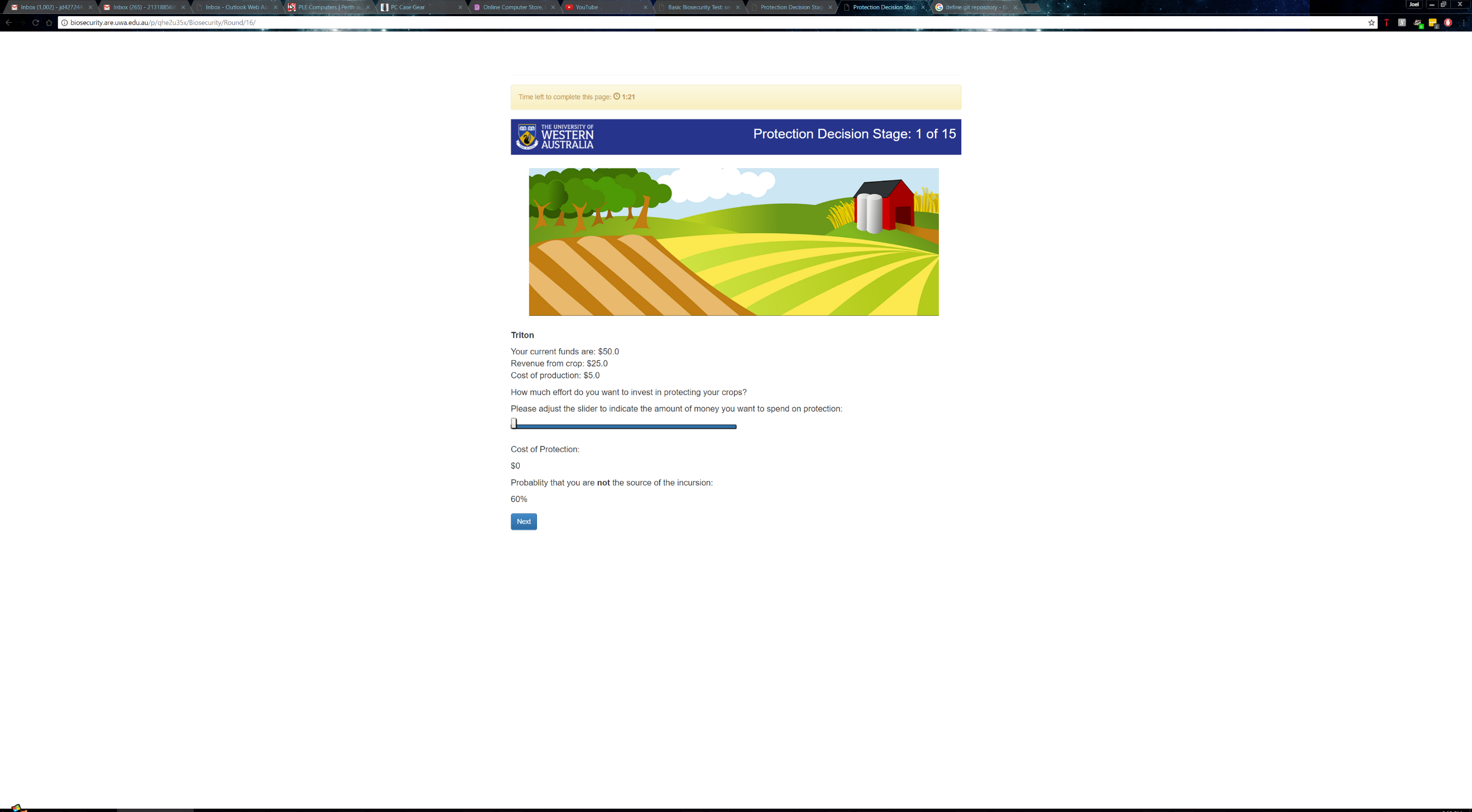
Testing Document  
*UWA Biosecurity Game*  
By: Joel Dunstan  
Personal: [jd427244@gmail.com](mailto:jd427244@gmail.com?subject=Test%20Document%20Question)  
Work: [joel.dunstan@uwa.edu.au](mailto:joel.dunstan@uwa.edu.au?subject=Test%20Document%20Question)  
<biosecurity.are.uwa.edu.au>



**Table of Contents**

[Testing Strategy 3](#_Toc481597039)

[Automated Testing 3](#_Toc481597040)

[Manual and Server Testing 5](#_Toc481597041)

[Test 1: The Frequency of Outbreaks 6](#_Toc481597042)

[How an Outbreak is Determined 6](#_Toc481597043)

[Test Specification 7](#_Toc481597044)

[Test Description 7](#_Toc481597045)

[Test Analysis Report 9](#_Toc481597046)

[The Average Frequency of Outbreaks 9](#_Toc481597047)

[Mode - Frequency of Outbreaks 9](#_Toc481597048)

[Average Funds at Round 5 - Frequency of Outbreaks 10](#_Toc481597049)

[Average Funds at Round 15 - Frequency of Outbreaks 10](#_Toc481597050)

[Test 2 – Automated Command Line Testing 12](#_Toc481597051)

[Test Specification 12](#_Toc481597052)

[Test Description 12](#_Toc481597053)

[Test Analysis Report 13](#_Toc481597054)

[Test 3 – Automated Browser Bot Testing 17](#_Toc481597055)

[Test Specification 17](#_Toc481597056)

[Test Description 17](#_Toc481597057)

[Test Analysis Report 20](#_Toc481597058)

[Windows 10 – Automated Browser Bot Test 20](#_Toc481597059)

[Android 7.1.1 – Automated Browser Bot Test 20](#_Toc481597060)

[Test 4 – Manual Browser Testing 22](#_Toc481597061)

[Test Specification 22](#_Toc481597062)

[Test Description 22](#_Toc481597063)

[Test Analysis Report 26](#_Toc481597064)

[Windows 10 - Manual Test 26](#_Toc481597065)

[Android 7.1.1 - Manual Test 27](#_Toc481597066)

[iOS 10.2.1 - Manual Test 28](#_Toc481597067)

[Test 5 – Server Performance 29](#_Toc481597068)

[Test Specification 29](#_Toc481597069)

[Test Description 29](#_Toc481597070)

[Test Analysis Report 31](#_Toc481597071)

[Server – Time Benchmark 31](#_Toc481597072)

[CPU Utilization 32](#_Toc481597073)

[Test Materials 33](#_Toc481597074)

[oTree\_Tags 34](#_Toc481597075)

[Glossary 35](#_Toc481597076)

[Documents/Spreadsheets for Reference 37](#_Toc481597077)

[References 38](#_Toc481597078)

**Objectives and Test Summary**

This document is focusing on the testing that was performed to ensure that the system and all its aspects were working as expected. This document will primarily focus on these aspects:

1) How outbreaks are determined and the amount of times outbreaks occur.

2) The general usability and functionality of all the pages and how they work together.

3) The websites compatibility across systems, web browsers especially due to its use of HTML5 specific code used within the webpages, and its use of Django, a Python framework.

4) The performance of the server, its backup strategies and recovery plans (if any can be done). Focusing on how many users the server can handle and creating a session and running a session with many participants, 100 being the acceptable amount.

5) The export of the data from each game for experimental purposes and its readability.

6) The full use of all these aspects all working together in one cohesive, usable system even by users who aren’t fully familiar with python programming

**Testing Strategy**

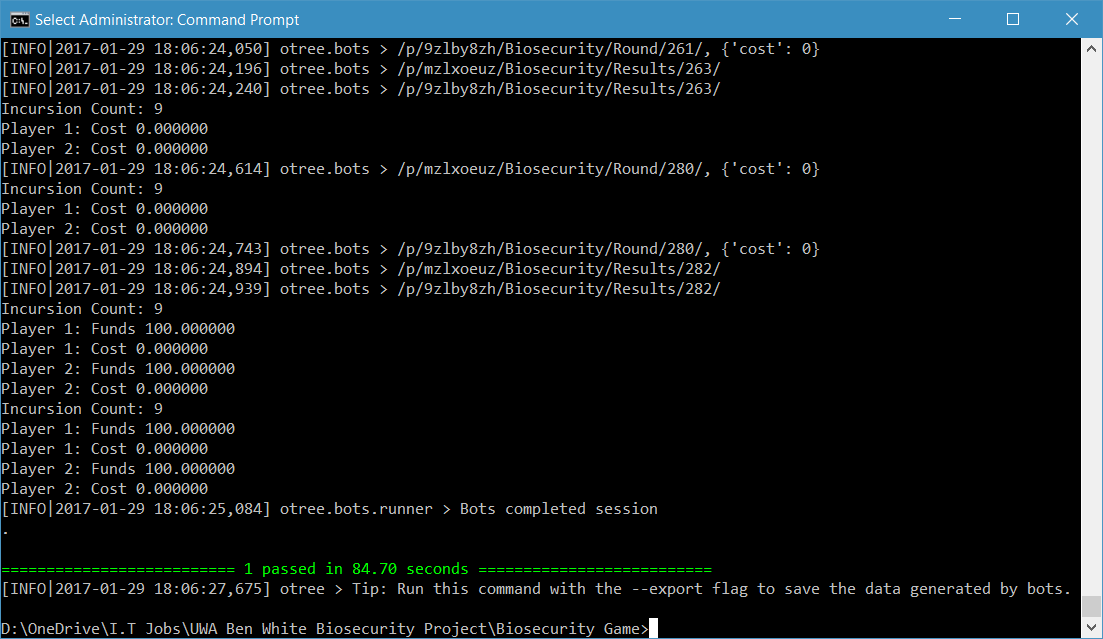
The testing strategy will be divided into two areas depending on what needs to be tested. One being automated testing which will be primarily used for software testing, however automated testing will be used when testing the server’s capabilities as browser bots can be used to mimic user behaviour on the website.

The overall strategy is a bottom-up approach, first testing the bare code itself using the command line (Refer to [Automated Testing](#_Automated_Testing) for more detail). From here I will move on to using browser bots to ensure that the logic works within a web browser and on different devices. Finally, I will conclude with running the game manually myself in each web browser on different devices to decide what are the most recommended web browsers and devices for this game.

### Automated Testing

Automated testing will be used in just about all areas of testing. The automated tests are run from code inside tests.py from every application inside the main oTree project directory. The project is divided into three applications, the Lottery Game, Biosecurity, and Results application. These tests can be run inside the command line using the command “otree test <session\_config\_name>” using numbers as arguments to conduct tests with a different number of bots. An example of such a test would be “*otree test basic\_biosecurity\_test 4”* which means that the command line will use the test.py from the Biosecurity application and run the session configuration ‘basic\_biosecurity\_test’ using 4 bots.

The automated tests do verification checks to see if the game produces the correct results given costs, player approvals, etc. If the game completes a test, this means the test ran without generating an AssertionError (this occurs whenever an assert statement failed, which is designed to validate functions, equations etc.) and without failing to complete every page with a form. In the case of a test being completed successfully then this will appear on the command line at the end of the test:



For the Lottery application, automated testing will always validate the control questionnaire for testing people’s understanding of the game is working correctly. The testing does this by inputting the incorrect answers for every question to see if it will fail, and then finally inputting the correct answers once the SubmissionMustFail (an assert statement that ensures that an input is invalid, a wrong answer to a question in the case) assert statements have been completed. For the play panels, the test randomly chooses A or B for the forms to make sure the data goes in through the form. Since there is no tangible way to verify that a person gets an exact result due to the random functions used to determine which game is the winning game in the Lottery application, no validation is done on the result.

Focusing on the Biosecurity Game, automated testing conducts itself using different scenarios. In each scenario, the test uses assert statements to ensure that the equations used in the game outputs the required result based on certain controlled inputs, i.e. when a player puts $0 as their cost for protection that round that the probability that they are **not** the source of the incursion is 60% or 0.6. To be specific the values that are tested throughout the game are the costs of protection for every single round, the funds at the end of each round, any group pledges that were performed, and finally the approvals by players. The automated tests conduct scenarios in the order mentioned below, which each scenario will show the values used for each of the values being tested above:

Random: A value is chosen between 0 and Max Protection for the cost of protection, funds are calculated and verified based on whether there was an outbreak or not. Due to the randomness of the frequency of an outbreak, funds will not be mentioned further when discussing the scenarios. Group pledging are always done randomly and thus will not be discussed again when describing the scenarios. Approvals of players are completely random.

Quarter: 25% Max Protection for the cost of protection and the approvals are always -6.

Half: 50% Max Protection for the cost of protection and approvals are always 3.

Three-Quarters: 75% Max Protection for the cost of protection and all approvals are always 6.

Full: Max Protection for the cost of protection, and approvals are always 0.

Half0HalfFull: Half of the bots do Max Protection, while the other half do no protection whatsoever and all approvals are random.

Bankrupt: No protection at all and all approvals are random.

There are default values used for every test:

* The maximum amount of protection that can be used is always 10
* The cost of upkeep is always 5
* The revenue is always 25
* The starting amount for every participant is 50
* The minimum amount of chance that someone is not the source of the outbreak is 60% (refer to [Test 1: The Frequency of Outbreaks](#_Test_1:_The))
* The number of participants per group is 4, and the number of participants is 4.
* A pledging round comes every 3 rounds, when pledging is turned on
* An approval by contribution takes place every round, when turned on.

After the bots complete the Biosecurity application they move on to the Results application. The Results application has a survey to which each bot automatically completes and just waits for correct pages to come forth until it has no more pages to complete.

One final note for automated testing is that the same code that performs the command line testing also performs the browser bot testing, where bots test the game inside a browser. Thus, when a bot completes a test within a browser, you can assume that the game works inside that browser.

### Manual and Server Testing

In contrast to automated testing, manual testing will involve running the game myself instead of bots completing the game. Manual testing will comprise of verifying the appearance of forms, buttons and the general usability of the site on different browsers and devices. As a final part of testing, manual testing will also reaffirm what the automated testing observed as I will be testing the same inputs and outputs that the bots used to complete the tests. On top of this, it will also confirm that oTree’s administration aspects work with the game as expected as well. Manual testing and the automated testing together provide a comprehensive software test of the game itself and its compatibility with many different browsers and devices.

In comparison to the software, server testing focuses on how the server performs under load. To put strain on the server, I will conduct tests with automated testing and a substantial number of participants. Doing this will mimic how the server will behave when so many participants are all on the server playing each game all at once. It’s here that changes to the amount of resources the server has available will be documented. No other server testing is required here, as I have no good means of backing up the database that will contain the data for all the sessions on a separate machine to the server, hence testing of the backup solutions will be restricted to getting any backups stored on the server itself.

## Test 1: The Frequency of Outbreaks

The first version of the Biosecurity application took an average of everyone’s protection and then generated a random number, and if the random number was higher than this, then an outbreak would occur. However, in this iteration, we now take a joint probability, this meant that calibration was needed to see the frequency of outbreaks and how participants would fare in different conditions depending on how they decided to play. In an ideal game where everyone was cooperative, there would be at least 3-5 outbreaks with participants gaining a profit. In contrast, if participants weren’t cooperating then there would be at least 8-15 outbreaks depending on how uncooperative they were and a loss from the starting amount of $25 (Later changed to $50 for ethical reasons, so participants had less chance of losing money), with the potential of a small bankruptcy (up to $10 in debt). This test involved changing the maximum amount of protection against biosecurity threats a player could provide in each round and the probability coefficient, or the minimum probability that a player is **not** the source of the outbreak. Before describing how I conducted the test, one should know how an outbreak is determined, and what happens when an outbreak occurs.

