ANSWERS

5.4 Priorities

Behaviours · Robot club · about 20 min

What this lesson is about

Avoid beats seek: behaviours that take over and run to completion.

- Seek alone
- Avoid takes over
- The rule

Questions

7 marks in all

1. With behaviours ordered by priority, who controls the robot each tick? [1 mark]

- ☐ **A** The most urgent behaviour that wants control
- ☐ **B** All of them, averaged together
- ☐ **C** The one that was in charge last tick
- ☐ **D** The lowest priority behaviour, always

> **Answer:** A. Higher layers only speak up when they must. When none does, the job at the bottom runs. This is called subsumption.

2. Put these layers in order, highest priority first. [1 mark]

Number the lines 1 to 3 to put them in the right order.

___ `Seek: head for the goal`

___ `Emergency: the battery is low, so stop`

___ `Avoid: something is close ahead, so back off and slide`

> **Answer:**
>
> ```
> Emergency: the battery is low, so stop
> Avoid: something is close ahead, so back off and slide
> Seek: head for the goal
> ```
>
> Safety beats not hitting things, and not hitting things beats getting to the goal. The actual job is the lowest layer.

3. Once avoid takes over, it runs for ten ticks whatever the sensor says. Why? [1 mark]

- ☐ **A** So the robot actually gets clear instead of flickering between avoid and seek
- ☐ **B** Because the distance sensor only works every ten ticks
- ☐ **C** So seek can finish its turn first
- ☐ **D** To save battery

> **Answer:** A. If avoid let go the moment the reading was fine, seek would steer straight back at the wall a tick later.

4. What does this program print? [1 mark]

```
def choose(battery, dist, avoiding):
    if battery < 20:
        return "stop"
    elif avoiding or dist < 22:
        return "avoid"
    else:
        return "seek"

print(choose(80, 50, False))
print(choose(80, 15, False))
print(choose(10, 15, True))
print(choose(80, 60, True))
```

Answer:

```
seek
avoid
stop
avoid
```

The first true test wins. A low battery beats everything, and an avoid that is still running carries on even when the way looks clear.

5. What does this program print? [1 mark]

```
state = "seek"
avoid_ticks = 0
for d in [50, 20, 18, 30, 15, 60]:
    if state == "seek" and d < 22:
        state = "avoid"
        avoid_ticks = 0
    if state == "avoid":
        avoid_ticks += 1
        if avoid_ticks >= 3:
            state = "seek"
    print(d, state)
```

Answer:

```
50 seek
20 avoid
18 avoid
30 seek
15 avoid
60 avoid
```

Avoid lasts three ticks. On the third, it hands back to seek, and the next close reading, 15, starts it again.

6. Does `steer_to`, the seek behaviour, need to know that avoid exists? [1 mark]

- ☐ **A** No: it just finds itself somewhere new and heads for the goal again
- ☐ **B** Yes: it has to undo the sideways slide
- ☐ **C** Yes: it must turn avoid off when it is done
- ☐ **D** No, because avoid never moves the robot

Answer: A. That is the beauty of layers. You can add one above without touching the ones below.

7. Why does `steer_to` only drive forward once the robot is facing within 25 degrees of the goal? [1 mark]

- ☐ **A** So the distance sensor looks where the robot is going
- ☐ **B** Because the robot cannot turn and drive at once
- ☐ **C** So it reaches the goal faster in a straight line
- ☐ **D** Because `drive` ignores speed while turning

> **Answer:** A. The distance sensor points forward. Facing the way you drive means avoid can see what you are about to hit.

The task: seek, but safely

Reach the green zone at (82, 82) without touching either wall.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

# maths: atan2, hypot, sin, cos, radians
import math

def wrapped(h):
    return (h + 180) % 360 - 180

def steer_to(x, y, speed=60):
    # where am I?
    px, py = position()
    bearing = math.degrees(math.atan2(x - px, y - py))
    error = wrapped(bearing - heading())
    # forward, sideways, rotation: -100 to 100 each, until the next command
    drive(speed if abs(error) < 25 else 0, 0, max(-60, min(60, error * 3)))

def near(x, y, cm=6):
    # where am I?
    px, py = position()
    return math.hypot(x - px, y - py) < cm

goal = (82 - 15, 82 - 15)
# do this 390 times (tick counts from 0)
for tick in range(390):
    if near(*goal, cm=5):
        # leave the loop
        break
    steer_to(*goal, speed=60)
    # pause 0.1 s (the robot keeps doing what it was told)
    wait(0.1)
# all motors off
stop()
```

The hint students can ask for: Head for the green zone at (82, 82), but avoiding things comes first: when the distance sensor sees something close, deal with that before seeking again.

A solution

```
from bugbot import *
connect()
def wrapped(h):
    return (h + 180) % 360 - 180
import math

def steer_to(x, y, speed=60):
    # one tick of driving towards (x, y), relative to the start: turn to face it, go when
roughly facing it
    px, py = position()
    bearing = math.degrees(math.atan2(x - px, y - py))
    error = wrapped(bearing - heading())
    drive(speed if abs(error) < 25 else 0, 0, max(-60, min(60, error * 3)))
```

```
def near(x, y, cm=6):
    px, py = position()
    return math.hypot(x - px, y - py) < cm
goal = (82 - 15, 82 - 15)          # relative to the start
state = "seek"
avoid_ticks = 0
for tick in range(390):
    if near(*goal, cm=5):
        break
    if state == "seek" and distance() < 22:
        state = "avoid"                  # avoiding takes over completely...
        avoid_ticks = 0
    if state == "avoid":
        drive(-10, 60, 0)                # ...back a touch and slide right
        avoid_ticks += 1
        if avoid_ticks >= 10:
            state = "seek"               # ...then seeking gets the robot back
    else:
        steer_to(*goal, speed=60)
    wait(0.1)
stop()
```

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