Drones

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.

Your drones will operate in three dimensions (x, y, z), constrained to a predefined area (x, y), or simply "map" (i.e. z is not constrained). The map models the real world. You'll have to navigate your drones and avoid collisions.

What does a Drone look like?

As our simulation is rather primitive, a drone is defined as follows:

Dimension	Value	Unit
Weight	2	kg
Length	1	m
Width	1	m
Height	0.5	m
Rotor count	8	1
Diameter of a rotor	0.25	m
Permitted maximum velocity	14	m/s

Commanding your Drones

Interfacing with your drones, you will have a small set of instructions that your program will have to emit, probably also process input from. By issuing commands you can for example control your drone's throttle (which influences the power of the rotors and therefore the thrust it generates), find out their location, velocity and much more. Commands are readable text, communicated either through TCP or standard streams.

Commands usually look like...

```
<COMMAND NAME> [<PARAMETER> | ...]
```

…a word that characterizes the command, followed by some optional parameters. It is not necessary to terminate the commands you are sending with a newline, however the order of parameters and completeness of a command is important.
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.

Responses will be one or more lines, each terminated by a newline (`\n`, the non-printable ASCII newline character, **not** a carriage return character `\r` or a combination of those).

Identifying a Drone

To consistently refer to drones, each drone is labelled with an ID. An ID is an integer, ranging from zero (inclusive) to number of drones (exclusive), very similar to indexing an array/list in many programming languages (C, Java, Go, JavaScript, …). The ID does not change during one simulation, so the same ID will always refer to the same drone. Also, IDs are consecutive, meaning that all integers ranging between zero (inclusive) and the number of drones (minus one) refer to a drone.

Physics

Our simulation (implemented in our simulator) is obviously not perfect (there is no such thing), but sufficiently complex for the matter of this contest. Your drones are octocopters (they have eight rotors). The thrust exerted by a single rotor is approximated as:

$$T = \sqrt[3]{\tfrac{\pi}{2} \cdot D^2 \cdot \rho \cdot P^2}$$

$$\rho = 1.225\ kg/m^3 \quad D = 0.25m$$

Where D is the diameter of a rotor, ρ is the air density, P is the power of the rotor.

The power of a rotor in turn is influenced by the *throttle*:

$$P = C * (throttle * R_{max} / 1000rpm)^f$$

$$C = 0.015W \qquad R_{max} = 10000rpm \qquad f = 3.2$$

Where C is the rotor constant, R_max is the maximum rotation frequency of the rotor, f is the rotor power factor, and t is the throttle.

This means you cannot directly change the thrust of your drone. Rather you have to set a throttle which in turn affects the frequency of rotation of the rotors of your drone, which in turn affects the power of the rotors, that is factored in when computing the thrust.

Hint: The gravitational force is not mentioned here, but it will affect your drone with a force of 9.80665 m/s^2 according to the International System of Units, NIST Special Publication 330, 2008 Edition (page 52).

Further Reading (not required for continuing with the contest):

http://www.wired.com/2014/05/modeling-the-thrust-from-a-quadcopter/
http://www.wired.com/2013/12/physics-of-the-amazon-prime-air-drone/
http://www.wired.com/2013/10/physics-of-the-new-shield-helicarrier/
https://quadcopterproject.wordpress.com/static-thrust-calculation/

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 (with the ground, with other drones, with obstacles, …)
- aborts the simulation in case hard constraints are violated (a crash, a malformed command, you are taking too long to complete the task at hand, you name it, …)
- 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/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 7)
test	mandatory	test case number (1 to 3)
port	optional	TCP port number to bind to

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.

Level Descriptions

Notation convention

Variables denoted with capital letters represent integers, while variables denoted with lower letters denote floating point numbers. You will see that when introducing new commands and explaining initial state.

Initial Input

When you first connect to the simulator via TCP or standard streams, it will write the level configuration and important initial state information to you. For example, the number of drones you have to control, further details about objectives, etc. Please consume that data and interpret it.

Level 1

Take off all your drones and let them climb to a given height without crashing (e.g. via wrong commands or illegal parameters) **or exceeding the permitted maximum velocity.**

You have 10 (virtual) seconds to accomplish that task. That is, not real time, but simulation time.

Model in the Simulator

1. Drone
    a. ID (integer)
    b. Position (3D [m], floating point)
    c. Velocity (3D [m/s], floating point, speed in axial directions), initially (0.0,0.0,0.0)
    d. Thrust direction (3D, floating point, normalized for computation, magnitude irrelevant), initially (0.0,0.0,1.0)

Initial Input Lines sent by the Simulator

`xmin xmax ymin ymax`	constrained area
`N`	the number of drones available (always 1 for this level)
`height`	minimum height to climb

Commands

The status command lets you request status information of a specific drone. You will get back its position and velocity in different directions.
Note that while the thrust orientation can be read using `STATUS`, it is not necessary to complete this level.

request	**response**	**description**
`STATUS` *`N`*		where *N* is the ID of a drone
	`x y z vx vy vz rx ry rz`	where *x, y* and *z* is the position *`vx, vy, yz`* is the velocity, and *`rx, ry, rz`* is the thrust orientation vector.

In order to accelerate a drone, you need to set a throttle. The throttle will influence the rotation frequency of the rotors on your drone, which in turn affects the generated power and therefore the generated thrust.

request	**response**	**description**
`THROTTLE` *`N x`*		where *N* is a drone ID and *x* is a number between 0.0 and 1.0 (both inclusive)
	`OK`	indicates that your command was processed

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

When you have sent all commands for all drones, you can advance in time with the following TICK command (i.e. use the physics to move any objects).

request	**response**	**description**
`TICK` *`dt`*		where *`dt`* is the amount of time in seconds you want the simulation to run (must not be negative)
	`t`	where *`t`* is the elapsed time since the simulation was started in seconds (test case not yet completed)
	`SUCCESS`	test case successfully completed