Qubot: A Bot That Can Finally Replace QA Testers?

COMS E6156 Topics in Software Engineering
Columbia University
Team: Gren Maju

Anthony Krivonos
ak4483@columbia.edu

Kenneth Chuen
kc3334@columbia.edu

In the original Qubot paper [5], we examined various contemporary approaches toward creating a software framework that performs automated exploratory tests, but never ultimately brought the framework into fruition. In this paper, we herein present Qubot—an open-source automated exploratory testing framework for Python—for performing fast, modular, and reproducible exploratory tests of websites using Q-Learning. Unlike past approaches to automated exploratory testing, Qubot uses provides a highly configurable testing interface for users, compatibility with Selenium Webdriver, and easy installation and setup using Pip. Qubot was put through a series of tests to determine its feasibility in the real-world—a field test on its performance across ten unique websites and a developer experience study. After struggling on various websites under test (WUTs), it was concluded that Qubot has yet to become a fully portable framework. However, the capability of multiple real-world testers to successfully complete tests using Qubot has provided hope for Qubot's use in a non-academic setting.

I. INTRODUCTION

A. Background

With ever increasing compute speeds in accordance with Moore's Law [6] and highly specialized processing units and programming languages, the rate of development within the sphere of software engineering appears exponential. However, no matter how fast our code compiles and runs, nothing will ever alleviate the need for software acceptance tests in productionalized applications.

In the *Qubot: Designing a Bot to Perform Autonomous Black Box Testing* [5] paper, we set out to design a software framework that mimics human Quality Assurance (QA) testers by performing automated exploratory tests. Exploratory tests are a subgroup of black box tests in which the functionalities of a software under test are tested in an ad-hoc manner through manual exploration of the software, while test cases are determined during, not before, testing. [1] By running an algorithm based on website information web-scraped via the popular Selenium Webdriver framework, we were then able to automate such tests. [10]

After thorough review of several software testing architecture proposals and automated black box testing approaches, we settled on creating an open source framework with tests that may be invoked through a simple JSON configuration file, in a fashion similar to that proposed in Hanna M. et. al.'s paper *Automated Software Testing Framework for Web Applications.* [2] Once this configuration file is provided, the WUT is then webscraped via Selenium Webdriver and then explored by a Q-Learning agent inspired by the approach in Harries et. al.'s *DRIFT: Deep Reinforcement Learning for Functional Software Testing* paper. [3] Whereas we will discuss the composition of the framework in greater depth in this paper, an overview on the chosen architecture and autonomous testing technique for Qubot are shown in Figure 1.

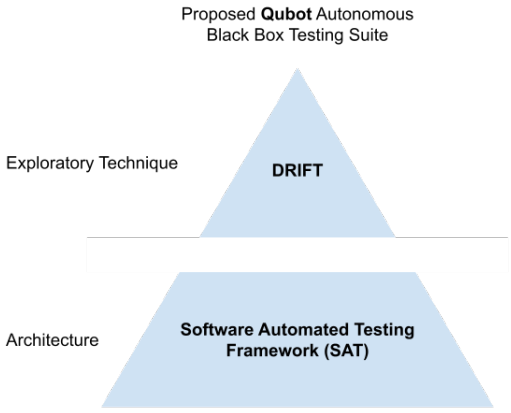

FIG. 1: Proposed Qubot Framework

B. Novelty Statement

Qubot was developed with the intention of becoming one of the few autonomous exploratory testing frameworks published for Python as of the present. It has the ability to crash test and simulate page flows for a wide variety of websites that do not require hyper-specific input generation nor difficult login mechanisms. Additionally, testers will have the option to specify their entire tests in a .qu JSON file, largely removing the need

for coding knowledge.

C. Value to User Community

Qubot's target community of users is the subgroup of software developers and engineers that do not have the monetary or temporal resources to devote a considerable amount of time into performing exploratory tests on their websites. This subgroup includes software engineers at startups and those working on solo projects. In choosing this user community, we decided to make Qubot open-source and available on the PyPi package index [9] for any developer to download and use as a command-line tool.

To determine the extent of Qubot's value to the aforementioned developer community, three overarching goals were enumerated during the designing of Qubot. These goals have been slightly altered during Qubot's development and are as follows:

- **Portability**: Can Qubot work on any WUT without changing the codebase, and can it work across WUT versions?
- **Intelligence**: Can Qubot effectively navigate its environment in a reasonable amount of time with minimum hardcoding?
- **Feasibility**: Does Qubot perform well across a wide range of websites consistently?

We will go back to these criteria after performing a field test and a developer experience test later on in this paper to assess whether the goals for Qubot have been achieved. If so, then we will be able to safely conclude that Qubot may, in fact, be a plausible alternative to manual exploratory testing.

II. OVERVIEW

A. Use of External Libraries

This framework is minimal in its use of external libraries. As mentioned in the Introduction, Selenium was chosen as the driver for Qubot. [10] Additionally, a custom OpenAI Gym environment was created for the Q-Learning agent. [8] Finally, the numpy library was used for minor calculations. [7] As Qubot learns on the fly by scraping interactive HTML elements off WUTs, it did not require external datasets.