### **How an Outbreak is Determined**

**MP** = **Max Protection =** (The maximum amount of protection a participant can put against biosecurity threats)  
  
**PC =** **Probability Coefficient =** (Essentially we moved the graph up so that $0 worth of protection doesn’t result in a 100% chance that someone is the source of the outbreak, e.g. if PC = 0.4, then there is a 40% chance that someone is **not** the source of the outbreak, or 60% chance that they’re the source of the outbreak)  
  
**C = Amount of protection a player put in that round** ($0 ≤ P ≤ MP)  
  
**P1, P2, P3, P4 = The probability that a player (4 players in this scenario) is NOT the source of the outbreak** (0 ≤ P1, P2, P3, P4 ≤ 0.9995). Will be denoted as **Px** when referring to the function for P1-4.

**RAND =** A random number between 0 and 1 generated by the server

**U =** The cost of the crops per round  
  
**R =** Revenue which a player receives when there is no outbreak  
   
1) Get the Cost Factor:

**CF =**

2)Determine the probability that a player is the source of the outbreak:

**Px =**3)Determine the probability of no outbreak for the group (the set of players containing P1­ – P4):

**Probability of No Outbreak (PNO) = P1** x**P2** x**P3** x**P4**

4) Determine the probability of an outbreak for the group:

**Probability of an Outbreak (PO) = 1 – PNO**

5) Now generate a random number between 0 and 1 and follow the inequalities below:

**if PO > RAND, Outbreak Occurs**

**if PO < RAND, No Outbreak Occurs**

6) Now determine each player’s profits or losses as per the following functions:

If there was an Outbreak, then: **Current Funds = Current Funds – C – U**

If there was no Outbreak, then: **Current Funds = Current Funds – C – U + R**

### Test Specification

In this test, we require seeing that the group’s actions proportionally affect how much the group the gains from the game. That means:

*As a group if each player is cooperative, then we must see that on average, each player should receive a sizable profit compared with starting amount, while still having outbreaks to prevent a lack of engagement from the game being too easy and the result being independent to the group’s actions.*

*Inversely, as a group if each player is uncooperative, then we must see that on average, each player should receive a loss from the game compared with the starting amount, while still allowing players to ideally to finish with an amount more than $0 to prevent lack of motivation and engagement in the game***.**

It is worth mentioning too, that while the above is a requirement, it won’t always be met due to the nature of joint probability and some of the inherent randomness in the game. The game should allow cases where players are *cooperative* and outbreaks still occur often. Inversely, the game should also allow cases where players are *uncooperative* and outbreaks don’t occur often. Such cases in this game should be rare though to get the best data possible, as the player’s actions must determine the outcome as much as possible despite the randomness that exists in the game.

### Test Description

To perform this test, you will need a terminal (CMD, Bash) where you will be able to perform the command line tests and export the data to an accompanying file (ideally, text, csv or some other file that’s human readable). The operating system you use, shouldn’t matter if you have a met these requirements (at the time of writing 02/04/2017):

* oTree 1.2.8 (or above)
* Postgres 9.6.1 (or above)
* otreechat (latest version)
* psycopg2 (latest version)
* Python 3.6 (or above)
* Have followed the instructions for oTree to use Postgres as per this page:  
  <http://otree.readthedocs.io/en/latest/server/ubuntu.html#database-postgres>
* Have followed the instructions for [oTree\_Tags](#_oTree_Tags)

Postgres here isn’t completely necessary to complete these tests, you could use MySQL, MSQL (Microsoft SQL) and many others, you must ensure it’s a Relational Database Management System that can withstand multiple concurrent transactions without issue.

In the terminal (in my case PowerShell), you’ll need to put the following command to perform the test 20 times for the sake of getting a reasonable sample size thus reducing the average being affected by outliers like those rare times when players do well despite being uncooperative or vice versa.

for($i=1; $i -le 20; $i++) {

otree test basic\_biosecurity\_test 4 > “Path\To\File\file<i>.txt”

}  
  
This will run the basic biosecurity test 20 times which includes all the different case scenarios which were as follows (and in the order, they will be performed):

* Random
* Quarter
* Half
* Three-Quarters
* Full
* Half0HalfFull
* Bankrupt

For more details on what these scenarios include refer to [Automated Testing](#_Automated_Testing). The above code will also produce text files which you will need to record the incursion count and the funds each player has at round 5 and the end of the game (round 15). How you the reader record these, I will leave to you, my method was to record the data into Excel spreadsheets which automatically calculated the average and mode of outbreaks (denoted as incursions in my documents as there was a change of terminology) and funds at rounds 5 and 15.

While you may not produce the exact same results due to the randomness of the game and the changes that were made while testing the frequency of outbreaks, you should be able to make the same conclusion as I did below.

### Test Analysis Report

In the tests, there was two values that changed throughout the tests which were:

1. The minimum chance that a player wasn’t the source of the incursion, or the **Probability Coefficient (PC)** as per [How an Outbreak is Determined](#_How_an_Outbreak).
2. The Maximum amount of protection or effort that a player can put against biosecurity threats, or the **Maximum Protection (MP)** as per [How an Outbreak is Determined](#_How_an_Outbreak).

The reason that these values were chosen is because these values adjust the frequency of outbreaks and how much funds each participant finished with more than any other values. For example, one could adjust the revenue, however $25 is a fair value a participant can gain and just as easily lose with carelessness, and a change in this value dramatically changes how the games works. In this test, we were looking for more finer calibration such as the calibration that took place in adjusting the **PC** and **MP**.

The defaults across all games were as follows:

* $5 for Upkeep **(U)**
* $25 for Revenue **(R)**
* 4 Participants
* $0 as the Minimum Amount of Protection
* 15 Rounds, thus we take the averages below as the average number of outbreaks occurring within 15 rounds.
* The Starting Funds is $25 for every participant (later changed to $50).

#### **The Average Frequency of Outbreaks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scenario ↓ | **MP PC →** | **MP =** $15 **PC =** 40% | **MP =** $15 **PC =** 50% | **MP =** $10 **PC =** 60% | **MP =** $10 **PC =** 70% |
| Random | | 10.25 | 9.2 | 7.9 | 6.4 |
| Quarter | | 13 | 12.05 | 10.95 | 8.7 |
| Half | | 9.7 | 8.65 | 8 | 6.4 |
| Three-Quarters | | 5.6 | 5.45 | 4.45 | 3.65 |
| Full | | 0 | 0.05 | 0 | 0 |
| Half0HalfFull | | 12.75 | 11.35 | 9.7 | 7.8 |
| Bankrupt | | 14.8 | 13.6 | 13.1 | 11.9 |

#### **Mode - Frequency of Outbreaks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scenario ↓ | **MP PC →** | **MP =** $15 **PC =** 40% | **MP =** $15 **PC =** 50% | **MP =** $10 **PC =** 60% | **MP =** $10 **PC =** 70% |
| Random | | 10 | 10 | 9 | 6 |
| Quarter | | 12 | 13 | 12 | 9 |
| Half | | 8 | 9 | 7 | 7 |
| Three-Quarters | | 4 | 5 | 4 | 3 |
| Full | | 0 | 0 | 0 | 0 |
| Half0HalfFull | | 13 | 12 | 10 | 6 |
| Bankrupt | | 15 | 13 | 14 | 13 |

#### **Average Funds at Round 5 - Frequency of Outbreaks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scenario ↓ | **MP PC →** | **MP =** $15 **PC =** 40% | **MP =** $15 **PC =** 50% | **MP =** $10 **PC =** 60% | **MP =** $10 **PC =** 70% |
| Random[[1]](#footnote-2) | | 5.26 | 11.30 | 43.51 | 47.93 |
| Quarter | | 0 | 3.75 | 23.75 | 48.75 |
| Half | | 6.25 | 7.5 | 40 | 36.25 |
| Three-Quarters | | 25 | 20 | 52.5 | 60 |
| Full | | 50 | 50 | 75 | 75 |
| Half0HalfFull[[2]](#footnote-3) | | -15 | -15 | 16.25 | 49.38 |
| Bankrupt | | 1.25 | 8.75 | 16.25 | 25 |

#### **Average Funds at Round 15 - Frequency of Outbreaks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scenario ↓ | **MP PC →** | **MP =** $15 **PC =** 40% | **MP =** $15 **PC =** 50% | **MP =** $10 **PC =** 60% | **MP =** $10 **PC =** 70% |
| Random[[3]](#footnote-4) | | -43.525 | -19.18 | 52.46 | 92.45 |
| Quarter | | -56.25 | -32.5 | 13.75 | 70 |
| Half | | -30 | -3.75 | 50 | 90 |
| Three-Quarters | | 16.25 | 20 | 101.25 | 121.25 |
| Full | | 100 | 98.75 | 175 | 175 |
| Half0HalfFull[[4]](#footnote-5) | | -106.25 | -71.25 | 7.5 | 48.13 |
| Bankrupt | | -45 | -15 | -2.5 | 27.5 |

Going through each test, first starting with **MP** at $15 and the **PC** at 40%, it was clear in these scenarios that with **PC** at 40% the game was too harsh. Even if every participant contributed at 75% of **MP** ($11.25) every single round, then on average everyone would still have less than what they started with. While it was possible to finish with a profit, the optimum amount of protection to ensure this all the time was around $13 or 90% chance that a participant is not the source of the outbreak.

The story was the same with **PC** at 50%, while the game was less harsh, it still had an unacceptable result at Round 15. If every participant contributed 75% or $11.25 every round, then participants would still lose $5 compared to the starting amount. With **PC** at 50%, there was notable improvements compared to 40%, on average there was one less outbreak in each scenario and it was now possible to make a reasonable profit, but this was rarer than it needed to be.

At this point, it was clear that changing the **PC** alone wasn’t enough, the game was still too harsh, participants would lose interest half way through the game as they would have lost money by round 5. A loss of interest results in the data would becoming useless as they would (arguably) be like if players were randomly choosing their protection instead of putting thought and time into their decision. I say arguably, due to players most likely heading towards $0 as the game progressed could result in participants providing protection that was erratic and much lower than they should if participants would like to recover from the loss. The lowering of protection by all participants results in an average outbreak frequency close to or worse than the computer producing random protection amounts each round.

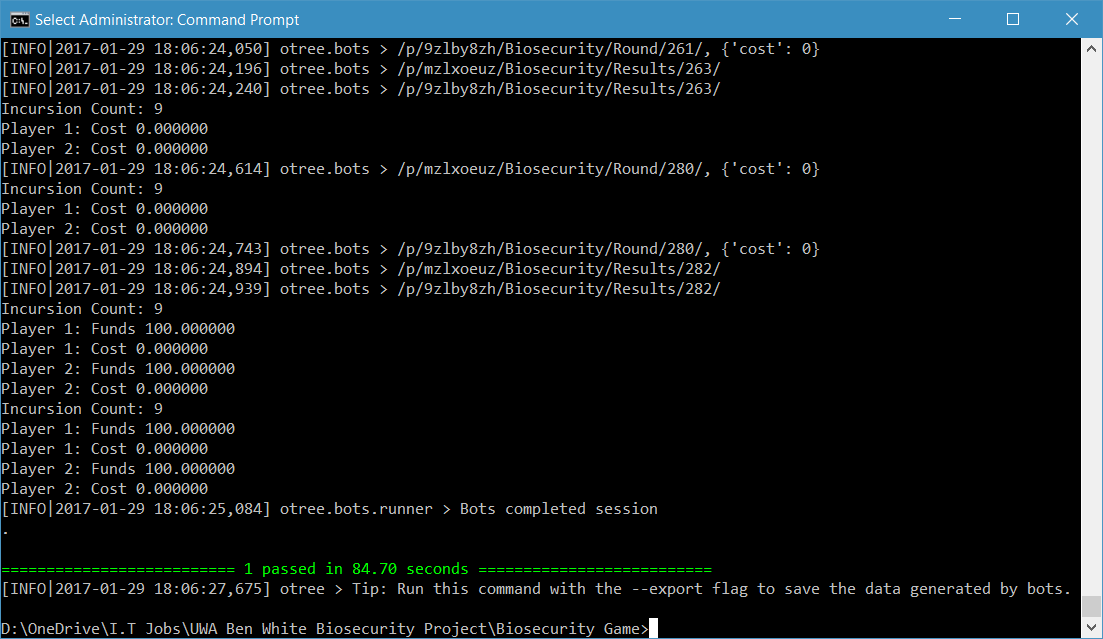
Given the results I had got from doing tests of **PC** at 40% and 50% with a **MP** of $15, I decided to try different values that would most likely make the game too easy. Thus, I decided to try a **MP** of $10 and a **PC** of 70%. The lower **MP** would ensure that participants gained more revenue when no outbreak occurred; this lowers the difficulty of the game and affects all the functions as said in [How an Outbreak is Determined](#_How_an_Outbreak_1). The rise of **PC** up to 70% also lowers the difficulty of the game as the lowest possible chance of no outbreak was 0.74 = 0.2401, or 24.01%. The combination of these together would ensure that the game was less difficult and more rewarding, while being less harsh when there was an outbreak. The results I got from this test reflected my initial thoughts, however, I did not realise how easy it would make the game. Participants could let chance decide the amount of protection compared to strategically always doing 50% **MP** ($5), and players letting chance decide their actions either did the same as or better than those would always did 50% of **MP**. Clearly, this is unacceptable, we need the results to reflect how participants acted during the game, not how chances dictated their profit or loss. At this point I could conclude that a **PC** of 40%, 50% and 70% didn’t meet the requirements as any one of this percentages would inevitably lead to participants being disengaged from the game, resulting in the data being useless for study.

This lead onto my final test, leaving the **MP** at $10, I changed the **PC** to 60%, this reduced the minimum probability of no outbreak to 12.96% (0.64 x 100). The results of this configuration of **MP** and **PC** met the requirements for the game’s frequency of outbreaks and the resulting participant’s funds. In this configuration, participants were rewarded for cooperating, as they provided more protection, the frequency of outbreaks reduced. Inversely, when participants were uncooperative, they suffered with a higher frequency of outbreaks and lower funds, with losses rarely going below $0 at the end of the game. During this test, I observed times when participants constantly doing $0 could still end the game with a positive result, some as high as $50. In a similar but opposite fashion, at the other end with 75% of **MP**, participants could get unlucky and end the game bankrupt, but such occurrences were rare.

To conclude, the final configuration chosen for **MP** and **PC** were $10 and 60% respectively, the averages and modes indicate that with players who cooperate will *most likely* be rewarded and players who are uncooperative will *most likely* suffer. With this configuration, we can rely on the data to reflect the player’s actions and their thought processes behind those actions.

