Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 11/03/2014 | 1.0 | Initial version | A. Emre Ünal |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**TABLE OF CONTENTS**

Revision History 1

1- Introduction 3

1.1- Document overview 3

1.2- Abbreviations 3

1.3- References 3

1.4- Conventions 3

2- Tests preparations 4

2.1- Hardware preparation 4

2.2- Software preparation 4

2.3- Other test preparation 4

2.4- Safety, security and privacy precautions 4

3- Tests descriptions 5

3.1- Choose sub-section name 5

# Introduction

## Document overview

This document is the software test plan of the TicTacToe game software development project. It contains the description of tests.

## Abbreviations

The TicTacToe game software project: “the game” or “the software”

The TicTacToe game software project’s GitHub repository page: “the repo”

## References

| # | Document Identifier | Document Title |
| --- | --- | --- |
| [R1] | SDPv1.2 | Software Development Plan of the TicTacToe game, version 1.2. |

## Conventions

JUnit test code implementation will follow the standard Java Programming Language code conventions, found on <http://goo.gl/srJN2t>.

# Tests preparations

This section contains tasks and recommendations before executing tests.

## Hardware preparation

The game requires nothing more than a functional machine for the local multiplayer mode testing.

For remote multiplayer mode testing, it is required to have a basic TCP/IP stack (which would require a network card) for 2 localhost instances.

## Software preparation

The client must have Java Runtime Environment v1.7, JUnit v4 and Eclipse v4.2 installed to run the game. To run local multiplayer mode tests, that’s all that is required.

To run remote multiplayer mode tests, a basic TCP/IP stack that supports localhost loop-back is required.

## Other test preparation

None.

## Safety, security and privacy precautions

None.

# Tests descriptions

You may organize tests by groups or list them in alphabetical order.

The most effective way is to list them in chronological order of execution for each phase. Test engineers won’t waste time searching for tests in the document.

## Local Tests

Section name may be:

* or category of test xxx
* or some other logic to group tests

Describe each test with the pattern below.

For most of tests, only a subset of fields in the table is used, mark N/A (non applicable) the unused fields.

|  |  |  |
| --- | --- | --- |
| **Test ID** | **Some ID assigned according to the naming chosen convention** |  |
| Test description | A brief description of the test |  |
| Verified Requirement | SRS-REQ-001 | Verification method: I,A,D,T  See the explanations and examples on the next pages |
| Initial conditions | The state of software before test | You may reference a procedure or it may be the result of previous test |
| Tests inputs | Input data from any test tool, input files name and location | You may reference a procedure to use the test tool |
| Data collection actions | Recording and post processing of output data | You may reference a procedure to record data with a test tool |
| Tests outputs | Output data files names and location, logs … | Give unique name out output data files. |
| Assumptions and constraints | If any, may be limited access to a tool, license … |  |
| Expected results and criteria | List here the results of test | And the criteria to evaluate the result |
| **Test procedure** |  |  |
| **Step number** | **Operator actions** | **Expected result and evaluation criteria** |
| 1 | Start foo | Foo is started |

Verification Methods:

* Inspection (I): control or visual verification
  + Control of the physical implementation or the installation of a component. The control verifies that the implementation or the installation of a component is compliant with the requirements of diagrams.
  + Control of the documentation describing a component. The control verifies that the documentation is compliant with the requirements.
* Analysis (A): verification based upon analytical evidences
  + Verification of a functionality, performance or technical solution of a component by analyzing the data collected by tests in real conditions, by simulation of real conditions or by a analysis report.
  + Analysis of test data or of design data is used as appropriate to verify requirements.
  + The verification is based upon analytical evidences obtained by calculations, like modeling, simulation and forecasting.
  + Analysis is used when an acceptable level of confidence cannot be established by other methods or if analysis is the most cost-effective solution.
* Demonstration (D): verification of operational characteristics, without quantitative measurement
  + Verifying a requirement by demonstration implies that the required functionality specified by a requirement is complete.
  + Demonstration is used when quantitative measurement is not required for verification of the requirements
  + Demonstration includes the control of the technical solutions specified by the non-functional requirements.
* Test (T): verification of quantitative characteristics with quantitative measurement
  + Verifying a functionality, performance or technical solution of a component by executing testing scenarios in predefined, controlled and traceable testing conditions.
  + Tests require the use of special equipment, instrumentation, simulation techniques, or the application of established principles and procedures,
  + Data produced during tests is used to evaluate quantitative results and compare them with requirements.