B. Development

Qubot is divided into six subpackages and subclasses that perform distinct tasks, culminating into the entire testing framework. We will go into slight detail about what each subpackage does in the following subsections.

1. Driver Class

The **Driver Class** provides the interface between Selenium Webdriver and Qubot. When a new instance of Qubot is created, the Driver is provided with basic information such as the URL to begin the test from and the maximum depth of website to traverse. Given that walking a graph of linked websites is very computationally costly, this maximum depth value is defaulted to 10 sites. Finally, after the driver accesses the WUT and crawls the current page, as well as the pages that are linked by it, it returns a `UITree` object that can be explored by the Q-Learning agent.

2. UI Package

The **UI Package** contains several classes to summarize HTML web elements and user interactions that take place in the web browser. Firstly, the `UIAction` class enum provides three available actions the driver can take when scraping the webpage—left-click an element, fill an input element with text, and navigate to a child element of the current element. Next, the relationship between elements is captured in a `UITree`, which contains `UITreeNode`s that are connected to their parents via a `UIAction`. Take the sample `UITree` shown in Figure 2.

```
NAVIGATE: <html id="" class="">
    NAVIGATE: <body id="" class="">
        LEFT_CLICK: <a id="" class="my-website">
        NAVIGATE: <div id="about-me" class="content">
...
```

FIG. 2: A Sample UITree Printed to Console

When the Q-Learning agent traverses this tree starting from the `<body>` element, it has the choice of traversing to the `<head>` element via the `NAVIGATE` action or to the `<a>` element via a `LEFT_CLICK` event, and so on.

3. Environment Package

The custom Q-Learning agent lives in the **Environment Package**. The `QLearningEnvironment` class subclasses the broader `Environment` class, which is a custom Q-Learning environment created with OpenAIGym skeleton code. In the former class, computations on the Q-function are performed and important statistics such as rewards and penalties are recorded.

The parameters for the Q-learning environment include the following:

- α: The learning rate. Higher values of α will consider more recent pieces of information.

- γ: Weighs the importance of future rewards. Higher values of γ more greatly consider long-term rewards.
- ϵ: Randomness parameter. Lower values of ϵ favor exploration (seeking out new actions and states) and higher values of ϵ favor exploitation (seeking out actions that produce greatest rewards).
- **ϵ-decay**: Rate at which to decrease the value of ϵ to introduce more randomness.

Using these parameters, the Q-function is updated according to the generic Q-learning update rule:

$$Q^{\text{new}}(s_t, a_t) = Q^{\text{old}}(s_t, a_t) + \alpha \tag{2.1}$$

$$\times [r_t + \gamma \times \max_a Q(s_{t+1}, a_t) - Q^{\text{old}}(s_t, a_t)], \tag{2.2}$$

where t is a training iteration and r_t is the value of the reward at time t. To simplify the configuration of the testing framework and to encourage code reuse, seven preset rewards functions are available to the tester, each of which induce different rewards and penalties based on which node in the tree the agent visits. Additionally, a parameter called `step_limit` was used to limit the number of actions the agent may take during each training and testing episode if it never finds a terminal state. [11]

4. Config Package

The **Config Package** is the culmination of each of the aforementioned classes and packages, and includes the `Qubot` class that is exposed to the tester. This class contains info on the terminating elements in the test (which elements the agent must find), a `Driver`, a `QLearningEnvironment`, and some auxiliary information.

There are many ways to invoke a test via the Qubot class. In all cases, the tester begins by installing the **hosted Qubot package** either globally or within a virtual Python environment via `pip3 install qubot`.

To begin a programmatic test (a.k.a. *Techinque 1*), which is more suited to engineers and testers with software development backgrounds, one may create a testing script such as that shown in Figure 3.

Here, a new `Qubot` instance is created, and it is designated to test the `https://upmed-starmen.web.app/` site. During testing, the bot is tasked with finding an HTML element containing the class attribute `SignIn_login_hcp__qYuvP`. Morover, during training, the bot is tasked with finding an HTML element containing the text "Log in." All driver parameters are provided in `driver_params` and Q-Learning parameters are provided in `model_params`. The preset reward function, `ENCOURAGE_EXPLORATION`, is designated in `reward_func`. Finally, when the bot comes across an input tag during webscraping, it will first look at the values provided in `input_values` to see if the type of the input it encountered matches. If so, it uses the provided text as input. Finally, both the training and testing operations are invoked with `run()`, and stats from these runs are printed at the end.

