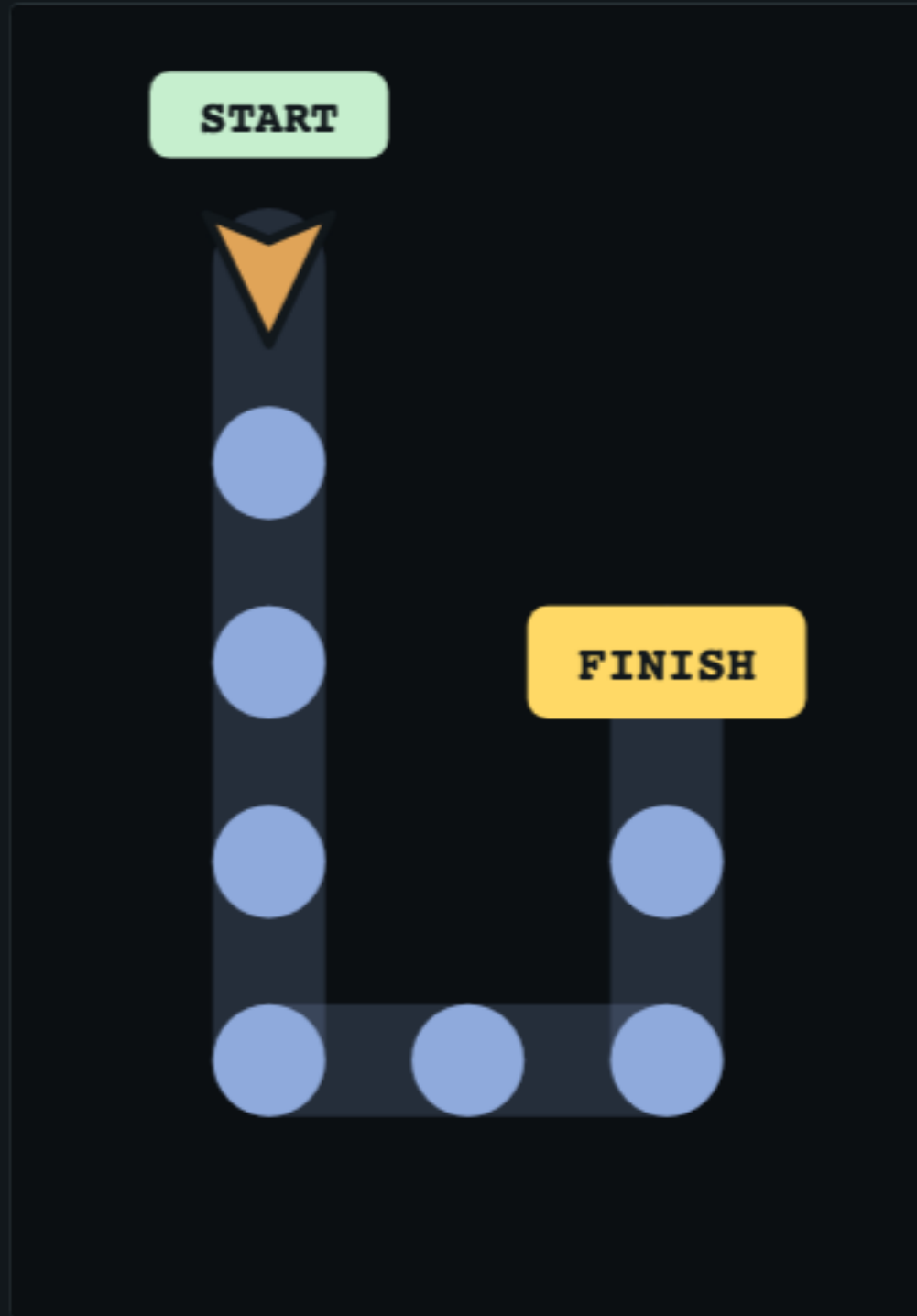
```
move_forward()
```

```
move_forward()
```

```
move_forward()
```

- The body cards sit **inside the frame**. Write the number in the blank.
- The runner re-runs the body **N times**, from wherever the turtle is *right now*.

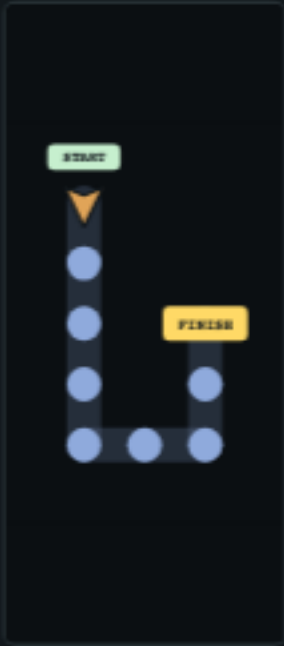
Try Level 1 Together



1. **Programmer:** lay cards from START to FINISH.
2. **Runner:** run them literally.
3. Bug? Fix the cards, re-run from START.
4. Teacher sign-off, then swap roles for Level 2.

