ALE Java Agent Tutorial

Marc G. Bellemare
mgbellemare@ualberta.ca

June 21, 2015

1 Requirements

To run this agent, you will need:

- Java 1.6
- ALE
- At least one ROM file
- Apache Ant
- Perl (optional)

Although not strictly necessary, we recommend the use of Apache Ant to build the Java agent.

This tutorial assumes that you have installed ALE 0.5 in a directory named `ale_0_5`. See the main manual (`/path/to/ale_0_5/doc/manual/manual.pdf`) for ALE installation details.

2 Installation/Compilation

Installing the Java agent is simple. Assuming you have extracted the Java agent package to the directory `ale_java_agent`, you may simply use Ant to build the jar file containing the agent code:

```
> cd ale_java_agent
ale_java_agent> ant jar
```

Voilà! Your Java agent is compiled and good to go.

2.1 Perl script setup

For your convenience, we offer a perl script that automates running ALE and the Java agent. To use this perl script, you should copy or link the ALE executable, configuration file and roms directory accordingly. Assuming that your roms are located at `/path/to/atari_roms` and that you have installed ALE at `/path/to/ale_0_5`, you may then (on Mac OS X and *NIX system):

```
> cd /path/to/ale_java_agent
/path/to/ale_java_agent> ln -s /path/to/ale_0_5/ale .
/path/to/ale_java_agent> ln -s /path/to/ale_0_5/stellarc .
/path/to/ale_0_5> ln -s /path/to/atari_roms roms
```

In particular, if running the Java agent from its package path, i.e.

```
/path/to/ale/doc/java-agent
```

you may simply do

```
doc/java-agent/code> ln -s ../../../ale .
doc/java-agent/code> ln -s ../../../stellarc .
```

Your `ale_java_agent` directory is now set up.

3 Running

If you did not perform the optional perl script setup (Section 2.1), or do not have perl installed, skip to Section 3.1 for information on running ALE via named pipes. The named pipes interface is somewhat deprecated and we encourage you to use the perl script if possible. The C++ fifo interface example

```
doc/examples/fifoInterfaceExample.cpp
```

can also easily be adapted to run your Java agent together with ALE.

Assuming that everything is installed correctly, you should now be able to run the HumanAgent using the provided perl script:

```
ale_java_agent> perl run_agent.perl space_invaders
```

3.1 Named Pipes

To communicate with ALE via named pipes, you need to start ALE and the Java agent in separate consoles. First, we create two named pipes:

```
/path/to/ale_0_5> mkfifo ale_fifo_in
/path/to/ale_0_5> mkfifo ale_fifo_out
```

Then we run ALE and the Java agent in separate processes. We specify the path of both named pipes using the `-named_pipes` option:

```
Terminal 1
/path/to/ale_0_5> ./ale -game_controller fifo_named
  /path/to/atari_roms/beam_rider.bin

Terminal 2
ale_java_agent> java -cp dist/ALEJavaAgent.jar ale.agents.HumanAgent
  -named_pipes /path/to/ale_0_5/ale_fifo_
```

4 Recording screen data

We provide facilities for recording received screen data to PNG files. We do so by passing the `-export_frames` command-line argument to the Java HumanAgent class. The perl script is already set up to pass all arguments beyond the first to the Java agent:

```
ale_java_agent> perl run_agent.perl beam_rider -export_frames
```

Frames will be saved in the `ale_java_agent/frames` directory, starting with `frame_000000.png` and subsequently incrementing the index. The equivalent command with named pipes is:

```
ale_java_agent> java -cp dist/ALEJavaAgent.jar ale.agents.HumanAgent
  -named_pipes /path/to/ale_0_5/ale_fifo_ -export_frames
```

For further information, see the class `ale.movie.MovieGenerator`. In `ale.agents.HumanAgent`, the variable `exportFramesBasename` defines the pathname to which frames are saved.

5 Code listing

To complete this tutorial, we give a list of the classes provided in the Java agent package, along with a short description.

1. `ale.agents`
    - `AbstractAgent` Abstract class; interfaces with ALE and the GUI.
    - `HumanAgent` extends `AbstractAgent`. Defines an agent controlled by the user via the keyboard.
    - `RLAgent` extends `AbstractAgent`. A simple learning agent that uses SARSA and ϵ-greedy.
2. `ale.gui`

- `AbstractUI` Abstract class; defines a user interface.
- `AgentGUI` extends `AbstractUI`. Defines a graphical user interface.
- `NullUI` extends `AbstractUI`. The `/dev/null` of uesr interfaces.
- `KeyboardControl`. Receives keystrokes and converts them to ALE actions.
- `ScreenDisplay`. Responsible for displaying images on the screen.
- `MessageHistory`. A helper class for display messages on-screen.

3. `ale.io`
   - `ALEPipes`. Communicates with ALE via stdin/out or named pipes.
   - `Actions`. Helper class mapping action names to integers.
   - `ConsoleRAM`. Encapsulates RAM data.
   - `RLData`. Encapsulates RL data.
4. `ale.movie`
   - `MovieGenerator`. Helper class to save screen data to PNG files.
5. `ale.rl`
   - `FeatureMap`. Maps screen data to feature vectors.
   - `FrameHistory`. Stores a list of recent frames.
   - `LinearModel`. A linear regression predictor. Used for approximating value functions.
   - `SarsaLearner`. The core SARSA algorithm.
6. `ale.screen`
   - `ColorPalette`. Abstract class; defines basic colour palette functionality.
   - `NTSCPalette`. Defines the NTSC colour palette (128 colors).
   - `SECAMPalette`. Defines the SECAM colour palette (8 colors).
   - `ScreenConverter`. Converts ScreenMatrix objects to Java image objects.
   - `ScreenMatrix`. Encapsulates screen data.