## Test 2 – Automated Command Line Testing

The purpose of this test is to verify that the python code is stable. To do this, tests are performed within a terminal which go through the scenarios as mentioned in [Automated Testing](#_Automated_Testing). To pass the test the game must be able to complete tests for all the different configurations of the biosecurity game. A test is complete when you observe the following inside a terminal:



### Test Specification

To complete the test and ensure the python code is working, you must have a computer that has the following installed:

oTree 1.2.8 (or above)  
Postgres 9.6.1 (or above)  
otreechat (latest version)  
psycopg2 (latest version)  
Python 3.6 (or above)

The operating system *shouldn’t* matter as Python and oTree do not have issues with compatibility across operating systems. Also ensure that you have followed the instructions as per [oTree Tags](#_oTree_Tags).

Upon the completion of this test, every single function and class written inside the program will have been tested multiple times, including the test code itself. The only requirement of this test is that it **must** complete every test.

### Test Description

The steps to perform the test are as follows:

1. Have a copy of the oTree project.
2. Inside a terminal and navigate to the directory where you stored the project.
3. Run the following command:  
   otree test basic\_biosecurity\_game > basic\_biosecurity\_game\_cmd\_test.txt
4. Once completed wait for the test to finish and ensure the text file was made (it doesn’t have to be a text file, you could export it a differently, such as into a csv or Excel file if you have the skills)
5. Now repeat 3) and 4) but with the following commands:  
   otree test freeform\_biosecurity\_game > freeform\_biosecurity\_game\_cmd\_test.txt

otree test monitoring\_biosecurity\_game > monitoring\_biosecurity\_game\_cmd\_test.txt

otree test pledging\_biosecurity\_game > pledging\_biosecurity\_game\_cmd\_test.txt

otree test mon\_pledging\_biosecurity\_game > mon\_pledging\_biosecurity\_game\_cmd\_test.txt

otree test aop\_biosecurity\_game > aop\_biosecurity\_game\_cmd\_test.txt

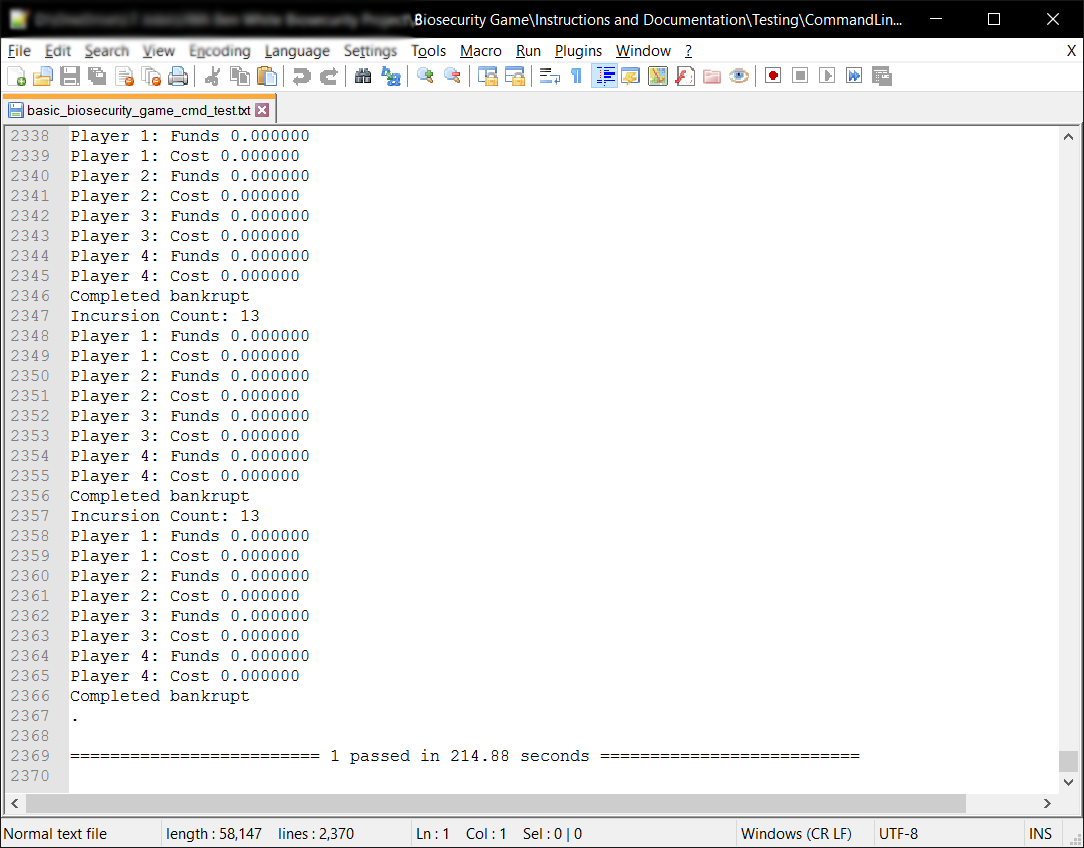
otree test aoc\_biosecurity\_game > aoc\_biosecurity\_game\_cmd\_test.txt

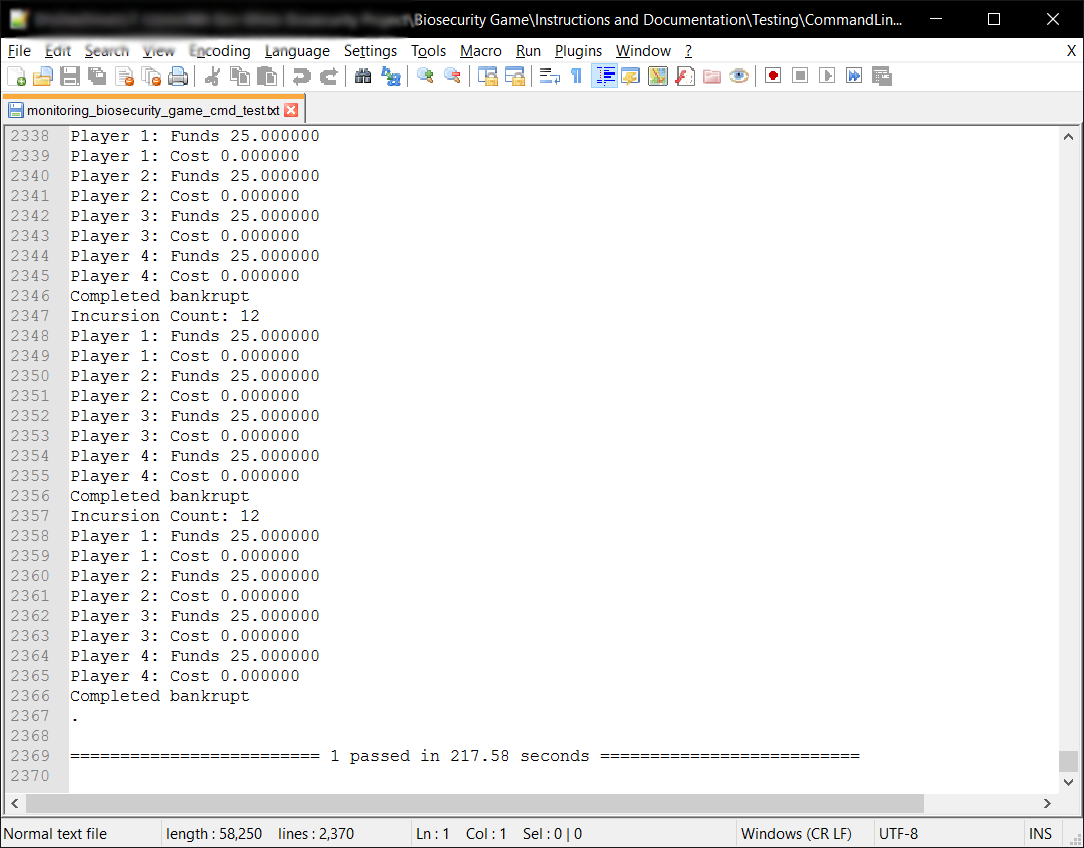
Once you have completed these steps and assuming you received no errors in your testing, then the testing is complete. If the testing failed, then it will most likely be one of 3 errors:

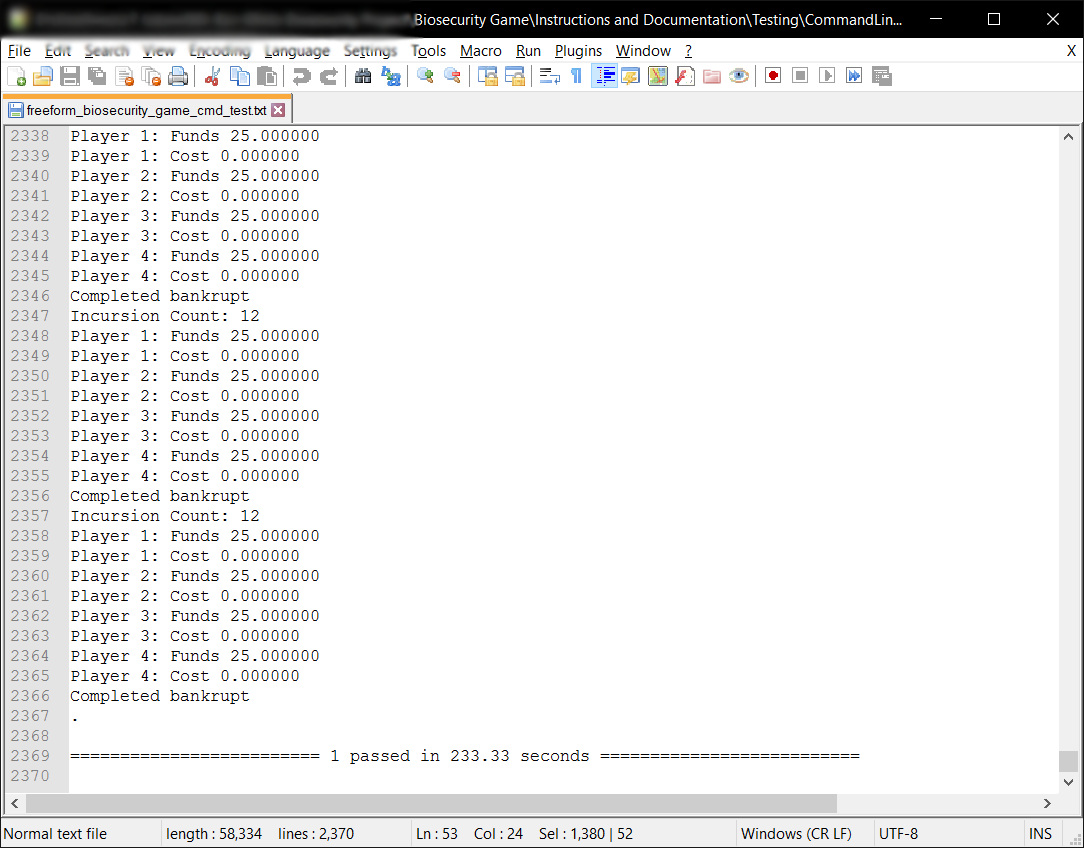
1. You haven’t followed the instructions in [oTree Tags](#_oTree_Tags)
2. It has generated an Assertion Error, in the traceback it should reveal which assert statement failed. Upon finding out which assert failed, debug the code to see what could have caused the Assertion Error.
3. You are running a future version of Python or oTree, its possible some features being used in this program might be deprecated and therefore no longer work with the version of Python or oTree that you are running. Hopefully the python code will keep up with such feature changes, but it’s not guaranteed especially if you’re reading this in 2019 or later.

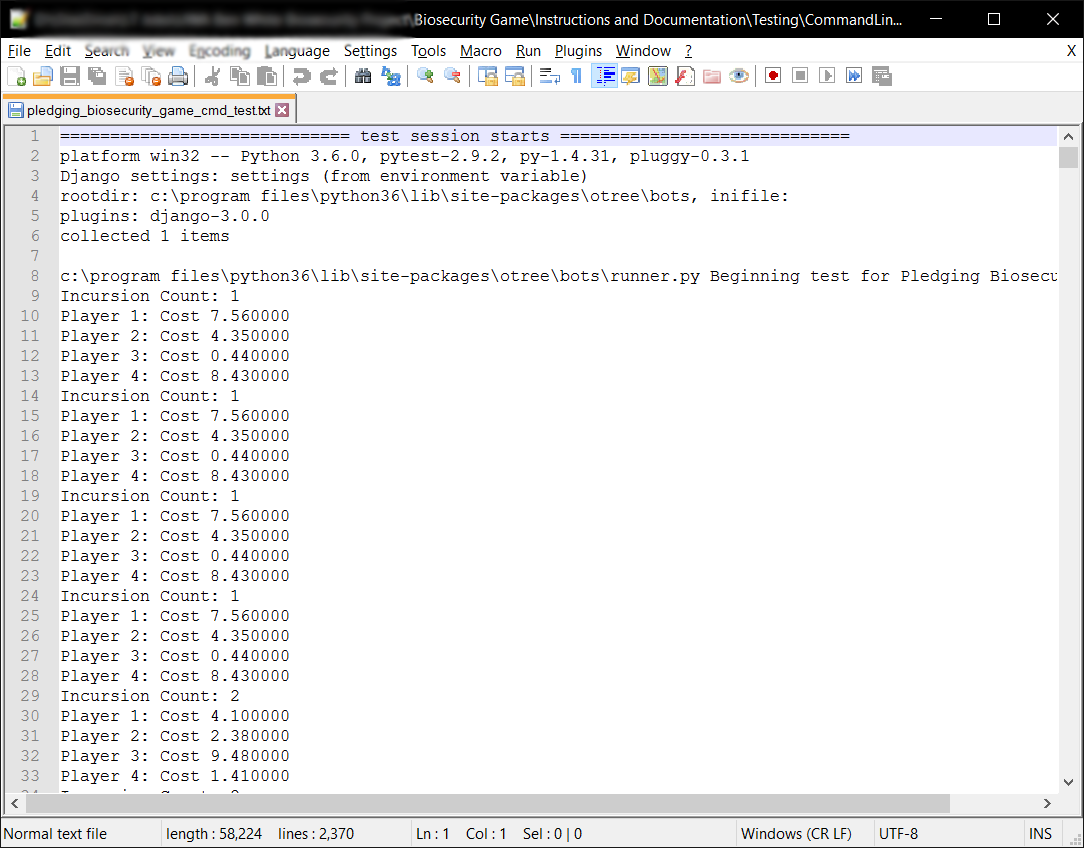
### Test Analysis Report

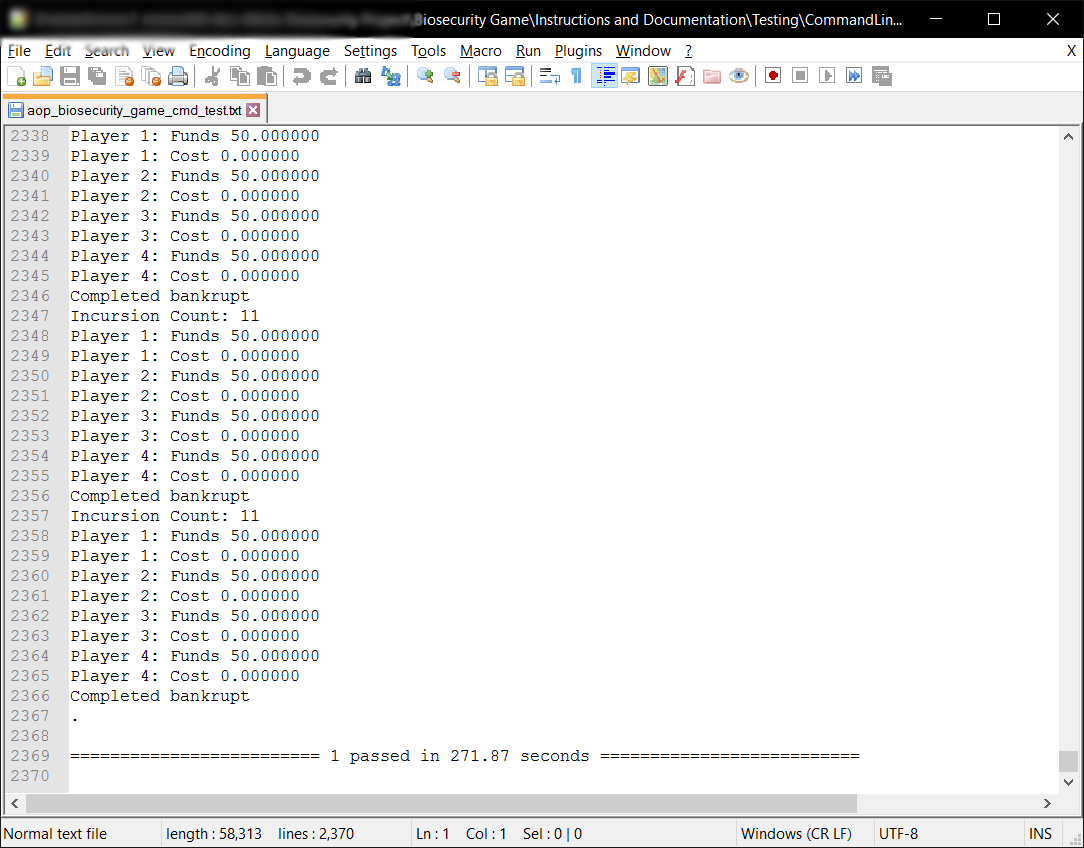
My testing was successful, my proof is below and inside the text files I made. The images below are from the text files, which have recorded that the test passed and the amount of time it took to pass the test.

