MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

TEST SPECIFICATION REPORT

CENG 492

MOON SOFTWARE SOLUTIONS

1. INTRODUCTION

1.1. PURPOSE OF TEST SPECIFICATION REPORT

This document is a guideline for testing software product which is being developed and maintained by MoonSoftwareSolutions. In very short terms, our software product is a 3-D online multiplayer game with textures, sounds, multiple players and more. The purpose of this document is to give information about the scope, method, resources and strategies of testing management as preparation for final release of the game. We are going to do extensive testing in order to;

- Ensure that product performs each function as intended.
- Demonstrate the product is free from defect.
- Improve software reliability, quality, and maintainability.
- Show that all internal components have been adequately exercised.

The document also identifies the modules and features to be tested, testing tasks to be performed and personal responsibilities for each task, module, item or feature.

1.2. SCOPE OF THE DOCUMENT

The document covers steps to be done during testing. This process includes test items, testing approach and pass/fail criteria for test items. While reading this document, it will useful to analyze taxim requirement analysis report and detailed design report to understand testing activities that are supposed to ensure functionality better.

In test items part, it will be explained which items or modules will be tested. There are mainly 11 modules. In the second part, testing approach will be stated namely which testing methods will be used during tests. There are many different kinds of testing methods but we are going to use general ones namely white box testing and black box testing. Also unit testing, integration testing and system testing which are done according to different bases will be explained. In the last part will be pass/fail criteria. These criteria are the minimum necessary requirements needed for game.

1.3. REFERENCES

The following documents were used for the preparation of this report:

- MoonSoftwareSolution Detailed Design Report
- MoonSoftwareSolution Requirement Analysis Report
- MoonSoftwareSolution Configuration Management Plan
- IEEE Standard for Software Test Documentation

2. TESTING PROCESS

2.1. TEST ITEMS

All the project items will be tested seperately but since the product include these modules, project as a whole will also be tested. The modules are independent as much as possible so that in case of bugs the correction will be made easier. The main modules are as follows;

- Graphics
- Sound
- GUI

- Physics
- Network
- AI
- Database
- Gamelogic
- File Control
- Gameplay
- Training

2.2. TESTING STRATEGY

White box and black box testing are terms used to describe the point of view a test engineer takes when designing test cases.

White box testing (a.k.a. clear box testing, glass box testing or structural testing) uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to identify all paths through the software. The tester chooses test case inputs to exercise paths through the code and determines the appropriate outputs.

Black box testing takes an external perspective of the test object to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid input and determines the correct output. There is no knowledge of the test object's internal structure.

These methods of test design are applicable to all levels of software testing: unit, integration, and system testing. The higher the level, and hence the bigger and more complex the box, the more one is forced to use black box testing to simplify. While this method can uncover unimplemented parts of the specification, one cannot be sure that all existent paths are tested.

There are generally 3 main levels of software testing carried out:

➢ **UNIT TESTING:** Unit testing is a test in which each unit (basic component, module) of the software is tested to verify that the detailed design for the unit has been correctly implemented.

  There are four major points to take care of while testing game units. They are:

  ✓ **Functionality Tests:** Functionality tests are the tests which are done in order to verify if a feature is working according to its design specification.

    Functionality tests are used in taXim project to examine the extent to which the main modules of the program meet expected functional requirements.

  ✓ **Performance Tests:** Performance tests are the tests conducted to evaluate the compliance of a system or component with specified performance requirements.

    Performance tests are used in taXim project in order to determine if the project as a whole(when all its modules are working) satisfies the performance constraints defined in the Detailed Design Report of Moon Software Solutions group.

- ✓ **Stress Tests:** Stress tests are the tests used for determining the stability of a given system under extreme conditions.
  Stress tests are used in taXim project to see the results of extreme use cases for some modules of the game. Also these cases are applied to the whole project in order to see the behavior of the program.

- ✓ **Edge Tests:** Edge tests are the tests used for determining the weaknesses of a given system.

  Edge tests are used in taXim project to detect the possible bugs in the game.

- ➢ **INTEGRATION TESTING:** Integration testing is a testing in which progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a whole.

- ➢ **SYSTEM TESTING:** System testing is a kind of testing in which the software is integrated to the overall product and tested to show that all requirements are met.

