From Helsinki with Love

Secret Agent 00L

My Name is Baud, James Baud. I have never seen my superior; when he contacts me, he simply calls himself "The Penguin". He recently sent me a LEGO Spybot to help me with my next mission. To cause havoc at the very heart of our enemies citadel. Using the latest technological gadgets, the mission initially seems impossible.

BY VOLKER SCHMITT

The Penguin's instructions are usually clear and concise, but there are some occasional surprises. His last message simply told me to pick up a parcel from a locker at the main station. Much to my surprise, the parcel contained a LEGO Spybot robot in its original packaging (shown assembled in the secret agent photo above); I picked it up and read the Penguin's instructions. They read as follows:

```
"Agent Baud! We need an agitator
on my adversary William
Wallopening's premises. His main
building is tightly secured.
Find a way in for the Spybot.
What ever you do, do not use
Windows. Additional information
and instructions to follow."
```

That was typical of the Penguin: *"Do not use Windows"* was one of his standing orders.

Construction Plans

Before I turned to the enemy's building, I first needed to study my little agent's helper. When I unpacked the Spybot, I discovered only the rump, an infrared transmitter, and a few bags of small parts. In addition to this, the parcel contained a serial cable with an infrared transmitter/receiver attached to one end (secret agent photo 1). The plug fitted exactly in the rear of the Spybot's rump – okay, that took care of hitching up my technical pal to my computer.

Time for the next task: Unfortunately the assembly instructions were not included in printed form, and had to be downloaded from a CD-Rom (that included software for Windows 98, ME and XP only). I tried to work around this obstacle using the Windows emulator **wine**. After successfully installing the Spybotics software and the *QuickTime* movie player, it looked like I might be on the right track – but my hopes were dashed. After launching the software, the intro flashed across the screen (secret agent photo 2). But unfortunately, the LEGO-*wine* software combination hung while looking for the cable (secret agent photo 3). I tried a variety of different configurations, but all to no avail.

To help *wine* find the robot, I suddenly hit on the plan of tricking the software into thinking it had already found it. I contacted a friend – a double agent (whose employees are not familiar with the *"no Windows "* standing order) – and asked him to get me a **User.ini** file for the first serial port (that is, */dev/ttyS0*). I

THE AUTHOR

Volker Schmitt is a mathematician and works for a large insurance company. He is familiar with multitasking programming from experience with mainframe PL/1 timers. In his leisure time Volker has a lot of fun with NQC and his LEGO Spybot.

GLOSSARY

wine *is not really a genuine emulator that emulates Windows on Linux. Instead wine imitates system and library calls to Windows and maps these to Linux calls [1].*

User.ini: *A user specific file containing program configurations that run on Windows (or wine).*

copied the file to the correct location in my *wine* configuration.

And although the *wine*-LEGO team did manage to get a bit further this time, the program hung again after only a short while. Surprisingly, *winex*, an alternative *wine* specially customized for games, crashed even earlier than *wine* itself and refused even to install the software.

Was this the end of my mission, simply because I had no construction plans for the Spybot? I took a closer look at the LEGO CD-Rom. Luckily for me, the individual construction steps were stored in JPG format in the *Shared/build/* directory. And as this CD ROM is supplied with the whole Spybot range, this is the place to look, no matter what construction plans you need.

A quick look at the box showed the article number for my Spybot Gigamesh G60: It was *3806*, and I couldn't believe my luck when I discovered that the JPG files *3806-*.jpg* had actually been assigned consecutive numbers in the correct order. LEGO provided short video sequences to support more complex steps. And this is where the *QuickTime* program I had previously installed came in useful. Typing

```
user@host:~$ wine "C:\Pro➘
gram Files\LEGO Software\Pro➘
ducts\QuickTime\QuickTime➘
Player.exe 3806-08a.mov"
```

displayed the required video on screen, no problem. This setup allowed me to assemble my technical collaborator without any difficulty. The Spybot has pressure sensors at the front, and drives on the left and right. LEGO provides a neat solution for the IR port. When you remove the serial cable, you can attach the laser cannon to the IR outlet and the light meter to the IR inlet port.