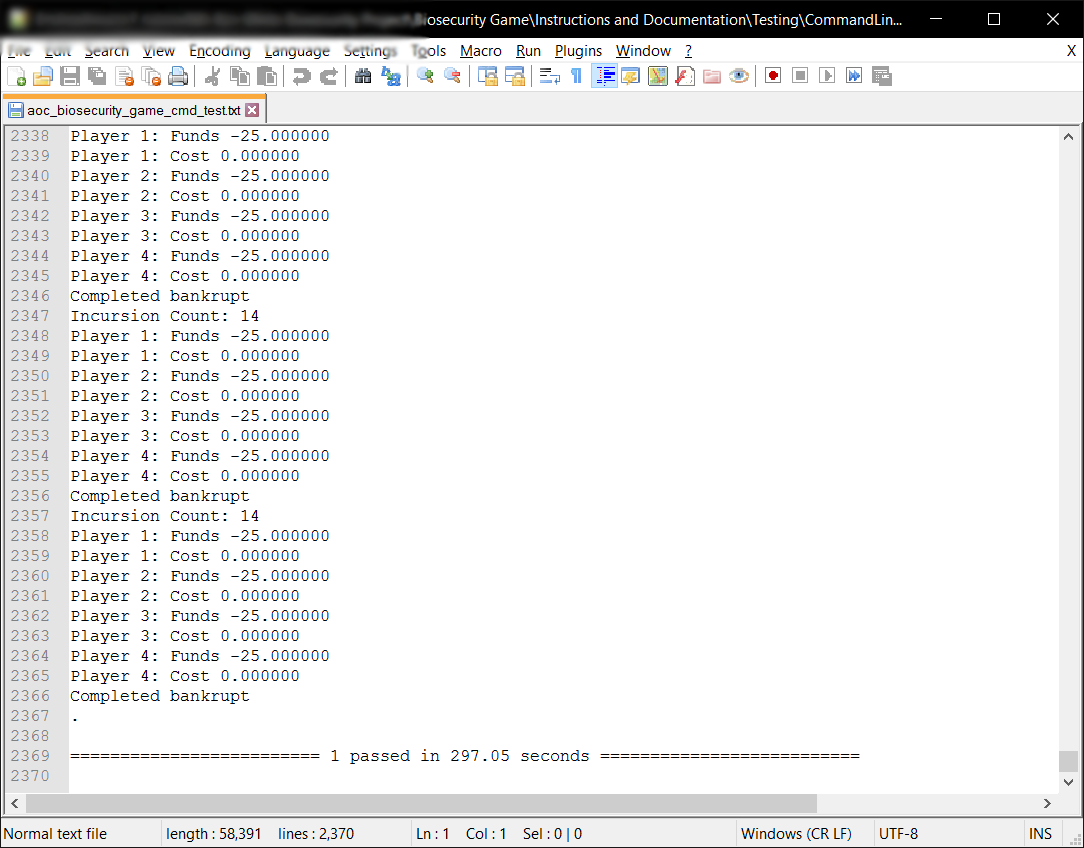












As the above proves, every single test passed and completed without issue.

## Test 3 – Automated Browser Bot Testing

The purpose of this test isn’t just to test the python code, it’s also to test the compatibility of Django, Python and oTree across multiple browsers and different devices. This test will reveal any issues the game has with specific web browsers and operating systems. In theory, oTree, Django and Python, and therefore the game should be able to run on any web browser, on any device, or is this the case or not?

### Test Specification

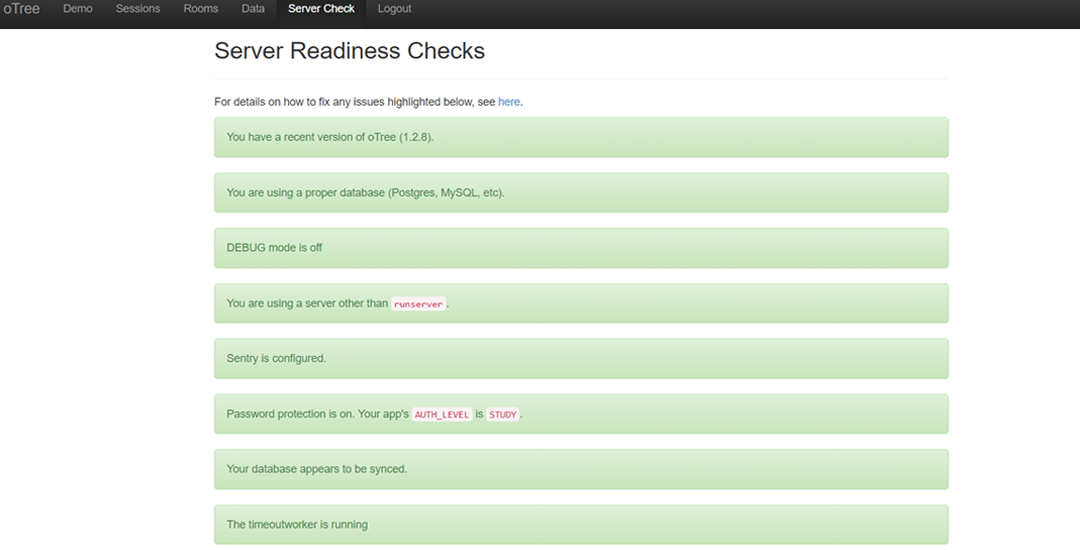
To perform this test, you need to have the following:

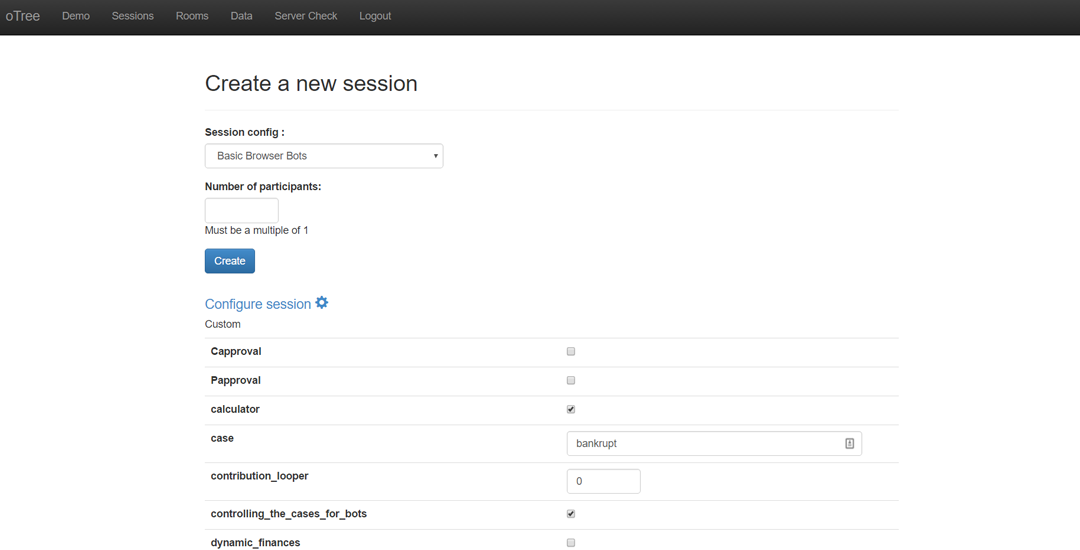
1. A web server that has completed the following test [Automated Command Line Testing](#_Test_2_–) to its requirements. This web server needs to be running the oTree program via otree runprodserver -p 80 (This will run the program and a Daphne web server that will put all input and output through port 80, or HTTP). You can, once you have gained my permission, run this test on biosecurity.are.uwa.edu.au.
2. Ideally you want to have as many different devices as possible, with each device having as many web browsers as you think appropriate. For my testing, I used a Windows 10 PC with Google Chrome, Mozilla Firefox, Opera, Internet Explorer and Microsoft Edge; a Nexus 6P an Android smartphone running Nougat 7.1.1 with Google Chrome, Mozilla Firefox and Opera; an iPad Mini 2 running iOS10.2.1 with Safari, Google Chrome, Mozilla Firefox and Opera. My test covered at least 75% of the market, as Android, iOS and Windows Users use the above web browsers with Google Chrome being the most used browser of them all (Anon., 2017; Anon., 2017).
3. Someway to record the results, you could use oTree’s inbuilt data retrieval. I recorded the results myself through paper and a pen and wrote down any difficulties I had running the tests.

To complete this test successfully, you need to have tested all the browsers you have chosen using the current market share of your time to decide what browsers to test. It is not a requirement of the test that it work on **every** web browser, the **only** requirement is that the test works on the browsers **you wish to support**. In my case I want to support Google Chrome in any version, Safari, Firefox and Microsoft Edge, thus it was a requirement that these web browsers are compatible with the game.

### Test Description

The steps to complete the test are:

1. Navigate to the web site that you are using to test this program, or use biosecurity.are.uwa.edu.au. You could run this locally, in which case navigate to 127.0.0.1:8000, unless you used the -p argument with otree runprodserver in which case navigate to whatever port you assigned for it.
2. Once here, go to ‘Server Check’ located at the top of the administration page and ensure that you are running using the most recent version of oTree (it was oTree 1.2.8 in my case), using a proper database, not using runserver (using otree runprodserver when launching the server), the database must be synced and timeoutworker is running. Observe the following image below, the green details as labelled by red arrows are the required settings to be configured:
3. Now that the server is ready to go, navigate to ‘Sessions’ in the top menu. Here you will be greeted with a list of all the sessions you have created and a button *‘Create New Session’*. Click this button, now select a ‘*Session Configuration’*, choose ‘*Basic Browser Bots’* and set the number of participants to 4.
4. Here comes the most important part for testing, click *‘Configure session’*, this will reveal parameters you can change. For testing you must change the case parameter, the case will determine what test scenario it runs, for example, bankrupt will run a case scenario where every player contributes $0 of protection per round in the biosecurity game. Do *‘bankrupt’* as your first case scenario.



4

1. Create the session and open all the links which will conduct the automated tests in your selected browser. Let them continue until they reach a page where it outputs “*Bot Completed”*.
2. To complete testing for the ‘Basic Browser Bots’, you must repeat steps 3-5, except you must run all these case scenarios successfully (in no specific order and **case specific,** you cannot do **bankrupt** as **Bankrupt**):
   1. **bankrupt** – Every player does $0 protection every round. You should have just completed this one.
   2. **half0halffull** – Half of the players do maximum amount of protection, half do $0 every round.
   3. **full** – Every player does maximum amount of protection every round.
   4. **threequarters** – Every player does 75% of maximum protection every round.
   5. **half** – Every player does 50% of maximum protection every round.
   6. **quarter** – Every player does 25% of maximum protection every round.
   7. **random** – Every player does a random amount of protection ($0 – Max Protection) every round.
3. Now complete steps 3-6, except choose the following *Session Configurations*:
   1. Freeform Communication Browser Bots
   2. Approval on Pledges Browser Bots
   3. Approval on Contributions Browser Bots
4. Now that you have completed all the session configurations, save the data using the data export page. Navigate to ‘Data’ on the top menu in the oTree administration page, and take the excel or csv sheet per app or through all-apps.

The test is completed to its requirements once you have conducted steps 1-8 on the browsers and the systems those browsers are running. This test will take a fair amount of time to complete, here’s some tips to complete it faster:

* You can run one browser bot session and complete the test for multiple browsers by opening (or copying) the links in different browsers across different devices. This is a lesson I learned in doing the manual tests, as iOS only allows 2 active tabs at the same time, meaning that to perform an automated test on an iPad you must constantly switch tabs. This resulted in much manual work for a supposedly automated test, which at the time meant that I didn’t complete the automated testing for iOS and resorted to Manual Testing for the iPad due to time constraints.
* You could create all the sessions first with their appropriate cases, and then proceed to run the browser bots for different cases simultaneously, although, be warned this will cause the web server to slow down. In my testing, which will be examined later, I could run around 20-30 bots simultaneously.
* If the browsers can’t even complete the *Basic Browser Bots* test successfully, chances are they will not complete the rest of the session configurations successfully as the other configurations put more stress on the web server and the web browser (and the device its running on) alike.

### Test Analysis Report

In my testing, Windows and Android all completed their tests successfully, iOS had some difficulties as mentioned [above](#_Test_Description), namely that it could only have 2 active tabs. This limitation by iOS on web browsers meant that every time you opened 4 links, only 2 of those links could be doing something in their respective processes. This resulted in me having to switch tabs constantly on iOS, in my case this was too much time to complete the test this way and had to move on to manual testing.

The below tables go through Windows, Android and all the web browsers I tested on these platforms, a tick will indicate that the browser passed the test, and cross will indicate it didn’t. I will explain why each didn’t complete the test below. As a reminder, the only requirement for the test was for the program to work on any web browsers that the tester (you) wishes to support. To meet this requirement, it needs to *complete every single scenario for each session configuration* as mentioned above in the [Test Description](#_Test_Description). Thus, in the tables below, a tick means it completed all the scenarios, in that session configuration.

