Report

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

[Recap 3](#_Toc156486006)

[Glossary 3](#_Toc156486007)

[Summary 4](#_Toc156486008)

[Goal 5](#_Toc156486009)

[Methodology 5](#_Toc156486010)

[Similar Products 5](#_Toc156486011)

[Environment 6](#_Toc156486012)

[Gym 6](#_Toc156486013)

[Extensibility 6](#_Toc156486014)

[Observation/Action Spaces 6](#_Toc156486015)

[Selenium 8](#_Toc156486016)

[Agent 9](#_Toc156486017)

[Modelling 9](#_Toc156486018)

[Rainbows! 9](#_Toc156486019)

[Implementation 10](#_Toc156486020)

[Considerations (ethics) 10](#_Toc156486021)

[Future 11](#_Toc156486022)

# Recap

During the AI-Advanced semester at Fontys, all students were tasked with working 10 weeks (week 8-18) on an innovative and data driven project.

The Leveraging reinforcement learning for automated testing project, is a product of an idea researched during the earlier Open Program of the same semester, during this period preliminary research was done to establish the feasibility, potential and innovation-ism of this idea. To learn more about the Open Program work done, it is recommended to read the Open Program [Plan](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Open%20Program/Open%20Program%20Plan.docx) and [Report](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Open%20Program/Open%20Program%20Report.docx).

The Preliminary research/analysis led to the [Project Plan of the Data Driven Innovation Challenge](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Data%20Innovation%20Challenge/DataDrivenInnovationChallenge%20Plan%20-%20Thomas%20Van%20der%20Molen.docx), in which background information, similar products and potential downsides were explored alongside the definition of the project’s goal and scope.

# Glossary

Throughout this document, several domain specific terms will be used, some of these terms have been expanded on below with a potential shorthand that might be used to avoid unnecessary repetition.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Reinforcement learning (RL) | A machine learning technique where an AI model learns from positive and negative feedback for given actions. |
| Agent | Within reinforcement learning, an agent refers to the system that utilizes the AI RL model to decide on actions to take and when to update given model. |
| Environment | A virtual representation allowing for observations and actions to be made to simulate a real-world scenario (used when training the model). |
| Webdriver | The interface that allows code to interact with a browser. |
| Web-application | A remote application that is (often) accessed via a browser as a website. |

# Summary

# Goal

The goal of this project is to utilize Reinforcement Learning to automatically test web-applications for anomalous system behaviors. By setting up proper RL environments and agents, the hope is that an RL agent can train and thereby test any web-application it is provided, no client-side (web-application) integration needed.

Another benefit of this approach is; an AI model will be able to not only target known weak points of an application, but can also explore new possible issues not previously encountered or even considered.

# Methodology

During this project, the [DOT framework](https://ictresearchmethods.nl/) has been used to research, investigate and iterate on various parts of the project, such as with the (but not limited to): Library, Lab and Workshop methods.

**Main question:**

*Can Reinforcement learning be leveraged to automatically identify anomalous behavior within a web-application?*

***Sub questions:***

* *How can a web-page (HTML, CSS, JS) be converted into a RL environment?*
* *How can an RL agent interact with a web-application?*
* *What anomalous behavior can be expected within a web-application?*
* *What factors should affect an RL agent’s behavior?*

# Similar Products

When considering the feasibility and work left to do, it is performing an [available product analysis](https://ictresearchmethods.nl/library/available-product-analysis/) (as described by the DOT framework). Some similar products found were:

* [Applitools](https://applitools.com/)
* [Testim](https://www.testim.io/)
* [Mabl](https://www.mabl.com/)
* [Perfecto](https://www.perfecto.io/)
* [Test.ai](https://test.ai/all-products)

These similar products all market themselves as automated testing platforms powered by AI. None of these however, do what I want to achieve as most of them are either glorified [GitHub copilot](https://github.com/features/copilot) for writing tests, or are based on visual testing which boils down to checking the validity of HTML (which to my understanding is done just as well by [google lighthouse](https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=nl) without any AI and is completely Open-Source).

# Environment

To test any web-application, an environment has to exist from where the web-application can be observed and interacted with. For human testing this environment is usually just a web browser such as [google chrome](https://www.google.com/chrome/) that supports many different ways for a user to interact with the web pages. Automated tools that programmatically interact with web-application such as [Cypress](https://www.cypress.io/), who create their own browser using [Electron](https://www.electronjs.org/), and run any interactions side-by-side with the application (fully integrated).

As this AI project should require zero integration on the website’s end, any direct integration systems will not work, instead a programmatical system will have to be used that allows for similar interactions/observations to be made as a human would.

## Gym

A popular standard to use when working with Reinforcement Learning is [Gym](https://gymnasium.farama.org/), this API standard tries to ensure that a certain life-cycle and process-flow are used throughout all RL environments, thereby making it easier to swap environments, which can be very useful when evaluating various models for example, or for future collaboration/resource sharing.

There are no Gym environments that suit the scope/use-case of this project, therefore a custom one will have to be created. Creating our own environment, gives the benefit of allowing us to architect and build the environment as we see fit.