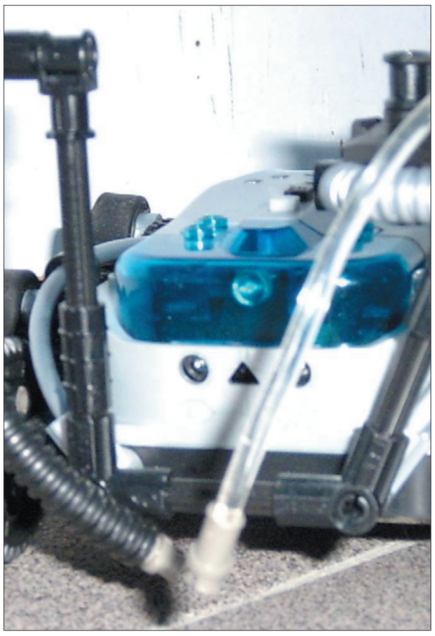
Figure 1: IR plug of the LEGO Spybot

Training!

My friend the double agent had not only supplied me a *User.ini* file, but passed on some information on the capabilities of the LEGO software. The software contains ten fascinating missions for the Spybot, which I could have used to prepare myself for the Penguin's mission – at least this is what I assume judging from the videos on the CD ROM. As I could not get the software to run, I had to make do with the standard program that LEGO downloads to the Spybot by default. After pressing the start button, the LEDs lit up, and the robot set off: After a short while, it turned slightly and set off again. Whenever it hit a wall, a signal from the pressure sensor made the Spybot say "Ouch".

Time to take a closer look at the Spybot's IR controller (secret agent photo 5). There are three modes: remote control, link and action mode. The four buttons on the controller allow you to operate the left and right drives in forward or reverse direction. In action mode, the red button fires the Spybot's laser cannon. These new insights made me confident of being able to complete the mission the Penguin had set me, and lo and behold, the next day the instructions popped up in my mailbox the next day:

```
"Baud! Another agent has
discovered that the alarm system
is triggered by infrared light.
Set the alarms off time and time
again during the night to make
our enemy think the alarm system
has broken down and switch the
system off. Zero hour coincides
with the next new moon."
```

Do-It-Yourself Spy Programs

As I could not make use of the standard programs on the LEGO CD, I intensified my secret inquiries and finally discovered something important at [2]. You can use the NQC ("Not Quite C") tool to program Spybots as of version 2.5r1 on Linux. You can either load the sources from the program's Website (using the *wget http://www.baumfamily.org/nqc/release/nqc-2.5.r1.tgz* command for example). Then use the following command

```
tar -zxvf nqc-2.5.r1.tgz
```

to unpack the archive file. This creates a new directory called *nqc-2.5.r1*, where you can execute

```
make
```

Agent Listing 1: forwards, back and stop with *forw_a_back.nqc*

```
#include "spy.nqh"      // every Spybot program needs this
task main()
{
  OnFwd(OUT_A+OUT_B);  // both drives ahead
  Wait(300);           // run for 3 seconds
  OnRev(OUT_A+OUT_B);  // both drives back
  Wait(300);           // run for 3 seconds
  Off(OUT_A+OUT_B);    // stop
}
```

Agent Listing 2: An NQC Inline Function

```
void two_three(int value_var, int
&reference_var)
{
  value_var=2;      // value_var keeps its
value until the end of the func
  reference_var=3;  // the value of 3 is
retained after the call to the func
}
```

and

```
make install
```

to install NQC. This assumes that you have installed the *bison* program previously, and this is not typically the case for standard Linux installations.

NQC

NQC programs have a similar syntax to C. Instructions are terminated by semicolons, and blocks are enclosed in curly brackets { }. Comments are indicated by /* and */ or follow // if they occupy only one line. The control structures *if*, *while*, and *for* correspond to their C counterparts. Additionally, NQC provides *repeat* and *until* keywords (see Box 1). Just like C, NQC needs a main program, which defines a task as follows: *task main()* (see Listing 1). This (and any other) task can contain instructions to the robot. Let's look at the following simple instruction as an example: *OnFwd(OUT_A + OUT_B)*.