#### **Windows 10 – Automated Browser Bot Test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Basic | Freeform | Approval on Pledges | Approval on Contributions |
| Chrome | ✔ | ✔ | ✔ | ✔ |
| Internet Explorer 11 | 🗙 | 🗙 | 🗙 | 🗙 |
| Microsoft Edge | ✔ | ✔ | ✔ | ✔ |
| Firefox | ✔ | ✔ | ✔ | ✔ |
| Opera | ✔ | ✔ | ✔ | ✔ |

#### **Android 7.1.1 – Automated Browser Bot Test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Basic | Freeform | Approval on Pledges | Approval on Contributions |
| Chrome | ✔ | ✔ | ✔ | ✔ |
| Firefox | ✔ | ✔ | ✔ | ✔ |
| Opera | ✔ | ✔ | ✔ | ✔ |

In these tests, I found all the browsers except Internet Explorer 11 to be compatible with the game. The main problem with Internet Explorer 11 was that it occasionally crashed during the test. The game could be restarted after a crash at the point where the game crashed, however this lack of stability was concerning from the start, and on top of that concern is Microsoft no longer supporting Internet Explorer 11. This means users should, if they are concerned about security at all, will abandon the web browser in favour of browsers like Edge or Chrome. Thus, with Internet Explorer 11’s lack of support and instability with the game, I will not support it.

As mentioned, in the [Test Description](#_Test_Description), I didn’t do any automated tests on iOS due to the limitation on web browsers put on by iOS. The limitation was that iOS didn’t allow any more than 2 active tabs at a time. On iOS, this makes sense as it increases battery life with less processor usage, and reduces the amount of data used by the device, meaning people on expensive 3G/4G (and potentially 5G) plans in the future will not suffer large costs. Unfortunately, in this context, it was not helpful and to complete an automated bot test, the tester would have to sit there constantly switching tabs on the device, spending just as much time (or longer) as completing a manual test. To be clear I will support iOS and the web browsers Safari and Chrome, as one will see from the manual testing below, they are successful and are quite good at running the game.

Finally, to discuss and compare the behaviour of browsers during these tests, on Windows 10, Chrome and Microsoft Edge were the fastest to complete their automated tests by far. With Opera and Firefox being comparable in speed to each other, but not as fast as Edge or Chrome. Thus, on Windows 10, I will support all browsers that passed all the automated tests, favouring Chrome, Edge and Opera. On Android, unsurprisingly, Chrome was the fastest browser, due to Android being an OS developed by Google, Chrome’s optimisation on Android would undoubtedly high priority. Thus, I will support Android’s Chrome Web browser, however I will advise, that people should not play this game on devices smaller than 7”, for an explanation of this refer to the [test report for the manual testing](#_Test_Analysis_Report).

To conclude, Google Chrome is the best web browser for the game to run on, the Manual Testing supports this, as below.

## Test 4 – Manual Browser Testing

The purpose of manual testing is to observe how the game behaves in real time, to see if the manual testing reflects the results of the automated browser bot testing. The tests also include any actions coming from administrators like moving the slowest players forward, checking the data as games progress. Mainly this test is about how the game looks and works on many different devices and browsers. This test is the final stage of testing the game.

### Test Specification

This test has the same specifications as [Automated Browser Bot Testing](#_Test_Specification), like that test you need to ensure it passes the test on the browser and devices you wish to support. To pass the test, the game must perform the following the requirements:

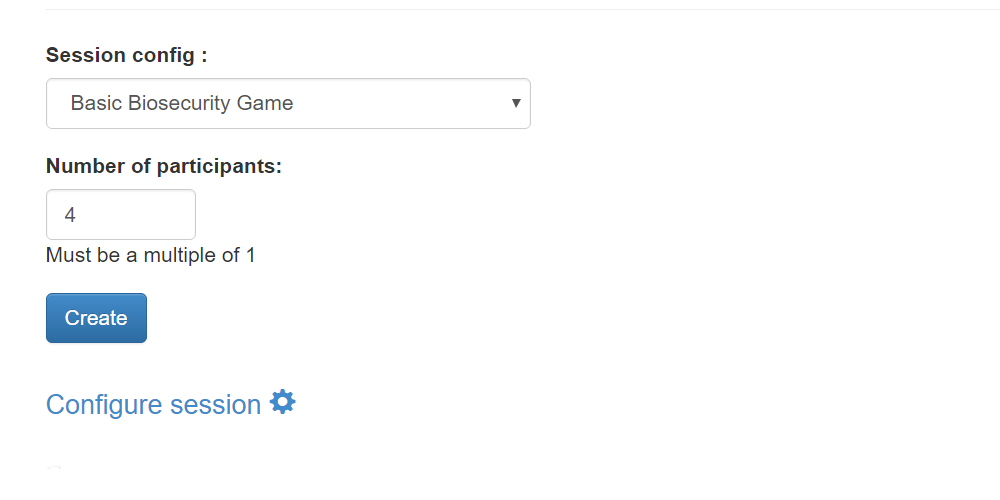
1. Complete the Lottery Game, perform at least 5 rounds of the Biosecurity Game, and display the results and Questionnaire properly at the end, with ***absolute success***, there must not be a single issue by this point.
2. The administrator must be able to use the ‘advance slowest user(s)’ functionality for at least 5 rounds of the Biosecurity Game.
3. The administrator can check the data while the game continues and the game dynamically updates the data appropriately, including the ‘Payments’ page (The payments page has been updated since then, it will show the wrong values if someone is losing in both the Lottery and/or the Biosecurity application).
4. To have met the previous 3 requirements in the session configurations:
   1. Basic Biosecurity Game
   2. Freeform Communication Game
   3. Approval on Pledges Game
   4. Approval on Contributions Game
5. Finally, have met requirements 1-4 on each device and all the browsers you wish to support on that device.

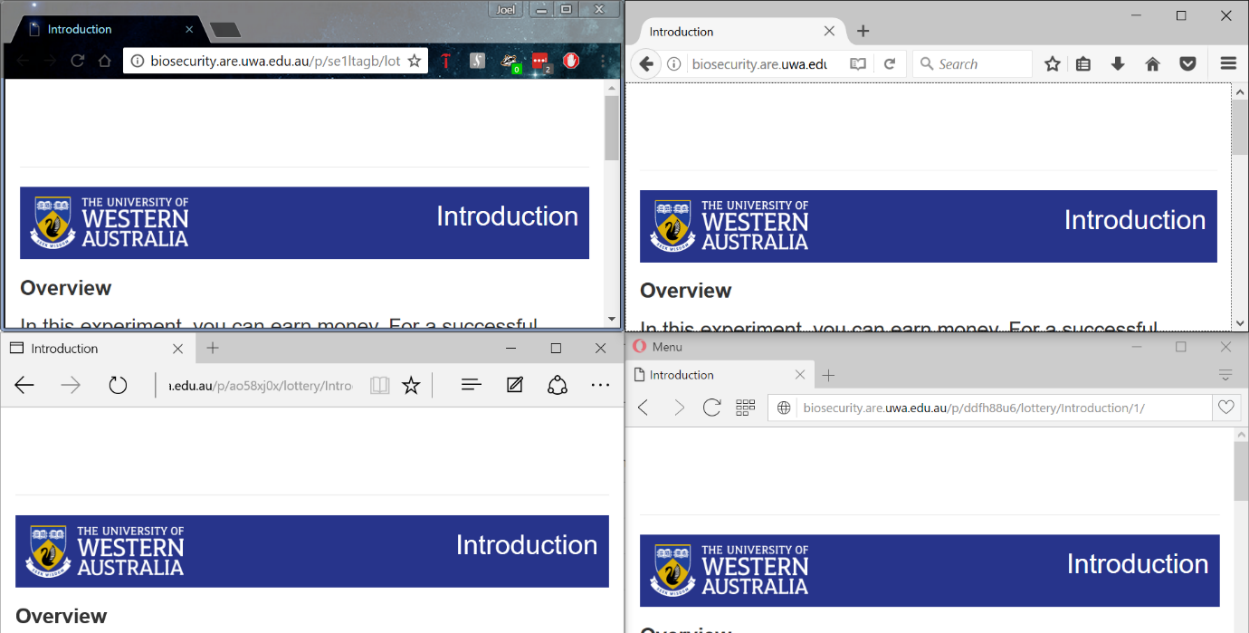
The browsers and devices I tested on are the same as the [Automated Browser Bots Tests](#_Test_Specification).

### Test Description

The steps to complete this test are as follows:

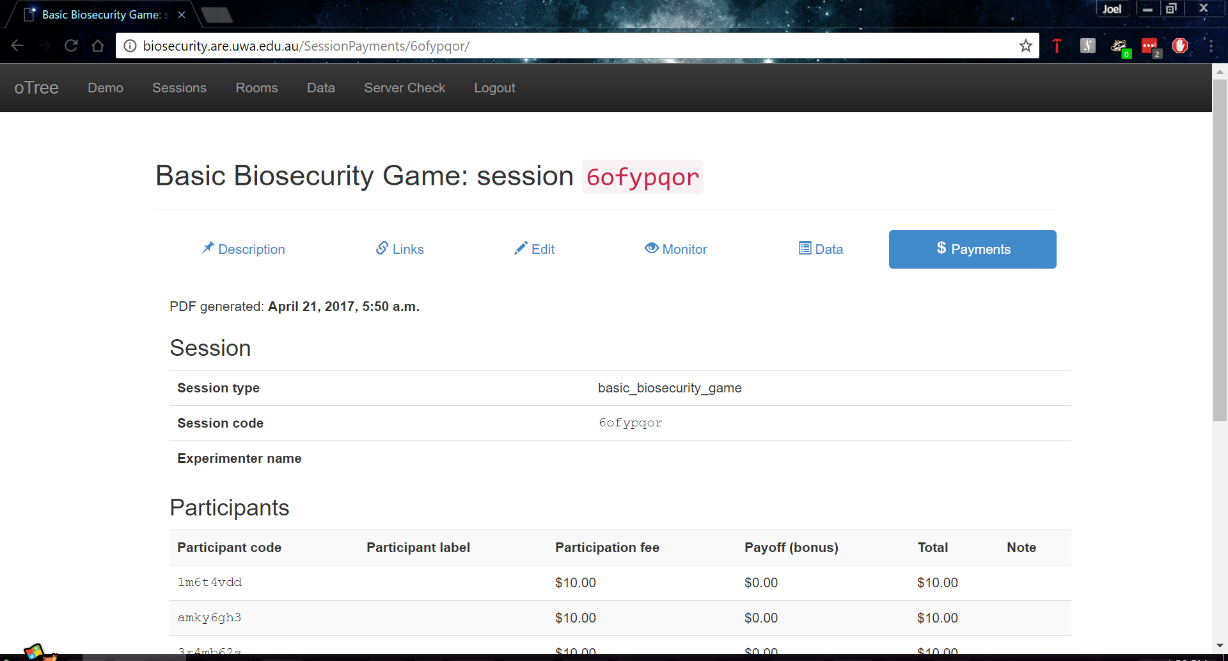
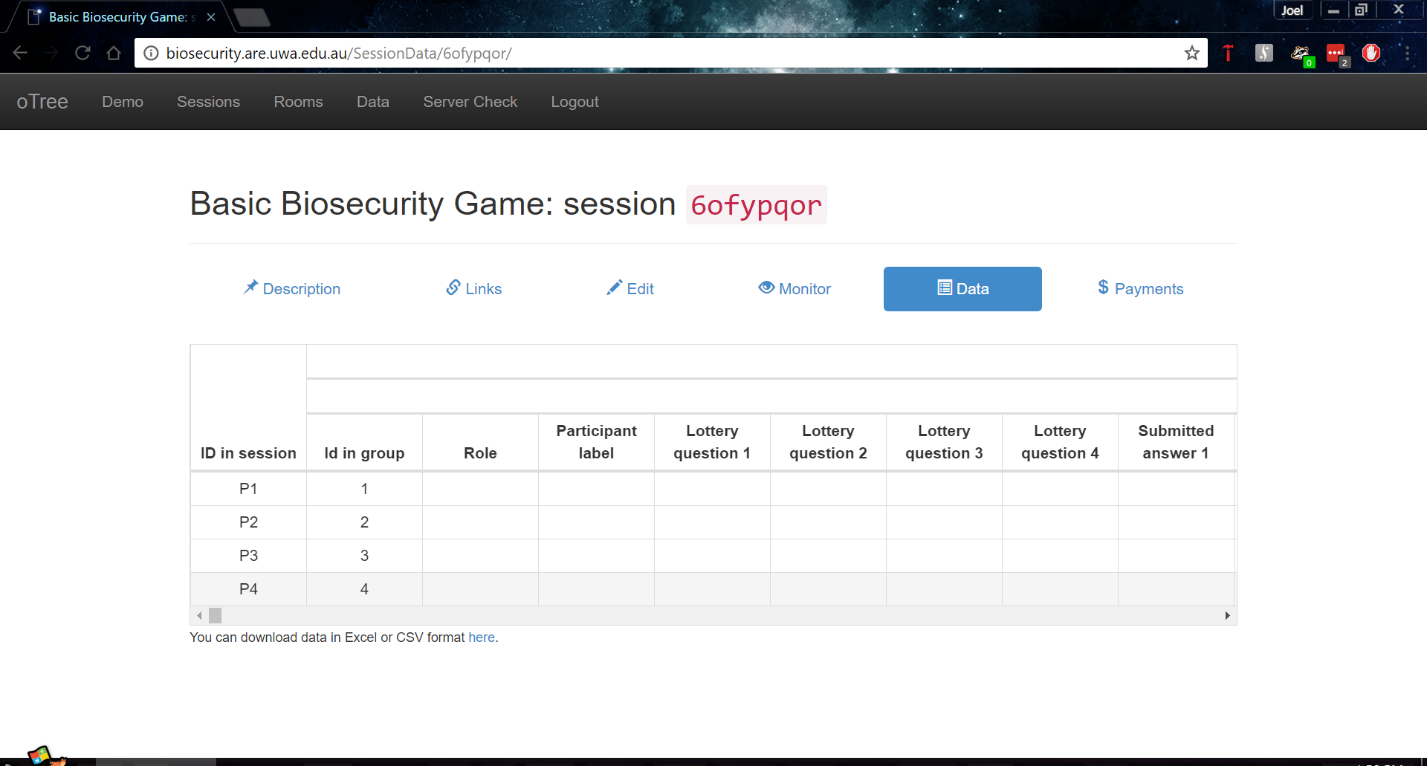
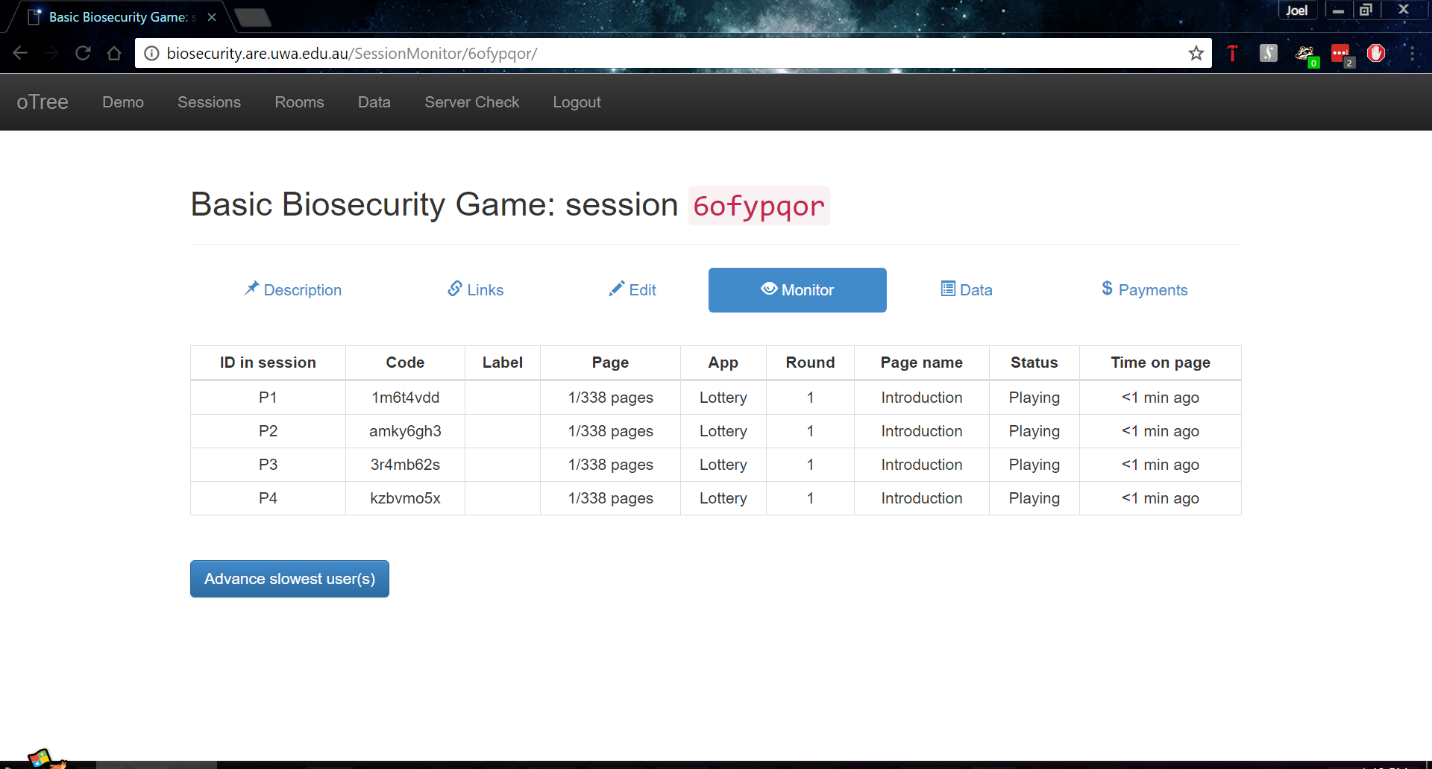
1. Navigate to the same website/server that you used to perform the Automated Browser Bot Testing, in my case this was biosecurity.are.uwa.edu.au. Also like the browser bot testing, ensure the server is still healthy as per [step 2 here](#_Test_Description).
2. Navigate to ‘Sessions’, and click ‘Create a new session’. Now in the list of session configurations choose ‘Basic Biosecurity Game’. Like the browser bot testing, put 4 as the number of participants and click create. Here there is no need to configure the session like the browser bot testing as we do not need to change any variables to test the game in its ‘default’ states as per the Session Configurations.