 Worksheet · page 2 · Level 1 – 01-s-bend

The Ladder: Seven Mazes



L1 · p2
s-bend
3x5 · faces S



L2 · p3
zigzag
5x5 · faces S
diamond



L3 · p4
weave
6x5 · faces S
diamond loops pay



L4 · p5
spiral
6x6 · faces E
diamond loops pay



L5 · p6
open-maze
6x6 · faces S
diamond trap



L6 · p7
one-way-maze
5x5 · faces S
one-ways



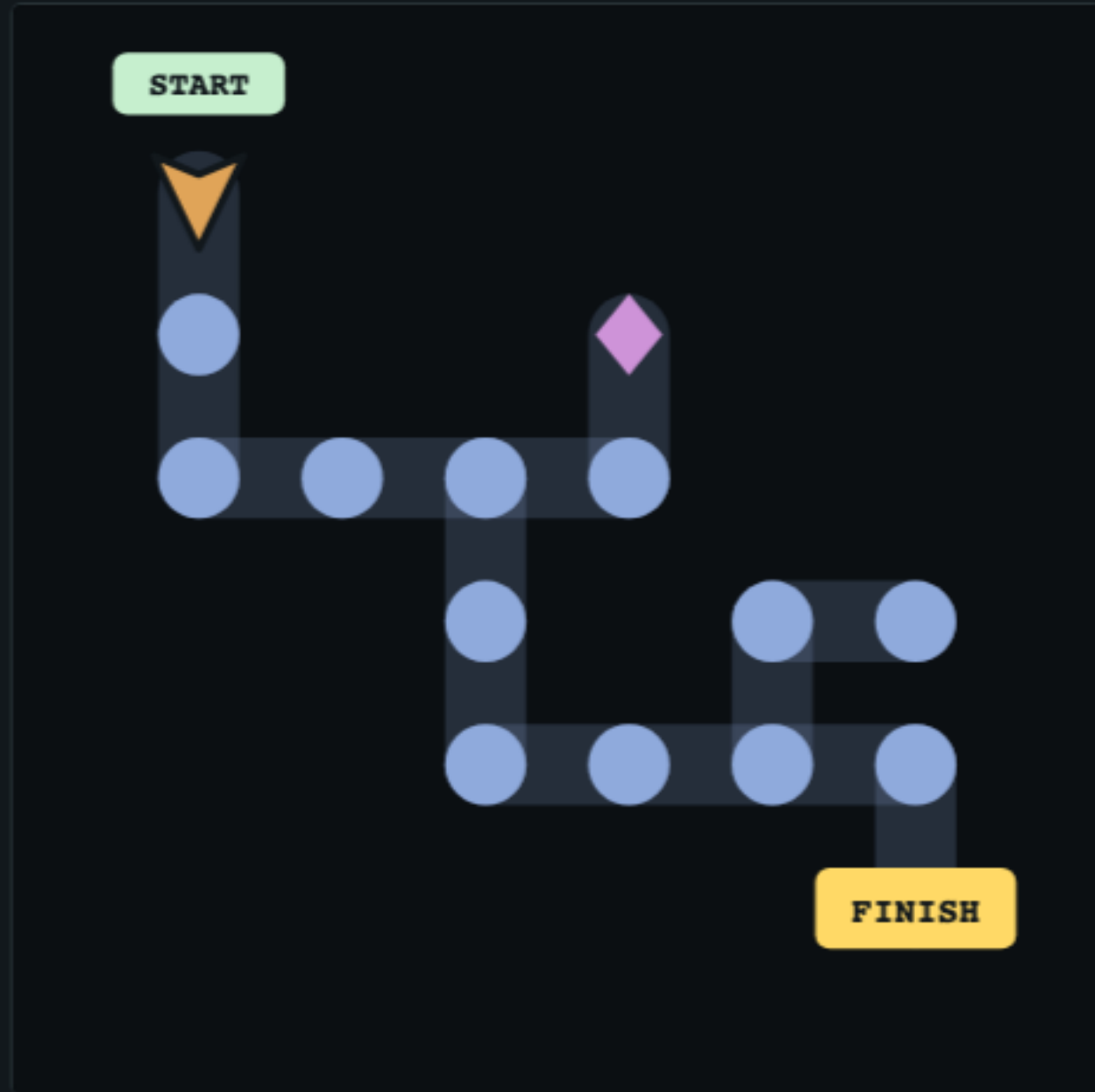
L7 · p8
long-maze
6x6 · faces W
one-ways trap def

// one maze per worksheet page, pages 2–8. Each new idea is briefed in the slides below this one.

