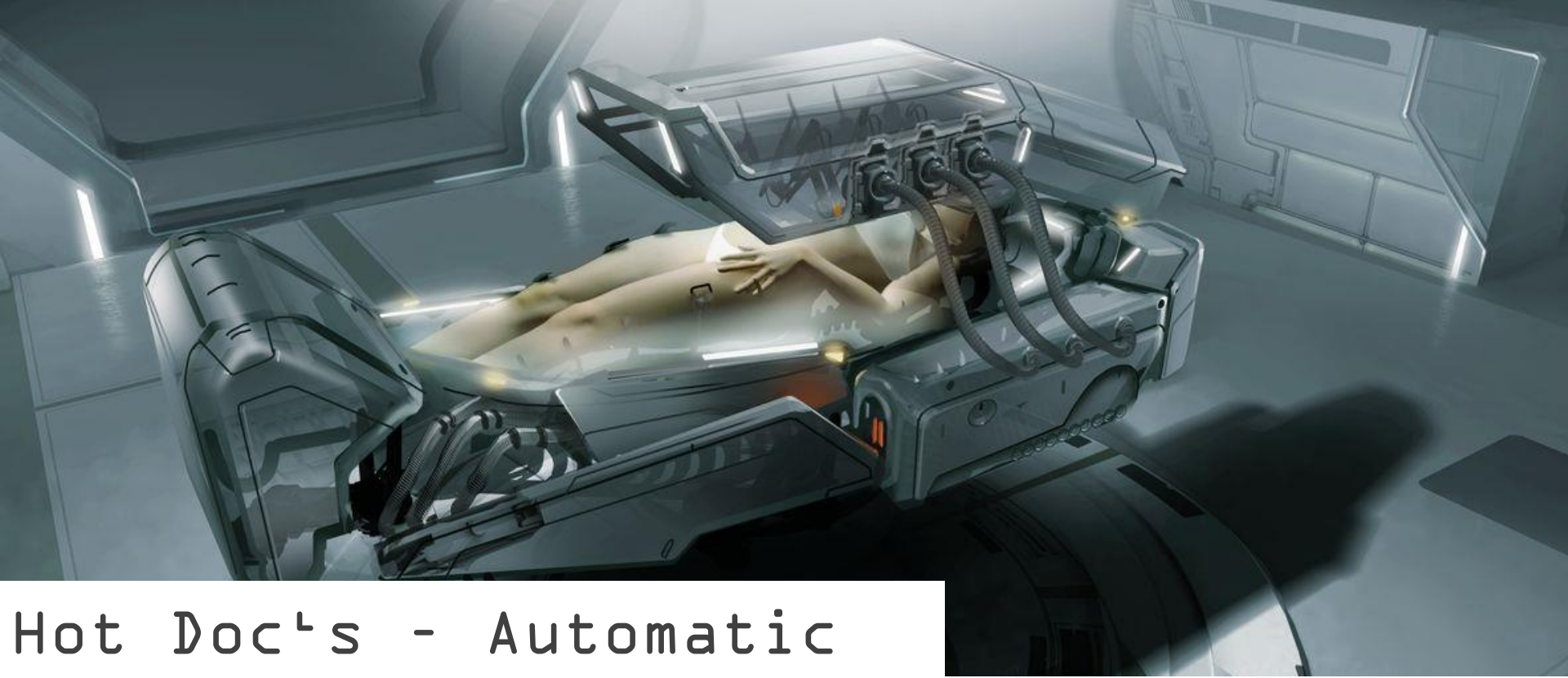


Hot Doc's - Automatic Surgery

Catalysts Coding Contest
Linz 2015

Hot Doc's

2099: A surgery is done in a surgery machine. Some people will make it at home. Others might go to a surgery centre.

Common to all is, that human interaction during the surgery is not necessary anymore.

Your doc is a computer!

The computer needs to move you in the right position for the next surgery step.

Your task in this CCC is to write a control program for the surgery table. Your software must make the right movements to bring the patient in an appropriate surgery position.

In every level you have to achieve goals in a simulator.
You are provided with a simulator and your program has to communicate with it in real time.

General Information

Info

The surgery table-surface consists of many thousand rods, which you can lift and lower. In a lifted position you can move an object or a body to the front, back, left or right.

