Drones 2D

You may have watched the slalom race where Marcel Hirscher was nearly hit by a drone.

And you may have followed the public discussions to use drones for delivering goods. This Catalysts Coding Contest picks up the topic and is about controlling (a fleet of) delivery drones to transport packages to customers.

Simulation Model

- Drones are identified by distinct IDs ranging from 0 to *n*-1 (when there are *n* drones).
- Each drone has a current *location*, *direction* and *speed*.
- The *location* is given as integer x- and y-coordinates of a cell in an infinite 2D grid.
- The *direction* is one of N, NE, E, SE, S, SW, W, NW.
- The *speed* is an integer from 0 to 5 grid units per iteration.
- The drones move simultaneously in iteration steps:
    - with direction N, the y-coordinate is increased by *speed*;
    - with direction E, the x-coordinate is increased by *speed*;
    - with direction S, the y-coordinate is decreased by *speed*;
    - with direction W, the x-coordinate is decreased by *speed*;
    - with direction NE, SE, SW, NW, x- and y-coordinates are both changed as combination of the above rules.

Example: move from (1,0) with *speed*=2 in SW-direction to (-1,-2):

...	...	...	...	...	...
...	(-1,1)	(0,1) NW	(1,1) N	(2,1) NE	...
...	(-1,0)	(0,0) W	(1,0) before	(2,0) E	...
...	(-1,-1)	(0,-1) SW	(1,-1) S	(2,-1) SE	...
...	(-1,-2) after	(0,-2)	(1,-2)	(2,-2)	...
...	...	...	...	...	...

The Simulator

All the computation regarding simulating the drones is covered by a component we call the "simulator". It …

- holds state of the whole simulation, including all objects that are being simulated;
- can output a portion of that state via commands;
- handles your commands and interaction with your drones;
- checks for constraints, such as drone crashes;
- aborts the simulation in case of malformed commands or violated constraints;
- tells you when you successfully complete a test case and creates a file (in its working directory) that you will need to upload for verification.

It is implemented using Java 8 and can be obtained from

https://storage.googleapis.com/coduno/drones-2d/simulator.jar

You can run the simulator from the command line with 2 or 3 positional arguments like this:

```
java -jar simulator.jar level test [port]
```

level	mandatory	level number (1 to 5)
test	mandatory	test case number (1 to 3)
port	optional	TCP port number to bind to

 For questions related to the level descriptions, go here
For other questions (organizational, technical, arbitrary), go here

If you omit the TCP port number, the simulator will process I/O using its standard streams. You'll have to redirect them appropriately.

Note: When running through our Coduno web interface, you can reach the simulator via simulator:7000 (that's hostname "simulator" and TCP port 7000). Memory limit on Coduno is 64 MiB.

Interacting with the Simulator

Interaction with the simulator is line based. Each line is terminated by a newline (`\n`, the non-printable ASCII newline character, **not** a carriage return character `\r` or a combination of those).

After starting the simulator

- first receive a line from it that specifies the test case you have to solve;
- then perform commands until the test case goal is reached:
    - send the request line
    - receive the response line

The syntax of the test case specification line and of the available commands is explained in the respective level descriptions.

Hint: Make sure you flush the stream. Often times, this will automatically happen if you write a newline. To be on the safe side, you may want to flush the stream manually.

Level 1

There is **one drone** (with *id*=0, initial *speed*=0 and *direction*=N).
Move it from its starting location to a target location.

Test case syntax:

```
startX startY targetX targetY
```

startX startY	starting location
targetX targetY	target location

Available commands:

request	**response**	**description**
STATUS *id*	*x y dir speed*	query current status of drone *id*
DIRECTION *id dir*	OK	set *direction* (N,NE,E,SE,S,SW,W,NW) of drone *id*
SPEED *id speed*	OK	set *speed* (0-5) of drone *id*
TICK	*ticks*	perform an iteration step: response is either SUCCESS when the goal is reached or else *ticks*, the number of iteration steps so far
	SUCCESS	

Reminder: The simulator aborts the simulation in case of malformed commands or violated constraints.

 For questions related to the level descriptions, go here
For other questions (organizational, technical, arbitrary), go here

Example:

test case	`2 1 -1 0`

...	...	...	...	...	...
...	(-1,1)	(0,1)	(1,1)	(2,1) start	...
...	(-1,0) target	(0,0)	(1,0)	(2,0)	...
...	(-1,-1)	(0,-1)	(1,-1)	(2,-1)	...
...	...	...	...	...	...

Possible solution:

Commands			**State**
request	**response**		**location**
`DIRECTION 0 W`	`OK`		(2,1)
`SPEED 0 2`	`OK`		(2,1)
`TICK`	`1`		(0,1)
`DIRECTION 0 SW`	`OK`		(0,1)
`SPEED 0 1`	`OK`		(0,1)
`TICK`	`SUCCESS`		(-1,0)