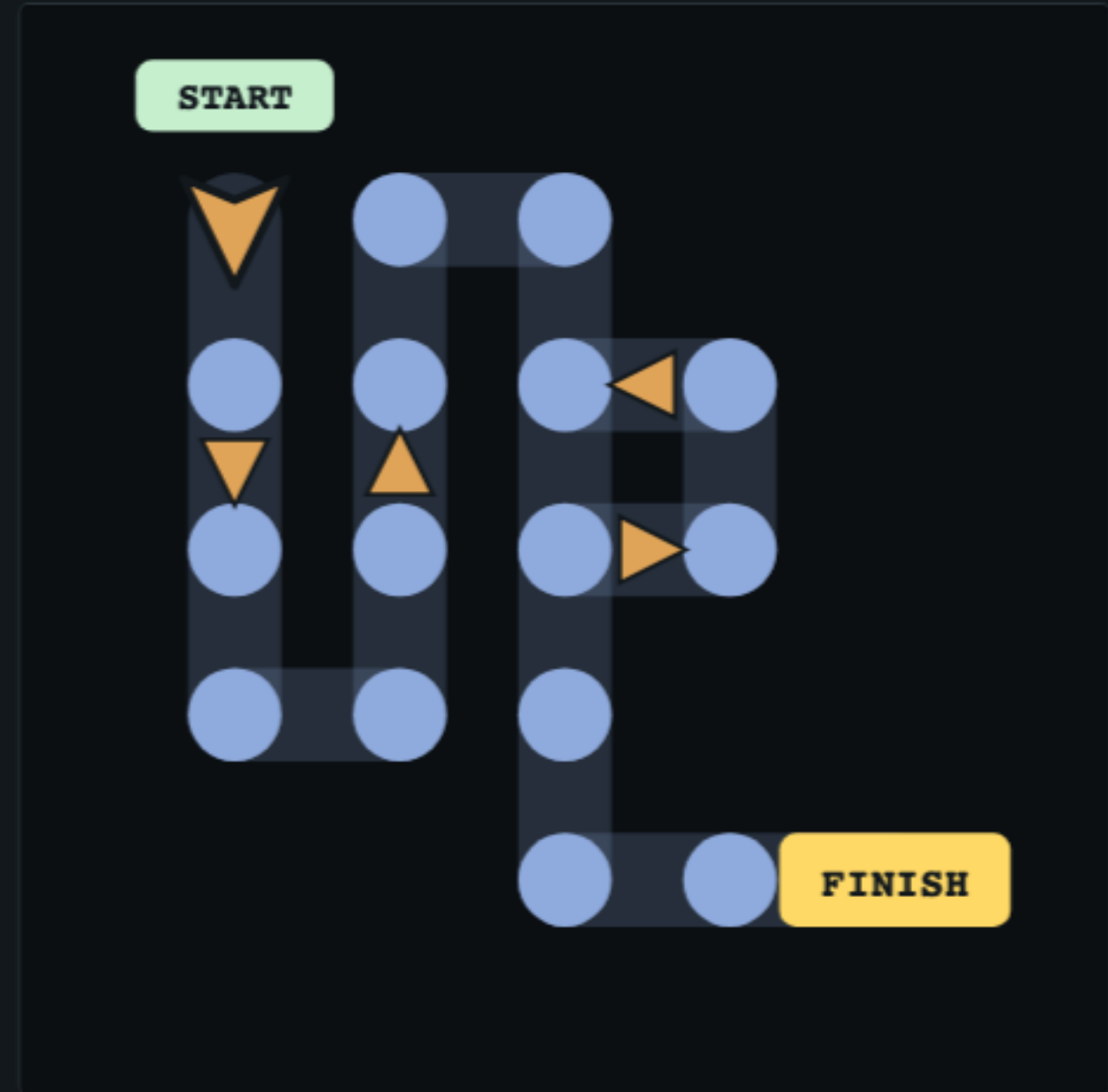
A maze game interface with a dark blue background. A light blue path leads from a 'START' button (top left) to a 'FINISH' button (center). The path is defined by light blue circles. A yellow arrow points right from the start, and a pink diamond points left from the finish.

