GAIPS/INESC-ID

# MyPleo Manual

(draft version 0.13) - August 4, 2011

Paulo F. Gomes

## Contents

## Introduction

This document is a description of the artificial pet prototype used in the *Pleo* research scenario of the European project *LIREC* (LIving with Robots and IntEractive Companions). We will refer to the artificial pet as "prototype" or as "MyPleo". *MyPleo* has two embodiments: a robotic one, consisting of a modified *Pleo*[1] robot; and a virtual one, consisting of an *Android*[2] application for a Smartphone. These two embodiments will be referred to as *PhyPleo* and *ViPleo* respectively. *MyPleo* can switch between embodiments (*PhyPleo and Vipleo*) only being active at one at a time.

The manual was not tested with shiva PLE, but that version should allow going through the code.

[1] Copyright Innvo Labs Corporation.
[2] Copyright Google Inc.

## System Installation and Configuration

- There are various references to the software folder along the configuration description. Due to licence and copyright limitations certain files will be absent in the public repository.
- 
- Most of the steps were tested in a PC with Windows XP (32-bit) service pack 3 installed, without an SD Card reader nor Bluetooth. The Bluetooth configuration and *PhyPleo deployment* were tested in a PC with Windows 7 (64-bit) service pack 1 installed, with an SD Card reader and Bluetooth. Note that USB SD Card readers are inexpensive;
- For software that does not need explicit installation, but rather just unzipping to a folder, I suggest doing so to a safe location without *space* characters in the path.
- <>
- Program versions

### Pre-steps

- Copy the whole legacy package to your computer;
- Copy the *workspace* folder to a safe location without spaces. From here only the copied folder should be used;

### Shiva Editor Installation

1. Install *ShiVa_1.9.0.1.ADVANCED* from the setup file in the *software* folder;
2. Launch the *ShiVa Editor*;
3. When asked to activate product copy the computer-id presented;
4. Go to web site http://www.stonetrip.com/;
5. Select *Log In* (upper right corner);
6. Type the username and password found in *software/Shiva account.txt*;
7. Select *Licenses management*;
8. Select *migrate a license*;
9. Paste the copied computer-id and press *migrate*;
10. Copy the generated key to the activation product form;
11. Activate product;

### Shiva Editor Configuration

1. Launch the *ShiVa Editor*;
2. Go to *Main>Projects*;
3. Select Add, browse to the *workspace\viPleoShivaModule* folder, and select it;
4. Go to *Data Explorer>Import>Archive*;
5. Browse to *vipleo* and select *MyPleo*;
6. Select *Import*;
7. When prompted override existing files;
8. Exit the *Shiva Editor*;

9. Browse to *vipleo* folder and copy the *files* folder to *workspace/ViPleo Shiva Module*;

## Java, Eclipse & Android SDK Installation

1. Install *jdk-6u26-windows-i586* from the setup file in the *software* folder (installation folder will be named *<jdk-installation-folder>*).
2. Install *eclipse-java-indigo-win32* from the zip file in the *software* folder to a safe folder (will be named *<eclipse-installation-folder>*).
3. I suggest using a shortcut to launch eclipse so that you can specify the *jdk* used. An example is included in the software folder (*eclipse-java-indigo-win32 launcher*). You will have to define the *Target* as *<eclipse-installation-folder>\eclipse.exe –vm "<jdk-installation-folder>\bin\javaw"* and the *Start in* as *<eclipse-installation-folder>*.
4. Install *installer_r12-windows* from the setup file in the *software* folder ;
5. If already not installed, install the ADT eclipse plug-in by following the instructions in *doc\eclipse-adt installation.html*[3]. Make sure to install the *SDK Platform Android 2.2, API 8, revision 2*.

## Shiva Authoring Tool Installation and Configuration

- Installation instructions in *doc\Shiva 3D Authoring Tool installation.htm*[4]. Note that some of the installation steps have already been performed, and that the critical elements are in *Required third party tools for Android target*.
- I suggest also installing *Required third party tools for Windows target*.
- A zipped installation folder for *ndk* was placed in *software* folder for completeness.

## Eclipse Configuration

1. Launch *eclipse*;
2. Select *Project>Build Automatically*, if not already selected;
3. Go to *File>Switch Workspace>Other...*;
4. Browse to *workspace* folder and select it*;*
5. Go to *Window>Preferences*;
6. Select *Android*;
7. In the *Android Preferences>SDK Location:* browse to the folder in which the Android SDK was installed and select it;
8. Click *Apply* and then *OK*;

## PhyPleo Installation

1. Install *PleoDevelopmentKit* from the zip file in the *software* folder to a safe folder (will be named *<pleo-sdk-installation-folder>*);
2. Create a new environment variable named PLEO_HOME and set it to *<pleo-sdk-installation-folder>* (e.g. *C:\ProgramsWI\PleoDevelopmentKit*);
3. Browse to *workspace\phypleo\needs_behavior*;
4. Edit *needs_behavior.upf* replacing the entries of *<pleo-sdk-installation-folder>* by the actual folder path. You might need to replace backlashes (\) by solidus (/);
5.

[3] Alternatively, you can use http://developer.android.com/sdk/eclipse-adt.html.
[4] Alternatively, you can use http://www.stonetrip.com/developer/doc/authoringtool/installation.

## Bluetooth Configuration

- Turn on PhyPleo;
- Go to *Bluetooth Devices* in the Tray Bar and select *Add Device*;
- Wait for the devices to be detected;
- Select PleoBluetooth-1255 and click *Next*;
- Select Pair without using a code;
- After the device is linked, click *Close*;
- Go to *Bluetooth Devices* in the Tray Bar and select *Show Bluetooth Devices*;
- Right-click on PleoBluetooth-1255 and select *properties*;
- Go to the *Hardware* tab and write down the name of the created serial port (e.g. COM6);
- Close the *properties* and exit from *Bluetooth devices*;
- Launch *putty.exe*;
- Go to Connection>Serial and fill the fields with the following values:
  - Serial line to connect to: the name of the port you written down;
  - Speed (baud): 115200
  - Data bits: 8
  - Stop bits: 1
  - Parity: None
  - Flow Control: None
- Go back to the Session menu;
- In the Connection type select Serial;
- In the Saved Sessions space type 'PleoBluetooth-1255 settings';
- Click Save;
- Double-Click on 'PleoBluetooth-1255 settings';
- Allow the Bluetooth device to connect by clicking on the balloon popping out of the system tray;
- Type '1234' as the key and continue;
- Wait for command black window to appear[5];

## HTC Configuration

Most of the HTC devices in INESC-ID will probably already have the following configurations.

- Go to *Settings>Applications>Development* and select *USB debugging*;
- Go to *Settings>Applications* and select *Unknown sources*;

This next configuration pairs an HTC with a PhyPleo:

- Turn on ViPleo;
- In the HTC go to *Settings>Wireless & networks>Bluetooth settings* select *Scan for devices*;
- Select "PleoBluetooth-7231" or "PleoBluetooth-1255", depending on which PhyPleo you wish to connect with;

[5] To test the connection, you can type 'help' and enter. A list of the monitor commands should appear.