To visualize the movements of your control software and to test it, you must use the simulator.

Simulator

Info

The simulator only does exactly what you tell him. He awaits your next command and reacts accordingly to it. The communication is entirely text-based. One line sent is one command and gets a line of response every time.

The simulator is written in Java and to run it you will need Java 7 (Update 6) or higher installed.

There are two different ways how you can communicate with the simulator. You only have to implement one method though:

- Console mode
- TCP mode

Simulator - Console Mode

Info

The simulator executes your program and redirects stdin and stdout.
If you choose this method your program has only to interact with the "console". The downside of this approach is that you can't easily debug your application. Of course you always can attach your debugger after the program has started. You can also write debug output to stderr, which will be shown in an debug window of the simulator.

Example start commands:

```
java -jar simulator.jar --level=level1.in --exec=mysolution.exe
java -jar simulator.jar --level=level2.in --exec="java -cp myclasspath my.company.Main"
java -jar simulator.jar --level=level3.in --exec="c:\python\python.exe solver.py"
```

Simulator - TCP Mode

Info

The simulator opens a tcp port and your program connects to it.

Example start commands:

```
java -jar simulator.jar --level=level1.in --tcp=7000
```

Simulator - Usage

java -jar simulator.jar --level=<level-description> [--exec=<solver>] [--tcp=7000]

Arguments:

`--level=<level-description>`
selects the level description file (optional, default: level.in)

`--exec=<solver>`
start in Console Mode

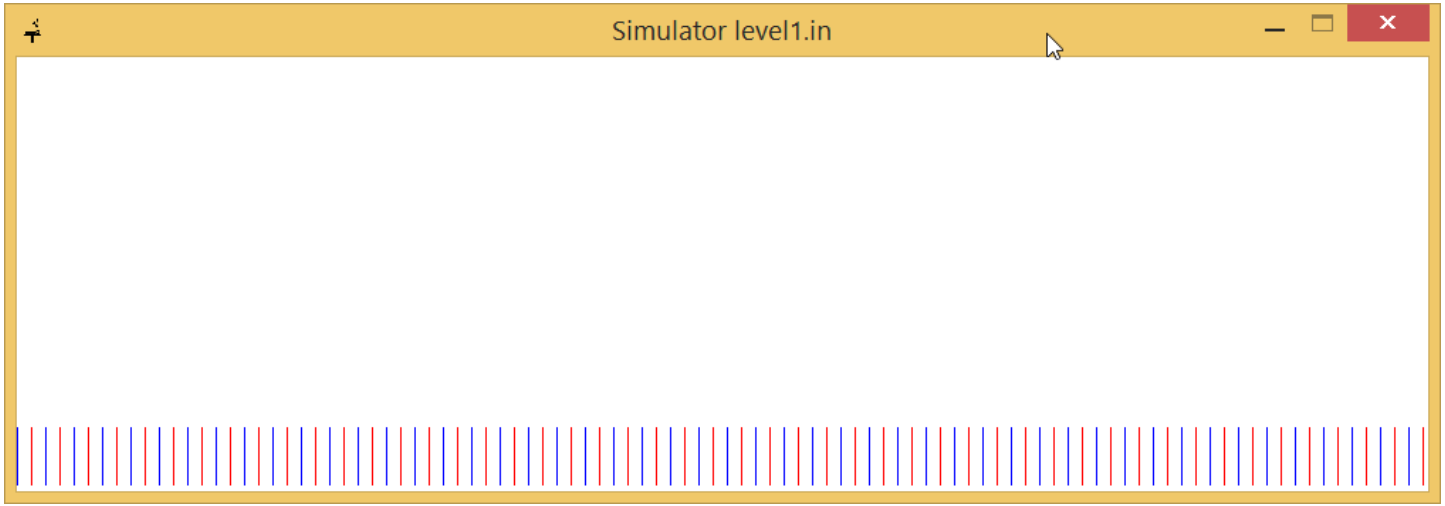
`--tcp=<port>`
start in TCP Mode

Simulator

Level 1

Your task is to connect to the simulator with one of the two methods, and achieve the level goals. After you have achieved all goals the simulator will write a “levelX.out” file to the current working directory (where X is the number of the current level). Simply upload the generated file to advance to the next level.