- Worksheet · pages 4–5 · Levels 3 and 4

Levels 5–6: Traps and One-Ways

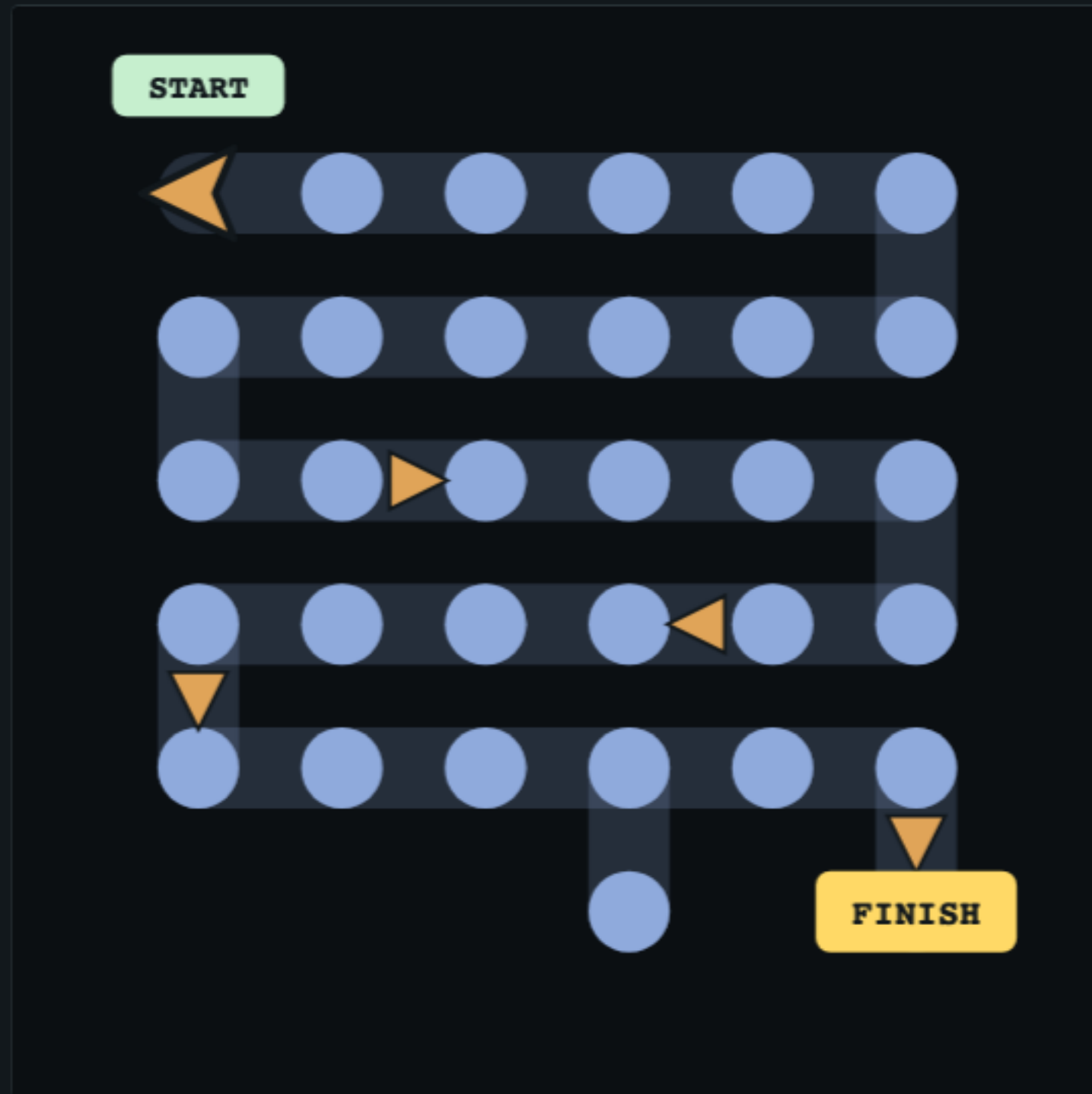


Level 5. No arrows, but one corridor is a trap. Diamond optional.



Level 6. One-ways: you may only *enter* a corridor the way the arrow points.

Level 7: Def a Jog



- The turtle starts facing **west**, but the first corridor runs **east**. What card do you need first?
- The same row is crossed five times.

```
1 def jog():  
2   # the same five moves, every row
```

» Stretch: **def** a jog card, then call it again and again.

Worksheet · page 8 · Level 7 – 07-long-maze

Your Turn: Work Through the Levels

- 1 Cut out the **command cards**. One set per pair.
- 2 Work the levels in order, one worksheet page each (pages 2–8).
- 3 Write your cards on the lines, then lay them out and run them.
- 4 Fill in *My count*, get a **sign-off**, swap roles, next level.

p2 Level 1

p3 Level 2

p4 Level 3

p5 Level 4

p6 Level 5

p7 Level 6

p8 Level 7

» Keep your counts. When the teacher calls time, we go through the solutions together.

```
$ check --together■
```

Check Together

Quick run through the solutions. Compare them with your cards and your count.

```
// was yours shorter? did you find a loop? what was the bug?
```