This contains quite a lot of information: two outputs, *OUT_A* and *OUT_B* (linked by a + character), are passed as parameters. The *OnFwd()* is itself a combination of *On()* and *Fwd()* and includes both the power on command and a direction.

Listing 1 also shows the *Wait()* function. This function expects to be passed a value in hundredths of seconds that the robot should carry on doing whatever it is doing. *OnRev(OUT_A + OUT_B)* switches the drives into reverse, and *Off(OUT_A + OUT_B)* stops the drives. The next step was to test the Spybot with a simple program, that made it go ahead for three seconds, and then back for three seconds, before finally stopping. The following command:

```
user@host:~$ nqc -TSpy -d ↲
forw_a_back.nqc
```

compiles the program and uploads it directly to the robot. If an error occurs during compilation, NQCs error messages provide enough information to troubleshoot the source code. The *-TSpy* option is mandatory for every Spybotics program; *-d* uploads the compiled program directly to the robot. Of course, this assumes that the cable is connected and the Spybot switched on. The first serial port is used by default; the *-S* parameter allows you to select another port, e.g. *-S/dev/ttyS1*.

Figure 2: LEGO Intro

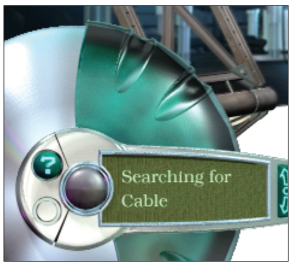

Figure 3: Looking for cables

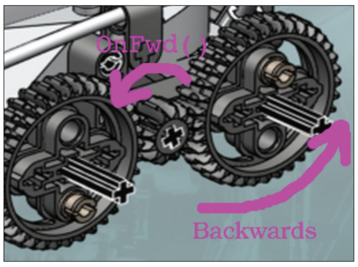

Figure 4: Converting to reverse direction

Forward March – in Reverse?

After uploading my *forw_a_back.nqc* program to the Spybot, and launching it, the Spybot first went into reverse (instead of moving forward) and changed direction after three seconds. This is not a program error, but a constructional whim. The gearwheel on the drive meshes directly with the larger gearwheels of my Gigamesh G60 (secret agent photo 4), so you would need to swap the forward and reverse directional commands in your programs, if NQC did

Box 1: NQC Control Structures

```
if(condition)
{conditional instruction}
else
{alternative instruction}

while(condition){instructions}

do{instructions}
while(condition)

for(initial
instruction;condition;subsequent
instruction)
{repeat instructions}

repeat(numerical expression)
{instructions}

switch(expression)
{
  case constant_expression_1:
  {instructions_1} break;
  ...
  case constant_expression_n:
  {instructions_n} break;
  default:
  {instructions} break;
}

until(expressions)
{expressions}
```

Figure 5: The IR Controller

not provide a *SetGlobalDirection()* command to do exactly that. The *OUT_REV* parameter simply inverts all of these commands. Box 2 contains a list of actuation commands for the Spybot.

Hierarchy

In addition to tasks, NQC also defines sub-routines (*sub()*) and inline functions (*void()*). The advantage of inline functions is that you can pass parameters to them using either **call by value** or **call by reference** techniques. NQC programs can only handle *integer* variables. Agent Listing 2 shows a typical inline function, which is called by its name (*two_three();*) and can be terminated and quit using the *return;* command. NQC provides sub-routines as an alternative; a total of 32 sub-routines can be defined per program. Although you cannot pass parameters to a sub-routine, they do have the advantage of allowing multiple callers to use a single instance in memory. Sub-routines cannot call other sub-routines.

You can define a total of 32 global and four local variables, and this is why variables should be defined as locally as possible. Variables can be manipulated using the same operators used with *integer* variables in C programming: +, -, *, /, % (mod), *abs()*, *sign()*, ++, –, ~ (bitwise negation), *!* (negation), >>, <<, ==, !=, & (bitwise AND), ^ (bitwise XOR), | (bitwise OR), && (AND), || (OR), >, <, >=, and <=.

