**ML-Bot Manual**

**10th September 2017**

# This manual explains how to install and work with our bot. It is consisted of three parts:

* Installation
* Running the bot
* The bot’s different modes

**Prerequisites and Installation:**

To install the bot you will have to add an extension to your browser. Please follow the “README” of the original bot:

<https://github.com/ErmiyaEskandary/Slither.io-bot>

And the extensions installation guide: <http://slither.jlynx.net/>

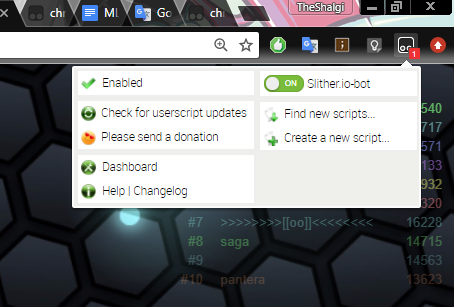
We present the following example on chrome:

# Browser: Google Chrome version 60

It may or may not work with other browsers or versions as the child nodes, representing web-page objects, might vary from one browser and version to another and are hard coded.

# Browser Extensions: Tampermonkey

This extension lets us inject the JS code to the slither.io html page, so it can work with our machine learning model. Otherwise we wouldn’t be able to use the already implemented browser logic chrome offers. Obviously we do not want to re-implement such logic, and we also chose not to interact with the server In a low-level manner - such thing would also demand a lot of work just for the interaction and parsing.



Here you can see our JS code called Slither.io-bot is ON. It is matched with the address slither.io

# Python: Version 3.5.2

Both the model and the proxy server are implemented in python. 3.5.2 is probably the most stable python version for Tensorflow in windows. That’s why we chose it.

# Python packages:

Tensorflow - version 1.1.0

Our model is using a Deep Neural Network and as for today we found Tensorflow as the most comfortable way to implement it, moreover, it is the state of the art technology for good performance.

Numpy - version 1.12.1

This widely-used package is very comfortable and very efficient for processing arrays (of any number of dimensions) so it was very useful for us.

Flask - version 0.12.2, Flask-CORS 2.1.2

Flask is a library we found helpful for communicating with the HTTP ajax requests sent from the JS.

Matplotlib - version 2.0.2

Used for creating graphs of the results of running some models

Scipy - version 0.19.0

Used for input visualization creations

# All the above packages, except for Tensorflow and Flask, are included in the installation of Anaconda

# Running the bot:

# Elements

## Remote Server

This server is where most of the processing of the game occurs. This server is in possession of slither, and we did not have the opportunity of learning it or changing it.

## Javascript Logic

This is the part of the code which runs under the context of our browser (mostly used chrome v60). It allowed us to participate in the game, transfer all the data which represents the game play environment. It is based on a JS code we found on the internet, implementing a heuristic-based AI-Bot.

Instead of choosing a heuristic based action it is now communicating with our model using the proxy server.

## Proxy Server

A communication script that is meant to transfer data from the JS code to the model, and also transfer the model’s output data back to the JS code.

## 

## ML Model - Python Logic

Most of our logic. Implementing a policy gradient descent model and a DQN model. For more details regarding those please see our article.

# Usage

Running our model Includes loading the JS code, specifically bot.user.js, to Tampermonkey (or parallel extension) and turning it on. This is done by clicking the “Create a new script” button, pasting the JS code of bot.user.js into the editor and saving. Now it is important to check the script is enabled, and to open or hard refresh the slither.io tab if it is already open.

The JS code includes several running modes. Worth mentioning:

Bot - enabling or disabling JS logic.

Auto respawn - whether to automatically restart another game after losing.

ML mode - has several options:

* Disabled - no Machine-Learning based model is running, runs the original heuristic AI bot.
* ML server mode - the machine learning model has control over the bot.
* IL-mode - imitation learning mode, using AI-heuristic based model has control over the bot, used by machine learning model in order to observe and imitate the AI bot.
* JS bot - the implementation of our model in JS. Is not actually capable of learn as the reinforcement learning function does not work well with the library CNN implementation, left for presenting the option, alpha ‘release’.

Visual debugging - active in both AI and ML modes, allows to visually see interior model’s properties.



Here you can see the bot is on, automatic respawn is enabled, the ML mode is ML server mode, logging is on and visual debugging is off.

Before starting a game, it is important to run the proxy-server script. Just open a cmd prompt and run “python server.py”. The server is waiting for requests and responses now. Now it is ok to start a game in one of the machine learning modes.

After running the proxy server and starting a game, it is time to run the specific model, more specifically the DQN model, called DQN\_model.py, or the policy gradient model, called policy\_gradient\_model.py. They do not require parameters in the command, their parameters are defined in parameters files called: Policy\_Gradient\_Params.json, DQN\_params.json, CNN\_params.json. They are responsible of critical flow variables, including the depth and shape of the NN, whether to load weights from a file and which file exactly and much more.

The parameters will be auto generated when you run one of the models for the first time.

**Possible Bug:**

The function getMyScore in the JS code is based on the location of a <div> element. In most of the browser it is the 17th child of the <body> element, but in some cases it is the 16th child.

If it’s not the 17th child in your browser you will see the on the console when running the DQN model that the average result is always 10. In that case try to change the child index in the function to 16. If that’s not working, debug it threw the developer tools.