Level 1 – 01-s-bend: solution

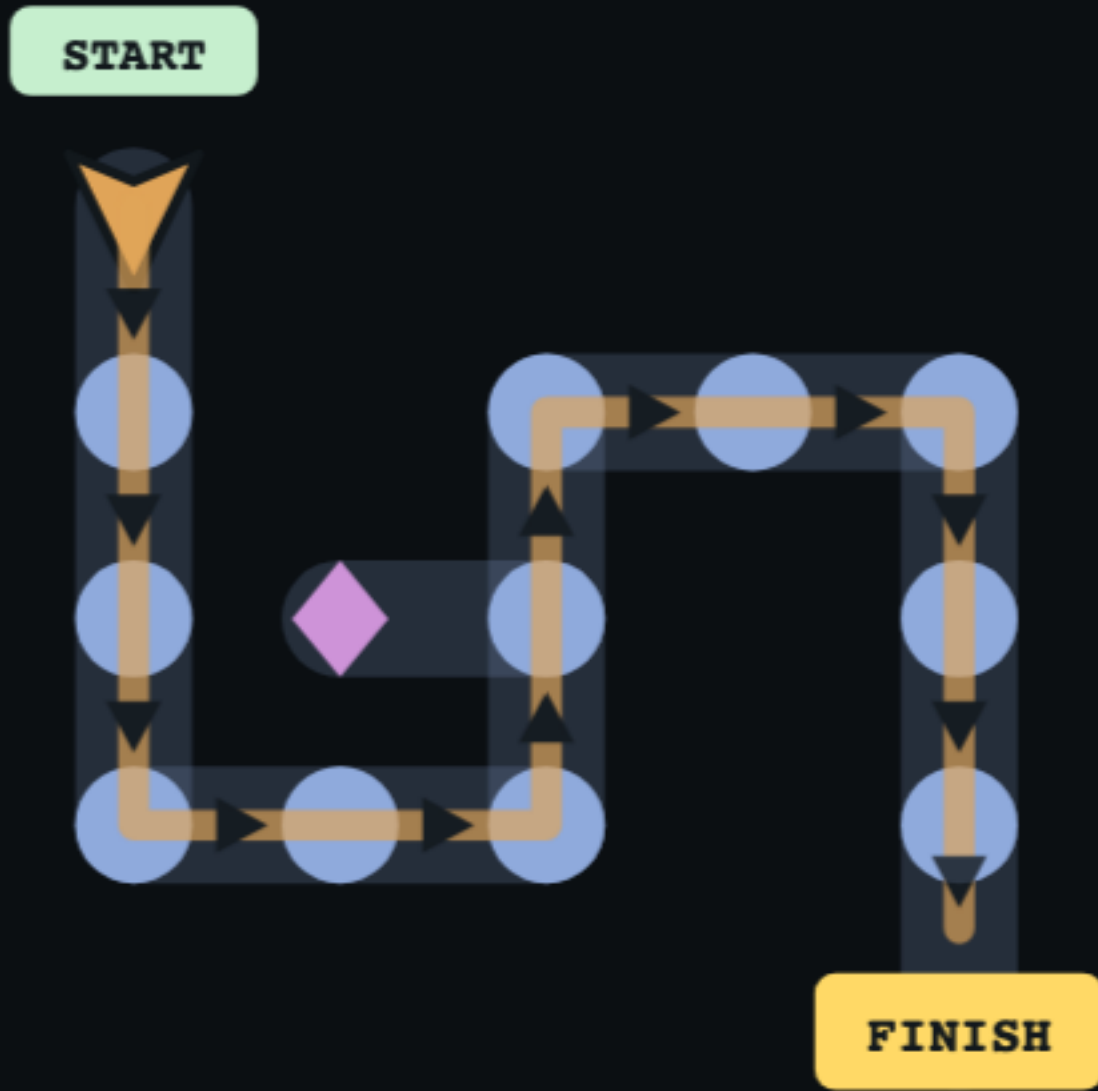


forward $\times 4$, left, forward $\times 2$, left, forward $\times 2$

8 forwards · 2 turns · 10 cards laid one by one

The first leg is four forwards in a row: `for 4: [forward]` . A first taste of a loop.

Level 2 – 02-zigzag: solution



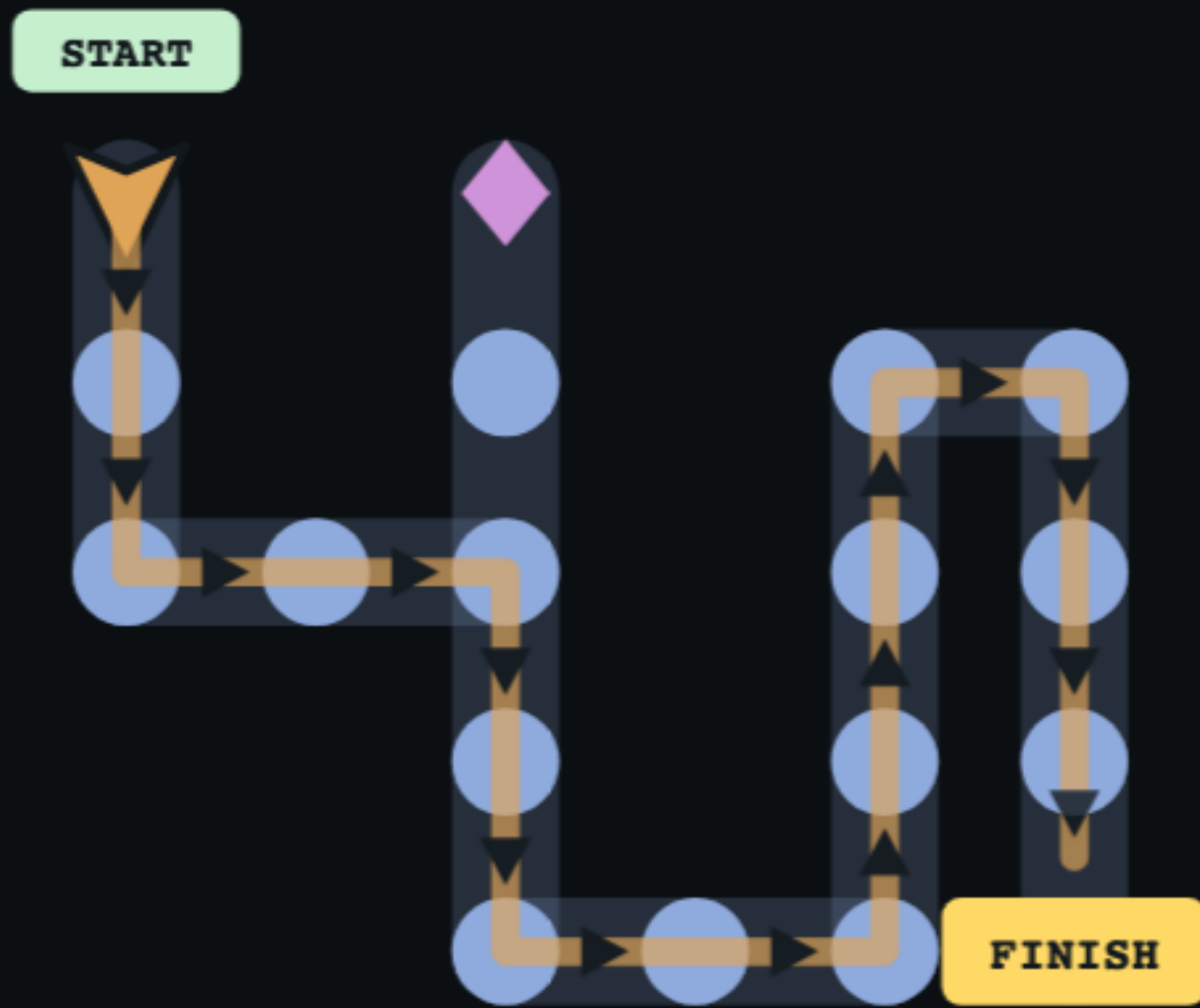
```
forward x3, left, forward x2, left, forward x2, right,  
forward x2, right, forward x3
```

