Testing Document  
*UWA Biosecurity Game*  
By: Joel Dunstan  
Email: [jd427244@gmail.com](mailto:jd427244@gmail.com)  
biosecurity.are.uwa.edu.au

Table of Contents

[Objectives and Test Summary 3](#_Toc479019413)

[Testing Strategy 3](#_Toc479019414)

[Automated Testing 3](#_Toc479019415)

[Hardware and Manual Testing 5](#_Toc479019416)

[Test 1: The Frequency of Outbreaks 5](#_Toc479019417)

[How an Outbreak is Determined 6](#_Toc479019418)

[Test Specification 7](#_Toc479019419)

[Test Description 7](#_Toc479019420)

[Test Analysis Report 8](#_Toc479019421)

[Test 2 – Automated Command Line Testing 11](#_Toc479019422)

[Test Specification 11](#_Toc479019423)

[Test Description 11](#_Toc479019424)

[Test Analysis Report 12](#_Toc479019425)

[Test 3 – Automated Browser Bot Testing 12](#_Toc479019426)

[Test Specification 12](#_Toc479019427)

[Test Description 12](#_Toc479019428)

[Test Analysis Report 13](#_Toc479019429)

[Test 4 – Manual Browser Testing 13](#_Toc479019430)

[Test Specification 13](#_Toc479019431)

[Test Description 13](#_Toc479019432)

[Test 5 – Server Performance 13](#_Toc479019433)

[Test Specification 13](#_Toc479019434)

[Test Description 14](#_Toc479019435)

[Test Analysis Report 14](#_Toc479019436)

[Test Materials 14](#_Toc479019437)

[oTree\_Tags 15](#_Toc479019438)

**Objectives and Test Summary**

This document is primarily focusing on testing all aspects of the system, web site and the program underlying the website biosecurity.are.uwa.edu.au. Here it will describe the behaviour of the system and the underlying logic that decides when outbreaks occur. Testing 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.

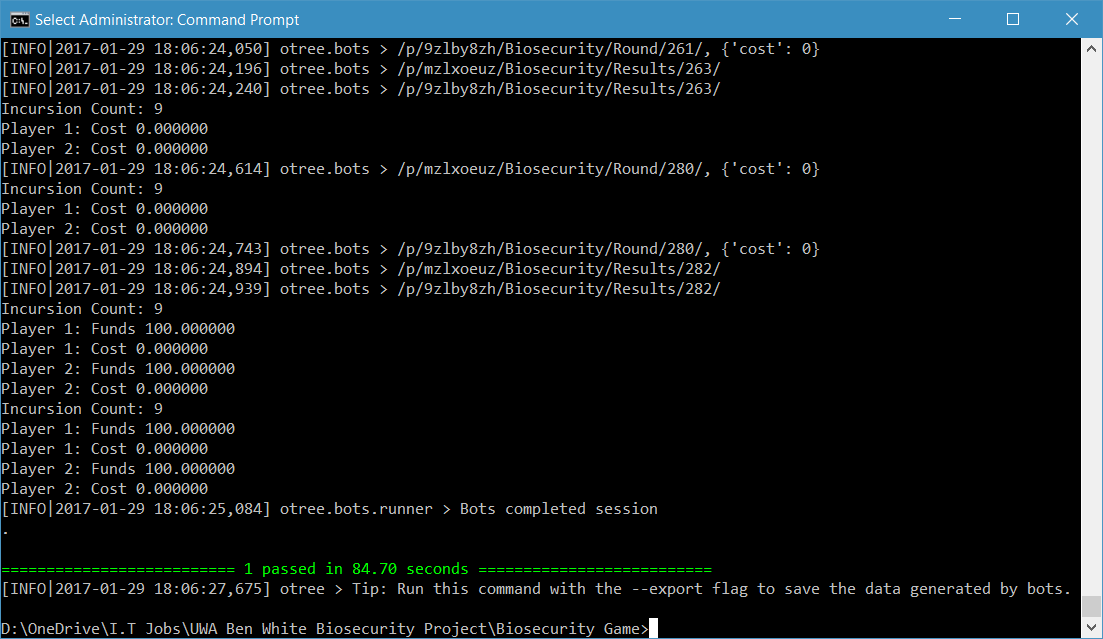
The overall strategy is a bottom-up approach test the bare code itself, using the command line as a way to test this (Refer to [Automated Testing](#_Automated_Testing) for more detail), from here I will move on to using browser bots to ensure that logic works within a web browser as well (While checking its compatibility with web browsers). Finally, I will conclude with running the game manually myself in each browser.