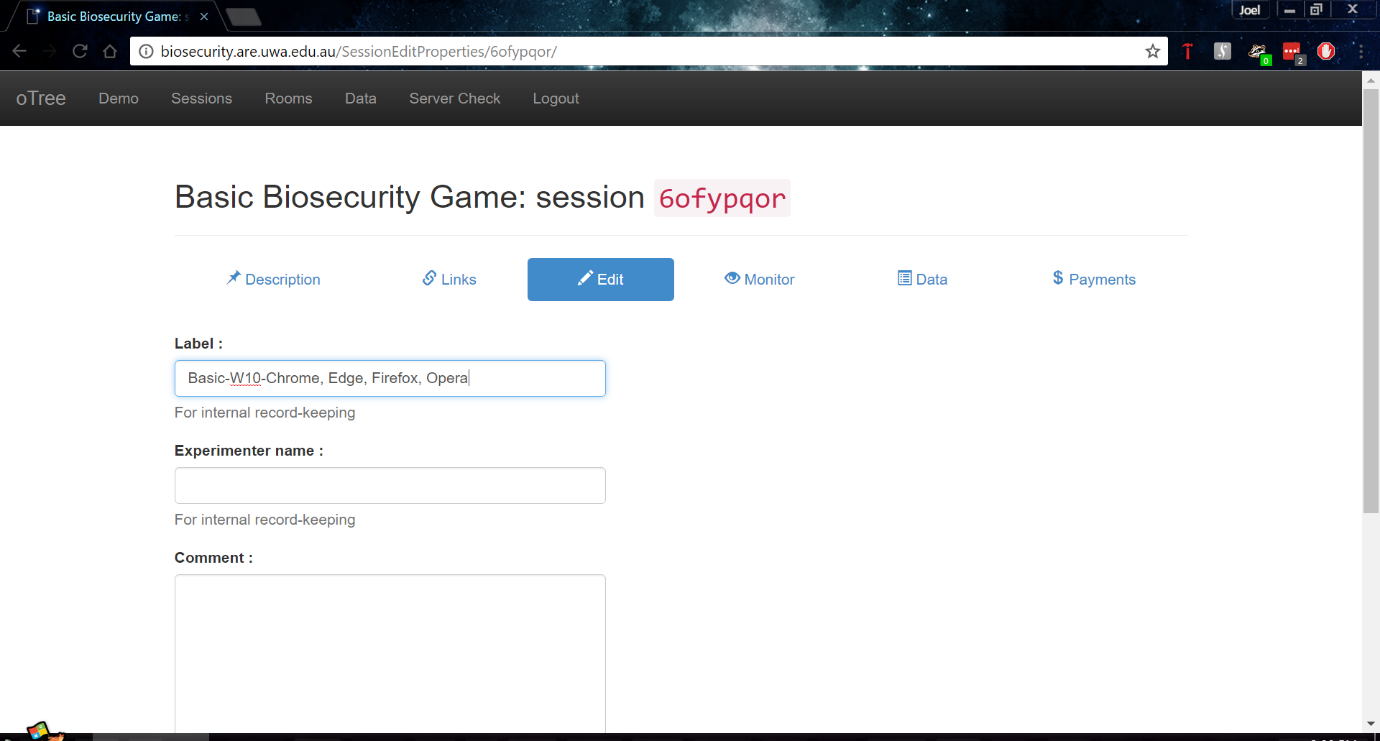
1. Now, here is when you have options as to how to perform the test, each will give you the same result even if there are issues. The first option is on each device and each browser, open the 4 links for every participant in new tabs on that one browser, meaning you will test each browser one by one with a full game. This is very time consuming, so my suggestion is to do a new browser for every participant as per the following example:  
   

This method will allow you to complete your tests more quickly and more efficiently, with a bonus of testing how browsers and devices interact with each other in a real world like scenario.

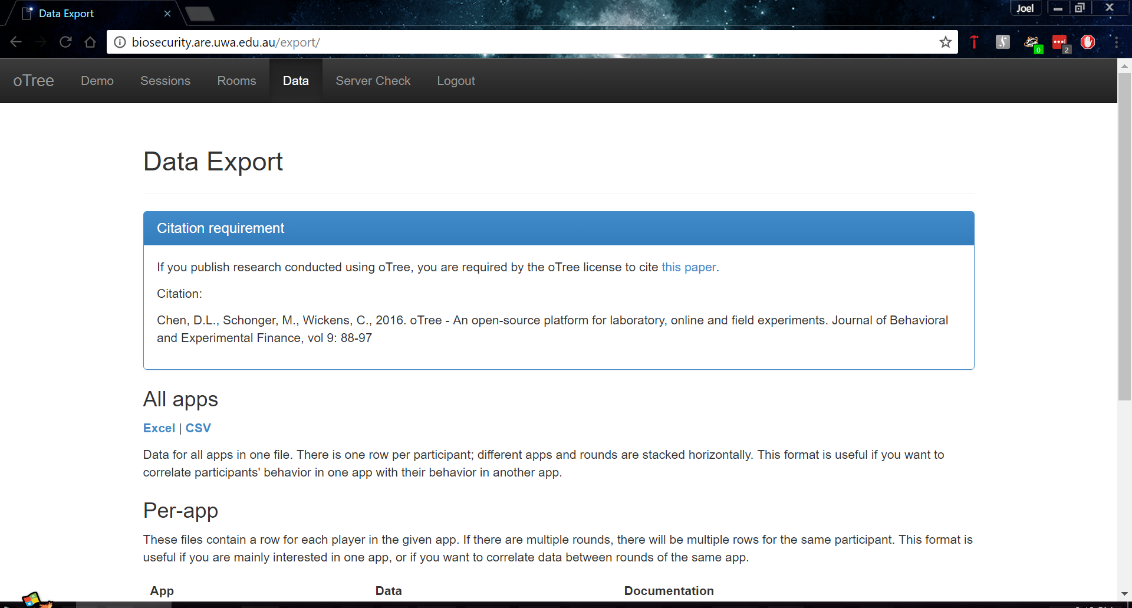
1. Now play the Lottery, and the first 5 rounds of the Biosecurity application yourself, in each round you must use different numbers in a random sequence to identify that the functions and algorithms work. To ensure that the calculations are working perfectly, use the first round of the Biosecurity Game as a chance for every player to do $0 or the maximum amount of protection to get expected and verifiable results. Test every slider, every button, observe how the elements look on screen (does font become unreadable on certain devices? Are the pictures too big for certain browsers or devices? Do the sliders work properly alongside the JavaScript that accompanies them? Etc.) ensure that the right information appears based on how you made the participants play the game. For example, in the instructions we don’t want chat instructions appearing when we aren’t in the ‘Freeform Communication Game’ session configuration.
2. Once you have completed 5 rounds yourself, check the ‘Monitor’, ‘Data’ and ‘Payments’ page, ensure that each page displays the correct information. The ‘Monitor’ page should show which page each participant is currently on and how long they’ve been on the page. The ‘Data’ page should show the data for the game as it progresses, ensure the data for the Lottery Game is there and that the first 5 rounds of the game have been recorded on the data page. The ‘Payments’ should show how much each player is owed as per the conversion rate and participation fee for the game, this page also dynamically updates with the game (again may be incorrect depending on if there are losses in the Lottery and/or Biosecurity application so far). Observe the following images below as to what the pages look like.



1. Navigate back to the ‘Monitor’ page, you should see that everyone has probably advanced a few pages while you were performing step 5. If they’re all still at the same page you left them and didn’t advance even a single page, then the timeouts might not be working as expected. If they did advance then continue the test by clicking ‘Advance slowest user(s)’ as seen on the ‘Monitor’ page, assuming you have completed all tests properly so far, clicking this will advance every user to the next page. You are to keep advancing every player till every single player reaches round 11, you will notice that not every player seems to be on the same page according to the monitor page, don’t worry, the monitor page is slower to update than the game, so they are most likely on the same page.
2. Now that you’re at round 11, perform the same checks as you did in step 5, ensure that the ‘Data’ and ‘Payments’ pages have updated accordingly.
3. Now complete the game youself, complete the Results application which includes a questionnaire and a final page that shows the results for the entire game. As one last check verify that the results reflect what it says on the ‘Data’ page, and you can verify the ‘Payments’ page yourself, make sure you take into account the participation fee and the conversion rate into your calculations.
4. Label the session with the session configuration it was and what devices and browsers you used to test it, this will show up when you extract all the data at the end. For example:



1. Now repeat steps 1-9 till you have tested all the session configurations as mentioned in the [Test Specification](#_Test_Specification_1) on every device and browser you wish to support.
2. Finally, once you completed steps 1-10 now extract the data using the ‘Data Export’ page, found in the oTree administration top menu as ‘Data’. Click ‘Excel’ inside the ‘All apps’ section. This will download an excel file with every single session configuration you tested, and if labled your sessions you will have proof that you completed the tests on your devices, and browsers.



### Test Analysis Report

As mentioned in the test specification, the test must pass with absolute success to pass the test. Thus, a tick in the tables below indicate that it completed the Lottery, Biosecurity and the Results applications without any issues. The issues I refer to here are ones that don’t allow someone to complete the game or to play the game as efficiently as other participants on other browsers.

#### **Windows 10 - Manual Test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Basic | Freeform | Approval on Pledges | Approval on Contributions |
| Chrome | ✔ | ✔ | ✔ | ✔ |
| Internet Explorer 11 | ✔ | ✔ | ✔ | ✔ |
| Microsoft Edge | ✔ | ✔ | ✔ | ✔ |
| Firefox | ✔ | ✔ | ✔ | ✔ |
| Opera | ✔ | ✔ | ✔ | ✔ |

While in the automated browser bot tests I noticed issues in Internet Explorer 11, these issues did not reveal themselves during the manual tests. However, Internet Explorer 11 has inconsistent behaviour, sometimes working perfectly, other times not at all, combining this with the lack of support from Microsoft in the future for the browser, I will not support Internet Explorer 11.

In this test, Chrome and Opera worked the best in terms of stability, not a single issue appeared at all while maintaining impressive performance as required from a modern web browser.

Microsoft Edge was *almost* the best, by appearances it was performing faster than Chrome and Opera, however the only aspect that let it down was its strange behaviour with the sliders. While dragging the slider it would work, while clicking on a different place on the slider wouldn’t always work.

Firefox worked great on Windows 10, no issues appeared at all and while the browser performed well, I had superior performance on Chrome, Opera and Edge. The only issue with Firefox is the inconsistency of the web browser across multiple operating systems, even those made by the same company. For example, in Windows 7 (which is what the lab machines run on in UWA behavioural psychology labs) Firefox refused to show the instructions on these machines despite the instructions relying on basic HTML and CSS for their appearance. As I will go into in Android and iOS tests below, Firefox while performing well in the manual tests showed too much inconsistency.

To conclude my testing for Windows 10, I will highly recommend using the latest version of Chrome to run the game and the administration of the game, it was the most stable across all tests and performed consistently. If Chrome isn’t allowed due to some restrictions on the OS or device itself, alternative browsers I will recommend are Opera and Microsoft Edge, just be aware of Edge’s strange slider behaviour.

#### **Android 7.1.1 - Manual Test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Basic | Freeform | Approval on Pledges | Approval on Contributions |
| Chrome | ✔ | ✔ | ✔ | ✔ |
| Firefox | 🗙 | ✔ | ✔ | ✔ |
| Opera | ✔ | ✔ | ✔ | ✔ |

Running the game on Android I observed that browsers had vastly different performance and behaviours when comparing them to each other. Chrome, as expected, ran perfectly without issue and with impressive performance. Opera was the same as Chrome except for being ever slightly slower.

