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](#_Toc472538762)

[Testing Strategy 3](#_Toc472538763)

[Test A 3](#_Toc472538764)

[Test Specification 3](#_Toc472538765)

[Test Description 3](#_Toc472538766)

[Test Analysis Report 4](#_Toc472538767)

[Test B 4](#_Toc472538768)

[Test Specification 4](#_Toc472538769)

[Test Description 4](#_Toc472538770)

[Test Analysis Report 4](#_Toc472538771)

[Test C 5](#_Toc472538772)

[Test Specification 5](#_Toc472538773)

[Test Description 5](#_Toc472538774)

[Test Analysis Report 5](#_Toc472538775)

[Test D 6](#_Toc472538776)

[Test Specification 6](#_Toc472538777)

[Test Description 6](#_Toc472538778)

[Test Analysis Report 6](#_Toc472538779)

[Test Materials 6](#_Toc472538780)

**Objectives and Test Summary**

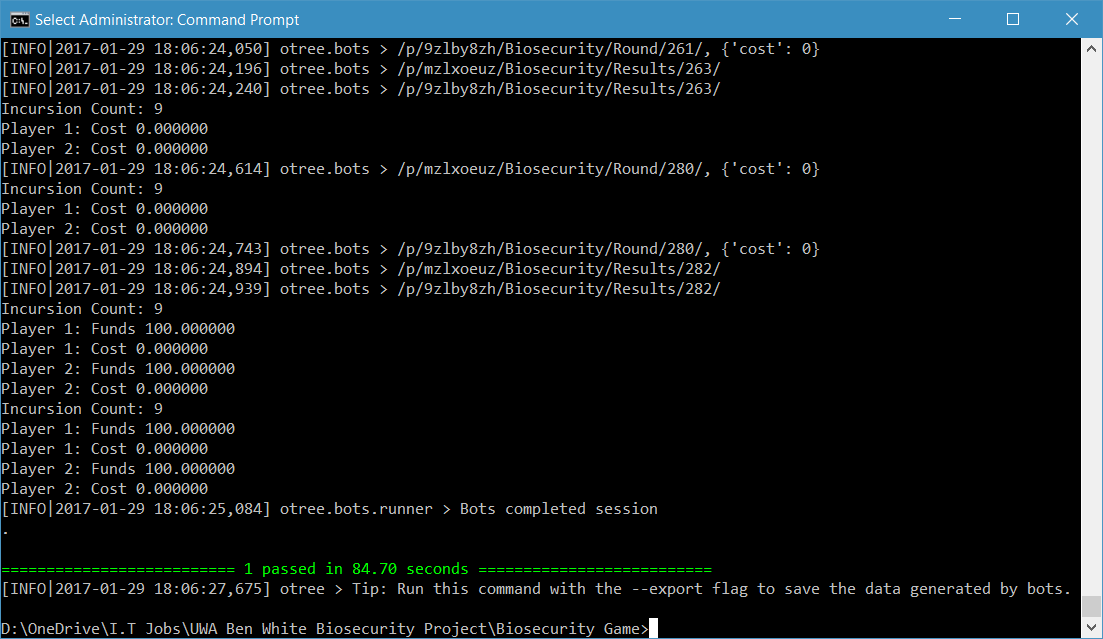
This document is primarily focusing on testing all aspects of the system, web site and the program underlying the website biosecurity.are.uwa.edu.au. Here it will describe the behaviour of the system and the underlying logic that decides when incursions occur. Testing will primarily focus on these aspects:  
  
1) How incursions are determined and the amount of times incursions 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 as just have to make sure the data goes in through the form as there is no real way to verify that a person gets the 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 incursion 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.

TwoSelfishTwoGood: 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 incursion is 50% (refer to Test – Incursion Count), 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 bots 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 will 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 A**

<<Introduction and overview for test A>>

**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 B**

<<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 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 C**

<<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 D**

<<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.