The Spybotics program restricts the total number of tasks to eight. Tasks normally run in parallel, that is, where the caller of a sub-routine waits till the routine has completed, task execution carries on without waiting. This technique allows you to write genuine multitasking programs in NQC. Box three shows the individual task commands.

The fact that the Spybot is equipped with sensors is particularly useful to my mission. My agent program will need to co-ordinate certain actions, depending on the status of the sensors. The first sensor is the pressure, which NQC refers to as *SENSOR_1*. The pressure sensor is normally set up to return Boolean values, that is 0 (status "not pressed") and 1 (status "pressed"). NQC refers to this as *SENSOR_MODE_BOOL*. A light sensor () *SENSOR_2* returns percentage values (0 through 100) and is referred to as *SENSOR_MODE_PERCENT* in NQC.

NQC additionally defines *SENSOR_MODE_RAW* for the Spybot – this is an internal representation of the sensor value for the robot (returning a value between 0 and 1024). You can query the status of a sensor by assigning a variable, e.g. *x = SENSOR_1*; additionally, using a query such as *if(SENSOR_1 = = 1)* allows you to discover whether the sensor has been pressed or not. Box 4 contains a list of sensor commands.

Sound

The Spybot can emit sounds. Box 5 provides an overview of the commands used to do so. You can access the laser either by doing *On(OUT_C);* or *SendVLL(**n**);* (where *n* is a value between 0 and 127). This leaves the commands that control internal timers, etc., and the event handlers with up to 16 different, customizable events. All I need for my task is the *Random(**n**);* command, which generates a random value between 0 and

Box 2: NQC Actuation Commands

```
OUT_A     // right drive
OUT_B     // left drive
OUT_V     // laser
outputs   // + combination of
          // OUT_A, OUT_B and OUT_C (if it makes sense)

On(outputs)           // on
Off(outputs)          // off
Fload(outputs)        // run out
Fwd(outputs)          // forward
Rev(outputs)          // reverse
Toggle(outputs)       // change direction
OnFwd(outputs)        // = On()+Fwd()
OnRev(outputs)        // = On()+Rev()
OnFor(outputs, time)  // 100time = 1 second

SetOutput(outputs, mode)
  // mode one of OUT_OFF, OUT_ON, OUT_FLOAT
SetDirection(outputs, direction)
  // direction one of OUT_FWD, OUT_REV, OUT_TOGGLE
SetPower(outputs, power)
  // power from 0 to 7 sets speed
x=OutputStatus(n)
  // returns current status
SetGlobalOutput(outputs, mode)
  // mode from OUT_ON, OUT_OFF   Toggles the availability of
  // control instructions to the ports on or off.
SetGlobalDirection(outputs, direction)
  // direction from OUT_FWD, OUT_REV, OUT_TOGGLE
  // Inverts all commands or reverts to norm, or changes
  // Important command for the Gigamesh G60
SetMaxPower(outputs, max_power)
  // max_power from OUT_LOW, OUT_HALF, OUT_FULL
  // Sets tops speed for the drives
x=GlobalOutputStatus(n)
  // returns general status
```

GLOSSARY

call by value: *In this case, only the value of a variable parameter is passed to the function. If the value of the parameter changes within the function, the original value is restored after leaving the function.*

call by reference: *Changes to a variable parameter within a function are retained after quitting the function. The & symbol in the function header defines a call of this type.*

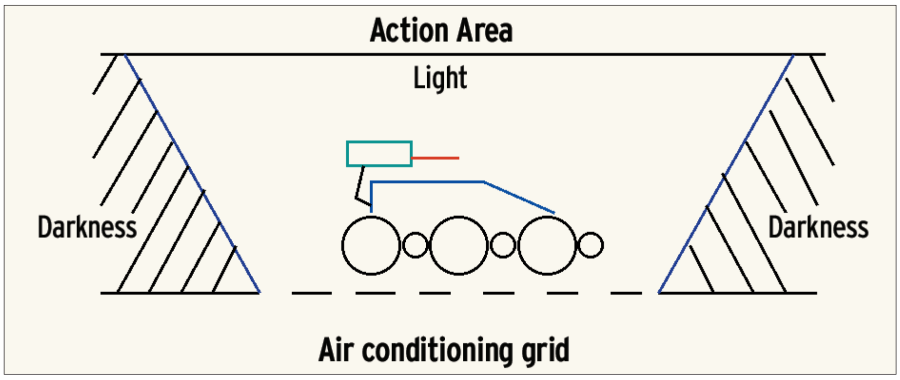

Figure 6: The Grid

n. The results of my research can be found at the website (or should I say secret drop site?) [2] with other additional information.