Firefox on the other hand had inconsistent performance and inconsistent issues across my manual testing. These issues appeared in the Lottery game, at one stage, Firefox didn’t let me use the radio buttons to select the options in the play panel which is troubling. Also, Firefox had inconsistent performance throughout the test, sometimes performing extremely well, other times being slower than Internet Explorer 11 on Windows which is an impressive feat for the wrong reasons. I observed such behaviour in the automated testing, however I hypothesized it was the substantial number of tabs open in the browser (10+ tabs) that slowed down the automated testing (which other browsers had no problem with). However, the manual testing revealed the browser had deeper problems, a lack of optimisation on a fundamental level in Android.

As a result, I will support the latest version of Chrome on Android, and the alternative I can recommend is Opera. Firefox, I just cannot recommend, it needs to improve on its performance and stability in general before I can support it.

As a final note on my Android testing in general, I got to see how the game appears on smaller devices. The game doesn’t look great on smaller devices, the lottery game for example becomes unreadable in portrait mode, due to test being aligned to the same spot. This behaviour was observed in the biosecurity game during certain stages too. While yes, you can play the game on Android devices, I suggest you play the game on a tablet, its appearance is too inconsistent on smaller devices. If you must play the game on smaller devices like a smartphone then play in landscape and prepare to do lots of scrolling, pictures and certain tables can appear quite large on a smartphone.

#### **iOS 10.2.1 - Manual Test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Basic | Freeform | Approval on Pledges | Approval on Contributions |
| Chrome | ✔ | ✔ | ✔ | ✔ |
| Firefox | ✔ | ✔ | ✔ | ✔ |
| Opera | ✔ | ✔ | ✔ | ✔ |
| Safari | ✔ | ✔ | ✔ | ✔ |

Finally, iOS 10.2.1 was simply a pleasure to play the game on, Chrome and Safari worked perfectly and their performance was exceptional. Firefox and Opera here performed well and worked without issue. iOS’ issues with multiple tabs didn’t show up during manual testing as we only needed one tab open to play, in fact this time, the ‘feature’ actually worked to its advantage as other tabs weren’t taking up system resources meaning it could allocate them to the game. I will fully support Chrome and Safari on iOS 10.2.1, while offering Opera and Firefox as an alternative to them, however the chances of someone using Opera and Firefox in this case is extremely low as Chrome and Safari are exceptionally impressive on iOS 10.2.1.

To conclude my testing of the game in general, the most recommended web browser across every device is the latest version of Google Chrome, the most popular web browser in the world (Anon., 2017; Anon., 2017). The only other highly recommneded web browser compared to Google Chrome at this time is Safari on iOS 10.2.1. Other browsers can be used, just they might have issues from time to time and/or may have inferior performance to Chrome.

**Test 5 – Server Performance**

The purpose of this test is to investigate the server’s performance under load and to see how far the server could be pushed before it got to a point where it couldn’t run the game anymore. This would result in a benchmark of the server, so one could in the future improve its performance by referring to this test as its point of reference.

**Test Specification**

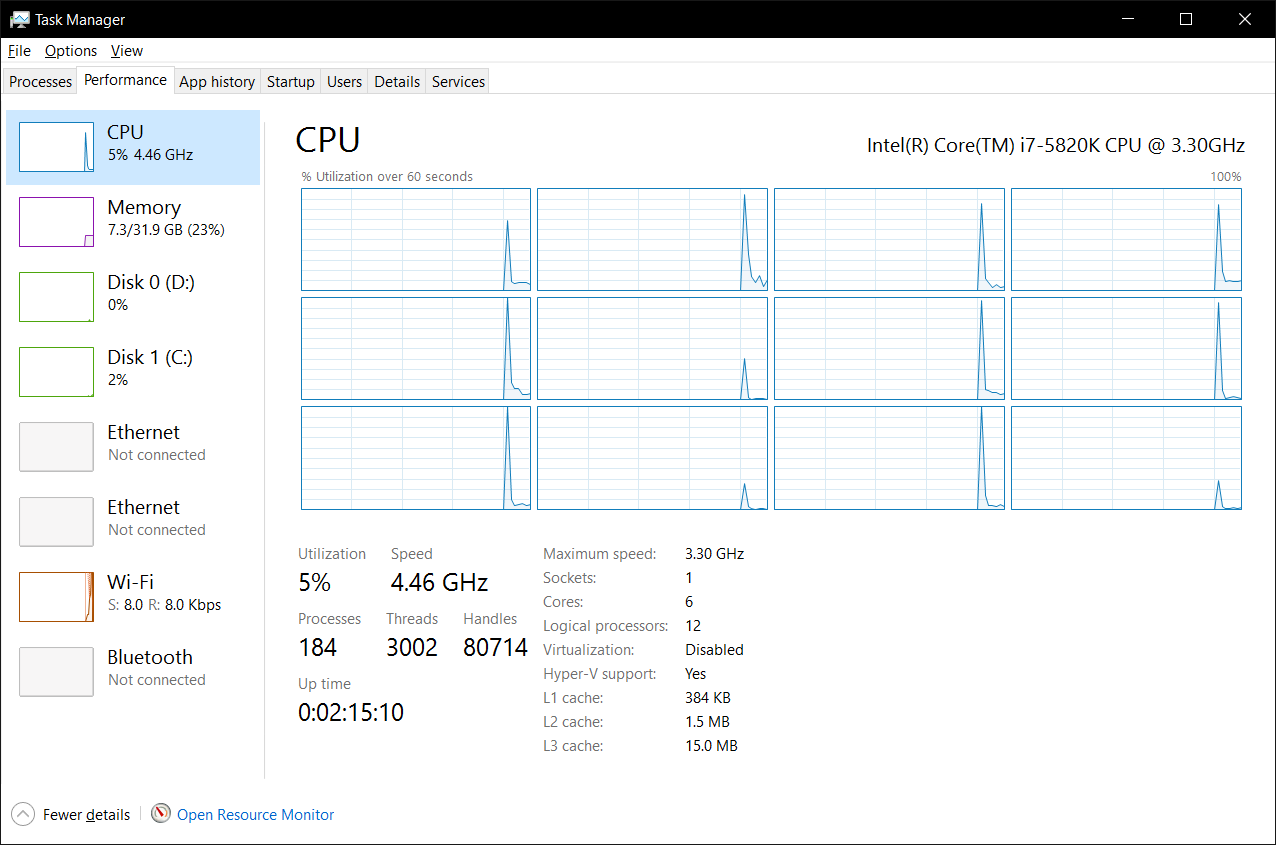
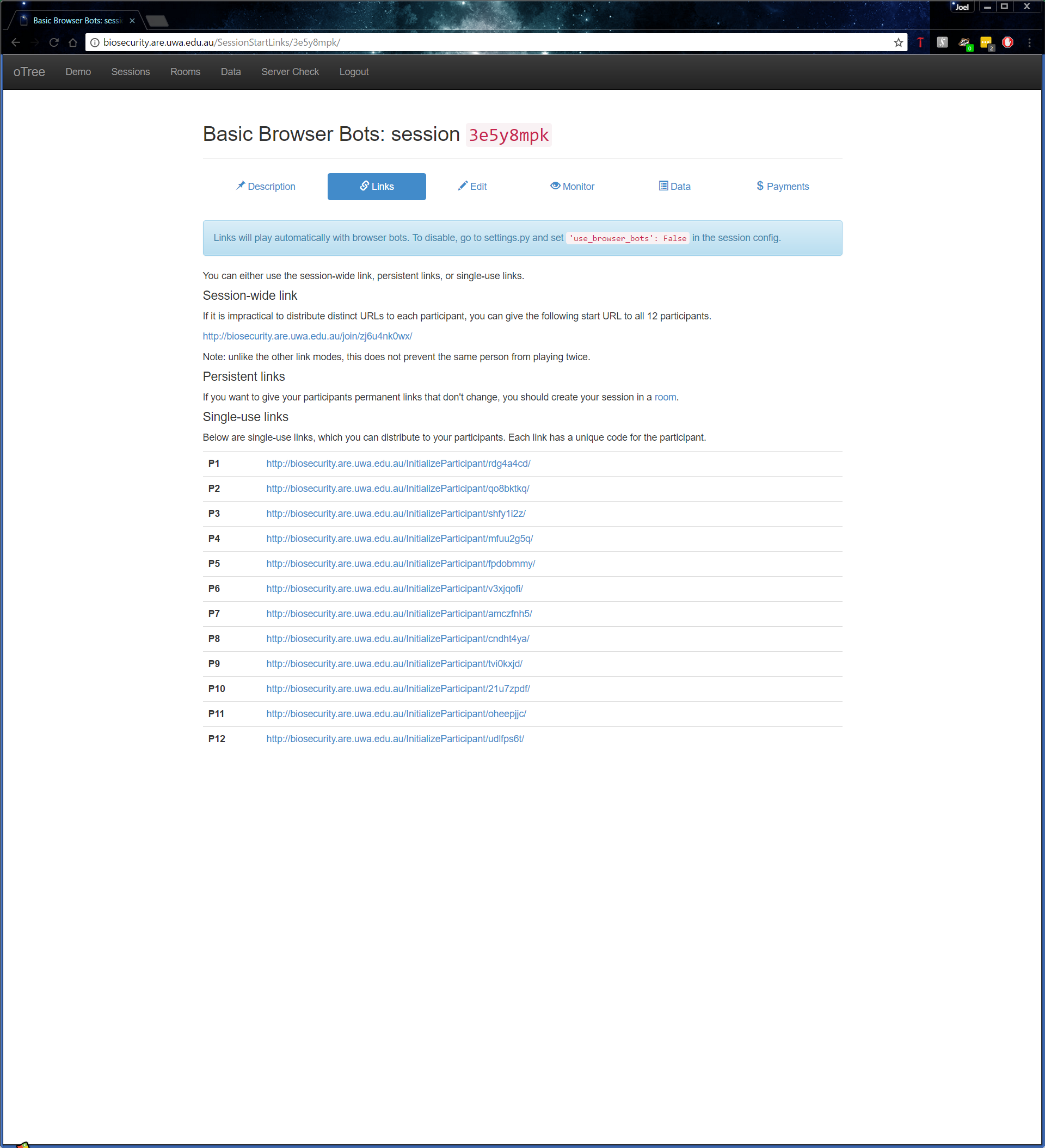
The idea of this test wasn’t to meet a certain requirement, but get a benchmark of the server. During this time, experiments are only happening within the lab and thus don’t need to support more than 20 players at a time, while this number may increase in the future that will be a case of adding more resources when the time comes.

Currently the server has 2 cores and 4 threads due to hyperthreading, 16GB RAM and is running Windows Server 2012 within VMware as a virtual operating system. The specifications of server currently were an upgrade from the 1 core, 2 threads and 4GB RAM. To perform this test, you want a computer that can handle a substantial number of tabs within Google Chrome (an excess of 200+ tabs) or multiple devices running Chrome to have up to 200 tabs open across them. I performed my test on my own desktop at home with an Intel i7-5820k a processor 6 cores, 12 threads combined with 32GB RAM running at 2666Mhz ensuring that any bottlenecks caused by the web browser and/or client device will be non-existent.

Also, as we will be timing in this test make sure you have a stopwatch available, most smartphones have one built in the clock settings.

**Test Description**

To perform this test, we will follow the steps below[[5]](#footnote-6):

1. Navigate to biosecurity.are.uwa.edu.au and login. You will need the username and password from an Administrator.
2. Do a server check by going to the ‘Server Check’ page and ensuring everything on that page is green to ensure everything is working as expected including Sentry which allows Administrators such as myself to receive the tracebacks for errors.
3. On another computer/device (I used my laptop), but you can do it on the same one, access the server using remote desktop software and connecting the university using uniconnect if outside UWA, instructions on how to do this are here: http://www.bits.uwa.edu.au/it-help/access/uniconnect/connecting. The server’s IP address is 130.95.176.155.
4. Once you have remote access to the server, open ‘Task Manager’ and go to the ‘Performance’ tab and make sure you’re observing CPU, like below:  
   
5. Now, going back to the computer (which was my desktop) where you have the website open in a browser, open the ‘Sessions’ in the main administration page. Now create a new ‘Basic Browser Bots’ session with 4 participants.
6. Open each link in a new tab starting the timer when the last tab was opened when doing 4 participants, when performing this test with more than 4 participants you need to open every single player link that isn’t a multiple of 4 first e.g.
7. Start the timer and begin opening the remaining tabs, record the time between opening the first tab that is a multiple of 4 to the last tab that you open. This will be your error margin, this will get significantly larger as you have more tabs to open.
8. Allow the test to keep going until every tab has said its finished and record the time it took to complete the test, this may be time consuming as you want to try to keep an eye on it in the case any errors do occur.
9. Complete steps 1-8 with the number of participants at 4, 12, 32, 60, 80, and 100
10. Complete steps 1-9 with the Session Configurations:
    1. Basic Browser Bots
    2. Approval on Contributions Browser Bots (AOC Browser Bots)

Now you may now try to improve the times recorded, if you attempt to do this ensure you follow the same strategy each time to keep consistent results, error margins and participants. You don’t have to follow my strategy exactly, you could start the timer from the time you open the first participant link and get the same conclusions.

**Test Analysis Report**

Before looking at the times it took the server to complete the game, I also observed how many participants the server could take when creating a session. At first, I found the limit to be 24 participants, however I found after debugging that I had assigned the names per session rather than per group. Since there are only 25 names to choose from it wouldn’t allow any more than 24 players at once when creating the session. After fixing this I found that the server could create a session with 1000+ participants well exceeding what the server could handle when participants are playing the game concurrently.

Now the results of the server performance tests are below, to remind you, the error margin is the amount of time it took to open the first tab that’s a multiple of 4 to the last tab you open. The time is given is MM:SS ± Error Margin.

#### **Server – Time Benchmark**

|  |  |  |
| --- | --- | --- |
| Number of Participants | Basic Browser Bots | AOC Browser Bots |
| 4 | 01:15 | 02:05 |
| 12 | 02:02 ± 00:05 | 03:52 ± 00:10 |
| 32 | 04:33 ± 00:20 | 08:10 ± 00:25 |
| 60 | 09:05 ± 00:20 | 14:10 ± 00:50 |
| 80 | 12:30 ± 01:00 | 20:00 ± 01:10 |
| 100 | 13:45 ± 01:15 | 26:00 ± 02:00 |

Looking at the time’s alone isn’t enough, I did record the CPU utilisation and when games started to finish to get a better idea of things might go if all the participants started at the same and how the game affects the server. Throughout the testing of the ‘Basic Browser Bots’ I noticed a CPU utilisation somewhere around between 50-75% on average occasionally spiking to 80-90% when the game went to 32+ participants. On average the first game to finish was well before the actual finishing time with participants finishing a minute or two before the rest of them completed the test.