The tester can also provide the entire Qubot configuration in a JSON file and run the same exact test with much fewer lines of code (a.k.a. *Techinque*

```
from qubot import Qubot, QubotConfigTerminalInfo,
    QubotConfigModelParameters, QubotDriverParameters,
    QubotPresetRewardFunc

qb = Qubot(
    url_to_test="https://upmed-starmen.web.app/",
    terminal_info_testing=QubotConfigTerminalInfo(
        terminal_ids=[],
        terminal_classes=["SignIn_login_hcp__qYuvP"],
        terminal_contains_text=[],
    ),
    terminal_info_training=QubotConfigTerminalInfo(
        terminal_ids=[],
        terminal_classes=[],
        terminal_contains_text=["Log in"],
    ),
    driver_params=QubotDriverParameters(
        use_cache=False,
        max_urls=10,
    ),
    model_params=QubotConfigModelParameters(
        alpha=0.5,
        gamma=0.6,
        epsilon=1,
        decay=0.01,
        train_episodes=1000,
        test_episodes=100,
        step_limit=100,
    ),
    reward_func=QubotPresetRewardFunc.ENCOURAGE_EXPLORATION,
    input_values={
        "color": "#000000",
        "date": "2021-01-01",
        "datetime-local": "2021-01-01T01:00",
        "email": "johndoe@gmail.com",
        "month": "2021-01",
        "number": "1",
        "password": "p@ssw0rd",
        "search": "query",
        "tel": "123-456-7890",
        "text": "text",
        "time":  "00:00:00.00",
        "url": "https://www.google.com/",
        "week": "2021-W01"
    }
)
qb.run()
print(qb.get_stats())
```

FIG. 3: Sample Qubot Programmatic Configuration

2), as shown in Figure 4. A full example of a Qubot configuration file can be found in the project's GitHub Repository. [4]

```
from qubot import Qubot

qb = Qubot.from_file('./qu_config.json')
qb.run()
print(qb.get_stats())
```

FIG. 4: Sample Qubot Programmatic Configuration w/ JSON Configuration

Lastly, Qubot can be run purely via the command line, and output a statistics file at the end of the test (a.k.a. *Techinque 3*). Since the command-line utility is automatically installed via `pip`, the tester must only supply the path of the configuration file. A valid command is shown in Figure 5.

```
qubot ./qu_config.json
```

FIG. 5: Sample Qubot Command-Line Operation

5. *Stats Class*

The **Stats Class** is a custom class that provides a versatile way to keep track of discrete, temporal, and continuous statistics such as training and testing times, as well as accumulated rewards. When invoked in the command line, Qubot will automatically generate an output JSON file with statistics from the last test.

6. *Utils Package*

The **Utils Package** contains a variety of methods used to fulfill miscellaneous tasks such as input generation and error checking.

C. Partitioning the Work

Among the authors, Anthony was responsible for architecting, developing, and pushing into production (on PyPi) the Qubot framework, whereas Kenneth was responsible for performing the field study and developer experience study, as well as for extending the Qubot GitHub repository with test results.

III. EXPERIMENTS

A. Field Testing

To test Qubot, we used 10 different websites of varying complexity, ranging from a simple static site generated resume page to a full-fledged homepage for a famous museum. In each case, a new Qubot instance was created and the WUT was webscraped in order to obtain a `UITree`. The objectives and parameters for each test are listed in the following subsections. Since there are no dependable heuristics for choosing the aforementioned parameters, various sets of values were tried for each WUT before being finalized.

1. *Field Test 1. Resume Website*

The objective of Field Test 1 was to navigate to the individual's full name at the top of page during the training iterations and then to navigate to the "Typography" header during the testing iterations, as shown in Figure 6.

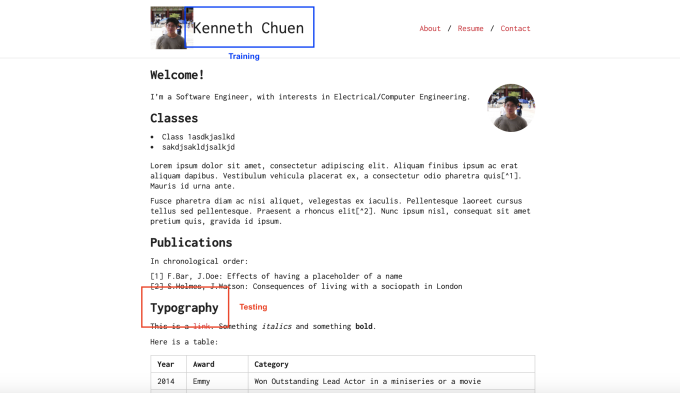

FIG. 6: Field Test 1 Objectives

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 7.

Parameter	*Value*
α	0.8
γ	0.8
ϵ	0.1
ϵ-decay	0.0001
`step_limit`	100
Training Episodes	1000
Testing Episodes	100

FIG. 7: Field Test 1 Parameters

2. *Field Test 2. Club Website 1*

The objective of Field Test 2 was to navigate to the "About Us" header during the training iterations, as shown in Figure 8, and then to navigate to the "Public Practice Schedule 2019-20" header on the same page during the testing iterations, as shown in Figure 9.

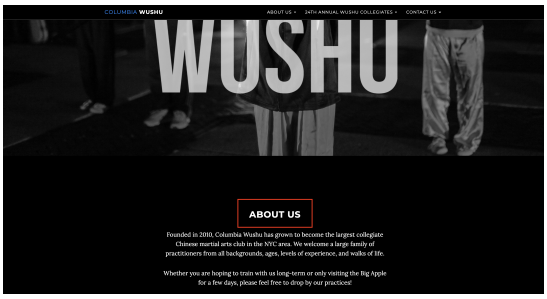

FIG. 8: Field Test 2 Training Objective

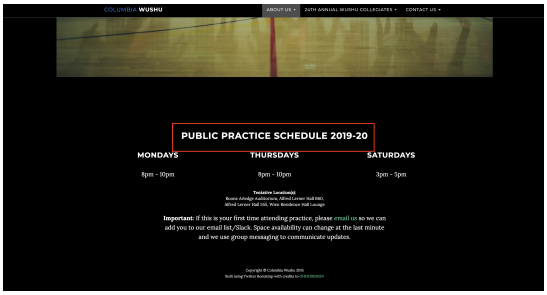

FIG. 9: Field Test 2 Testing Objective

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 10.