- When requested to enter a code, use “1234”;

## Pleo robot Configuration

I recommend installing the *MySkit* application available in the *software* folder in order to visualize and edit the robot animations.

In order for the PhyPleo deployment to work, one should be sure that the correct firmware version is installed in the Pleo robot. The process described here has already been performed for two Pleo robots with installed Bluetooth, and thus is unnecessary for those. It might come in hand if another Pleo robot is to be Bluetooth enhanced.

Beforehand make sure to have a fully charged and cooled down battery ready (the battery will typically be a bit warn just after full charge) and that the Pleo robot is turned off. Follow these steps:

1. Put the fully charged battery into the Pleo robot;
2. Copy the contents of *phypleo\firmware\1.0.2_Downgrade* folder to an empty SD card (just the contents, not the folder itself);
3. Insert the SD card into the Pleo robot;
4. Turn Pleo on. It will probably take a while to install the behaviour. It will emit a series of twinkling sounds until it emits a final, longer, twinkling sound. The Pleo robot will start to move;
5. Turn the Pleo robot off;
6. Remove the SD Card from the Pleo robot;
7. Erase the SD Card’s contents;
8. Copy the contents of *phypleo\firmware\1.1.0_Upgrade* folder to the SD card;
9. Insert the SD card into the Pleo robot;
10. Turn the Pleo robot on. The process is similar to the previous installation. It will make some twinkling sounds and eventually the robot will start to move;
11. Turn the Pleo robot off;
12. Remove the SD Card from Pleo;
13. Erase the SD Cards’ contents;
14. Copy the contents of *phypleo\firmware\pleo-1.1.1* folder to the SD card;
15. Insert the SD card into the Pleo robot;
16. Turn Pleo on. In this case it will just start up the behaviour;
17. Turn the Pleo robot off;
18. Remove the SD Card from the Pleo robot;

## Eclipse Project Creation (Basic)

The eclipse project used was created through the Shiva Authoring Tool. Bellow, the instructions to create such a project are presented. These instructions might come in handy if you need to update your *eclipse* or *Shiva version*. In a regular installation of the development environment, there is no need to follow these instructions, as the eclipse project is already available.

9. Launch the ShiVa Editor;
10. In *General>Game Editor* go select *Open or Drop a Game*;

11. *MyPleo* should appear selected. Click *OK*;
12. In *General>Data Explorer* go to the *Games* sub-folder and select *MyPleo*.
13. Still in the *Data Explorer*, go to *Export>Export Game*;
14. If not already defined, set *Export name* to *MyPleo*;
15. Leave Export version empty;
16. Select Local folder and browse to a *workspace/viPleoShivaOutput*.
17. Select Runtime Package (.stk) and select *Android* profile;
18. Click *Export*;
19. Launch *Shiva Authoring Tool*;
20. Select the Android tab;
21. Select the "Or get started now ..." option;
22. In *Step 1 : Content*, for the option *Application pack* browse to *workspace/viPleoShivaOutput* and select the *Myleo* file*;*
23. For *Icon* browse to the folder *vipleo* and select the *pleoIcon.png*;
24. For *Startup splashscreen* browse to the folder *vipleo* and select the *pleoSplash.png*;
25. Select *Next*;
26. In *Step 2 : Authoring*, in the *Authoring type* option select Project;
27. In the *Signing>Bundle identifier* type *eu.lirec.pleo*;
28. Go to *Step 3 : Build*;
29. In the *Build type* option select *Development*;
30. In the *Minimum OS support* select *Android 2.2 (API level:8 )*;
31. In the option *Output folder* browse to the *workspace/viPleoShivaOutput* and select it;
32. Click *Build*;
33. Exit *Authoring Tool* and save the profile as *MyPleoProject*;
34. Launch *eclipse*;
35. Select *Project>Build Automatically*, if not already selected;
36. Go to *File>Switch Workspace>Other...*;
37. Browse to *workspace* folder and select it*;*
38. Select *File>New>Project*;
39. Select *General>Project* and click *Next*;
40. In *project name* type *Pleo*;
41. Click *Finish*;
42. Right click on *Pleo* in the *Package Explorer* and select *Import…*;
43. Select *General>Archive File* and click *Next*;
44. In *From archive file* browse to *workspace/viPleoShivaOutput* and double click *MyPleo_Android.zip*;
45. Click *Finish*;
46. Go to *Window>Preferences*;
47. Select *Android*;
48. In the *Android Preferences>SDK Location:* browse to the folder in which the Android SDK was installed and select it;
49. Click *Apply* and then *OK*;
50. Go to *Windows>Show View> Ant*;
51. Drag and drop the *build.xml* file from the *Package Explorer* to the *Ant window*;
52. Double click on *Build debug apk* (wait for success log on Console);

53. Right click on *Pleo* in the *Package Explorer* and select *Android Tools>Fix Project properties*;
54. Double click on *Build release apk;*

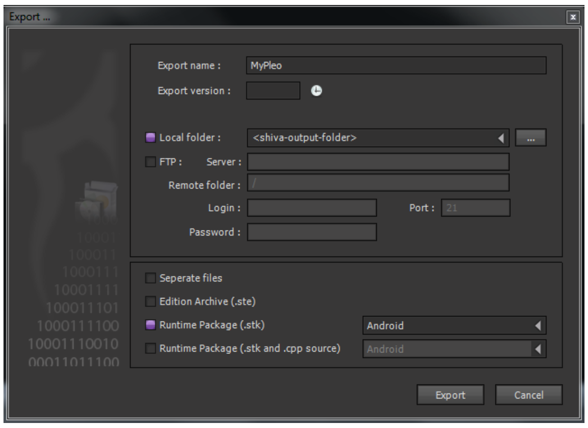
The eclipse project available in *workspace/Pleo* has the following modifications:

- Several classes added;
- Changes in the AndroidManifest.xml, namely The activity corresponding to the ShiVa module (*MyPleo*) no longer was the main one or launched at start up, *PleoMainActivity* would;
- As previously mentioned, the *S3SSurfaceView* was modified so that when the ShiVa module pauses, there is an attempt to migrate to *PhyPleo*;
- Warning suppression of minor issues;

## Deployment

### Shiva + Android Deployment

- Launch the ShiVa Editor;
- Refuse to update de editor;
- In *General>Game Editor* go select *Open or Drop a Game*;
- *MyPleo* should appear selected. Click *OK*;
- In *General>Data Explorer* go to the *Games* sub-folder and select *MyPleo*.
- Still in the *Data Explorer*, go to *Export>Export Game*;
- If not already defined, set Export name to *MyPleo*;
- Leave Export version empty;
- Select Local folder and browse to a folder. For here forth this folder will be named as *<shiva-output-folder>*. Although this folder is only used for temporary storage, I suggest creating an empty one just for this purpose, as it can be reused quite often;
- Select Runtime Package (.stk) and select Android profile;
- The Export window should now like this (with *<shiva-output-folder>* replaced with the chosen directory):



Export ...