12 forwards · 4 turns · 16 cards laid one by one

Two runs of 3: a loop fits but does not pay yet.

» Diamond detour (optional): `left, forward, around, forward, left`.

Level 3 – 03-weave: solution



```
forward x2, left, forward x2, right, forward x2, left,  
forward x2, left, forward x3, right, forward, right,  
forward x3
```

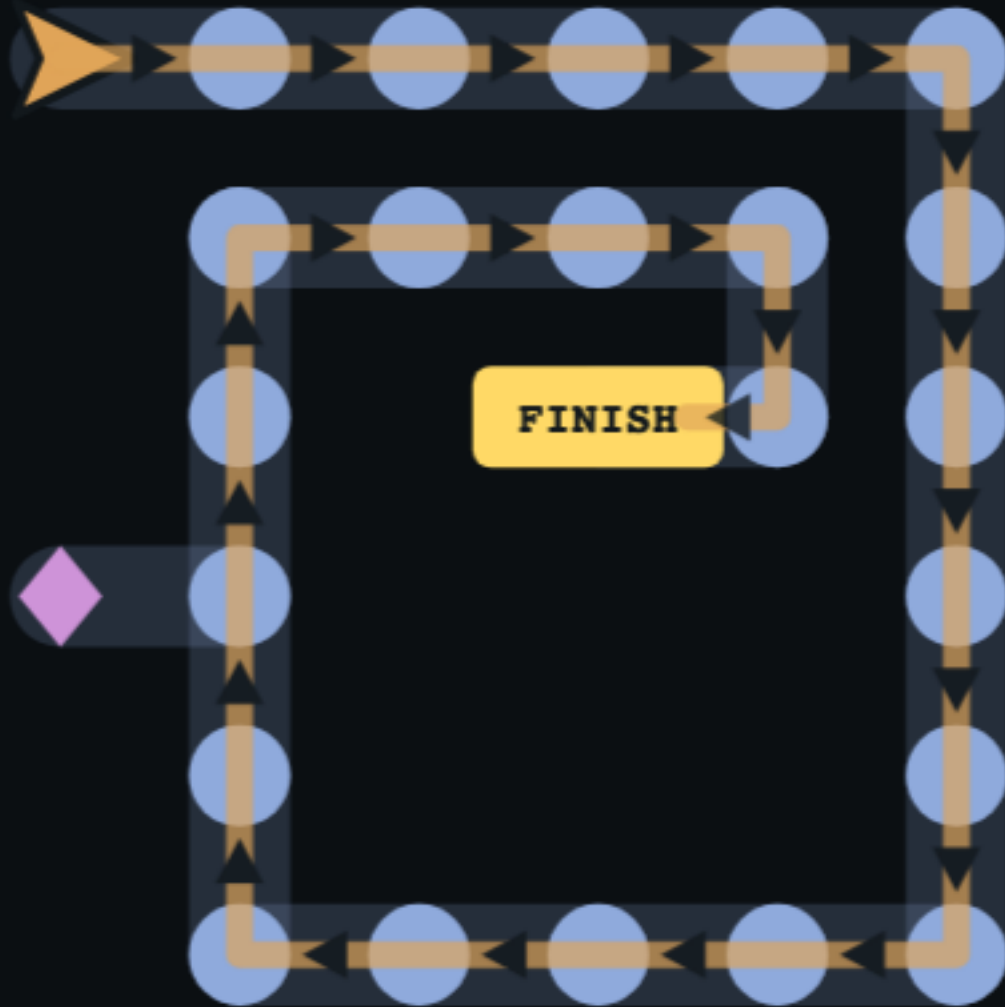
15 forwards · 6 turns · 21 cards laid one by one

Two runs of 3: `for 3: [forward]` replaces three cards each. First level where a loop is really cheaper.

» Diamond detour (optional): `left, forward x2, around, forward x2`.