Parameter	*Value*
α	0.6
γ	0.8
ϵ	0.1
ϵ-decay	0.001
`step_limit`	200
Training Episodes	2000
Testing Episodes	2000

FIG. 10: Field Test 2 Parameters

3. *Field Test 3. Club Website 2*

The objective of Field Test 3 was to navigate to the "Learn to Code" copy during the training iterations, as shown in Figure 11, and then to navigate to the footer on the same page during the testing iterations, as shown in Figure 12.

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 13.

4. *Field Test 4. Product Landing Page*

The objective of Field Test 4 was to navigate to demo GIF during the training iterations, as shown in Figure 14, and then to navigate to the testimonial header on the same page during the testing iterations, as shown in Figure 15.

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 16.

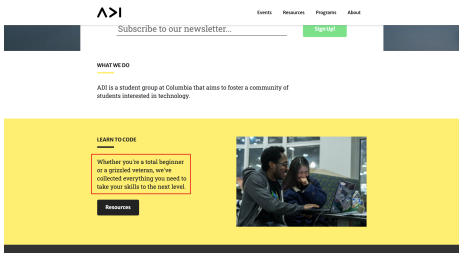

FIG. 11: Field Test 3 Training Objective

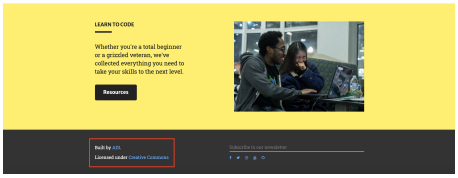

FIG. 12: Field Test 3 Testing Objective

Parameter	*Value*
α	0.9
γ	0.9
ϵ	0.1
ϵ-decay	0.01
`step_limit`	100
Training Episodes	2000
Testing Episodes	2000

FIG. 13: Field Test 3 Parameters

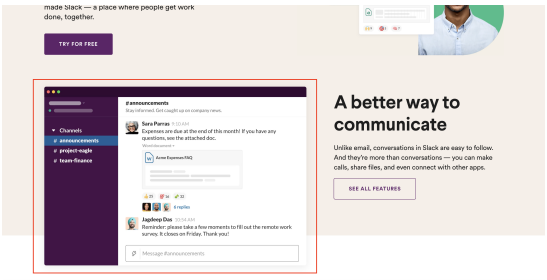

FIG. 14: Field Test 4 Training Objective

5. *Field Test 5. Museum Website*

The objective of Field Test 5 was to navigate to "Reserve Tickets" button during the training iterations, as shown in Figure 17, and then to navigate to the "Register" button on a new page during the testing iterations, as shown in Figure 18.

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 19.

6. *Field Test 6. Single Page Blog*

The objective of Field Test 6 was to navigate to the "Setup the gym" header during the training iterations, as shown in Figure 20, and then to navigate to the "References" header on the same page during the testing iterations, as shown in Figure 21.

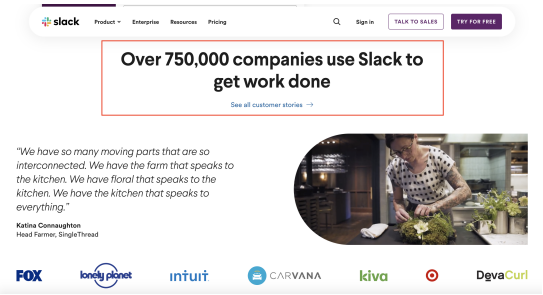

FIG. 15: Field Test 4 Testing Objective

Parameter	*Value*
α	0.6
γ	0.8
ϵ	0.1
ϵ-decay	0.001
`step_limit`	200
Training Episodes	2000
Testing Episodes	2000

FIG. 16: Field Test 4 Parameters

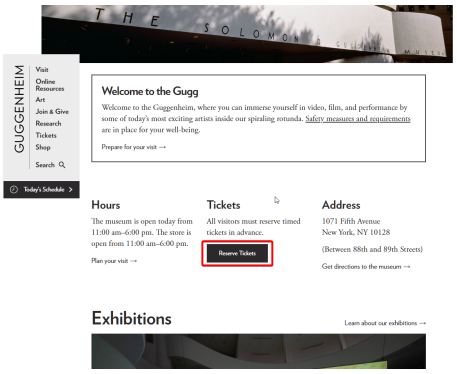

FIG. 17: Field Test 5 Training Objective

The Q-Learning parameters used by the agent in

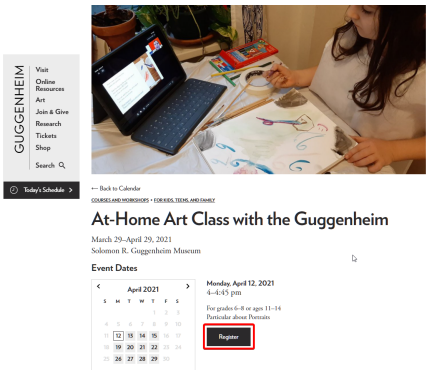

FIG. 18: Field Test 5 Testing Objective

Parameter	*Value*
α	0.8
γ	0.8
ϵ	0.9
ϵ-decay	0.0001
`step_limit`	1000
Training Episodes	1000
Testing Episodes	100

