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](#_Toc156573429)

[Glossary 3](#_Toc156573430)

[Summary 4](#_Toc156573431)

[Goal 5](#_Toc156573432)

[Methodology 5](#_Toc156573433)

[Similar Products 5](#_Toc156573434)

[Environment 6](#_Toc156573435)

[Gym 6](#_Toc156573436)

[Extensibility 6](#_Toc156573437)

[Observation/Action Spaces 6](#_Toc156573438)

[Selenium 9](#_Toc156573439)

[Agent 10](#_Toc156573440)

[Modelling 10](#_Toc156573441)

[Rainbows! 11](#_Toc156573442)

[In Practice 12](#_Toc156573443)

[Considerations (ethics) 14](#_Toc156573444)

[Future 15](#_Toc156573445)

# 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

During this project, it was attempted to research and develop an automated testing tool levering AI, as from research it was found that no similar products are available yet, while it seems to be a feasible area for AI to expand into.

For this challenge, it was decided that Reinforcement Learning would be the methodology/AI technology that would be used.

To utilize Reinforcement Learning for automated web testing, first an Environment had to be created for the RL agent to interact with and observe while testing/training, this environment has been created utilizing the chrome webdriver and Selenium, with OpenAI’s Gym API standard to allow individuals to easily expand on this project or use the environment for their own ideas.

Secondly, a model had to be developed for the RL agent to use for making decisions on the most optimal actions to make, for this a Rainbow Agent was used that was modified to more effectively steer the model to this project’s use-case and support the complex environment it works within.

Lastly, ethical considerations were very much taken into account when working on this project, as this has become an increasingly popular but also scary topic within AI, and thus should be taken seriously when working with new innovations in the domain.

# 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 1 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-phase, where an action-phase is terminated by the defined interaction act (e.g. an action-phase 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

Reinforcement learning systems consist of two components (when simplified) that interact with one another via the agent, these being the [environment](#_Environment) in which the actions of the system are applied and observations are obtained from, and the [model](#_Modelling) that is trained to predict the best possible action for the current state (and possibly actions) of the environment.

A grey and white person's profile

Description automatically generatedFor the model to obtain the necessary information to predict the best action, and the given action to be executed in the environment, an agent can be seen as the middle-man, exchanging the information between the two systems, thereby managing the application-flow.

## Modelling

There are many techniques that might be used to take the place of the “model”, as all it is supposed to do is give the agent the action it should take, this could be as simple as just randomly choosing an action or made more complicated by adding reasoning to what action to take.

Many RL models, work on the idea of a [quality table](https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/), this (Q-)table essentially contains every possible state and will assign a “quality score”, to the available actions for that given state, thereby being able to find the best possible action for the current state by looking for the action with the highest quality score.

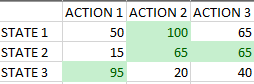


Figure 2 Example of a simple Q-table (green indicating the best action for given state)

These Q-tables can be changed and tweaked during training, thereby allowing the model to find the best state-actions on its own. For a more in-depth explanation on how these systems are developed in practice, I would highly recommend [this resource](https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187).

These tables can work surprisingly well when the state are not very complex, such as the tiles on a game board. When the state becomes more complex, for example chess, the amount of possible states becomes too large to [feasibly process in a simple table](https://medium.com/@samgill1256/reinforcement-learning-in-chess-73d97fad96b3). To handle these more complex states (e.g. a state with a large number such as a car’s speed) a [neural network](https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871) is used to store a representation of the states with the output layer being the possible action (for which the output node with the highest activation is often seen as the best possible action).

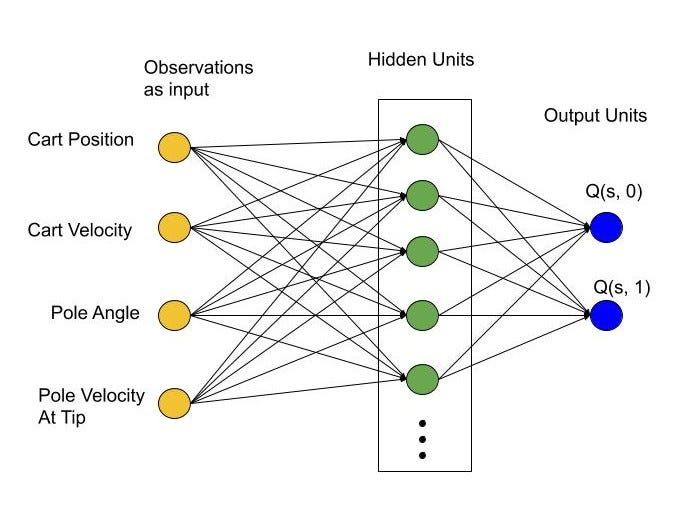


Figure 3 Example of a simple game board with limited states

### Rainbows!

Over time, many different methods to try and improve these models have been proposed and implemented, sometimes to [staggering success](https://medium.com/@smartboy91221/openais-pioneering-contributions-to-reinforcement-learning-805f6a24e7f1). A lot of the problems that these improvements try to overcome still exist in the commonly used models such as the previously discussed [convergence problem](#_Convergence_Stagnation).