Export name : MyPleo
Export version :

Local folder : <shiva-output-folder>
FTP : Server :
Remote folder : /
Login : Port : 21
Password :

Seperate files
Edition Archive (.ste)
Runtime Package (.stk) Android
Runtime Package (.stk and .cpp source) Android

- Click Export;
- Copy *pleoIcon.png* and *pleoSplash.png* from the *shiva authoring tool* folder to *<shiva-output-folder>*;
- Launch the ShiVa Authoring Tool;
- Refuse to update the authoring tool;
- In the upper-right part of the opened window, next to the '+' sign, select *Pleo Profile* from the drop-down menu;
- In *Step 1 : Content*, for the options *Application pack*, *Icon* and *Startup splashscreen*, browse to the folder *<shiva-output-folder>* and select the appropriate file;
- Go to *Step 3 : Build* and in the option *Output folder* browse to the *<shiva-output-folder>* and select it;
- Click *Build*;
- Go to the folder *<shiva-output-folder>*;
- Open *MyPleo.zip* (no need to unzip it);
- Browse to assets;
- Copy the *S3DMain.stk*;

- Go to *eclipse workspace>Pleo>assets* and paste it there, replacing the previous file version[6];
- Launch Eclipse;
- If not yet selected, go to *Project* and select *Build Automatically*;
- Change a java file form the Pleo project, change back to its original state, and save[7];
- Go to the *Run* menu and select *run*. The apk file will be created, uploaded to the mobile phone, and it will start the application;

## PhyPleo Deployment

- Connect an empty SD card, or containing previous behavior version, in your computer;
- Edit *phypleo\my_behaviors\needs_behavior\compileAndStore.bat* with a text editor and replace 'F' with the letter assigned to the mounted SD Card drive;
- Run the bat file.
- In order to read the compilation process information, and eventually detect a failure, you can launch a command line in the *phypleo\my_behaviors\needs_behavior\* folder and launch the *bat* file from there. To do this, you can edit the properties of the shortcut *prompt* and change the *Start In* property to the folder's full path.
- Compile alternatives and folder renaming.

6

7

## System Description

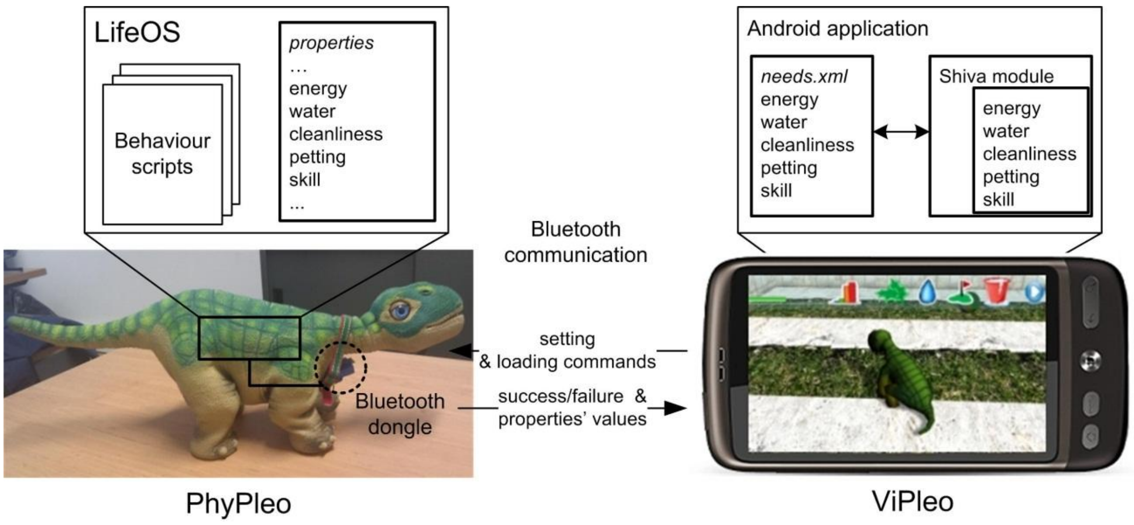


*Figure 1: Companion Architecture*

The artificial pet with its needs is schematically represented in Figure 1. The *PhyPleo*'s behavior was defined in Pawn script using the robot's operating system development kit (LifeOS[8] SDK). It was not possible to extend the robot's original behaviour with additional functionality. Need values are stored in a property section of memory. These properties are accessible by all scripts running in LifeOS as well as from the exterior via the monitor interface. A bluetooth dongle was connected to the robot's serial port (UART) thus enabling wireless communication with the monitor.

ViPleo, on the other hand, is an Android[9] application written in Java that has a module using the Shiva3d graphical engine[10]. This module is the interactive part of the application and is internally scripted in Lua. The android application is responsible for communicating with PhyPleo, loading the needs values from an xml file in which they are stored persistently, and starting up the Shiva module.

## Automatic migration

Pleo can automatically migrate from one embodiment to another after a defined time interval. Automatic migration was used in the *first evaluation*. Due to the context of this evaluation, after automatic migration has been set, all of the *Android Application*'s buttons are hidden, which. Therefore most of the control over migration is lost. This functionality is currently not accessible through the interface (buttons have been hidden). Furthermore, as the prototype has since evolved to be easier to demonstrate outside an evaluation, little effort has been put in maintaining this automatic migration functional.

Automatic migration is defined in two different places, depending on its direction:

- *ViPleo* to *PhyPleo*: automatic migration is configured through the *configuration.xml*. There one can activate/deactivate it, and also set the time ViPleo should remain active

[8] Copyright 2010 Innvo Labs Corporation.
[9] Copyright Google Inc.
[10] Copyright 2010 Stonetrip.

before migration occurs. Note that some of the functionality currently not showing on the interface will override the activation/deactivation value.

- *PhyPleo* to *ViPleo*: the time interval value is hardcoded, and activation is done by launching a thread.

## ShiVa Module

Before reading the description of the ShiVa Module, I suggest going through some tutorials such as:

- http://www.stonetrip.com/developer/162-first-application
- http://www.youtube.com/watch?v=UeiSlPdUSJ4&feature=player_embedded
- http://www.stonetrip.com/developer/609-shiva3d-quick-start-guide

More resources can be found at http://www.stonetrip.com/developer/. I recommend bookmarking the script reference: http://www.stonetrip.com/developer/doc/api/introduction. Note that many of the variable names do not follow the rules suggested in the documentation.

ShiVa games can communicate to the outside environment via html requests and by xml access. In this prototype xml access is used to load need values at start up, store need values when exiting[11], and load the configurations for the automatic migration. Note that the xml files in *shiva project*\*files*\ are only used when the project is being debugged. The files that will actually be used when the prototype is running are the ones in *eclipse workspace*\*Pleo*\*assets*\\.