Changes made to the code will be documented inside each test’s results.

### Automated Testing

Automated Testing will be used in just about all areas of testing using the tests.py from every application inside the oTree Project, the project was divided into three applications, Lottery Game, Biosecurity, and Results. These tests can be run inside the command line using oTree test <session\_config\_name> using numbers as arguments to conduct tests with a different number of participants. 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 application Biosecurity and run the session configuration ‘basic\_biosecurity\_test’ using 4 participants.

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, then it indicates that the game ran without an AssertionError which in oTree occurs whenever an assert statement failed or a game failed to finish due to oTree having no progress in the game (it doesn’t progress to the next 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 Game, automated testing will always double check that the quiz for testing people’s understanding of the game is working correctly giving an incorrect answer to see if it will fail, and then giving the correct answers. For the actual Lottery Game, itself, it randomly chooses A or B for the forms to make sure the data goes in through the form as there is no real way to verify that a person gets an exact result, due to the randomness of the selection of the winning game for every single session.

Focusing on the Biosecurity Game, automated testing conducts itself using different scenarios, the first being random where for every single form it produces a random value and performs checks on these values by replicating the same equations used in the code of the game and then comparing the results of the game with the results of the test using assert statements. 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. In the random scenario, we use random values for every single form required. The automated tests conduct other scenarios in the order mentioned below, which each scenario will show the values used for each of the values being tested above:

Quarter: 25% Max Protection for the cost of protection, funds are dependent on whether there was an outbreak or not and hence random and will not be mentioned further, group pledging is done randomly and the approvals are always -6.

Half: 50% Max Protection for the cost of protection, group pledging is done randomly and approvals are always 3.

Three-Quarters: 75% Max Protection for the cost of protection, group pledging is done randomly and approvals are always done at 6.

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

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

Bankrupt: No protection at all, group pledging and approvals are random.

There are default values used for every test throughout, the maximum amount of protection that can be used is always 15, the cost of upkeep is always 5, the revenue is always 25, the starting amount for every player is 25, 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 is 4 which all participants are all in a single group. Finally, a pledging round comes every 3 rounds and every round is when an approval by contribution takes place.

Finally, the Results has a survey to which each bot automatically completes and just waits for correct pages to come forth.

The same code that lies within the test.py for every application that does the command line testing also does the browser bot testing and therefore it’s safe to conclude that if a browser bot test completes without a fault, then the game’s logic is compatible with the browser.

### Hardware and Manual Testing

The manual testing will involve simply running the game manually instead of having automated tests complete the game. Manual Testing will be heavily used during compatibility testing to verify that each form works as we expect in each web browser. Manual Testing will include verifying the data that oTree produces every time a session is run, this will be the least tested part of the entire program as it simply loads data from a database which I don’t have much control of as oTree by design puts all the data inside the database by design without me as a user specifying every single round that I want to store the data.

Hardware testing will be conducted to see how the server performs under load. To put strain on the server I will conduct tests with automated testing and large amounts 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 major changes to the amount of resources the server has available will be documented. No other Hardware 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 iteration of the biosecurity game 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, 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 ≤ 0.999). 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 to see that 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 something readable by humans). The operating system you use, shouldn’t matter as long as you have a met these requirements (at the time of writing 02/04/2017):

oTree 1.2.7 (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>

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

And finally refer to here for the note on otree\_tags:

[oTree\_Tags](#_oTree_Tags)

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 to reduce the average being affected by any 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. 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.
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.

The reason that these values were adjusted is because these values adjust the frequency of outbreaks and how much funds each player finished with more than any other values. While one could adjust the revenue each player, $25 is a good value as a player can gain that amount and just as easily lose it with carelessness, any higher the game would be too easy, any lower too hard. As one will see from my results the same rings true for both **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 player.