Examples of tests methods:

Inspection:

* Verify that the color of background is blue,
* Verify that the user manual has the CE mark on its cover
* Verify that the PC has 4Gb memory
* Verify that firmware version on electronic card is 1.0.1

Demonstration

* Verify that when the user closes the window, a confirmation message appears
* Verify that the file is saved in the output directory
* Verify that the result is shown
* Verify that if a value is out of range, a warning is displayed

Analysis:

* Verify that the statistical distribution of results of xxx algorithm is a Gaussian with mean=x and stdev=y, when input data are blah blah
* Verify that the linear regression of results of xxx algorithm is a line for which value is 1 on the y-axis, at zero on the x-axis,

Test:

* Verify that a file of 1Gb is processed in less than 3s
* Verify that the response time of the server is 15ms with 20 simultaneous requests

Examples of tests:

Inspection

|  |  |  |
| --- | --- | --- |
| **Test ID** | **T-SRS-REQ-001** |  |
| Test desc. | Verify that the user manual contains the intended use |  |
| Verif. Req. | SRS-REQ-001, | Inspection |
| Init. Cond. | XXX Software is started and idle |  |
| Tests inputs | N/A |  |
| Data collection | N/A |  |
| Tests outputs | N/A |  |
| Assum & constr | N/A |  |
| Expected results and criteria | The user manual contains the intended use | Same IU as the one found in risk analysis report ref. xxx |
| **Test procedure** |  |  |
| **Step number** | **Operator actions** | **Expected result and eval crit** |
| 1 | Open user manual, doc ref xxx, section: Introduction | The IU is located in the introduction, the text is the same as the one found in risk analysis report ref. xxx |
| 2 | Open Help/User manual | The online user manual is opened |
| 3 | Go to Section 1 | The section 1 is displayed, it contains the IU as the one found in risk analysis report ref. xxx |

Demonstration

|  |  |  |
| --- | --- | --- |
| **Test ID** | **T-SRS-REQ-001** |  |
| Test desc. | Verify that the xxx software allows to choose on protocol in the list of recorded protocols |  |
| Verif. Req. | SRS-REQ-001, | Demonstration |
| Init. Cond. | XXX Software is started and idle |  |
| Tests inputs | N/A |  |
| Data collection | N/A |  |
| Tests outputs | N/A |  |
| Assum & constr | N/A |  |
| Expected results and criteria | Chosen protocol is selected | See last step for criteria |
| **Test procedure** |  |  |
| **Step number** | **Operator actions** | **Expected result and eval crit** |
| 1 | Open the list of protocols | The list of protocol is displayed |
| 2 | Select a protocol | The protocol summary is displayed |
| 3 | Validate the choice | The protocol is selected. The name of the protocol is displayed at the top of the screen. |

Analysis

|  |  |  |
| --- | --- | --- |
| **Test ID** | **T-SRS-REQ-001** |  |
| Test desc. | Verify that the xxx software computes the xxx result with yyy algorithm |  |
| Verif. Req. | SRS-REQ-001, | Analysis |
| Init. Cond. | XXX Software is started and set in “Verbose” mode |  |
| Tests inputs | Script xxx to generate data | See procedure xxx on how to use script |
| Data collection | Log file xxx-verbose.log and post-processor script |  |
| Tests outputs | Output file post-processor script and graph: xxx-post-process.txt and xxx-post-process.png | See procedure xxx on how to use script |
| Assum & constr | Do not run data generator script with more than 1000 loops, or post process is too long |  |
| Expected results and criteria | xxx-post-process.txt and xxx-post-process.png | The post process graph is a gausian  The post process values are:  Chi2 = … Mean = … stdev= … |
| **Test procedure** |  |  |
| **Step number** | **Operator actions** | **Expected result and eval crit** |
| 1 | Set log mode in “verbose” and restart software | Software is started, log file name xxx-verbose.log is created in xxx directory |
| 2 | Run xxx data generator script | Data binary file generated |
| 3 | Open data file with software | Software processes the file and a msg is displayed when finished |
| 4 | Run xxx post process and wait end of post processing | xxx-post-process.txt and xxx-post-process.png are generated |
| 5 | Open files | The post process graph is a gausian  The post process values are:  Chi2 = … Mean = … stdev= … |

Test

|  |  |  |
| --- | --- | --- |
| **Test ID** | **T-SRS-REQ-001** |  |
| Test desc. | Verify that the xxx software receives data from yyy |  |
| Verif. Req. | SRS-REQ-001, | Test |
| Init. Cond. | XXX Software is started  YYY simulator is started | See procedure yyy on how to use YYY simulator |
| Tests inputs | N/A |  |
| Data collection | N/A |  |
| Tests outputs | N/A |  |
| Assum & constr | YYY simulator work only in the range of zzz |  |
| Expected results and criteria | Data received and processed | See below |
| **Test procedure** |  |  |
| **Step number** | **Operator actions** | **Expected result and eval crit** |
| 1 | Send data with YYY simulator | Simulator displays msg “data sent” |
| 2 | Open xxx window of xxx software | Sent data are displayed, with date-time of reception |
| 3 | Press run button | Sent data are processed, result is displayed.  Value of result is … |

Test

|  |  |  |
| --- | --- | --- |
| **Test ID** | **T-SRS-REQ-001** |  |
| Test desc. | Verify that a user displays result in less than one minute |  |
| Verif. Req. | SRS-REQ-001, | Test |
| Init. Cond. | XXX Software is started |  |
| Tests inputs | N/A |  |
| Data collection | Stopwatch |  |
| Tests outputs | N/A |  |
| Assum & constr | Users shall have received basic training to use |  |
| Expected results and criteria | Results displayed in less than one minute |  |
| **Test procedure** |  |  |
| **Step number** | **Operator actions** | **Expected result and eval crit** |
| 1 | For 3 users, run the main use scenario 3 times. | Note duration of execution for each run of each user. |
| 2 | Compute the mean of durations | Mean is less than 1 minute. |