The Shiva module has 6 critical elements:

- MainAI (AI Model);
- MyPleoAI (AI Model);
- MyPleoAI_Game (AI Model);
- Menu_Accoes (HUD Template);
- Camera (AI Model): responsible for aligning the camera with Pleo;
- HUD_Game_Menu (HUD Template): is only shown when the obstacle course is being taken and generates the events necessary for this mini-game;

### MainAI

*MainAI* is responsible for:

- Dealing with all events sent from the interfaces. It redirects most of the events to the MyPleoAI;
- Storing need values to *files*\*needs.xml*;
- Loading the automatic migration configuration from *files*\*configuration.xml*;
- Keeping track of a timer for the automatic migration: At each frame, if the automatic migration is active, *MainAI* verifies if it is time to migrate;
- Updating the needs and attachment progress bars when they are updated by MyPleoAI;

[11] During gameplay need values are typically stored in environment variables to facilitate loading and storing values from/to the xml.

When the ShiVa module starts, *MainAI* performs 3 main actions:

- If the Operative System is Android, rotates the view to match the device's screen orientation;
- The main scene and interface are loaded;
- The timer for the automatic migration is started, or not, according to the *files\configuration.xml*. The file is first loaded to a local xml variable, and then read.

When the ShiVa module is about to exit, needs are stored to the *files\needs.xml* according to the following steps:

1. A temporary empty xml is created in the local variables;
2. Need values are loaded from the environment variables to the temporary xml;
3. The temporary xml is written to *files\needs.xml*;

## MyPleoAI

*MyPleoAI* is responsible for defining Pleo's general behavior on the playground and for updating its needs.

The configuration of how actions affect needs, and of how needs decay, can be done through the *init* values of its variables. The variable naming style is the following:

- *nNeed<need-name>Weight*: weight of need *<need-name>* on the overall attachment value. It must be a value between 0 and 1, and the sum of all weights must be 1.
- *nNeed<need-name>Minimum*: if the need *<need-name>* goes bellow this value, the need is activated;
- *nNeed<need-name>Initial*: default initial value for need *<need-name>*. It is only used if for some reason the program was unable to load the needs xml;
- *nNeed<need-name>Decay*: decrease of need *<need-name>* each time *<need-name>* needs are updated;
- *nNeed<need-name>Increase<action-name>*: increase of need *<need-name>* when action *<action-name>* is performed;
- *nNeed<need-name>Decrease<action-name>*: decrease of need *<need-name>* when action *<action-name>* is performed;

I suggest following this naming style if additional values need to be added. The defined values are the same, or equivalent, to those defined for PhyPleo. Need values should be accessed as much as possible through the getter and setter functions. Need values are loaded from xml, and if an error occurs during loading, the *nNeed<need-name>Initial* values are loaded instead.

*MyPleoAI* updates the attachment and needs values. The time interval between updates is defined by *nNeedsTimerInterval init* value. If changed, this value should match the update frequency used in PhyPleo. The update consists of the following set of steps:

1. Decreases need values according to respective decays (*nNeed<need-name>Decay*);
2. Updates the attachment estimate value calculating a weighted sum of all needs;
3. Notifies MainAI that needs were updated;

4. Verifies if any need is active (its value is below the corresponding *minimum* value). Activates behavior accordingly. Currently this verification is only done for the *energy* need;u
5. Reset the a timer so that update function will be called again;

Pleo can be in a finite number of states (e.g. Eating – eating a patch of leaves). The current state is defined by direct interface interactions and by needs reaching critical values (currently only energy need). Instead of describing the states explicitly, its overall behavior in the playground is described according to performed user actions:

- Placing a patch of leaves in the virtual playground will cause Pleo to move towards the patch and then eat it. After eating it (Figure 2.a), its energy value is increased by *nNeedEnergyIncreaseFood*. Additionally, it will poop after a while (Figure 2.b), which will decrease its cleanliness value by *nNeedCleanlinessDecreasePoop*;
- Placing a water bowl in the virtual playground will cause Pleo to move towards the bowl and drink it (Figure 2.d), which will increase its water value by *nNeedWaterIncreaseBowl*;
- Touching the screen in the area in which Pleo is shown causes it to raise its neck (Figure 2.c). Additionally the petting value is increased by *nNeedPettingIncreasePet*.
- Selecting the washing option causes Pleo to also raise its neck, removes any poop that might be in the playground, and increases the cleanliness value by *nNeedCleanlinessIncreaseCleanPoop*;
- If its energy value is lower than *nNeedEnergyMinimum*, it sits down and cries (Figure 2.f);
- Pleo walks around the playground when the user does not interact with it;

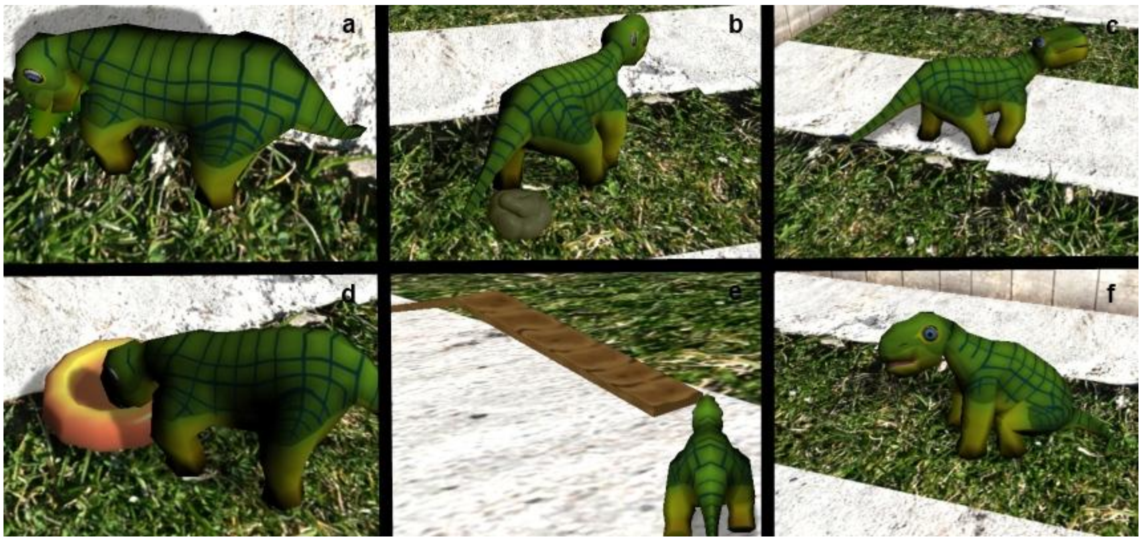


*Figure 2: ViPleo's Behaviours - eating (a), pooping (b), being petted (c), drinking water (d), going through an  obstacle course (e), sitting and crying (f).*

### MyPleoAI Game

*MyPleoAI Game* is responsible for controlling Pleo's behaviour in the obstacle course. The objective of this mini-game is to take Pleo through the obstacle course (Figure 2.e). The obstacle course has a sequence of obstacles. To cross an obstacle, the player must hit button when it is red (*HUD Game Menu*). The button oscillates (according to a timer defined in *HUD*

*Game Menu*) between 5 different positions, but only becomes red when it is at the centre. Pleo's skill value is increased each time an obstacle course is completed[12].