Many of these improvements however, are not exclusive to one-another and can be used in unison, these models that implement many different “components” (improvements to the generic RL model) has been creatively dubbed the “Rainbow Model”. For a short explanation of some of the most common modules and DQN’s in general, I would highly recommend watching [this video](https://www.youtube.com/watch?v=lUpW1OlJmsc) by Tristan Behrens.

There are many different rainbow models out there, as everyone builds their own based on their use-case (which makes it significantly more difficult to use them without the necessary background knowledge of all the components). For this project, a modified version of the [rainbow model by Curt-Park](https://github.com/Curt-Park/rainbow-is-all-you-need) has been used, the noteworthy improvements this model provides are the: Double models, Experience replay and N-step learning, with the version used in this project, having a modified neural network pre-processor to properly handle the more complex states our environment provides, as well as an expanded [noisy layer](https://arxiv.org/pdf/1706.10295.pdf) system oppose to the [epsilon greedy approach](https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/).

# In Practice

The [Environment](#_Environment), [Agent](#_Agent) and [Rainbow model](#_Rainbows!), can now finally be used together to autonomously/automatically test any web-application we would wish to target, which can be reproduced via [this training notebook](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Data%20Innovation%20Challenge/Feasibility/WebAgentTraining.ipynb).

Most RL models, aim to achieve an as high as possible reward, in as little steps possible. For this project however, we would like the model to keep exploring new possibilities, or even search deeper on an already rewarding action path, this approach however, has [possible convergence downsides](#_Convergence_Stagnation) which have been explored earlier as part of the [environment](#_Environment), while extra additions have been made directly to the model to try and [encourage this behavior](#noisylayer), as shortly mentioned as part of the [rainbow model](#_Rainbows!).

The model has been trained and tested on several target domains, such as [youtube.com](https://www.youtube.com/), the [BasicErrorLog.html page](https://github.com/Thomas-Molen/FHICT-S7-AI/blob/main/Data%20Innovation%20Challenge/Feasibility/Environments/BasicErrorLog.html) and (a personal [Open Source web-app](https://github.com/Thomas-Molen/WebAdventure)) [WebAdventure](https://webadventure.thomasmolen.com/). When looking at the findings for WebAdventure for example, it can be seen that over time the RL agent finds multiple reward points, while never fully converging on a single path (as in this case the reward over time would end up in a pattern).

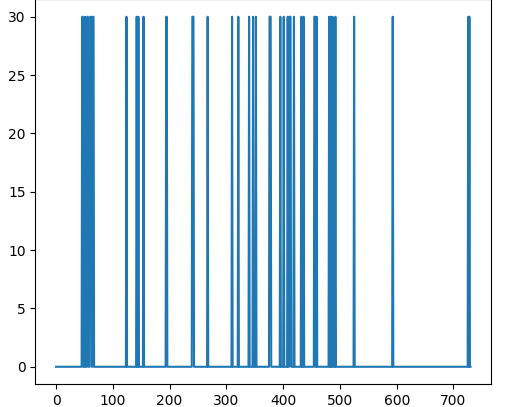


Figure 4 Rewards over time when training on WebAdventure

When looking at the loss chart, we can measure how much the model is trying to correct its internal DQN where high loss means the network is adjusted significantly, note that this loss factor has slight noise to it which is caused by the [project specific noise layers](#noisylayer) to ensure the model never fully convergence and becomes complacent of its current strategy.

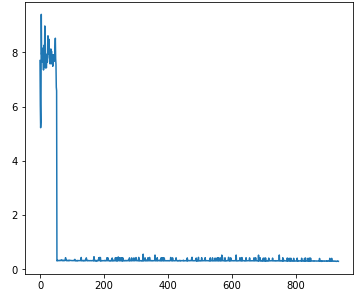


Figure 5 Loss over time when training on WebAdventure (default reward table)

The adjustments the model is making to its DQN can be tweaked due to the [easily adjustable reward structure of the environment](#_Extensibility), reducing the given rewards can for example make the model less convergent causing it to grab less onto a proven strategy and thus explore more extreme variations from known-good strategies.