Time to Get Serious

Now I had everything I needed, to start programming the Spybot, I set off to do a spot of reconnaissance, hiding in front of the target building. I couldn't seem to find an entry point into the building, but then I had an idea – the air-conditioning vents. On the outside they were secured only with a light grating, and would provide a useful access point for the small robot.

Disguised as a pizza delivery boy, I took a closer look at the structure of the building, paying close attention to the air-conditioning. Air-conditioning ducts connected the various levels of the building with large grids allowing the required airflow.

My plan was beginning to gel: I would plant the Spybot in the building at night using one of the poorly protected grids for access and just let it run from there. The light would increase each time the Spybot passed over a grid. After passing the grid, the light level would drop again – the light sensor should be able to pick that up. I could then tell the robot to back up to the grid and start making some noise to set off the alarms before getting the hell out of there.

There would be a ten percent drop in luminosity when the robot moved from a grid back to a covered part of the duct; querying the light sensor should allow me to pick this up. The lightcheck() task in the NCQ program, *helsinki.nqc* (to be found at *www.linux-magazine.com/issue/35* and on the subs CD), would do that job. I also wanted the Spybot to browse the aisles at random, randomly opting for a direction (that is using the Random() function to turn left or right) when it encountered a t-junction or a dead-end. The obstacle() task took care of that.

The important thing at this point was to avoid the *lightcheck()* and *obstacle()* functions obstructing each other. That would be fatal to the mission, if the Spybot kept running into a dead-end simply because the ambient light had unexpectedly changed. I decided to use a semaphore technique for both tasks: before performing an action, a task sets a global variable, *sem*, to prevent any other tasks from occurring. Tasks will only perform specific actions, if this variable is 0. While this is happening, the variable is temporarily set to 1 to prevent other tasks from occurring.

The *helsinki.nqc* program also contains a sub-routine called *wait()* that makes the Spybot wait on the spot for the first half hour – enough time for me to make a getaway. After a few test runs in my lab, I was able to customize the *wait()* times, to allow the Spybot to perform 90 degree turns on the spot.

I sent a message to the Penguin, telling him I was ready. On the night, I opened up the air-conditioning grid just a crack, sent the Spybot off, and got out of there. Half an hour later, I could hear the first alarm bells going off in the target building. ■

Box 3: Task Commands

```
start taskname; // launches a task
stop taskname;  // stops a task
StopAllTasks(); // stops all tasks
```

INFO

[1] *wine* homepage: *http://www.winehq.com*
[2] NQC: *http://www.baumfamily.org/nqc/*

Box 4: Sensor commands

```
sensor   // SENSOR_1 or SENSOR_2
n        // 0,1 for SENSOR_1 or SENSOR_2
mode     // SENSOR_MODE_BOOL, SENSOR_MODE_PERCENT, SENSOR_MODE_RAW

SetSensorMode(sensor,mode) // Sets the mode for a sensor
ClearSensor(sensor)        // Only used for resetting sensors that
// measure impulses or revs
SensorValue(n)             // x=SensorValue(0) corresponds to x=SENSOR_1
SensorMode(n)              // returns the mode
SensorValueRaw(n)          // returns the internal sensor value
```

Box 5: NQC Sound

```
PlaySound(sound)           // sound can be
  // SOUND_CLICK, SOUND_DOUBLE_BEEP, SOUND_DOWN
  // SOUND_UP, SOUND_LOW_BEEP, SOUND_FAST_UP
 (frequency, duration) // PlayTone(440,50) will play
  // a standard pitch A for half a second
MuteSound()
  // Stops all sound and toggles to mute mode
UnmuteSound()
  // Reinitializes sound output and toggles sound output back on
ClearSound()
  // removes sound from memory
```