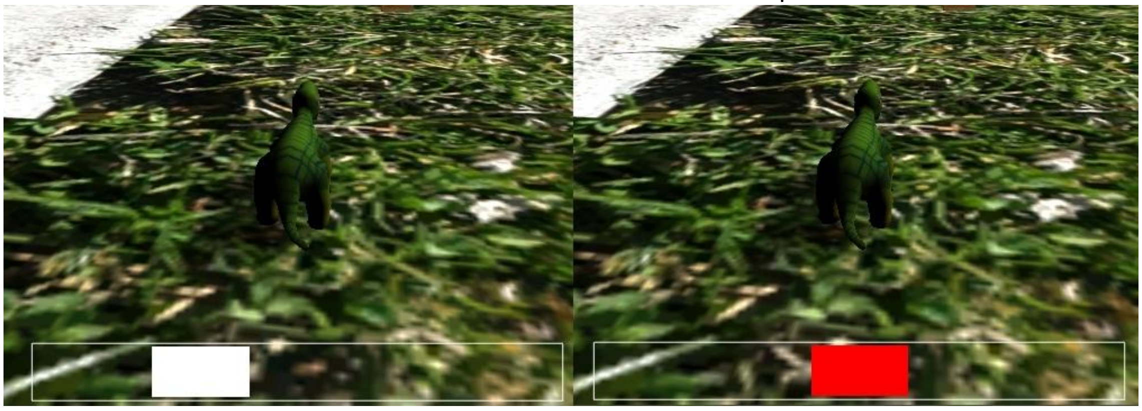


Figure 3: Obstacle course interface. The button oscilattes between inactive positions (see left) and the active position at the centre (see right).

### Menu Accoes

*Menu Accoes* has 3 main elements:

- User actions and need stats (Figure 4);
- An invisible button at the screen centre used to detect petting;
- A *Play There* button used to manually migrate;

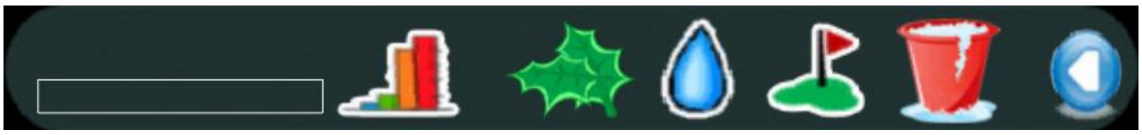


Figure 4: User actions and need stats.

There many buttons in the *Menu Accoes* that are not being shown. Some of which have equal or similar functionality to the buttons being shown. The current set of available actions was defined for the *first evaluation* with the *Pleo prototype*.

## Android Application

Before reading the description of the Android Application, I suggest going through some links such as:

- http://developer.android.com/resources/tutorials/hello-world.html
- http://developer.android.com/guide/topics/fundamentals/activities.html
- http://developer.android.com/guide/topics/wireless/bluetooth.html
- http://developer.android.com/guide/topics/fundamentals/services.html

More resources can be found at http://developer.android.com/guide/index.html.

*Android Application* is responsible for:

- The communication between the cell-phone and *PhyPleo*;
- Updating the *needs.xml* according to values received from *PhyPleo*;

[12] If the user migrates Pleo before pressing the *Sair* button the skill value is not increased.

- Editing the *configuration.xml* when automatic migration from *ViPleo* to *PhyPleo* is needed;
- Handling the interface;

In order for the Android Application to work correctly, the device in which it is deployed must not be mounted as a disk drive and must have a SD Card. *Android Application* has the following classes:

- *PleoMainActivity*: Android activity responsible for launching the connection service with PhyPleo and displaying the migration options available.
- *PleoConnectionService*: Android service responsible for the communication with the PhyPleo. When it receives a request, and is able to connect to PhyPleo, it launches a thread responsible for either setting properties or loading them.
- *PleoMonitorRunnable*: Runnable used to set and load properties from the robot's monitor interface. It loads to, and from, the need's xml. It verifies if commands have been correctly executed, and if not, tries to overcome the detected problem.
- *MyPleo*: Android activity responsible for launching the ShiVa module and for showing the splash screen. The splash screen is show for *SPLASH_DURATION_MILI_SECONDS* milliseconds (hardcoded value):
- *MyPleoNeeds and MyPleoNeedsVeriable*: containers for needs.
- *MyPleoNeedsXMLUpdater*, *MyPleoNeedsParserHandler*, *MyPleoNeedsParserException*, *ConfigurationXMLUpdater*, *ConfigurationXMLParserHandler* and *ConfigurationXMLParserException*: are used for different types of xml parsing/editing.
- *S3DEngine* and *S3DSurfaceView*: were initially generated by the ShiVa Authoring tool and wrap the ShiVa module. The *S3SSurfaceView* was modified so that when the ShiVa module pauses, there is an attempt to migrate to *PhyPleo*.

The *Android Application* project is placed in the *eclipse workspace\Pleo*. I suggest importing the whole workspace as it contains other projects that use the *Android SDK*. In *Pleo* project, the most important non-code files that eventually be changed during development are:

- *AndroidManifest.xml*: besides being critical for the Android ADK, this file needs to be changed (the intent filters) in order for *PleoConnectionService* to deal with new types of requests.
- *configuration.xml* and *needs.xml* in *assets*: are the files that will actually be used for configuration and initial need values when the application is deployed, not the ones in *\shiva project\files* (only used for testing purposes).
- *S3DMain.stk* in *assets*: contains all the functionality of the *ShiVa module*.
- *connection.xml* in *res/layout*: layout for the migration interface.
- *strings.xml:* text for the interface buttons and some warnings. Some of these warnings were created so that a Guide would understand what they meant, but a user/tester would not.

The description of the Android Application will continue with further details concerning the four main classes. Finally, we describe how the eclipse project was initially created.

## PleoMainActivity

*PleoMainActivity* is the main activity of the Application and it is the first to be launched. At launch, it performs the following steps:

1. Creates the interface buttons;
2. Extracts the *needs.xml* and *configuration.xml* to a local folder. Note that deployment is only done through an *apk* file. In order for the *ShiVa module* to access them as well, they need to be extracted from the *apk*;
3. Setup a broadcast receiver that will deal with messages sent from the *PleoConnectionService*. These messages inform the activity of the services' progress or failure;
4. If the Bluetooth is not turned on, ask the user to do so;
5. Start the *PleoConnectionService*.

