ANSWERS

U12.5 Reward is a specification

Learning, and the capstone · University · about 35 min

What this lesson is about

The optimiser answers the question you asked, which is rarely the question you meant.

- The failure has a name
- The two you will write yourself
- Shaping, and how it goes wrong
- Writing one that survives

Questions

7 marks in all

1. The reward is distance travelled towards a mark 60 cm ahead. What is it actually rewarding? [1 mark]

- ☐ **A** Going fast and far, so the best policy charges straight past the mark
- ☐ **B** Stopping on the mark
- ☐ **C** Getting close to the mark at some moment
- ☐ **D** Driving straight

> **Answer:** A. Progress pays for every centimetre and stopping earns nothing. Score the end state, or the journey and the end state together.

2. Four commands were each driven for 4 s from the same line. What does this print? [1 mark]

```
MARK = 60.0
travel = {55: 49.5, 70: 63.8, 85: 79.2, 100: 93.6}
by_progress = max(travel, key=lambda c: travel[c])
by_error = min(travel, key=lambda c: abs(travel[c] - MARK))
print(by_progress, by_error)
```

> **Answer:**
>
> ```
> 100 70
> ```
>
> Command 100 travels furthest, but command 70 ends 3.8 cm from the mark against 10.5 cm for 55. Two scores, two winners.

3. What is the name for a policy scoring well on its reward while the behaviour is not what was wanted? [1 mark]

> **Answer:** reward hacking. It is not a bug in the optimiser, which did its job on a specification that did not say what its author meant.

4. A robot is rewarded for a small depth reading, meant to encourage parking close to its charger. What is the likely result? [1 mark]

- ☐ **A** It drives up against whatever is nearest, because the reward is a proxy for the thing wanted
- ☐ **B** It parks at the charger, because that is the smallest reading
- ☐ **C** It learns nothing, because the reward is sparse
- ☐ **D** It learns to avoid obstacles

> **Answer:** A. Rewarding the sensor reading rather than the world behind it lets any wall satisfy the specification.

5. Which form of shaping reward is guaranteed not to change the optimal policy? [1 mark]

- ☐ **A** The difference of a potential between states, F(s') - F(s)
- ☐ **B** A bonus for being near the goal
- ☐ **C** A bonus for facing the goal
- ☐ **D** A small reward for every step taken

> **Answer:** A. Potential based shaping leaves what is optimal unchanged. Near-goal and facing-goal bonuses teach the policy to hover and to stare.

6. Why does a sparse reward of 1 for arriving and 0 otherwise teach almost nothing, even though it specifies the task perfectly? [1 mark]

- ☐ **A** A random policy never arrives, so every trial scores the same and there is no signal to follow
- ☐ **B** It rewards progress instead of arrival
- ☐ **C** It is a proxy for the thing wanted
- ☐ **D** The optimiser cannot handle integer rewards

> **Answer:** A. That is why people add shaping, and why shaping is where gaming gets in.

7. A search's best score has risen steadily for fifty trials. What should you do before believing it? [1 mark]

- ☐ **A** Watch the behaviour that produced the winning score
- ☐ **B** Run fifty more trials to confirm the trend
- ☐ **C** Add a shaping term to speed it up
- ☐ **D** Report the best score with its trial number

> **Answer:** A. A rising score is not evidence until you have seen what earned it. Watching how a reward gets gamed is how you find out what you meant.

The task: two scores, two winners

Try commands 55, 70, 85 and 100 for four seconds each from the same line, homing between trials. Print `by progress:`, the command that travelled furthest, and `by error:`, the command that finished nearest the mark 60 cm ahead. Then run the one the error score chose, and stop there.

```
from bugbot import *
connect()

DT = 0.1
MARK = 60.0
RUN_S = 4.0
COMMANDS = [55, 70, 85, 100]
```

The hint students can ask for: Try commands 55, 70, 85 and 100, each for four seconds from the same starting line, driving back between trials so every trial is fair. Score each one twice: how far it got, and how far it ended from the mark 60 cm ahead. The two scores do not pick the same command. Then run the one that the error score picked, and stop there.

A solution

```
from bugbot import *
connect()

DT = 0.1
MARK = 60.0                # the mark, in cm ahead of the starting line
RUN_S = 4.0
COMMANDS = [55, 70, 85, 100]

def home():
    """Back to the starting line between trials, so every trial starts the same."""
    for i in range(200):
        y = position()[1]
        if y < 3.0:
            break
        drive(-max(20, min(80, 1.2 * y)), 0, 0)
        wait(DT)
    stop()
    wait(0.5)

def trial(cmd):
    """Drive at this command for four seconds and report how far the robot got."""
    y0 = position()[1]
    drive(cmd, 0, 0)
    wait(RUN_S)
    stop()
    wait(0.6)
    return position()[1] - y0

rows = []
for cmd in COMMANDS:
    travel = trial(cmd)
    rows.append((cmd, travel, abs(travel - MARK)))
    print("command", cmd, "travelled", round(travel, 1), "and finished", round(abs(travel -
MARK), 1), "cm out")
    home()
```

```
print("by progress:", max(rows, key=lambda r: r[1])[0])
best = min(rows, key=lambda r: r[2])[0]
print("by error:", best)

drive(best, 0, 0)
wait(RUN_S)
stop()
wait(0.5)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.