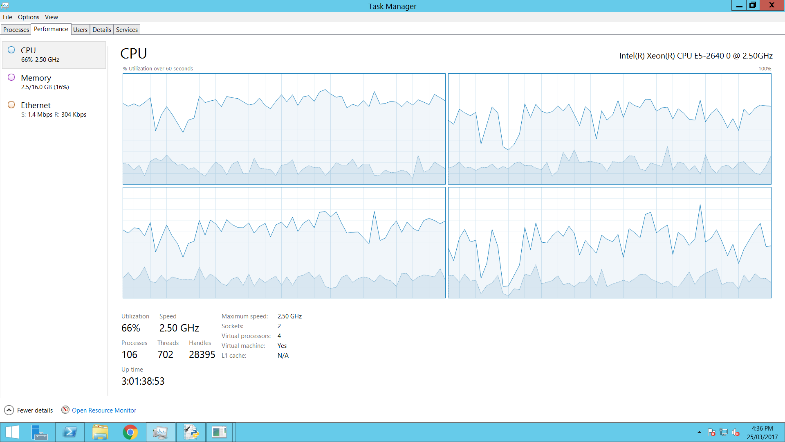
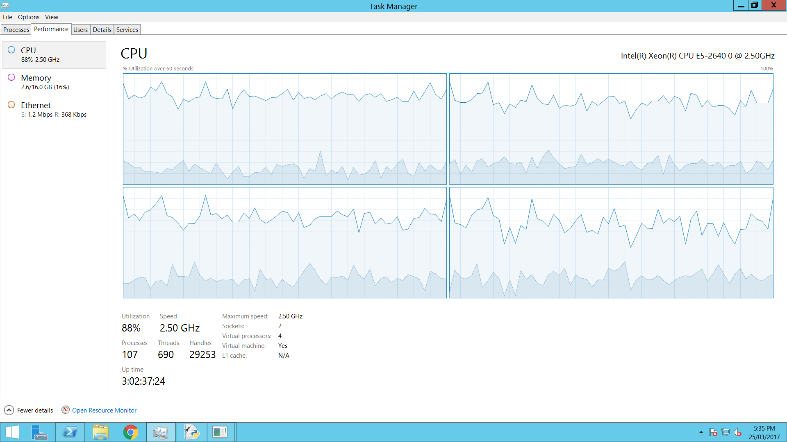
Going from the times and the utilisation and from what I observed during the tests, I believe the server can handle an absolute maximum of 50 participants concurrently with the amount of resources it currently has. At 50 participants, the game would be slow but verging on unplayable.

There is one limitation to this test, this test uses bots which push the server to constantly take in http requests, timeout requests and other requests, much more than what would happen if people were playing. I believe this to be so, as players will always take longer than the bot to decide their actions, thus when 50 players are playing concurrently, it doesn’t necessarily mean it’s taking every single player’s data once a second, its taking the data of 50 players over a minimum time of around 10-20 seconds (or longer) and a maximum of 90 seconds. This could result in the game being much smoother with 32 or 60 players in a real-world experiment than it was in testing.

I did attempt to see if I could improve the times above by adding more runworkers to run with the command otree runprodserver, unfortunately I saw no difference on my local machine or running on the server, each time being produced being within margin of error. It did make the administration page and the creation of sessions perform better, but not enough to warrant running extra runworkers. The main bottleneck and limitation to the game itself performing better is the use of Daphne as the HTTP server. Daphne as observed by others [here](https://groups.google.com/d/msg/otree/oWoypNd4O0c/CkhRfH3LAwAJ) needs some load balancing to properly utilise multiple cores and therefore increase performance of the game.  
  
To conclude, if you’re trying to improve the performance of the server rely on the times for 32, 60 and 100 participants and a CPU utilization of 50-90% depending on the number of participants you are testing with as your benchmark for the server. For reference on the CPU utilization refer to the images below:

#### **CPU Utilization**

60 Participants



100 Participants

## Test Materials

The test materials required will be a PC (Linux, Windows or OSX, Windows preferably as the other devices have browsers can have those that OSX and Linux can have, while Windows has IE and Edge), Android smartphone or tablet, and an iOS phone or tablet. With these 3 devices one will be able to test web browsers that will be used by most participants including Chrome, Firefox, Internet Explorer 11, Safari, Microsoft Edge and Opera.

The server will need to be on during the testing period, however one should not have to worry about the server being on due to it being on 24/7 except for about 5 mins in the middle of the night once a week. You could run your own server, if you wish to do this follow the instructions here:

<http://otree.readthedocs.io/en/latest/server/intro.html>

The instructions will inform you how to create and run a server, including pulling a git repository using SSH if you’re project is on here or have access to my repository which contains the project. The instructions will also tell you how to create a Postgres database and then use that database for the oTree project. Some extra notes from myself here is that I advise that if you do create your own user as they do in the instructions, ensure that the user has the authority to **create** databases and manage the one you have made. Without this authority, you won’t be able to use this Postgres user for testing.

Theoretically, the game should work no matter what OS you’re running on your own personal device. The web server runs all the Python code and produces the Django templates which your web browser views as HTML. However, to be thorough, each device should have as much web browsers as possible for testing to ensure a comprehensive compatibility test through the automated and manual testing.

## oTree\_Tags

For this program to work you will need to adjust the oTree package files, as this program requires an adjusted otree\_tags file. There are two ways you can do this:

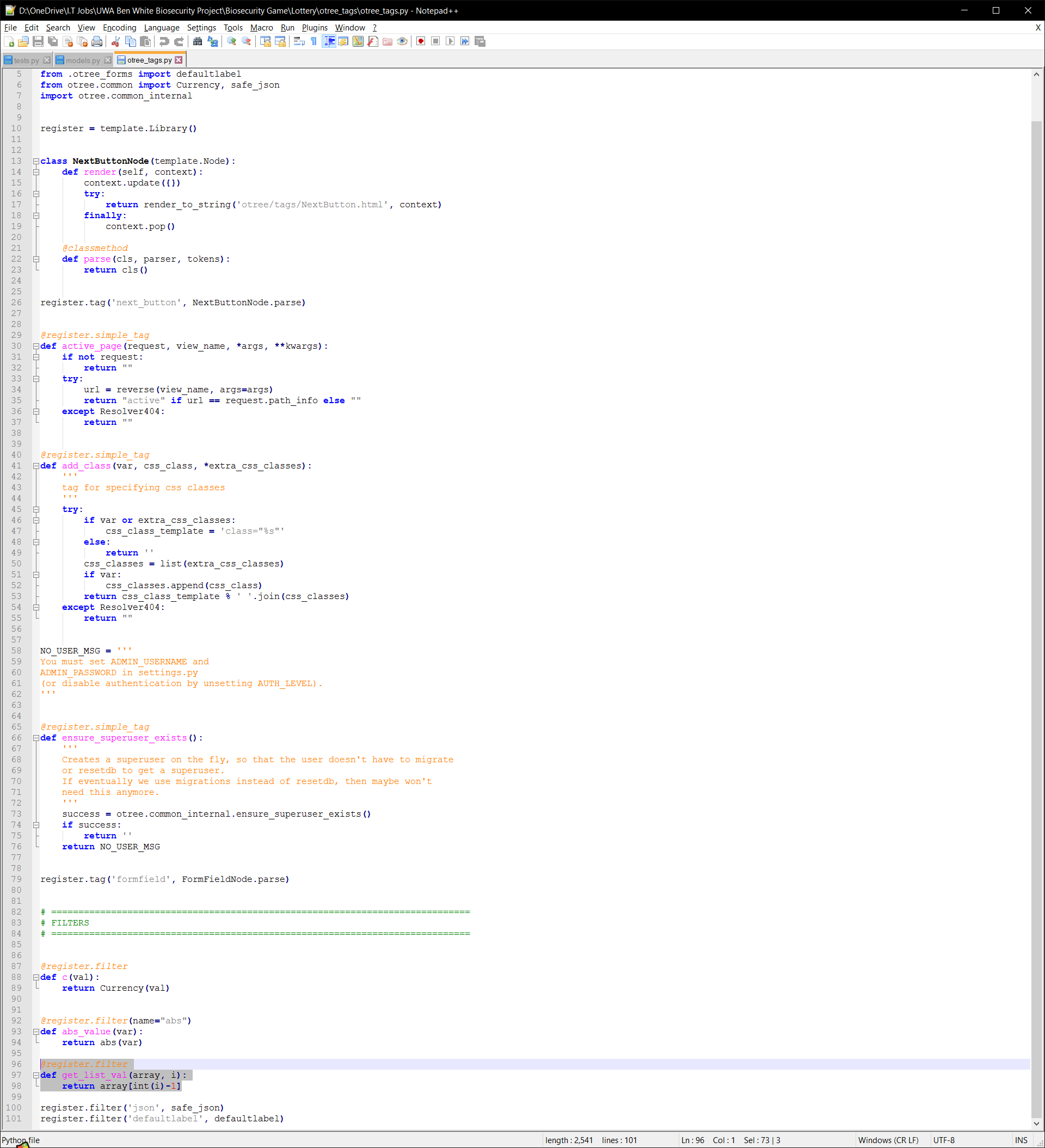
1) Assuming you cloned the repository or have the program files available to you, inside the Lottery app folder named *Lottery*, there should be a directory named otree\_tags, go in there.  
Copy the otree\_tags file inside a paste the file inside this directory:

<Home>\Python<version>\Lib\sitepackages\otree\templatetags\  
  
<Home> refers to your account files and folders which will be located differently depending on your OS, please refer to the Python documentation and how you installed Python on your machine to find the directory we’re looking for. Also, keep in mind the folder will be named differently depending on your Python version, so refer to what version you have as well to fill in <version>.

2) Instead of copying the file in the repository, find this file below (refer to above to understand what <Home> and <version> mean):

<Home>\Python<version>\Lib\sitepackages\otree\templatetags\otree\_tags.py

In this file add the code as per the image below:



## Glossary

* oTree: “oTree is a framework based on Python and Django that lets you build:
  + Multiplayer strategy games, like the prisoner’s dilemma, public goods game, and auctions
  + Controlled behavioural experiments in economics, psychology, and related fields
  + Surveys and quizzes” (oTree, 2016)
* Python: A high-level programming language that allows you to design and implement computer programs on any machine.
* Django: A web framework written in Python used to create complex, database driven websites.
* HTML: Stands for **H**yper**T**ext **M**arkup **L**anguage, it’s the language used to produce text, images, colour, graphics and links on the world-wide web. Every single page you access on the internet uses HTML.
* Web Server: The computer/device/machine that runs the necessary programs and services to allow people to access the website on the internet and run the oTree project using Python and Django on a web browser without a client’s computer needing Python, Django or any of the necessary programs to run the game. The term web server is also used when talking about the services or programs that deal with the input and output through HTTP, e.g. Daphne Web Server.
* Web Browser or Browser: An application that is used to retrieve and present information and graphics sent by a web server.
* CSS: Stands for **C**ascading **S**tyle **S**heet, it is a language used to format how HTML elements will appear on a web browser.
* Relational Database: A database that properly recognises many to many relationships and is structured in a way to store information efficiently and effectively.
* Postgres: A Database Management System (DBMS), to which a DBMS allows one to create and manage databases.
* HTTP: Stands for **H**yper**T**ext **T**ransfer **P**rotocol, a high-level protocol designed to take and receive requests for HTML pages on the internet, it is a major part of communicating on the world-wide web.
* Operating System: Often used as an acronym **OS**, low-level software that allows a computer to manage basic functions like scheduling tasks, providing API’s for other programs etc. Windows, Android, OSX and iOS are examples of operating systems.
* IP Address: Stands for **I**nternet **P**rotocol **Address**, it is your “location” in a network, you can think of it like your home address, a unique set of numbers (and letters if using IPv6) that identifies your computer.
* Hardware: A physical part, component of a computer
* Software: A virtual program that can run a computer/device/machine created using a programming language, and designed for a certain purpose.
* Session: In the context of oTree, “a session is an event during which multiple participants take part in a series of tasks or games.” (oTree, 2016)
* Protection: In the context of the game, this is the amount of effort a player contributes during a round in the biosecurity game in terms of probability of the player **not** being the source of the outbreak. This is not to be confused with Cost which is still the amount of effort a player contributes, but represented in dollars.
* Outbreak: In the context of the game, this is an event when pests have infected the player’s crop for that round and therefore get no revenue.
* GitHub or Git – A version control system that tracks any changes made to a piece of software from one person or amongst a group to coordinate and organise themselves on how to keep changing the software, if it needs changing.
* Repository – Git uses repositories as sources of code, the repository is the main source of the code to which a group will coordinate from, as Git stores the information about the project such as changes made to the code.
* Runworker – These are processes that are run when executing the commands for running an oTree server. Each process appears as a terminal on the desktop, they deal with the navigating and loading of the pages.

## Documents/Spreadsheets for Reference

* [Incursion, Funds, Cost Tracking and Averages 40%](https://1drv.ms/x/s!AphuEhKWv8YUgud3kOETcJLi2eU_-w)
* [Incursion, Funds, Cost Tracking and Averages 50%](https://1drv.ms/x/s!AphuEhKWv8YUgvtbCTmQmAEED5fuOg)
* [Incursion, Funds, Cost Tracking and Averages 60%](https://1drv.ms/x/s!AphuEhKWv8YUgvxMeKyyD9coBlN1kw)
* [Incursion, Funds, Cost Tracking and Averages 70%](https://1drv.ms/x/s!AphuEhKWv8YUgvwa5y-jbL8sfywNmw)
* [Load Testing Basic Biosecurity Game 4-200 Participants](https://1drv.ms/x/s!AphuEhKWv8YUg4MnpDI9VmqpioJ_Sw)
* [Load Testing AOC Biosecurity Game 4-80 Participants](https://1drv.ms/x/s!AphuEhKWv8YUg4MrHluooBuqcJ1HLw)
* [40% Test files for Frequency of Outbreaks](https://1drv.ms/f/s!AphuEhKWv8YUguhr1j_hk_1tX4-AhA)
* [50% Test files for Frequency of Outbreaks](https://1drv.ms/f/s!AphuEhKWv8YUguhp-XuwAADG57LX1Q)
* [60% Test files for Frequency of Outbreaks](https://1drv.ms/f/s!AphuEhKWv8YUgvwrFvToDFj9cxlpCQ)
* [70% Test files for Frequency of Outbreaks](https://1drv.ms/f/s!AphuEhKWv8YUgvwZvUhG4wjDzPBbeA)
* [Command Line Testing Files](https://1drv.ms/f/s!AphuEhKWv8YUgu16rtR9Trv-C8VgAg)
* [Manual Testing File](https://1drv.ms/x/s!AphuEhKWv8YUg4Bqqmhlyg8MkdIEYw)

## **References**

Anon., 2017. *Net Market Share.* [Online]   
Available at: https://www.netmarketshare.com/  
[Accessed April 2017].

Anon., 2017. *W3 Counter.* [Online]   
Available at: https://www.w3counter.com/globalstats.php  
[Accessed April 2017].

Chen, D., Schonger, M. & Wickens, C., 2016. oTree - An open-source platform for laboratory, online and field experiments.. *Journal of Behavioral and Experimental Finance,* Volume 9, pp. 88-97.

oTree, 2016. *oTree Documentation.* [Online]   
Available at: http://otree.readthedocs.io/en/latest/  
[Accessed April 2016].

1. These averages were the combined average of the 4 players at Round 5, for more detailed results on how each player performed refer to the Excel Spreadsheets or perform your own. [↑](#footnote-ref-2)
2. Same as 1 [↑](#footnote-ref-3)
3. These averages were the combined average of the 4 players at Round 5, for more detailed results on how each player performed refer to the Excel Spreadsheets or perform your own. [↑](#footnote-ref-4)
4. Same as 3 [↑](#footnote-ref-5)
5. The test instructions will assume you have done Tests 1-4 and have the administration instructions available to you. [↑](#footnote-ref-6)