Data Challenge Feasibility Research

Data Driven Innovation Challenge

Thomas Van der Molen

|  |  |
| --- | --- |
| **Project Information** | |
| Project members | Thomas van der Molen |
| Project Name | Leveraging reinforcement learning for automated testing |

Table of Contents

[Introduction 3](#_Toc152337894)

[Glossary 3](#_Toc152337895)

[Web-page interactions 4](#_Toc152337896)

[Selenium 4](#_Toc152337897)

[Showcase 5](#_Toc152337898)

[Selenium to RL Environment 7](#_Toc152337899)

# Introduction

This document contains summarizations for the set-up and results of feasibility studies performed for the separate components necessary, established in the [Project Plan](../DataDrivenInnovationChallenge%20Plan%20-%20Thomas%20Van%20der%20Molen.docx) (it is highly recommended to read the Project plan first).

To utilize a reinforcement learning agent to find anomalous behavior within a web application, several large road blocks will have to be proven feasible first. These feasibility tests are performed in accordance to the [DOT Framework](https://ictresearchmethods.nl/), and will span across many research strategies such as **Library** with documented research, **Workshop** via jupyternotebooks, and **Lab** utilizing an MVP.

## Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Reinforcement learning (RL) | A machine learning technique were the model (agent) learns from positive and negative feedback for given actions. |
| Agent | Within reinforcement learning, an agent refers to the model that performs actions and updates its internal knowledge of them based on the feedback received. |
| Environment | A virtual instance of a web-page that will be interacted with. |
| Webdriver | The interface that allows code to interact with a browser. |

# Web-page interactions

To interact with a website two things have to happen; first, we need to obtain the page information from the website, secondly we need to interact with elements on the website.

Thankfully, I am not the first one to have these requirements, as web scrapers, hackers, and tools to allow website to run in an app (such as a mobile app), have all created great tools for this.

For this research, we will be using the chrome service with [their webdriver](https://sites.google.com/chromium.org/driver/), as google has very mature development support for automated usage of their browser, and chrome is the [most popular browser in use currently](https://gs.statcounter.com/) (so any anomalous behavior encountered on chrome, should impact a large portion of a general user base).

A pie chart with different colored numbers

Description automatically generated

## Selenium

To programmatically interact with the browser and thus webdriver provided by chrome (google), [Selenium](https://www.selenium.dev/) will be used, this Open-source framework is primarily used for automating functional testing of web browsers (e.g. automatically testing interactions) and had become very popular with websites such as [LinkedIn](https://www.linkedin.com/), [WordPress](https://wordpress.com/) and [Shopify](https://www.shopify.com/) using it for their automated testing.

Selenium has support for [many languages and webdrivers](https://www.kualitatem.com/blog/selecting-the-scripting-language-for-new-webdriver#:~:text=Selenium%20WebDriver%20supported%20languages%20include,PHPUnit%2C%20and%20FitNesse%2C%20etc.), allowing us the freedom to switch to a different technology stack completely if necessary.

Selenium will be able to take care of both obtaining page information and interacting with the page programmatically all through the webdriver.

### Showcase

As a short showcase and part of the feasibility testing, a Selenium notebook (<Selenium.ipynb>) has been created, that will interact with a very simple testing Environment ([BasicErrorLog.html](Environments/BasicErrorLog.html)). This testing environment is an HTML page that contains a single target button, with an eventlistener that will throw an error when clicked.

A screenshot of a computer

Description automatically generated

Figure BasicerrorLog.html

Using Selenium in python, only requires [Selenium to be installed via pip](https://pypi.org/project/selenium/) and the [chrome driver it connects with](https://googlechromelabs.github.io/chrome-for-testing/) (assuming chrome itself is [installed already](https://www.google.com/chrome/)).

After installing the prerequisites, a chrome service and webdriver can be initiated and sent to retrieve the required webpage.

A screenshot of a computer program

Description automatically generated

Figure Initializing environment

After this, the driver can be interacted with directly to obtain information of the browser/webpage and interact/modify this information (Rule of thumb: If you can do it in javascript, you can do it with the driver, and if you can’t with javascript you probably can still do it with the driver).

After initializing the driver and obtaining the correct environment information, interactions can be performed with the page, such as finding a button and clicking it.

A screenshot of a computer screen

Description automatically generated

Figure Interacting with environment

Any information of use to us can be retrieved from the driver as well, such as the browser logs where the error was thrown to.

A screenshot of a computer

Description automatically generated

Figure Reading browser logs

All functionality of the python API for Selenium is also [documented on their dev page](https://www.selenium.dev/selenium/docs/api/py/).

## Selenium to RL Environment

Reinforcement learning agents require knowledge of their current available actions, against the current state. It uses this to form a quality mapping of what actions will be best to perform in which scenario.

To make the agent interact with the web-page in a meaningful way (to find possible anomalous behavior), it will need to be “steered” in a direction that might lead to situation were mistakes can happen, such as [keywords of interest](https://en.wikipedia.org/wiki/Sentiment_analysis), or interactable objects (such as elements with [eventlisteners](https://www.w3schools.com/js/js_htmldom_eventlistener.asp)).

These different types of objects can be obtained from the webdriver to create the current state of the webpage and all possible actions we allow the agent to perform. Due to the [ever evolving single-page-applications of the modern day](https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application), these values are continuously changing, thankfully smart people have already come up with RL models that can handle continuous states such as [DQNs](https://paperswithcode.com/method/dqn).