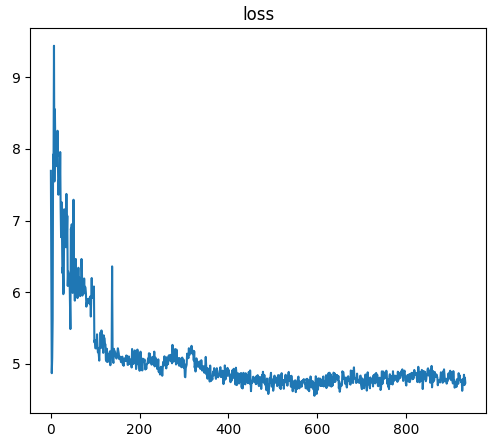


Figure 6 Loss over time when training on WebAdventure (adjusted reward table)

A training phase of the model on WebAdventure has also been recorded and can be found via [this Youtube Link](https://youtu.be/9nUJt1Go_is).

# Considerations (ethics)

#### Malicious use

Due to the AI model being built to identify possible anomalies within a web-applications functionality, it can very easily be used maliciously to find weak points within an application that can may be exploited by bad actors.

#### Energy/Environment

In recent years, the concerns of how technology has grown to wastefully consume more and more energy has become a [big talking point](https://www.theverge.com/2023/10/10/23911059/ai-climate-impact-google-openai-chatgpt-energy), especially with the resource intensive block-chains and large AI models of the current day.

This project does not aim to train a RL model to be extremely good at finding specific bugs/anomalies but instead utilizes to training process of such a model to rapidly explore a large range of possibilities. This process is sadly very resource inefficient as most of the results gained from such a training phase are thrown away and re-done the next time a web-application is being tested.

#### Open Source

Given the Open Source nature of the project, including the environment and research/implementations done for this project, collaboration is very much encouraged. However, due to this project being accessible to anyone, other people could use it in cases that it was not initially intended (think of the [malicious use consideration](#_Malicious_use)). Furthermore, as the project currently does not even fall under any kind of license such as the [MIT open source license](https://opensource.org/license/mit/), it can be used or abused in any manner with no repercussions which could reflect badly on the original creator.

#### Regulatory Compliance

A topic within AI that has been started to gain significant traction recently (especially with large models such as [ChatGPT using unauthorized data for training](https://www.informationweek.com/machine-learning-ai/openai-s-chatgpt-generates-lawsuits-over-data-use)) has been the legal justifications with training and using AI models and the potential need for governmental regulatory institutions to step in. As the project is developed to essentially scrape any web-application’s website for any signs of malicious use, it wouldn’t be a stretch to imagine [similar legal limitations](https://www.imperva.com/blog/is-web-scraping-illegal/) to be made for the legality of this tool compared to web scrapers.

#### Data Collection

While the legality of applying the tools developed in this project can fall under the same (very loose) rulings as web scrapers as discussed in [Regulatory Compliance](#_Regulatory_Compliance), the authorized use and collection of the data of any given targeted website might not fall to the person performing the automated testing, and it should be considered that authorization might first need to be given by the web-application’s owners. While the AI model, can only access functionality any user could feasibly access via a web browser (which includes edge-cases such as clicking disabled buttons), a case could be made that morally it could be morally wrong to specifically seek out and collect potentially harmful information of these unexpected behaviors.

#### Unforeseen consequences

If you are open to some “science-fiction-y” futures, an extremely well trained and large enough AI model, might be able to identify techniques to efficiently break login systems, causing a new form of brute force login attacks to be possible that were previously not considered, in a similar case as is now being discussed for [quantum computing](https://www.rand.org/pubs/commentary/2023/09/when-a-quantum-computer-is-able-to-break-our-encryption.html).

# Future

During this project, proof has been created of not only the feasibility of leveraging AI for automated web testing, but a functional version has been produced, this however does not mean that this tool is perfect.

Functionality for interacting with input elements such as text fields, was outside of the scope for this project, but would greatly expand on the possible test-cases that the tool could cover in the future.

Furthermore, plans were made to allow the web environment to record the actions taken by the AI model for replay in the future such as when reproducing a found bug. While this functionality has been considered when developing the environment, extra work would have to be done to implement this completely.

Adding an action-replay/history system to the environment, could also further improve the reward distribution by dynamically changing the reward given for a previously encountered error, thereby further encouraging the model to keep searching for new errors.

Lastly, as [discussed before](#_Convergence_Stagnation) action-phases are used to base reward distribution and processing on for this environment as it is a better representation of how the model should interact with the environment, however currently there is no consistent structure to fully reloading the target web-application generally contained within an “episode”, as an episode might last an arbitrary amount of action-phases depending on the complexity of the application itself. Finding a good way to scope episodes would improve the testing process and avoid edge-cases where website changes are brought over between what should be separate episodes.