WORKSHEET

A3.9 Project: mission control

Data structures · A level · OCR H446 1.4.2, AQA 7517 4.2.1.1, Eduqas A500QS 1.1 · about 30 min

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

A mission read from a file into a circular queue, moves looked up in a dictionary, position tracked as a vector, and an undo stack to bring the robot home.

- The brief
- The mission file
- Plan
- Step 1: the mission into a queue
- Step 2: moves as vectors
- Step 3: put it together

Questions

6 marks in all

1. A robot must be able to undo its moves, most recent first. Which data structure should hold the moves? [1 mark]

- ☐ **A** A stack
- ☐ **B** A queue
- ☐ **C** A hash table
- ☐ **D** A vector

2. Commands arrive by radio faster than the robot can carry them out. They must be carried out in the order received, using a fixed amount of memory. Which structure fits best? [1 mark]

- ☐ **A** A circular queue
- ☐ **B** A stack
- ☐ **C** A dictionary
- ☐ **D** A linked list

3. Which of these statements are true? [1 mark]

Tick every answer that is true.

- ☐ **A** A dictionary gives fast look-up of a value by its key
- ☐ **B** A stack returns items in the order they were added
- ☐ **C** Adding a displacement vector to a position vector translates the position
- ☐ **D** A hash table keeps its keys in sorted order

4. This program works out a position by dead reckoning. What does it print? [1 mark]

```
DIRECTION = {"forward": [0, 1], "right": [1, 0], "backward": [0, -1]}
pos = [0, 0]
for word, cm in [("forward", 20), ("right", 15), ("backward", 5)]:
    d = DIRECTION[word]
    pos = [pos[0] + cm * d[0], pos[1] + cm * d[1]]
print(pos)
```

5. Items go from a queue onto a stack and then off the stack. What does this program print? [1 mark]

```
queue = ["a", "b", "c", "d"]
stack = []
while queue:
    stack.append(queue.pop(0))
out = []
while stack:
    out.append(stack.pop())
print(out)
```

6. Why does the mission program load every command into the queue before the robot moves? [1 mark]

- ☐ **A** A mission too long for the queue is found before anything happens
- ☐ **B** The robot cannot read a file while it is driving
- ☐ **C** A queue can only be filled once
- ☐ **D** It makes the queue sort the commands

The task: mission control

Build the program from the brief. `slide(v)` is given: it drives the displacement vector `v = [x, y]` in cm (x to the right, y forward). Use fixed-size arrays with pointers for the queue and the stack, not the list's own `append`, `pop` or `insert`, and do not call `position()`. 1. Read every line of `mission.txt` (`<word>,<cm>`, where `<word>` is `forward`, `backward`, `left`, `right` or `abort`, and `<cm>` is a whole number) into a circular queue of size 8, wrapping the pointers with `%`, before the robot moves. 2. Keep a stack of size 8 with a `top` pointer, a dictionary `DIRECTION` from each direction word to its unit vector, and a position vector `pos` starting at `[0, 0]`. Do not compare the word with each direction name. 3. While the queue is not empty, dequeue a command: - A move: drive it with `slide`, push `(word, cm)`, add the move to `pos`, and print `at <pos>`, for example `at [0, 30]`. - `abort`: print `abort: <n> commands left in the queue`, where `n` is the number still in the queue. Then, until the stack is empty, pop a move, drive it in reverse, update `pos`, and print `undo <word> <cm>`, for example `undo forward 25`. Then stop taking commands. 4. Finally print `home at <pos>`.

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

def slide(v):
    """Drive by the displacement vector v = [x, y] in cm: x is sideways (right positive), y
is forward."""
    if v[0] > 0:
        right(50, distance=v[0])
    if v[0] < 0:
        left(50, distance=-v[0])
    if v[1] > 0:
        forward(50, distance=v[1])
    if v[1] < 0:
        backward(50, distance=-v[1])
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a3-9-project-mission-control/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Give each command a priority in the file, so an `abort` jumps the queue as soon as it is read. Which kind of queue do you need?
2. Store every position the robot reaches in a dictionary from `(x, y)` tuple to the step number, and report if the mission ever visits the same place twice.

3. The stack and queue are both size 8. What should happen if a mission has 9 moves before its abort? Change the program so it refuses to start rather than failing half way.