Writing Bots for Maize

Stephen Wattam

November 16, 2012

Contents

1 Introduction

This is a rough guide on getting started with writing your own bot for Maize. It describes the conventions and data format used in the Bot API, as well as pointing at some of the more advanced features available.

As an overview, you will be writing a class that controls a virtual agent as it moves around a virtual maze. You will be provided only with a small amount of data about the surroundings of the bot, and can only instruct it to turn left or right, and move forward or backwards. The aim is to reach the goal (or finish).

2 World Format

The maze you are navigating is represented internally by three quantities:

1. **A boolean matrix**—This represents the wall and space structure of the maze. Each cell holds a boolean value: true for the presence of a wall, false for a space;
2. **The co-ordinates of the start**—The (x, y) co-ordinates of the start position in the above;
3. **The co-ordinates of the finish**—The (x, y) co-ordinates of the goal in the above.

The maze is rendered on-screen using conventional display co-ordinates, *not* those conventionally used for graph display. This means that the top-left corner is $(0, 0)$, and the bottom-right is (i, j).

During a test, your bot will be positioned at the start co-ordinates, and the `nextMove()` method will be called once for each time tick, unless your bot's current position is equal to the finish. All mazes are solvable (that is, the generation algorithm generally ensures it), and static.

3 Bot API

Bots are run by calling a re-entrant API on an existing instance of a class that implements the `Bot` interface. This comprises three methods, all of which must be implemented:

- `nextMove()`—Is called once per time tick, and returns a value from the `Direction` enumeration in order to indicate which way the bot should move. The full format for this is listed below, in Section 3.1;
- `getName()`—Returns a name for the bot. Used in lists and the UI to identify your creation;

- `getDescription()`—Returns a sentence or two about how your bot works, and any state information you may wish to check on.
- `start()`—Called once each time the bot is run (regardless of whether it is being run on a new maze or not). Can be used to clear any bot state or generate pre-computed data.

In addition to this simple API, there are two classes, `AdvancedBot` and `StateBot`, which add a higher level API to the above. They are documented below in Section 3.2.

3.1 Bot Data Format

Each call to `nextMove()` provides the bot with information that is similar to a real world maze-solving digital mouse. This is limited to:

- `view`—A 3×3 boolean matrix representing the immediate surroundings of the bot, formatted as (x, y). Note that this is rotated in a similar manner to the Maze, with $(0, 0)$ being the top-left corner:

$(0,0)$	$(1,0)$	$(2,0)$
$(0,1)$	bot	$(2,1)$
$(0,2)$	$(1,2)$	$(2,2)$

  Note that the view matrix is rotated so that **the bot always faces towards** $(1, 0)$. It is possible to rotate it to face north by calling `Orientation.rotateToNorth()`.
- `x`—The integer x co-ordinate of the bot's current position in the maze.
- `y`—The integer y co-ordinate of the bot's current position in the maze.
- `o`—An integer taking one of the values from the `Orientation` class, representing the bot's current orientation (which way it is facing relative to the maze). This can take one of the following values:
  - `Orientation.NORTH` = 0
  - `Orientation.EAST` = 1
  - `Orientation.SOUTH` = 2
  - `Orientation.WEST` = 3
- `fx`—The integer x co-ordinate of the finish square.
- `fy`—The integer y co-ordinate of the finish square.

The method should return an integer, between 0 and 3 (inclusive), representing which way the bot should move. The easiest way to do this is to use the `Direction` class, which has four values coded into it:

- Direction.FORWARD = 0
- Direction.BACK = 1
- Direction.LEFT = 2
- Direction.RIGHT = 3

Note that `Direction.LEFT` and `Direction.RIGHT` will instruct the bot to *turn* left or right, not strafe.

3.2 Alternative Interfaces to Bot

In addition to the `Bot` interface, two other classes exist which provide further functionality. If you wish to use these, I recommend reading their javadoc documentation, where a full API is given:

- `AbsoluteBot`—Rotates the view matrix to always face north, and understands commands in terms of compass points, rather than forward/backward. When using this bot, override `calculateMove()` and return a value from `Orientation` to move.
- `StateBot`—Supports retaining just one piece of information by calling `setState()` and `getState()`;
- `AdvancedBot`—Supports a queue of actions, held in a buffer. When the buffer is empty, `nextMove()` is called for more instructions. Contains high-level APIs that take advantage of this functionality (*e.g.* `strafeRight()`, `rotate180()`).

3.3 Further Considerations

We have attempted to minimise the technical requirements for your bot, however, some remain (mainly due to the design of Java). The sample code in Section 4 covers all of these, and I recommend you start from it (or from the source of another bot).

- In order to save/load bots, your class must implement `java.io.Serializable`.
- In order to use the `Orientation` and `Direction` classes, you must import them .
- Your bot **must** be in the package that corresponds to its directory relative to `RunMazeUI.class`. This is, typically, `bots` (if in doubt, check the config file).

4 Template Bot Code

```java
package bots;
import maize.*;
import java.io.Serializable;

public class EmptyBot implements Bot, Serializable {

    /** Implementation of the Bot interface.
     * @see Bot
     */
    @Override
    public int nextMove(boolean[][] view, int x, int y, int o, int fx, int fy){
        // Behaviour Code in here
        return Direction.FORWARD;
    }

    /** Implementation of the Bot interface.
     * @see Bot
     */
    @Override
    public String getName(){
        return "Bot Name";
    }

    /** Implementation of the Bot interface.
     * @see Bot
     */
    @Override
    public String getDescription(){
        return "A quick bit of info about the bot.";
    }

    /** Implementation of the Bot interface.
     * @see Bot
     */
    @Override
    public void start(){
        // reset code for starting a maze
    }
}
```

A Listing of Method Signatures

A.1 Orientation

```
/** NORTH direction */
public static final int NORTH = 0;

/** EAST direction */
public static final int EAST = 1;

/** SOUTH direction */
public static final int SOUTH = 2;

/** WEST direction */
public static final int WEST = 3;

// Get the name of an orientation as a string
public static String getName(int o);

// Rotate to fit current context
public static boolean[][] rotateToNorth(boolean[][] view, int o);

// Rotate clockwise
private static boolean[][] rotateCW(boolean[][] mat);
```

A.2 Direction

```
/** FORWARD move */
public static final int FORWARD = 0;

/** BACKWARD move */
public static final int BACK = 1;

/** RIGHT move */
public static final int RIGHT = 2;

/** LEFT move */
public static final int LEFT = 3;

// Get the name of a direction as a string.
public static String getName(int d);
```