Level 4 – 04-spiral: solution

START



forward $\times 5$, right, forward $\times 5$, right, forward $\times 4$, right,
forward $\times 4$, right, forward $\times 3$, right, forward, right,
forward

23 forwards · 6 turns · 29 cards laid one by one

Runs of 5, 5, 4, 4, 3: five loop cards stand in for 21 of the 23 forwards.

» Looped: for 5: [forward], right, for 5:
[forward], right, for 4: [forward], right,
for 4: [forward], right, for 3: [forward],
right, forward, right, forward

[illegible]

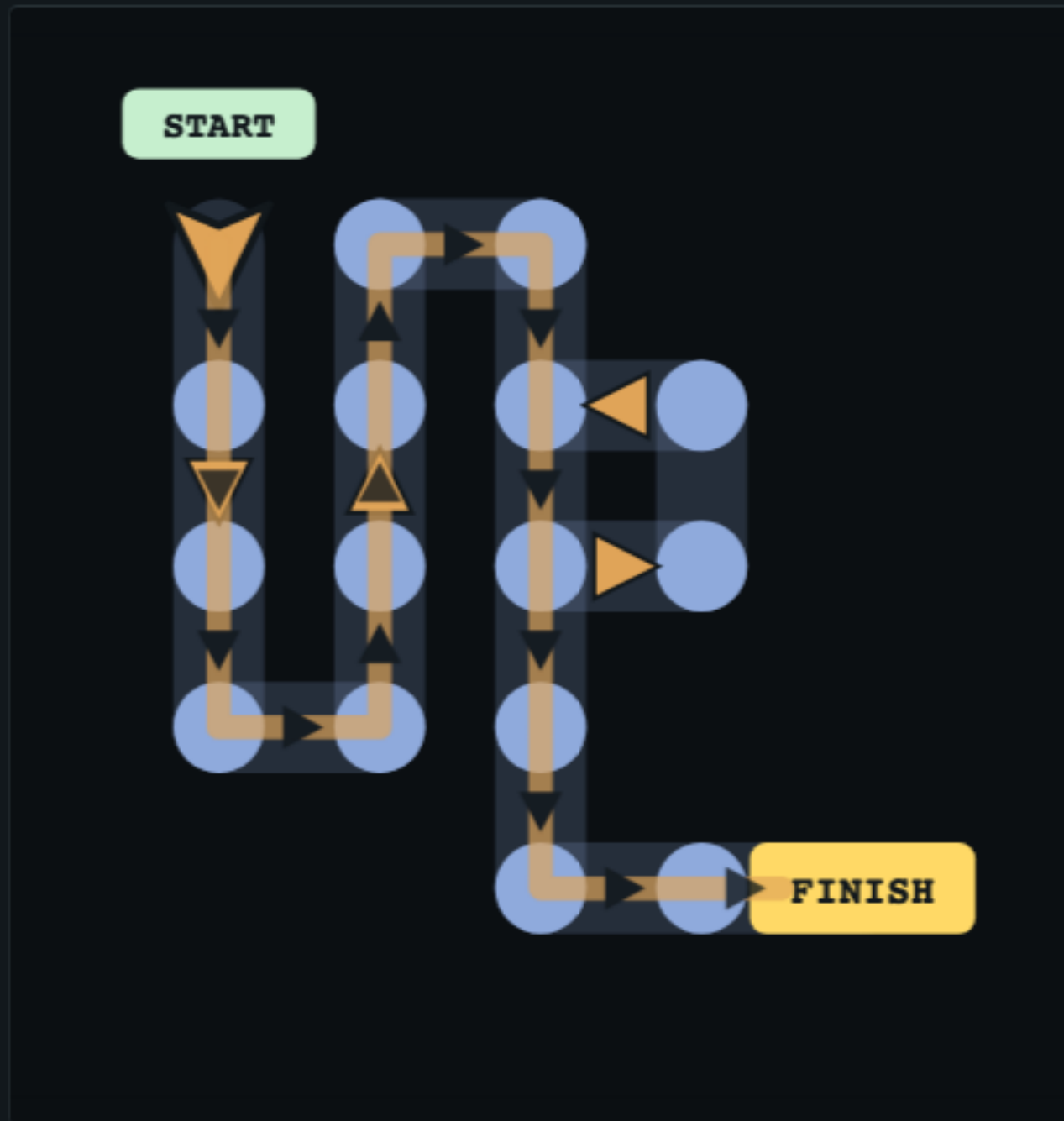
forward x2, left, forward x2, right, forward x2, left,
forward x3, right, forward

10 forwards · 4 turns · 14 cards laid one by one

Shorter than Level 4 (10 forwards vs 23) but the maze fights back: there is a trap, and the arrows are gone. Difficulty is not route length.

» Diamond detour (optional): forward, left, forward, around, forward, right, forward.