FIG. 19: Field Test 5 Parameters

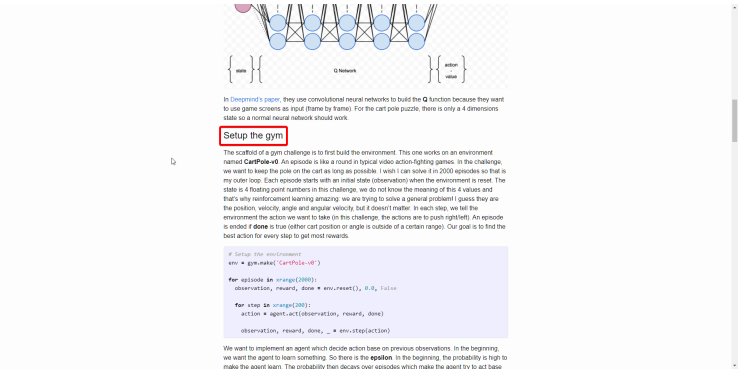

FIG. 20: Field Test 6 Training Objective

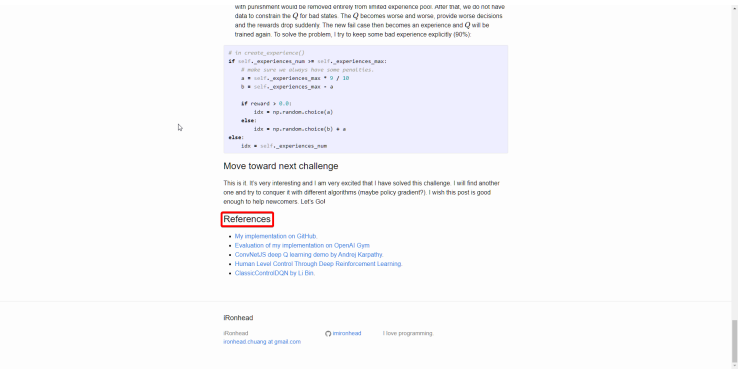

FIG. 21: Field Test 6 Testing Objective

traversing the built `UITree` are shown in Figure 22.

Parameter	*Value*
α	0.8
γ	0.8
ϵ	0.1
ϵ-decay	0.0001
`step_limit`	50
Training Episodes	1000
Testing Episodes	100

FIG. 22: Field Test 6 Parameters

7. *Field Test 7. Course Website*

The objective of Field Test 7 was to navigate to the "Reference material" bullet point during the training iterations, as shown in Figure 23, and then to navigate to the database setup instructions on a different page during the testing iterations, as shown in Figure 24.

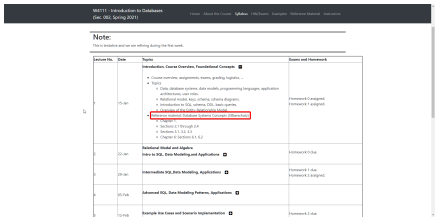

FIG. 23: Field Test 7 Training Objective

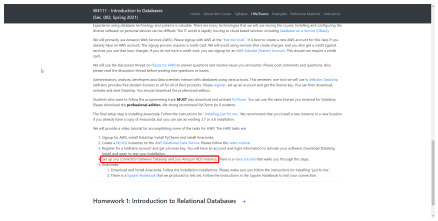

FIG. 24: Field Test 7 Testing Objective

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 25.

8. *Field Test 8. Nonprofit Website*

The objective of Field Test 8 was to navigate to the name of a student under a blurb during the training iterations, as shown in Figure 26, and then to navigate to the "COI Policies and Resources" header on a different page during the testing iterations, as shown in Figure 27.

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 28.

9. *Field Test 9. Climate Change Website*

The objective of Field Test 9 was to navigate to information on Afghanistan during the training iterations, as shown in Figure 29, and then to navigate to the information on Uganda on a different page during the testing iterations, as shown in Figure 30.

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 31.

Parameter	*Value*
α	0.9
γ	0.9
ϵ	0.9
ϵ-decay	10
`step_limit`	1000
Training Episodes	100
Testing Episodes	10

FIG. 25: Field Test 7 Parameters

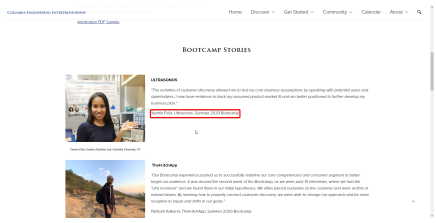

FIG. 26: Field Test 8 Training Objective

FIG. 27: Field Test 8 Testing Objective

Parameter	*Value*
α	0.8
γ	0.8
ϵ	0.9
ϵ-decay	0.0001
`step_limit`	1000
Training Episodes	1000
Testing Episodes	100

FIG. 28: Field Test 8 Parameters

10. *Field Test 10. Famous Person's Website*

The objective of Field Test 10 was to the "Biographies" header during the training iterations, as shown in Figure 32, and then to the all-caps disclaimer on the same page during the testing iterations, as shown in Figure 33.