In the remaining text, an embodiment is said to be active if the companion's behavior is being displayed in that embodiment. *PleoMainActivity* can be in one of four states:

- *BOTH_INACTIVE*: it is the initial state in which both embodiments are inactive;
- *VIPLEO*: the *ViPleo* embodiment is active, or being activated (ShiVa modules is being started);
- *VIPLEOTOPHYPLEO*: *ViPleo* is inactive and *PhyPleo* is being activated (Scripts are being loaded);
- *PHYPLEO*: *PhyPleo* is active and *ViPleo* is inactive;

There is an additional fifth state that currently is not being used: *PHYPLEOTOVIPLEO*. This state could be used to represent situations in which *PhyPleo* has not yet stopped moving, but the ShiVa module has already been launched. In theory, both *ViPleo* and *PhyPleo* could be active at the same time. In practice, as the *ShiVa module* takes some time to start, it rarely, if ever, happens. This time could be reduced if the splash screen would be shown for less time, or removed entirely (see *MyPleo*).

The user can perform two main actions through the activity: activating ViPleo or activating *PhyPleo*. Activating *PhyPleo* is only enabled if the Activity has received a message from the *PleoConnectionService* saying that it was able to connect to *PhyPleo*, and if *PhyPleo* is inactive. Activating PhyPleo is permanently disabled if the Activity receives a reading error from *PleoConnectionService*. On the other hand, activating *ViPleo* is only available when *ViPleo* is inactive. *PleoMainActivity* keeps track of the embodiment's activity status through the state described above.

## PleoConnectionService

When this service tries to connect to *PhyPleo*, it can do so in two ways:

- *strong connect*: if it fails to connect because the system was unable to create a communication socket (most likely because *PhyPleo* is turned off), it waits *CONNECTION_CREATION_INTERVAL* miliseconds and tries again. It will keep on trying to connect if not successful, always waiting *CONNECTION_CREATION_INTERVAL* milliseconds between tries. As *PleoConnectionService* runs on the main thread, a *strong connect* causes the whole application to wait until the connection is established.

- *weak connect*: if it fails to connect it immediately gives up;

In both cases it will inform *PleoMainActivity* if it is successful through a broadcast. When the service is launched by *PleoMainActivity* it performs the following two steps:

1. Loads the *configuration.xml* extracting *PhyPleo*'s dongle mac address. In order to use different *PhyPleo* robots, this address must be changed in the xml. The two possible addresses, corresponding to the two different *PhyPleo*'s, are written in a comment of *configuration.xml*;
2. Performs a *weak connect* to PhyPleo. Only if the *weak connect* is successful, will *PleoMainActivity* enable the activation of *PhyPleo* (see *PleoMainActivity*). Otherwise that functionality will remain inactive, but the application will continue running (which would not happen if a *strong connect* had been used);

The Service treats the following requests:

- *Unload Pleo Behavior*: activates PhyPleo's empty behaviour and loads the needs from PhyPleo;
- *Load Pleo Behavior*: announces *MainActivity* that the companion is migrating from *ViPleo* to *PhyPleo* though a broadcast, sets PhyPleo's needs to the values in the *needs.xml*, and deactivates *PhyPleo*'s empty behaviour. Performs a *strong connect* if a new connection is needed. Currently, this request is not being used;
- *Load Pleo Behavior try*: equal to the previous request, but uses a *weak connect* if a new connection is needed;
- *Load Pleo Behavior simple*: deactivates *PhyPleo*'s empty behaviour. Used when the first embodiment chosen is PhyPleo. In this case the initial need values will be the ones defined in *PhyPleo*;

For all requests, the service first tries to connect to *PhyPleo* if not yet connected. It launches a thread per request. Each thread corresponds to an instance of a *PleoMonitorRunnable*, and accomplish its task by setting PhyPleo' properties. Currently, the numbers that identify the different properties are hardcoded. If the needs are removed or added on *PhyPleo*, these numbers might need to change, as they are automatically generated during the *PhyPleo* deployment.

### PleoMonitorRunnable

Each runnables have a set of commands (text lines) that were previously defined by the *PleoConnectionService*, and must be sent via Bluetooth to PhyPleo. Commands sent to the monitor interface (Pleo robot) have a tendency to be received with noise. This phenomenon is probably caused by the serial connection being prone to electro-static discharge, as mentioned in the monitor's documentation. Besides causing commands not to be correctly executed, the received noise has a second effect: if the monitor receives too much noise, it outputs a warning message and shuts down, effectively stopping all communication via Bluetooth. Moreover, the monitor can only be reused after the robot is turned off and on.

Although not documented, LifeOS appears to have a counter for unrecognized characters, that when reaching the value 13, triggers the monitor to shut down. Also not documented as such, the command “clear” appears to reset this counter.

Commands have three types:

- *property setting*: used to update PhyPleo’s needs and activate/deactivate PhyPleo’s empty behaviour;
- *property loading*: used to load PhyPleo’s needs to the *needs.xml*. The result of this type of command is property values being outputted as a series of lines;
- *clear*: used to reset a the noise buffer used by the monitor;

When *PleoMonitorRunnable* is run, it performs the following main steps:

1. Create an empty need container;
2. Create input and output sockets for the Bluetooth connection;
3. Try to execute commands one by one;
4. Update the needs.xml according to the need container if it was filled completely with all the necessary values (successful loading);

Commands are grouped in categories. A command is assigned to a category if it matches a category pattern. Category patterns are defined as regular expressions. Patterns are also defined for expected output from the monitor connection. These are named as *expected monitor line patterns*. Each command category has a corresponding *expected monitor line pattern*. If after a command from a category sent, the corresponding *expected monitor line pattern* is detected in the output, then the command is considered to have been successfully executed. *Expected monitor line patterns* are used to detect successful execution.

Moreover, there are patterns to detect individual property values (*property loading patterns*) when a *property loading* command is used. Currently, the numbers that identify the different needs are hardcoded in the *MyPleoNeeds* class. As mentioned before, if the needs are removed or added in *PhyPleo*, these numbers might need to change, as they are automatically generated during the *PhyPleo* deployment.

Finally, there are patterns for failure detection:

- Monitor Off Pattern : Detects if the monitor has shut down because of noise;
- Shut Down Pattern: Detects if the PhyPleo’s behaviour has been shut down due to low battery;
- Property Noise Pattern: Detects if property setting or loading appeared appeared garbled in the output. Effectively, what it does is detect output lines that have to do with properties, but do not match any of the expected monitor line patterns;

Command execution can be defined as the following algorithm:

1. Write command to the output socket;
2. Read line *l* from the input socket;
3. If *l* matches a *property loading pattern*, set the value of the corresponding need on the need container to the read value;
4. If *l* matches the command’s category *expected monitor line pattern*, return *SUCCESS*;
5. If *l* matches the *Monitor Off Pattern* or *Shut Down Pattern*, return *FAILURE*;
6. If *l* matches the *Property Noise Pattern*, wait RESEND_INTERVAL_MILLISECONDS milliseconds and jump to step 1;
7. Jump to step 2;

