Scalatron Server Setup

About Scalatron

Scalatron is an educational resource for groups of programmers that want to learn more about the Scala programming language or want to hone their Scala programming skills. It is based on Scalatron BotWar, a competitive multi-player programming game in which coders pit bot programs (written in Scala) against each other.

The documentation, tutorial and source code are intended as a community resource. Feel free to use, copy, and improve them!

Table of Contents

Overview

Scalatron consists of a game server application running on a central computer. This server hosts the bot tournament by continuously simulating short rounds of the Scalatron BotWar game. Players participate in the tournament by publishing their bots into the server.

To publish a bot, a player has to provide the server with a `.jar` archive file that contains the bot's control function, which was implemented in Scala and then compiled. There are two approaches to doing this, which in the Scalatron documentation are referred to as the "serious" and "casual" paths, respectively.

The "Serious" Path

The "serious" path is intended for experienced programmers planning for a longer bot coding session (5-7 hours). On this path, bots are built locally by each player, using an IDE or command line tools. The bots are then published into the tournament by copying them into the plug-in directory on the central computer from which the game server loads them at the start of each round.

Pros:

- Players can work with the editor of their choice and can use the build tools they are already accustomed to.
- Working with an IDE allows for very comfortable debugging with breakpoints and single-stepping.
- Each player ends up with a fully configured Scala build environment and is able to keep coding in Scala beyond the bot coding session.

Cons:

- The server administrator has to set up network sharing for the plug-in directory.
- Each player has to set up a local working environment (IDE, SBT or command line).
- Based on experience, setting this up can take anywhere from 30 to 60 minutes.

The "Casual" Path

The "casual" path is intended for less experienced programmers or for shorter bot coding sessions (2-3 hours). On this path, players write, build, debug and publish their bots in a web browser. The browser-based development environment is provided by an embedded web server which is hosted inside the game server and requires no setup by the user.

Pros:

- Players do not have to install or configure anything.
- The server administrator does not have to configure a network-shared directory; just unzip the distribution and double-click the application.

Cons:

- Debugging is only possible via state watches and debug output in the browser; no breakpoints can be set in the code and no variables can be inspected at run time.
- After the tournament, players leave without a working local development environment.

Setup Steps

The "Serious" Path

1. Select a computer to be the central server
2. Unzip the Scalatron distribution on that server
3. Configure a directory for sharing over the network with read/write access for all players
4. Make sure that a Java Runtime Environment is available on the server computer
5. Launch the game application
6. Have fun!

The "Casual" Path

1. Select a computer to be the central server
2. Unzip the Scalatron distribution on that server
3. Make sure that a Java Runtime Environment is available on the server computer
4. Launch the game application
5. Have fun!

The following sections provide more detailed instructions.

Installing the Scalatron Server

Installation is required for both the "serious" and "casual" paths.

The following setup steps were tested on MacOS X 10.6.

- **Optional**: provide a dedicated computer for the Scalatron game server which you will use during the live coding workshop. Change the name of that computer to something users will easily recognize on the network, such as `Scalatron`. On MacOS X, you can do this in the System Preferences app via the icon Sharing.
- **Optional**: create an account that you will use to run the Scalatron game server during the live coding workshop, for example for a user `Scalatron` with standard user rights. On MacOS X, you can do this in the System Preferences app via the icon Accounts.
- Log in as the user account that you will use for running the game server. In this example setup, that account would be `Scalatron`.
- Now copy the compressed Scalatron distribution archive of the version you want to use (e.g. `scalatron-0.9.7.zip`) into the parent directory and decompress it there. This will create a sub-directory called `scalatron-0.9.7` containing a directory `Scalatron`. Move that `Scalatron` directory below your user directory. On MacOS X, the path to this directory would then be `/Users/Scalatron/Scalatron`.