The Q-Learning parameters used by the agent in traversing the built `UITree` are shown in Figure 34.

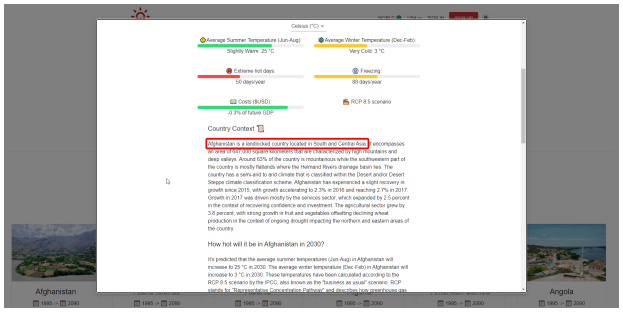

FIG. 29: Field Test 9 Training Objective

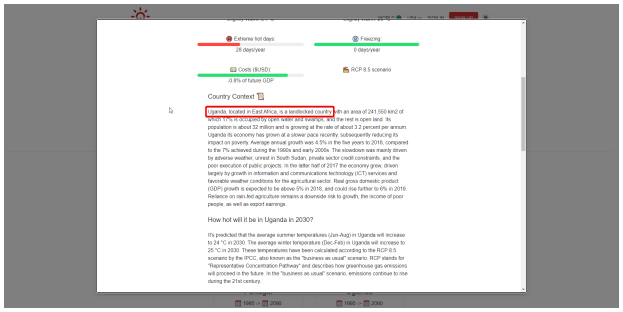

FIG. 30: Field Test 9 Testing Objective

Parameter	*Value*
α	0.8
γ	0.8
ϵ	0.9
ϵ-decay	0.0001
`step_limit`	1000
Training Episodes	1000
Testing Episodes	100

FIG. 31: Field Test 9 Parameters

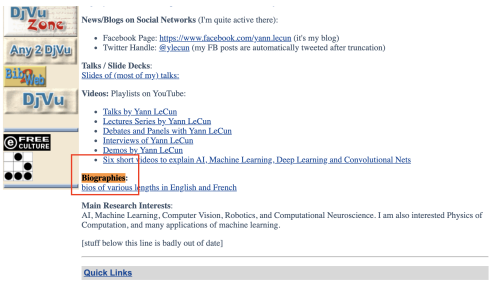

FIG. 32: Field Test 10 Training Objective

B. Developer Experience Study

Three of our peers who have various levels of experience in software development and testing were chosen to partake in the developer experience study over video conferencing. First, they were asked to thoroughly read the `README.md` file in the GitHub repository for Qubot until they felt comfortable with installing and using the library.

Second, they were asked to run Qubot on the `https://upmed-starmen.web.app/` site, which is the site on which Qubot was tested in the previous paper, in order to find the "Get Started" button. As they completed this task, we took note of their approaches as they were timed, and then asked them subjective questions based on their experiences with Qubot.

The metrics obtained in this test are described in the next section.

IV. RESULTS

A. Field Testing Results

The results from our ten field tests are summarized Figure 35, and they were, unfortunately, not very promising.

Qubot evidently accrued the most amount of rewards during training on the simple one-page resume website (Field Test 1), as shown in Figure 36. However, it consistently plummeted to under $-100,000$ in training penalties across most of the tests, as shown in Figure 37

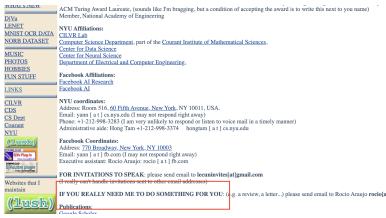

FIG. 33: Field Test 10 Testing Objective

Parameter	*Value*
α	0.8
γ	0.8
ϵ	0.9
ϵ-decay	0.0001
`step_limit`	200
Training Episodes	1000
Testing Episodes	1000

FIG. 34: Field Test 10 Parameters

for Field Test 1. From this, we can conclude that Qubot is unable to generalize what it had learned during the training episodes and merely induced penalties during testing episodes for each of the tests that ran successfully.

Many tests did not run successfully due to issues with the webscraping script (see the "Error" column), which lost references to elements and threw test-terminating errors in every case.

It was also found that Qubot usually converged to a 99% exploitation, as opposed to an exploration, probability, often after failing to navigate to a terminal node within the first 1000+ training episodes. This is largely due to the fact that Qubot cannot simulate complicated mouse movements such as hovering over elements, resulting in hundreds of unexplored HTML tags.

The rate of rewards accrued across the ten tests is a proxy for the rate at which Qubot performs successful website navigation to a target element, and so it is fair to say that Qubot was not very *intelligent* in its attempt to reach a terminal state. All tests that did not result in a webscraping error, besides the first and second tests, manifested reward accrual trends such as in 39 and 39.