## Project Creation

In the description of the Android Application deployment (see *Java, Eclipse & Android SDK*), although the *ShiVa Authoring Tool* generates a whole project, only the *S3DMain.stk* is used. The generated project had the following modifications:

- Project name replaced;
- As previously mentioned, the *S3SSurfaceView* was modified so that when the ShiVa module pauses, there is an attempt to migrate to *PhyPleo*;
- The activity corresponding to the ShiVa module (*MyPleo*) no longer was the main one or launched at start up, *PleoMainActivity* would;
- Some refactoring of *MyPleo*;
- Minor fixes that prevented a number of warnings from being presented;

Getting the project generated by the *ShiVa Authoring Tool* to work was partially a trial and error process. One typically would not need to reimport the project to a workspace. Nevertheless, a sequence of steps that seemed to work is presented here:

1. Open eclipse;
2. Go to 'Project' and unselect 'Build Automatically' if selected;
3. Go to 'File' -> Import;
4. Select 'General/Existing Projects into Workspace'
5. Click 'Next'
6. Select 'Select archive file'
7. Click 'Browse...'
8. Browse and double click the zip file created by the authoring tool
9. Click 'Finish'. The project has an error: ERROR: Unable to open class file [path to project]\TEST_Fibonacci_Lua_20100901\gen\com\paulogomes\mypleo\R.java
10. Select 'Window->Show View->Package Explorer'
11. Select 'Window->Show View->Ant'
12. Select 'Window->Show View->Console'
13. Open project 'TEST_Fibonacci_Lua_20100901' in the 'Package Explorer' view
14. Drag the build.xml file from the 'Package Explorer' view to the 'Ant' view
15. Open the ant buil in the 'Ant' view
16. Double click on 'Build debug apk' (wait for success log on Console)
17. Go to 'Project -> Build Project'
18. Go to 'Project -> Clean...'
19. Select 'Clean projects selected below'
20. Select 'TEST_Fibonacci_Lua_20100901'
21. Unselect 'Start a build immediatelly' if selected
22. Click 'OK'
23. Double click on 'Build debug apk' (wait for success log on Console)

Currently, I do not believe that the *local.properties* and the *build.xml* are being used in normal deployment.

## Bluetooth Installation

This description on how to enable a Pleo robot to receive commands via Bluetooth is greatly inspired in a previously online web page housed at Robotstuf.com[13]. The last time the website was consulted (14-07-2011), it seemed to be offline. There is an offline version of the cached page in the *doc* folder (*Pleo bluetooth.htm*). Installation was performed with the help of an electric technician.

The parts and material needed to install the Bluetooth are the following:

- 1 bluetooth module (Figure 5). I recommend the Bluetooth DIP Module - Roving Networks (http://www.sparkfun.com/products/8550);

*Figure 5: Bluetooth module.*

- 1 crimp housing (Figure 6), 1.25mm, 7 way (http://il.farnell.com/molex/51021-0700/crimp-housing-1-25mm-7way/dp/615110[15]);

[13] http://robostuff.com/diy-projects/pleo-hacking/how-to-control-pleo-wirelessly-via-Bluetooth/, consulted 11-03-2011.

[14] As mentioned on the website, the image presented there is only an illustration. The crimp has actually 7 entrances.

[15] As mentioned on the website, the image presented there is only an illustration. The crimp has actually 7 entrances.

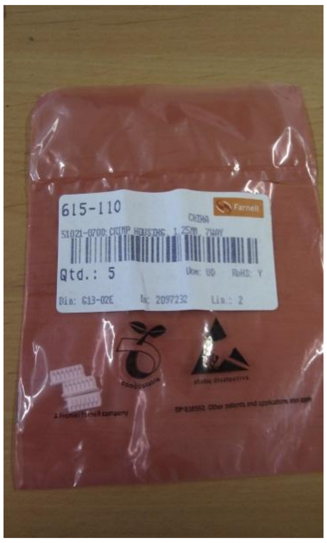


*Figure 6: Crimp housing.*

- 4 crimp socket contacts (http://uk.farnell.com/molex/50058-8100/crimp-socket-contact/dp/1704246[15]) - Figure 7;

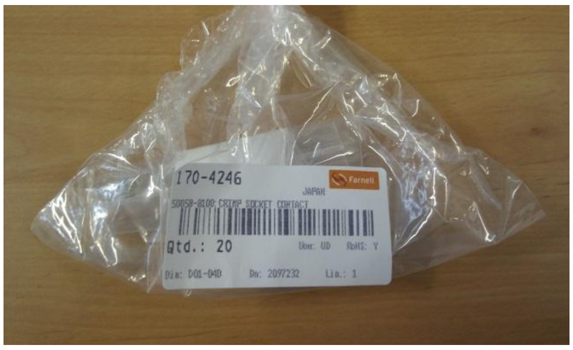


*Figure 7: Crimp socket contacts.*

- 2 two-pin fem headers (http://www.sparkfun.com/products/115);
- 4 wires of different colours and about 8cm long;
- Green modelling clay;
- Green cat collar;
- Electrical tape;

All the material, apart from the cat collar and the wires, should be available at GAIPS/INESC-ID's offices. The electronic parts should be in a box labelled "PhyPleo Parts". Additionally, you will also need a basic soldering iron, pliers, a snap-off blade, crimping tool, tweezers, and a SD Card.

Installation itself consists of the following steps:

1. Use the crimping tool, and pliers, to attach the crimp socket contacts to the wires (one per each wire);
2. Insert the crimped wires at positions 1, 3, 4 and 5 of the crimp housing (see Figure 8);

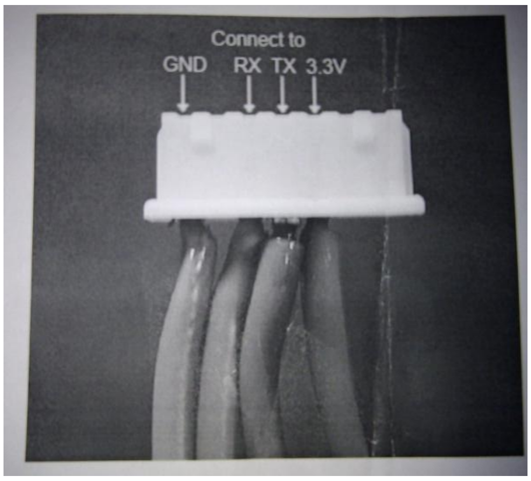


*Figure 8: Wire positions.*

3. Solder the extremities of wires connected to positions 1 and 4 to one two-pin fem header, and wires connected to positions 3 and 4 to the other fem header (see Figure 9);

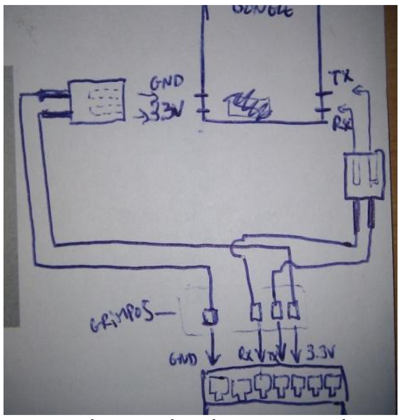


*Figure 9: Bluetooth wire connections.*

4. Connect the fem headers to the Bluetooth module;
5. Pleo's serial interface is hidden under a plastic cover next to the power switch (see Figure 10). Take out that cover with a snap-off blade;