**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 player 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 player 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 player contributed 75% or $11.25 every round, then players 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, players would lose interest half way through the game as they would have lost money by round 5. A loss of interest would lead to players not caring about the game enough and therefore the data would become useless as they would (arguably) be just like if players were randomly choosing their protection. I say arguably, due to players most likely heading towards $0 as the game progressed meaning that the protection that players would provide would be erratic and towards the lower end. The lowering of protection by all players would result 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 players 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. 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 in percentage 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. Players could let chance decide the amount of protection compared with 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**. A **PC** of 70% is just as bad as the game being too difficult. 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 players 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 player’s funds. In this configuration, players were rewarded for cooperating, as players provided more protection, the frequency of outbreaks reduced. Inversely, when players were uncooperative, players 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 also observed times when players constantly doing $0 or 60% chance that they’re not the source of the outbreak, that players could still end the game with a positive result, some as high as $50. The same was at the other end with 75% of **MP**, players 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**

<<Introduction and overview for test B>>

**Test Specification**

<<The Test Specification lists the requirements whose satisfaction will be demonstrated by the test. It lists the methods tested, and describes the conditions of the test.>>

**Test Description**

<<The Test Description is used as a guide in performing the test. It lists the input data and input commands for each test, as well as expected output and system messages. If you find that you are unable to describe expected output numerically, use a natural language description. A test description consists of

* Location of test (hyperlink to test)
* Means of Control: Describes how data are entered (manually or automatically with a test driver)
* Data
  + Input Data
  + Input Commands
  + Output Data
  + System Messages
* Procedures: The test procedure is often specificed in form of a test script.

**Test Analysis Report**

<<The Test Analysis Report lists the functions and performance characteristics that were to be demonstrated, and describes the actual test results. The description of the results must include the following:

* Function
* Performance
* Data measures, including whether target requirements have been met

If an error or deficiency has been discovered, the report discusses its impact.>>

**Test 3 – Automated Browser Bot Testing**

<<Introduction and overview for test C>>

**Test Specification**

<<The Test Specification lists the requirements whose satisfaction will be demonstrated by the test. It lists the methods tested, and describes the conditions of the test. >>

**Test Description**

<<The Test Description is used as a guide in performing the test. It lists the input data and input commands for each test, as well as expected out put and system messages. If you find that you are unable to describe expected output numerically, use a natural language description. A test description consists of

* Location of test (hyperlink to test)
* Means of Control: Describes how data are entered (manually or automatically with a test driver)
* Data
  + Input Data
  + Input Commands
  + Output Data
  + System Messages
* Procedures: The test procedure is often specificed in form of a test script.

**Test Analysis Report**

<<The Test Analysis Report lists the functions and performance characteristics that were to be demonstrated, and describes the actual test results. The description of the results must include the following:

* Function
* Performance
* Data measures, including whether target requirements have been met

If an error or deficiency has been discovered, the report discusses its impact.>>

**Test 4 – Manual Browser Testing**

<<Introduction and overview for test D>>

**Test Specification**

<<The Test Specification lists the requirements whose satisfaction will be demonstrated by the test. It lists the methods tested, and describes the conditions of the test.>>

**Test Description**

<<The Test Description is used as a guide in performing the test. It lists the input data and input commands for each test, as well as expected out put and system messages. If you find that you are unable to describe expected output numerically, use a natural language description. A test description consists of

* Location of test (hyperlink to test)
* Means of Control: Describes how data are entered (manually or automatically with a test driver)
* Data
  + Input Data
  + Input Commands
  + Output Data
  + System Messages
* Procedures: The test procedure is often specificed in form of a test script.

**Test 5 – Server Performance**

<<Introduction and overview for test D>>

**Test Specification**

<<The Test Specification lists the requirements whose satisfaction will be demonstrated by the test. It lists the methods tested, and describes the conditions of the test.>>

**Test Description**

<<The Test Description is used as a guide in performing the test. It lists the input data and input commands for each test, as well as expected out put and system messages. If you find that you are unable to describe expected output numerically, use a natural language description. A test description consists of

* Location of test (hyperlink to test)
* Means of Control: Describes how data are entered (manually or automatically with a test driver)
* Data
  + Input Data
  + Input Commands
  + Output Data
  + System Messages
* Procedures: The test procedure is often specificed in form of a test script.

**Test Analysis Report**

<<The Test Analysis Report lists the functions and performance characteristics that were to be demonstrated, and describes the actual test results. The description of the results must include the following:

* Function
* Performance
* Data measures, including whether target requirements have been met

If an error or deficiency has been discovered, the report discusses its impact.>>

**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.  
  
Theoretically, the game should work no matter what OS you’re running on your own personal device as the 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.

