CPL

JavaScript
2014

Abstract

This document is a guide to learn some basic programming in JavaScript. It is an integral part of the course on "Comparative Programming Languages" of prof. Frank Piessens.

Contents

1 FightCodeGame **1**

1.1 Setup 1

1.2 Debugging 2

2 Implement a basic robot **2**

3 Building your robot **3**

1 FightCodeGame

The practical session for CPL on JavaScript is about coding your own robot and battle against others. In this session, we will rely on the programming platform of the website `http://fightcodegame.com`. The platform consists of a local fight game engine that simulates fights between robots. The server-side of part of the platform, which keeps track of challenges and rankings, will not be used.

FightCodeGame is a free to play game, centered around the creation of so-called robots (or tanks) that have to fight on a battlefield against others. You – as a developer – have to think about the AI of the robot and implement it in JavaScript on top of a given API.

1.1 Setup

For this exercise session you will only need a working web browser. The documentation of the API of the robots can be found on http://fightcodegame.com/docs/.

This exercise session can be done via the online web site http://fightgamecode.com. In case of emergency, you are provided with a local copy of the web site. First, download the `cpl-javascript.tar.gz` file from Toledo. Next, open a terminal and run the following commands.

Unpack the tarball in your home directory `/home/<studentnumber>`:

```
tar zxf fightcode.tar.gz
```

Run the local copy:

```
cd /home/<studentnumber>/cpl-javascript/fightcode
python -m SimpleHTTPServer 8888
```

Next, open a new browser window and go to http://localhost:8888.

1.2 Debugging

To help you while programming your robots, you can rely on your browser's built-in functionality for debugging.

- https://developer.chrome.com/devtools/docs/javascript-debugging
- https://developer.mozilla.org/en/docs/Debugging_JavaScript

2 Implement a basic robot

```
1 function Robot() {};
2 Robot.prototype.onIdle = function(ev) {};
```

Listing 1: Example code of a very minimal robot.

The skeleton code of an extremely simple robot is shown in Listing 1. The robot consists of a plain Javascript constructor function, named `Robot`. The engine behind FightCodeGame, called the fight engine, will use this constructor function to create an instance that will fight against other robots.

```
1 var Robot = function(robot) {
2 };

3 Robot.prototype.onIdle = function(ev) {
4     var robot = ev.robot;
5     robot.ahead(10);
6     robot.back(10);
7     robot.turn(90);
8 };
```

Listing 2: Example code of a simple robot.

The fight engine will simulate a battle between robots. To do so, it will call event handlers that can act on specific events that happened during the fight. These event handlers can be implemented by your robot. For example, when your robot runs out of actions during the game loop, the fight engine will call the `Robot.onIdle` event handler. A simple robot that moves around a bit, is shown in Listing 2.

Task: Enhance the robot from Listing 2 so that it can win from all three scenarios. You can choose a different opponent in the dropdown list on the battlefield (see Figure 1). Rely on the documentation to learn what event handlers can be installed and what information event objects contain.

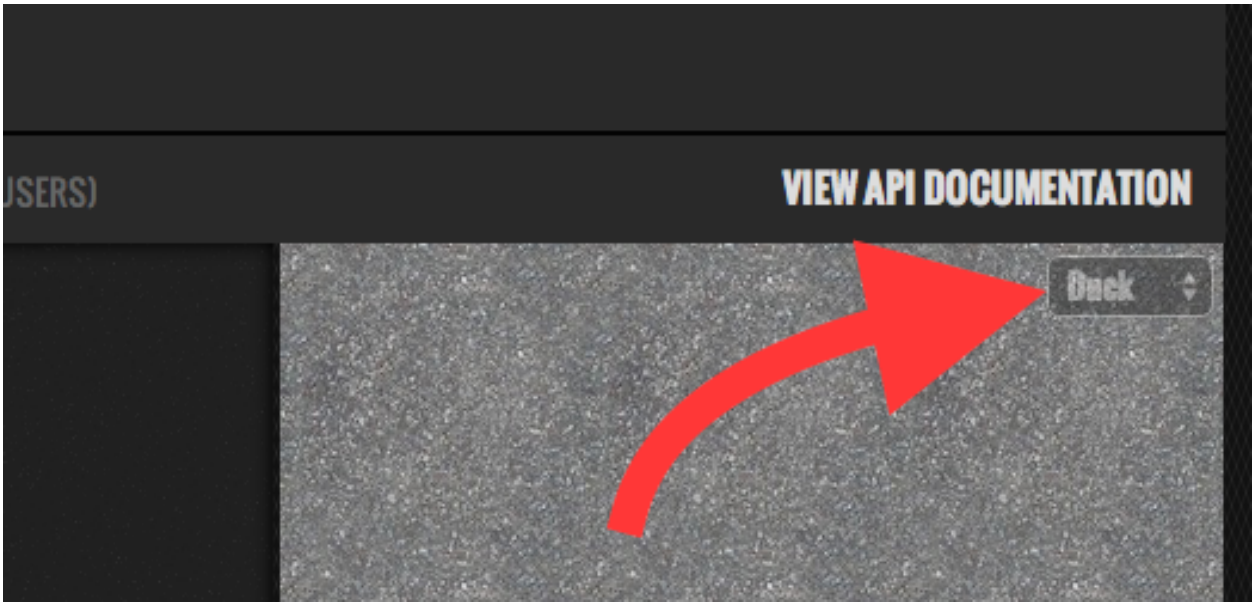

Figure 1: The top right corner has a dropdown list to test your code on different AIs, with `Duck` being the standard.

Task: Make clever use of the ability to clone your robot via `robot.clone()`. This will have a certain impact on your code as you can't be sure anymore that e.g., every scanned robot is an enemy (it could be your clone). Carefully read the documentation about the `clone()` function.

3 Building your robot

```
var Robot = new RobotBuilder()
    .on("idle", /* handler */)
    .on("wallCollision", /* handler */);
```

Listing 3: The use of a 'RobotBuilder' object in practice.

Task: Use the builder pattern to implement a builder object `RobotBuilder`. Transform your original robot code so that it uses the `RobotBuilder` as exemplified in Listing 3.

Task: Guarantee that the `RobotBuilder` is called with `new`. What goes wrong/different if you forget the `new`? Think about the effect of `new` when applied on a function constructor. How can a constructor check if got called with `new`?

Task: Each event handler receives a robot event `ev`. Enhance your builder object so that you can simply write handlers that have direct access to `ev.robot`. Do this by wrapping the original handler with a custom closure in the `on` function of the builder object. To invoke functions, search for `Function.call()` or `Function.apply()`.

```
function directRobotHandler (robot) { /* directly access robot instead of via ev.robot */ }
var Robot = new RobotBuilder().on("idle", directRobotHandler)
```