### Extensibility

While this environment will be created with the use-case of this project in mind, the environment is also architected to keep future expansion or other use in mind, making this environment not just useful for this scope but allowing others to also use it in the future. To accomplish this, the environment (besides being completely [Open Source](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Data%20Innovation%20Challenge/Feasibility/Environments/Gym/WebEnvironment.py)) is fully configurable without editing any source code, allowing users to easily change or tweak the reward tables, specifying the target domain/website, etc.

### Observation/Action Spaces

An important part of any environment, is to have robust definitions of the available observations and actions that an AI model might use to determine their actions, within Gym these are called [Spaces](https://www.gymlibrary.dev/api/spaces/). Defining proper spaces can be very difficult as you must give enough information to make substantiated actions, while not giving unnecessary information or too many options as it could confuse any model trying to learn.

During the feasibility research, much time was spent on how to correctly establish an observation and action space for these RL models within a web environment, and it is highly recommended to read [the full research](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Data%20Innovation%20Challenge/Feasibility/Data%20Challenge%20Feasibility%20Research%20-%20Thomas%20Van%20der%20Molen.docx) for a more detailed explanation on the considerations made.

#### Dynamic Spaces

A screenshot of a video

Description automatically generatedWhen considering an interactive website, many observations or possible actions might change over time, think of the amount of buttons that might appear when navigating from page to page.

Figure Youtube landing page buttons (13 interactables)

If the Observation state would be the HTML itself for example, it would be constantly changing in size and contents, or with the action state, the number of interactable elements (such as buttons, input fields, etc.) changes constantly.

The constantly changing nature of these pages can be represented in the observation/action spaces, however, most RL models that are currently being developed do not account for this and expect these spaces to be of a static shape.

While attempts have been made, such as this [very interesting research paper](https://arxiv.org/pdf/1905.03970.pdf), most are still very limited in application and hard to reproduce and thus will be considered as outside of this projects scope, but could be an [interesting topic for the future](#_Future).

#### Observation Space

The Observation space is used by RL models to determine the current state of the environment; thus it is very important to properly represent the current state of the environment through this space. The features from a state are generally used to ‘steer’ the agent into making actions that will lead to the desired outcome.

The features that were decided on to be used are: page metadata (page title, page URL), count of interactable elements, count of encountered diagnostic messages (errors, warning, information, etc.), count of HTML validation errors (such as an improperly closed element) and the amount of actions taken since [last interaction](#_Action_Space).

One feature that ended up not being used is the count of keyword, this was implemented using a query algorithm that looks for a pre-defined list of words on the current page.

The keyword-count however, made the model worse at converging to a viable action strategy, this could be expressed in the time it would take the agent to find a consistent number of errors, which with the feature would take ~10.000 actions, while without it could be as low as ~100 actions.

#### Action Space

The action space is generally seen as a list of all available actions that the agent can make in the environment for the current state, within webpages this could have easily been a list possible interactions with all available interactable elements, this method however would require a dynamically sized action space, which as [previously encountered](#_Dynamic_Spaces) would not be suitable for most RL models and therefore the current scope.

Instead, an action space similar to that of [Turing complete systems](https://en.wikipedia.org/wiki/Turing_completeness) is used, these systems generally attempt to use as little actions options as possible while still allowing a system to perform any thinkable action. Using this mindset within our web environment, we can store an internal list of all interactable elements with 2 possible actions: select the next element in the list, and interact with the currently selected element. Using this Turing complete-like action system, our agent will be able to still interact with every possible element in the environment, while only requiring a static actions space (of 2 actions).

#### Convergence Stagnation

With the very popular Q-learning structure for reinforcement learning, one problem that is often encountered is a problem with the model [never converging, or finding its end objective](https://towardsdatascience.com/convergence-of-reinforcement-learning-algorithms-3d917f66b3b7) (this can be due to many factors, such as local minimums, unclear reward structures, etc).

This is also an issue with the proposed statically scoped (turing-esq atomic steps) action space, as a single action does not achieve much in the environment as it might just be cycling through possible action options before choosing one.

Most implementations of reinforcement models, expect each action and reward to be relatively very important to the performance of the model, for us however this is not the case as a combination of actions will lead to a new state with possible rewards attached.

Two possible solutions for this, is to reduce the importance of a single action, however this does not actually solve the problem as it will make convergence that much harder, due to the model getting very small rewards that are hard to learn from. A second and preferred option however, is to take multiple actions into a single action scope, where an action scope is terminated by the defined interaction act (e.g. an action scope might be nextitem->nextitem->nextitem->interact) and give a calculated reward based on all the actions taken at once (which would also slightly speed up the model’s performance due to the environment state calculations being shorthanded as they do not change completely).

## Selenium

We have [established a methodology for observing and interacting with a web environment](#_Gym), now we just need to find a way to actually perform the actions within an actual website, without requiring the website to do any pre-integration, such as with [other automated testing tools](#_Environment).

To programmatically interact with the web-application [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.

A pie chart with different colored numbers

Description automatically generatedTo get Selenium to interact with a browser, a third tool will be needed called a webdriver, this will handle all communication between the programmatic Selenium and the website. While there are [several web drivers to choose from](https://www.selenium.dev/documentation/webdriver/drivers/), chrome’s [webdriver](https://sites.google.com/chromium.org/driver/) with [their services](https://googlechromelabs.github.io/chrome-for-testing/) was chosen for this project, 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).

# Agent

## Modelling

### Rainbows!

# Implementation

# Considerations (ethics)

# Future