## 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:

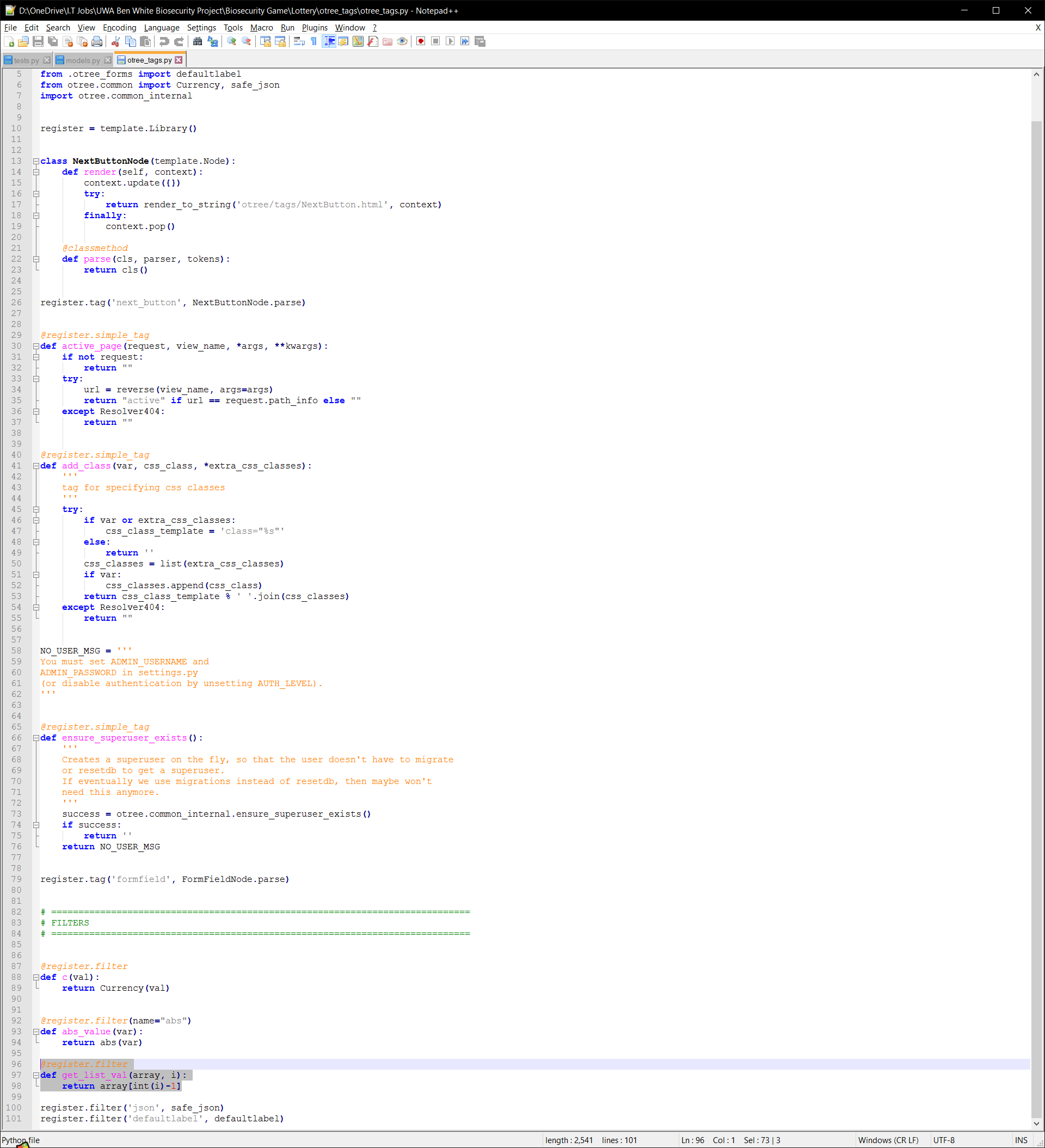
1) Assuming you cloned the repository or have the program files available to you, inside the Lottery app folder named *Lottery*, there should be a directory named otree\_tags, go in there.  
Copy the otree\_tags file inside a paste the file inside this directory:

<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 in order 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:



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)