Level 6 – 06-one-way-maze: solution

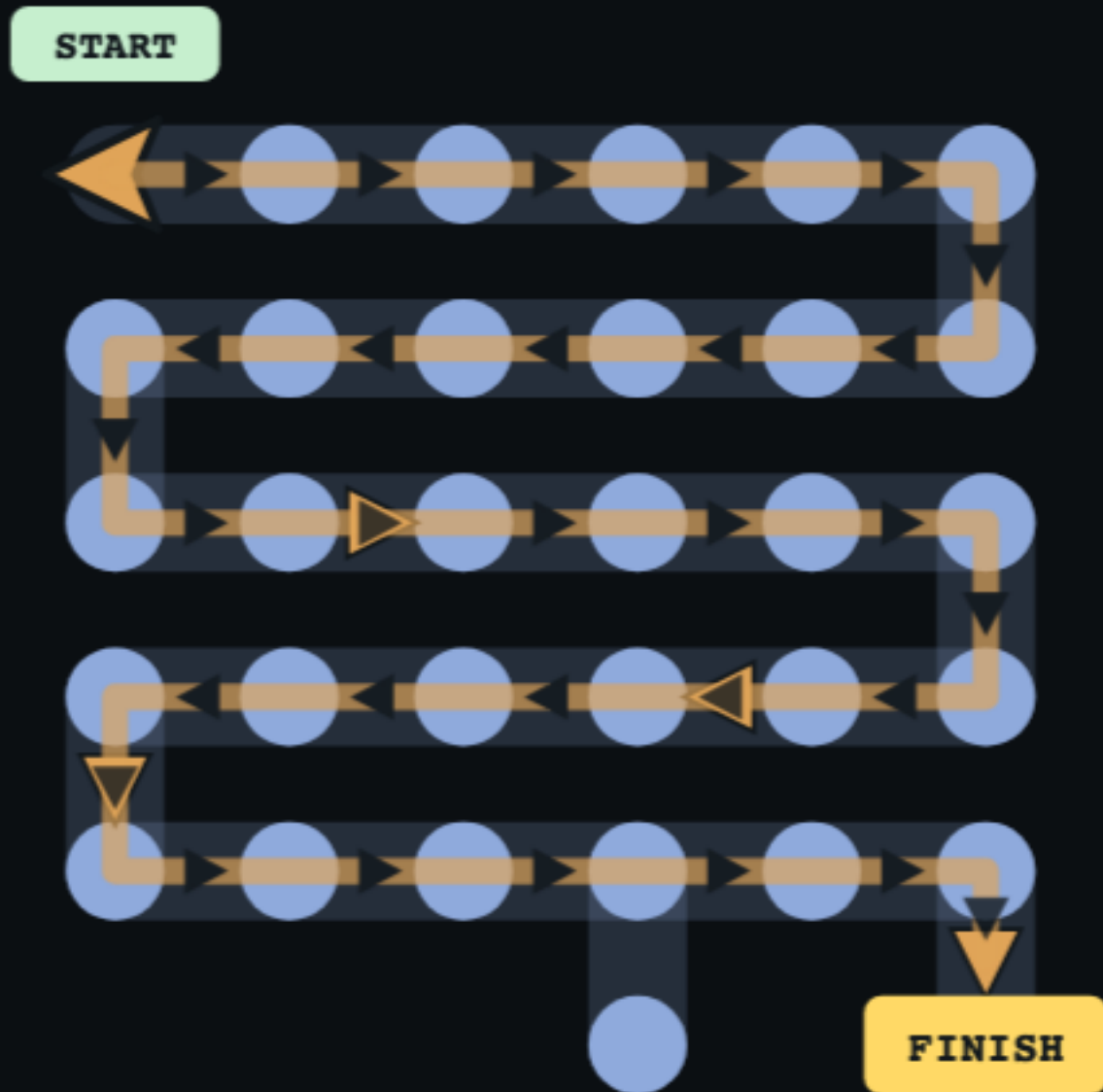


forward $\times 3$, left, forward, left, forward $\times 3$, right, forward, right, forward $\times 4$, left, forward $\times 2$

14 forwards · 5 turns · 19 cards laid one by one

The little loop of corridor is never a shortcut: the arrows say which way you may *enter*.

Level 7 – 07-long-maze: solution



around, `jog()`, right, forward, right, `jog()`, left, forward, left, `jog()`, right, forward, right, `jog()`, left, forward, left, `jog()`, right, forward with `def jog(): forward x5`

30 forwards · 10 turns · 40 cards laid one by one

Starts facing west, first corridor runs east, so the first card is `turn_around()`.

» Count check: 26 cards on the table = 1 **def** + 5 body cards + 20 play cards.

Your Cards Are Code

Each card is a one-line function. Same names, same order.

```
1 def move_forward():  
    2 t.forward(1)  
  
4 def turn_left_90():  
    5 t.left(90)  
  
7 def turn_around():  
    8 t.right(180)
```

» The same cards work in code. One card, one function, one line inside.

Def It, Call It

```
1 def jog():  
    2 for _ in range(5):  
        3 move_forward()  
  
5 turn_around()  
6 jog()  
7 turn_right_90()  
8 move_forward()
```

- `def` names a block. **Nothing happens** when it is read.
- The call, `jog()`, runs it once from wherever you are.
- Level 7 needs it five times.

Things You've Learnt This Session:

- ✓ the seven cards, and the loop rule: re-run the body N times, from wherever you are
- ✓ def a block, then call it: nothing happens until the call
- ✓ programmer lays, runner follows, a wrong turn is a bug
- ✓ cards are code: you were programming all lesson