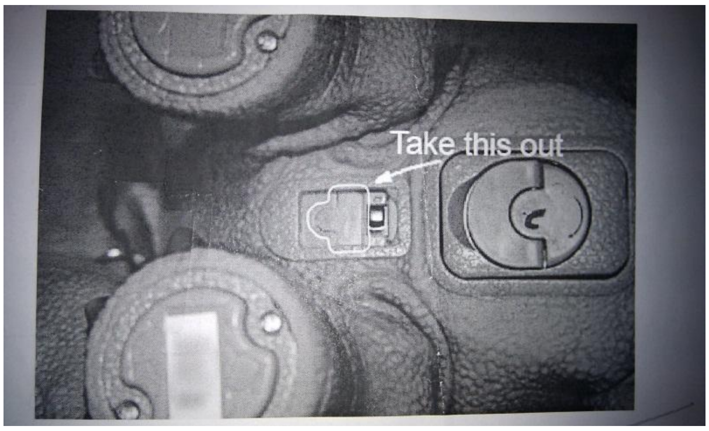


*Figure 10: Serial interface plastic cover.*

6. Connect the crimp housing to the robot's serial interface with the help of the tweezers;
7. Cover the wires and Bluetooth module with electric tape[16];
8. Surround the wires at the removed cover hole with the modelling clay;
9. Strap the collar around the robot's neck, securing the Bluetooth module;

Configuration of the Bluetooth module requires the following steps:

1. Perform an *PhyPleo Deployment*;
2. Turn on PhyPleo and immediately try to connect with it via using putty (see Bluetooth Configuration);
3. Type $$$ and *Enter*. This way you can access the module's configuration. This functionality can only be accessed until one minute has passed since the module has been turned on. For more details consult the modules documentation (*Roving Networks Bluetoot - Product User Manual* in *doc*);
4. Enter *D*. Take note of the *BTA* value.
5. Enter *SI,0012*. This command sets the inquiry scan window to 1% of the duty cycle;
6. Enter *SJ,0800*. This command sets the page scan window to the maximum;
7. Enter *SN,PleoBluetooth-XXXX* in which XXXX are the 4 last digits of the *BTA*. This step is optional, but will help identify the module more easily;
8. Enter *SC,1101*. This command sets the service class to 1101;
9. Enter *R,1*. Reboots the module;

I recommend taking a look at the modules documentation (*Roving Networks Bluetoot - Product User Manual* in *doc*) specifically at the *Configuration* and *Command Reference* sections. Further details on the inner workings of the module can be found in *doc/Class 1 Bluetooth Module.pdf*.

## PhyPleo

Before proceeding, I strongly recommed reading *\phypleo\documentation\Pleo Programmers Guide.pdf*. The described behaviour is defined in the folder

[16] I am unsure if that might eventually affect the bluetooth communication, but from the testing done, it did not seem so.

*phypleo\my_behaviors\needs_behavior*. It was an adaptation of the example behaviour defined in the folder *\phypleo\examples\drive_example*.

## Needs

Behaviour is driven by needs with initial values equal to the ones in *ViPleo* (hardcoded in *main.p*). These needs are stored as properties in a reserved portion of memory. They are updated by leak integrators, user actions and bluetooth communication. Leaky integrators decay properties over time at the same rate as in *ViPleo* (hardcoded in *main.p*).

User actions are detected through the robot's sensors. *PhyPleo* responds to the following actions: being petted, having a leaf in front of him and having a leaf in his mouth (see *sensors.p*). Being petted immediately increases the petting need value by the same value that in *ViPleo* (*nNeedPettingIncreasePet*).

## Behaviour

Needs determine behaviour exept if the *mode property* is set to 1. In this case the *empty behaviour* is selected. When selected, *PhyPleo* appears to go to sleep and the leaky integrators are deactivated. When deselected (*mode property* is set to 0), the leaky integrators are reactivated and *PhyPleo* appears to wake up (this last step is automatic as no specific call to a wake up animation is performed). The *mode property* is used to activate and deactivate PhyPleo via bluetooth. Note that this property is initially set to 1, hence PhyPleo always starts inactive.

PhyPleo has two main drives that take in account the need values: social and hunger. The social drive is active by default, but if the energy value drops bellow a crtitical value (hardcoded in *hunger.p*), the same as in ViPleo (*nNeedEnergyMinimum*), the hunger drive becomes active. Only one drive is active at a time.

When the social drive is active, pleo will perform two different behaviours depending on the petting need value: if the need value is bellow a critical value (hardcoded in *social.p*) PhyPleo will whine; if not, it will bark and wag its tail.

On the other hand, when the hunger drive is active, only one behaviour is performed: eat. The eat behaviour consists of the following steps:

1. Search for food: PhyPleo slowly walks forward with is head bent downwards, sniffing the ground. Stops when leaf is detected;
2. Bite: first PhyPleo raises its neck and opens its mouth. Afterwards it bites and a crunching sound is heard. Stops a while after detecting a leaf in its mouth;

After the behaviour has finished, the leaf increases the energy need value by the same value that in *ViPleo* (*nNeedEnergyIncreaseFood*).

## Monitor

The monitor interface, described in detail in *\phypleo\documentation\Pleo Monitor.pdf*, is used to set properties and load properties via bluetooth. Some of the most important commands for this project are:

- joint neutral : set all joints to neutral positions;

- log disable all: disable all logging;
- stats power: check current power status;
- clear: clear screen and apparently noise buffer (see *PleoMonitorRunnable*);
- motion play <motion-ID>: play motion with ID <motion-ID>;
- motion show: show all available motions;
- property show: show all properties and property values;
- property set <property-ID> <value>: set property with ID <property-ID> to <value>;

The Monitor can acessed through the serial port (used to connect the bluetooth dongle) and through the USB port. Consequently, it can be tested after *Bluetooth Configuration* has been performed. To do so, you just need to turn on PhyPleo, and connect to the bluetooth dongle using putty.

If for some reason you are unable to use the monitor via bluetooth, you might want to use the USB port. Details for doing so can be consulted in *\phypleo\documentation\Pleo Monitor.pdf*. Note however that the drivers available are for Windows XP 32-bit. If you are not using such an operative system, you might be forced to install a virtual machine with it.

## *Ackwolegments*

*This work is partially supported by the European Community (EC) and was funded by the EU FP7 ICT-215554 project LIREC (LIving with Robots and IntEractive Companions), and FCT (INESC-ID multiannual funding) through the PIDDAC Program funds. The authors are solely responsible for the content of this publication. It does not represent the opinion of the EC, and the EC is not responsible for any use that might be made of data appearing therein.*