- The structure you want to end up with in either case is roughly the following:

```
/Users
      /Scalatron
         /Scalatron
            /bin
               Scalatron.jar
            /bots
               /BrownianMotion
                  ScalatronBot.jar
            /doc
            /src
```

- Now either open a console, navigate to the `bin` directory and launch the app, using e.g.

```
cd /Users/Scalatron/Scalatron/bin
java -jar Scalatron.jar
```

- Or just double-click on the `Scalatron.jar` file in the `bin` directory.
- The game server will tell you where it is looking for plug-ins (the default location is a `bots` directory that resides next to the `bin` directory from which the app was launched) and which address players can point their browsers to (the default is the host name of the server computer with port 8080, e.g. "http://scalatron:8080").

Sharing the Plug-in Directory

This step is not required for the "casual" path.

- **Create a Network Share**. Once you got the server application to launch, share the bot directory so that the players can upload their bot files. On a game server running MacOS X, you can do this as follows:
  - open the System Preferences application
  - click the icon Sharing
  - in the list Shared Folders in the middle, click + to add a new shared folder
  - this may require you to enter an administrator password
  - then browse to the `bots` folder (e.g. at `/Users/Scalatron/Scalatron/bots`) and add that folder as a network share
  - set the permissions on the `bots` folder so that everyone can Read & Write
- **Verify the Network Share**. Test that it works. On a participant computer running MacOS X, you would do this as follows:
  - open a Finder window
  - in the main menu, select Go > Network
  - under Network, you should now see an entry for the game server computer (e.g. `Scalatron`)
  - click on it
  - you should now briefly see a message `connecting...`
  - then a folder called `bots` should appear

  - click on it
  - if you created the file structure on the server by unzipping the Scalatron archive (e.g. `scalatron-0.9.1.zip`) then the folder should now already show a range of directories: the example bots, e.g. `BrownianMotion` and `FoodFetcher`. Each directory should contain a Scala source file and a `ScalatronBot.jar` plug-in.
  - now try to create a new player directory in the `bots` folder. If you were doing this for a player (i.e., programmer) called Tina, then the new directory would be called `Tina`.
  - control-click on the new directory and select Get Info. This will pop up a window where you can see the file system path on the local system for the player's remote plug-in directory. It will be something like `/Volumes/bots/Tina`.
  - now copy one of the existing `ScalatronBot.jar` files into the `Tina` directory.
  - if that works, you have the proper file access permissions
- **Adjust Permissions**. Setting the permissions on the `bots` folder on the game server so that everyone can Read & Write lets users create new directories for their plug-ins in the bots folder. But it will not allow the game server to write files into those directories while the game is running. This, however, is required so that the debug logging explained in the Scalatron Tutorial works. To make this work, you need to expand the permissions on the shared folder. On MacOS X you can do this on the game server as follows:
  - open a Finder window
  - browse to the bots folder, e.g. at `/Users/Scalatron/Scalatron/bots`
  - control-click to open a context menu, then select Get Info
  - in the window that pops up, click the lock at the bottom right to make changes; this may prompt you to enter an Administrator password
  - the permissions on the folder should already be set so that everyone can Read & Write
  - now click on the little toothed wheel at the bottom and select Apply to enclosed items...
  - this will propagate the access permissions to all the enclosed plug-in directories
  - now the server running the bots should be able to create log files in their plug-in directories

Configuring the Scalatron Server

This step is optional for both the "serious" and "casual" paths.

The Scalatron game application can be launched either from the command line or by double-clicking it. This will launch the application in its default configuration, which un most cases will be sufficient. To launch it with a different configuration, you must use the command line. You can display a list of command line options by using the single option `-help`:

```
     java -jar Scalatron.jar -help
```

The parameters available on the command line fall into three groups, as follows:

```
java java-options -jar Scalatron.jar scalatron-options botwar-options
```

Java VM Options

The options for configuring the Java Virtual Machine apply where the command line option example above contains the string `java-options`. Here you can provide the usual JVM launch options, including::

```
-server
-Xmx512M
```

Recommended options:

```
-server -Xmx512M
```

Scalatron Server Options

The options for configuring the Scalatron server framework apply where the command line option example above contains the string `scalatron-options`. These include:

• `-plugins directory`	Scalatron plugin base directory (default: `../bots`)
• `-verbose yes|no`	print verbose output (default: `no`)
• `-headless yes|no`	run without visual output (default: no)
• `-rounds int`	run this many tournament rounds, then exit (default: unlimited)
• `-webui directory`	directory containing browser UI (default: `../webui`)
• `-users directory`	directory containing browser-edited bots (default: `../users`)
• `-port int`	port to serve browser UI at (default: `8080`)

Recommended options:

- You will generally not need to modify any of these settings if you have installed the server by unzipping the Scalatron distribution, since all directories will be in the places where the server expects them.
- If for some reason you would like the server to load its bot plug-ins from a different directory (for example because you have an existing network share), you can point the server to that directory using `-plugins` option.

BotWar Game Options

The options for configuring the BotWar game run by the Scalatron server apply where the command line option example above contains the string `botwar-options`. These include:

• `-frameX int|max`	window width (pixels; `max` = full screen; default: 640)
• `-frameY int|max`	window height (pixels; `max` = full screen; default: 500)
• `-steps int`	steps per game cycle (default: `5000`)
• `-maxfps int`	maximum steps/second (to reduce CPU load; default: unlimited)
• `-x int`	game arena width (cells; default: depends on plugin count)
• `-y int`	game arena height (cells; default: depends on plugin count)
• `-perimeter option`	arena perimeter: `none`, `open`, or `closed` (default: `open`)
• `-walls int`	count of wall elements in arena (default: x*y/300)
• `-zugars int`	count of good plants in arena (default: x*y/250)
• `-toxifera int`	count of bad plants in arena (default: x*y/350)
• `-fluppets int`	count of good beasts in arena (default: x*y/350)
• `-snorgs int`	count of bad beasts in arena (default: x*y/500)

Recommended options:

- You will generally not need to modify any of these settings.
- The server determines the `-frameX` and `-frameY` parameters using the size of the screen on which the window will appear. Adjust only if that does not give adequate results.
- The server determines the `-x` and `-y` parameters based on the number of plug-ins loaded each round. The default will be a square arena whose cell count increases with the player count. Adjust only if that does not give adequate results.
- For the initial experimental phase (focus on quick turn-around) use around 1000 steps per game round: `-steps 1000`
- For the later refinement/competition phase (focus on longer evaluation) use up to 10000 steps per game round: `-steps 10000`
- You can significantly change the nature of the game by reducing or increasing the number of entities of various kinds that the simulator sprinkles into the arena. Try `-snorgs 300` or `-walls 0`.
- If your server CPU load is too high and the browser-based Scalatron IDE is sluggish to respond to user requests (e.g. builds take a long time), try throttling the CPU load of the simulation loop by specifying a maximum frame rate with the `-maxfps` option, for example using `-maxfps 20`.

Examples

```
java -server -jar Scalatron.jar -plugins /Users/Scalatron/Scalatron/bots
java -server -jar Scalatron.jar -x 200 -y 150 -frameX 640 -frameY 500
```