All rods can only be moved to the left or to the right. The simulator displays the rods in a side view. A rod has a distance of 10 units to the adjacent rods.



Simulator

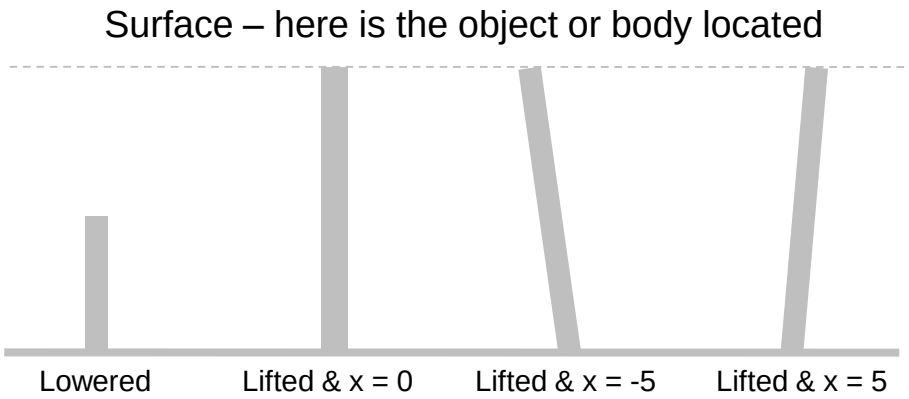
Level 1

A position change of a rod follows the following steps:

- Selected lowered rods will be lifted to their correct position (without moving the object / body – it just adds friction)
- Unselected rods are lowered
- Selected lifted rods will be moved to their new position (affecting the object / body)

Moving a body is done by rods in lifted state either from the last step or the current step. The movement of the body is a result of the average movement of all lifted rods. A moved rod may rub against a stopped rod. The movement of the body is reduced by this friction.

The maximum excursion of a rod is +/- 5 units. An excursions higher than this value is automatically limited by the simulator during execution of the MOVE-command.



Simulator

Level 1

Goal Requirements:

- Lower every second rod
- Lift the lowered rods to position -5 and lower the rods in between
- Move the lifted rods to position 5
- Lift the lowered rods to position 0
- Lower the lifted rods which are in position 5
- Lift the lowered rods to position 0

End condition:

- All rods are in the starting position, which is: lifted and no deviation to any direction

Constraints:

- At any given moment at least half of the rods have to be in the lifted state
- You have to follow the steps above one by one, don‘t try to achieve to steps with a single move command

Simulator Communication Protocol

Info

The communication with the simulator is text and line based. Every message is one line of text. Each message sent by your program will be followed by a response from the simulator.

Available client messages / commands:

MOVE rodNumber deviation {rodNumber deviation}

`rodNumber := <integer>`

- starting with 1
- listed rod are raised or moved
- non-listed rods are lowered (if not already lowered)

`deviation := <integer>` move rod in mm (range from -5 to 5)

- The deviation can be positive or negative and is an integer.
- Actuators which aren‘t included in the command are automatically lowered
- Valid ‚**MOVE**‘ commands are acknowledge with ‚**OK**‘
- Invalid ‚**MOVE**‘ commands will be answered with ‚**ERROR**‘ followed by a description of the error

Exampel:

```
MOVE 1 2 4 -5 3 -2
OK
```

Simulator Communication Protocol

Info

EXIT
- Exits the simulator.
- Returns ‚**OK**‘ if the level goal has been reached
- Returns ‚**ERROR**‘ followed by a error description in case the goal hasn‘t been reached yet

GET_POSITION
Returns the position (x coordinate) of the center of gravity as floating point number
Example:

GET_POSITION
255.23

GET_NUMBER
Returns the number of rods
Example:

GET_NUMBER
100

Simulator Communication Example

Info

Client command	Simulator response
GET_NUMBER	5
MOVE 2 0 4 0	OK
MOVE 1 5 2 0 3 5 4 0 5 5	OK
MOVE 1 5 3 5	OK
MOVE 1 -5 3 -5	OK
MOVE 1 -5 2 0 3 -5 4 0 5 -5	OK
MOVE 2 0 4 0	OK
MOVE 1 0 2 0 3 0 4 0 5 0	OK
EXIT	OK