Test	*Train Rewards*	*Test Rewards*	*Test Penalties*	*Error?*
1	14009953	−111663	219263	No
2	−18734324	−191093009	193530209	No
3	−96793143	−96698876	96698876	No
4	N/A	N/A	N/A	Yes
5	−249807545	−2520917	2520917	No
6	−12512500	−126250	126250	No
7	−2474098	−26691	27241	No
8	N/A	N/A	N/A	Yes
9	N/A	N/A	N/A	Yes
10	N/A	N/A	N/A	Yes

FIG. 35: Field Test Results

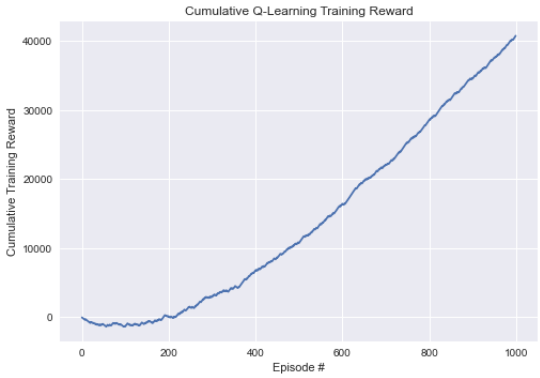

FIG. 36: Example Training Reward Accrual (Field Test 1)

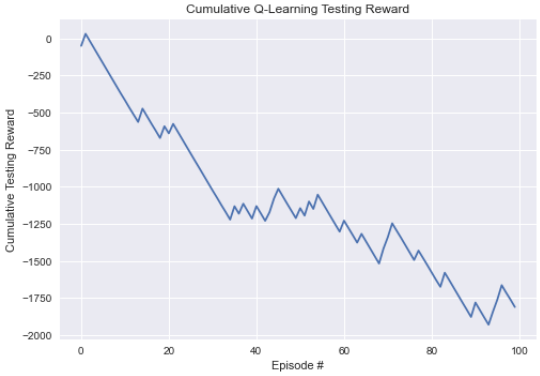

FIG. 37: Example Testing Reward Accrual (Field Test 1)

FIG. 38: Example Unsuccessful Training Reward Accrual (Field Test 3)

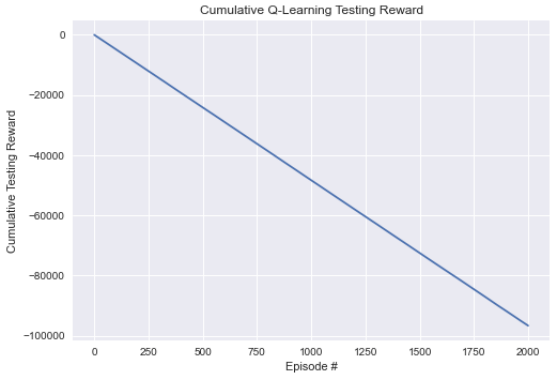

FIG. 39: Example Unsuccessful Testing Reward Accrual (Field Test 3)

Finally, despite Qubot's ability to detect crashes during the webscraping process, we failed to detect any crashes over these ten tests, and so it is inconclusive as to whether or not Qubot's crash detection function is *feasible*.

The results from each of these tests may be viewed and reproduced by running the `tests/field_tests.ipynb` file in the Qubot GitHub repository.

B. Developer Experiences and Reactions

The table in Figure 40 shows the results from asking three participants, with obfuscated names, to find the "Get Started" button using Qubot on the `https://upmed-starmen.web.app/` site.

Tester	*Time (s)*	*Technique*	*Failed?*	*Reason*
Joe E.	312	3	No	N/A
Deb B.	598	3	No	N/A
John Q.	866	3	Yes	`GeckoDriver` issue

FIG. 40: Field Test Results

Joe and Deb, who successfully found the "Get Started" button, were able to do so within 10 minutes, including installing the Qubot library. John, however, failed to complete the test at all because the `GeckoDriver` dependency used by Selenium Webdriver could not be installed nor found on his system.

Next, it was found that *Technique 3*, which involves running Qubot in the command-line, was favored among all the testers. According to Deb, "it seemed like the fastest [approach] and I forgot how to run a Python script."

We examined the JSON configuration files each individual used in their tests and noticed that all of them lacked both a driver and model parameters. According to Joe, who has a background in Machine Learning, he had "no idea what each of the model parameters do in this context nor what their right values are, so [he] just left them blank." Deb and John, however, admitted that they were simply racing against the clock and chose to omit as much as they can from the configuration file just to get the test to run.

The raw responses from this study may be found in `tests/experience_study.xlsx`.

V. CONCLUSION

A. Challenges

Developing and testing this framework within a one-month span while juggling other curricular activities came with an abundance of challenges. Primarily, performing ten field tests with a buggy webscraping engine was the most time consuming task of all. Constructing `UITree`s for certain websites took between ten minutes to an indefinite amount of time (due to cyclical links), which prevented many websites from being tested.

We had originally planned to perform a third test in which we compared Qubot's performance to that of Selenium-AI, a small repository found on GitHub that also performs exploratory testing using Selenium Webdriver. However, due to a lack of documentation on setting up this platform and an outdated Docker image containing the source code to Selenium-AI, we were unable to continue with the comparison. Spending a surplus amount of time attempting to perform this comparison also impacted our ability to perform the ten field tests and developer experience study.