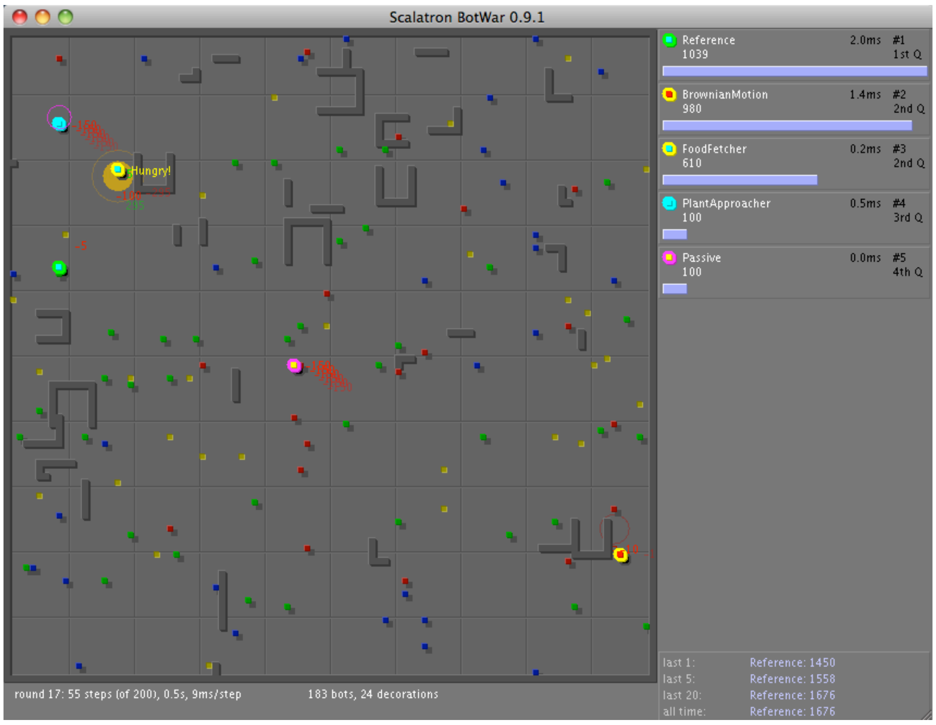
Keyboard Commands

The following keyboard commands are available by typing into the game server's display window while games are in progress:

• `space`	freeze/unfreeze the action
• `'1'`	no delay between simulation steps
• `'2'`	delay = 50ms
• `'3'`	delay = 100ms
• `'4'`	delay = 250ms
• `'5'`	delay = 500ms
• `'6'`	delay = 1000ms
• `'7'`	delay = 2000ms
• `'h'`	show/hide bot horizons
• `'r'`	abort current round, reload all plug-ins and start next round; round counter and leaderboard are not affected; useful while developing bots to prompt immediate re-scan for updated plug-ins
• `'+/-'`	step focus through players; useful for highlighting a particular player's bot

Screen Layout

When launched, the Scalatron server application displays a window that will look something like this:



```
Reference        2.0ms  #1
1039                    1st Q
BrownianMotion   1.4ms  #2
980                     2nd Q
FoodFetcher      0.2ms  #3
610                     2nd Q
PlantApproacher  0.5ms  #4
100                     3rd Q
Passive          0.0ms  #5
100                     4th Q
```

```
round 17: 55 steps (of 200), 0.5s, 9ms/step          183 bots, 24 decorations
```

```
last 1:    Reference: 1450
last 5:    Reference: 1558
last 20:   Reference: 1676
all time:  Reference: 1676
```

The display consists of the following components:

- top left / main area: the game arena. This is the playing field in which bots compete.
- bottom left area: the statistics panel. Shows operational data for the game server:
  - the index of the game round currently in progress
  - the number of steps computed so far and the total number to compute
  - the real-time duration of the game round so far, in seconds
  - the average per-frame time during the current game round, in milliseconds
  - the number of bots (players, beasts, plants and walls) in the game
  - the number of decorations (explosions, bonks, bonus breadcrumbs etc.)
- top right area: the score board. Displays, for each player and sorted by score, the following information:
  - at left, from the top:
    - the player's bot icon
    - the player's name (the name of the plug-in directory)
    - the player's score
  - at right, from the top:
    - the time spent in the plug-in's control function, in milliseconds. If the game slows down because some player is hogging CPU resources, you'll be able to see it here.
    - the rank of the player (e.g. "`#1`")
    - the quartile of the player (e.g. "`1st Q`"). The quartile is based on the player's relative rank. It affects the distance from which beasts in the game can sense the player's master bot.
- bottom right area: the leader board. Displays the player(s) that had...
  - the highest score(s) in the most recent round
  - the highest average score(s) across the most recent 5 rounds
  - the highest average score(s) across the most recent 20 rounds
  - the highest average score(s) across all rounds