Lastly, choosing the correct parameters for Qubot's Q-learning agent was yet another difficulty we came across. It was found, eventually, that values of α and γ around 0.9 were most performant, and that large `step_limit` values yielded generally better results. This later seemed obvious, as larger step limits give more time for the agent to find the terminating elements and to accrue rewards.

B. Threats to Validity

Threats to validity in this paper that have already been mentioned include the small set of WUTs used in the field tests, the small sample size of developers who have tested the Qubot library, and the lack of parameter optimization.

C. Future Work

In the future, we first hope to extend the tests presented in this paper to include crash tests, wherein Qubot's performance on volatile websites may be assessed. Moreover, we hope to optimize the model parameters chosen prior to testing and to, perhaps, develop a heuristic for choosing parameters in the future.

If we were given a chance to reconstruct the developer experience study in the future, we would tweak is such that time is no longer a constraint, as Deb and John mentioned that being timed led them to use fewer of Qubot's available features.

Finally, on a larger scale, we hope to improve Qubot's webscraping capabilities so to minimize crashes and to speed up `UITree` creation, as well as to include more thorough documentation on its usage in the `README.md` file.

D. Notes from Authors

1. *Anthony's Self-Evaluation*

I am thankful for having been able to write a comprehensive paper comparing exploratory testing techniques and then to culminate my findings into the open source Qubot project described in this paper. The biggest challenges I have personally encountered over the course of my midterm paper and this final project include the amount of hours painstakingly poured into developing a webscraper that produces `UITree`s as well as implementing the OpenAI Gym environment from

scratch. It was a struggle to get Qubot to work on multi-page sites, and I would like to continue improving Qubot until it works consistently across WUTs.

I have learned an abundance of new concepts and ideas throughout my work on Qubot, from state-of-the-art exploratory testing models in literature to parsing DOM elements in Selenium, interfacing with OpenAI, and publishing command-line tools to PyPi. Most importantly, however, I improved in my ability to work as a team to fulfill both the development and the testing parts of this project. Hopefully, I will get more chances to work with others on Qubot as this open-source library starts to receive pull requests from other GitHub users.

2. Kenneth's Self-Evaluation

This project (and the course overall) exposed me to applications of machine learning in web development, although this statement says more about my exposure to web development compared to machine learning. Working with Anthony on Qubot gave me valuable insights on both the user and developer side of web development.

The greatest challenge I faced was resolving the **GeckoDriver** issue (missing from PATH) on my personal computer. I greatly sympathize with John Q., and can now truly appreciate *anaconda* and other environment based utilities. The next most difficult task was creating navigatable tests containing input fields. As Qubot identifies input fields by the *type* tag, any fields that lack a corresponding tag (e.g email) cannot be filled in. Qubot is also not capable of interacting with radio buttons.

In creating the tests, I learned how to use Jupyter notebook to write portable tests. From the user study, I now feel more comfortable with collecting user feedback in terms of the process and data collection. As a developer and tester, working on Qubot helped me realize why there are so many cries for standardized best practices.

VI. REFERENCES

[1] Fowler, M. 18 November 2019. *Exploratory Testing.* `https://martinfowler.com/bliki/ExploratoryTesting.html`

[2] Hanna, M, Aboutabl, A.E, Mostafa, M.M. 2018. *Automated Software Testing Framework for Web Applications.* International Journal of Applied Engineering Research (2018). Research India Publications. `http://www.ripublication.com/ijaer18/ijaerv13n11_141.pdf`

[3] Harries et. al. 16 July 2020. *DRIFT: Deep Reinforcement Learning for Functional Software Testing.* 33rd Deep Reinforcement Learning Workshop (NeurIPS 2019). Vancouver, Canada. Microsoft. `https://www.microsoft.com/en-us/research/publication/drift-deep-reinforcement-learning-for-functional-software-testing/`

[4] Krivonos, Anthony (Author). 13 February 2021. *Qubot*, `experiments.ipynb`, GitHub repository. `https://github.com/anthonykrivonos/qubot/blob/main/experiments.ipynb`

[5] Krivonos, Anthony. 23 February 2021. *Qubot: Designing a Bot to Perform Autonomous Black Box Testing.* COMS E6156 Topics in Software Engineering. Columbia University.

[6] Moore, G. E. September 2006. *Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.* IEEE Solid-State Circuits Society Newsletter, vol. 11, no. 3, pp. 33-35. `https://ieeexplore.ieee.org/abstract/document/4785860`

[7] *Numpy.* 14 April 2021. NumPy. `https://numpy.org/`

[8] *OpenAI Gym.* 14 April 2021. OpenAI. `https://gym.openai.com/`

[9] *PyPi.* 14 April 2021. Python Software Foundation. `https://pypi.org/`

[10] *Selenium WebDriver.* 20 February 2021. The Selenium Browser Automation Project. `https://www.selenium.dev/documentation/en/webdriver/`

[11] Sutton, R.S., Barto, A.G. 04 January 2005. *Reinforcement Learning: An Introduction.* The MIT Press. Cambridge, Massachusetts. `http://incompleteideas.net/sutton/book/ebook/the-book.html`