2.3. UNIT TESTING

2.3.1. Graphics

Graphics unit is the most important part of our project since taXim project is a 3D computer game at first sight. Graphics concept is a wide area, thus needed to be tested very carefully. We will test graphics unit in all four subgroups of unit testing. Graphics testing will be done by Yusuf OSMANLIOGLU.

As functionality test, we need to test whether if our graphics engine works properly. For this purpose we need to check the answers of the following questions:

- Does it load .ive files properly?
- Does it show textures of the objects qualified enough?
- Do cameras work without twinkling?
- Do objects disappear when they are far enough?
- Does engine remove objects from drawable list, when they are out of the cameras' scope?
- Does it show different levels of detail of objects as it is supposed to do?

As performance test, we need to test if our graphics engine can handle loading and rendering of numerous objects meanwhile. Performance of the graphics is very important for project taXim because we need to keep frame rate higher than 30fps for the sake of being "real time". For this purpose we need to check the coming points:

- Does it able to load a subpart of the map without critical performance loss?
- Does adding and removing of drawables frequently affect the performance seriously?
- What quality limit should be put to objects and textures so as to have reasonable game play with reasonable performance?

As stress test, we need to test what are the upper limits for the performance, with respect to object quality, texture quality and object number. For this purpose we need to find answer to coming questions:

- What is the upper limit for the number of objects that can be loaded, without affecting other units?
- What is the upper limit for the number of object that can be rendered in each frame?

- What is the upper most quality level for the 3D object in LOD1(level of detail)?
- What is the upper most quality level for the textures of the objects?

As edge test, we need to test what are the bottlenecks for project taXim with respect to graphics engine. For this purpose we will test the points we are suitable for bugs. So coming questions must be answered:

- What happens when the limits found in stress test are exceeded?
- What happens when minimum hardware requirement is not found on the running system? How should program behave?

2.3.2. Physics

Physics is also another very important part of the project taXim. This is mainly because the reality feeling comes from the application of physics rules in the game. Since perfect computations are impossible for physics, physics engine makes approximations. Due to the time limit for making computations, reasonable results for physics are a serious trade off. So testing of physics are important to keep this approximation level in a reasonable rate. Test related to physics will be made by Osman ŞAHİN.

As functionality test, we need to test if basic physics rules are applied as they are supposed to be. For this purpose we need to answer the following questions:

- Does gravity works fine?
- Does acceleration of gravity reasonable?
- Does collision of vehicle with the terrain and environmental objects are working properly?
- Do tires of the vehicle obey physics rules? Do their motor power modeled realistic enough?
- Does parameters of motor, vehicle mass, tire mass, tire turning angle limit etc. adjusted and working properly?

As performance test, we need to test how serious does physics affect the overall performance. For this purpose, we need to consider following points:

- Does changing setting collision on/off frequently for objects affect performance seriously?
- How much does it affect the performance having mesh collision for the vehicle instead of box collision?
- How much do physics calculations affect the frame rate performance?
- How much does frame rate affect the physics calculations performance?
- How much does it affect the performance to increase the precision of calculation for collisions and ODE joints?

As stress test, we need to test what are the boundaries for reasonable physics applied with highest performance. For this, we need to answer following questions:

- What is the upper most precision value to be applied for joints with reasonable results?
- What is the upper most number of objects that are collidable in one environment?

As edge test, we need to test implementation of physics engine in order to find weak rings. For this purpose we need to consider following points:

- What happens to physics when frame rate drops down 30fps limit?
- What happens to physics when the above limits are exceeded?

2.3.3. Network

The types of tests that are to be applied to network modules are Functionality Test, Stress Test and Edge Test. The test scenarios are built by Mustafa Özpınar.

In Functionality Test, the unit is tested with both white-box and black-box testing. Considering the former, all the separate parts of the module is tested. For example, whether it can handle the connections perfectly, packets are sent and received correctly, the connection with database and also the operations are successful. With black-box testing, the module is given some specific inputs and the output is compared with the expected one. For instance, a user logins or signups, then the corresponding table in the database is checked for the inserted data.

Stress test is also important in this module since the game is a multiplayer one. Many connections are setup at the same time by means of a test program and the performance and the response of the module is tested.

By edge-testing, the unit is checked for some unexpected or extreme inputs. This is required for stability of the system. To illustrate, the system should not corrupt even if a packet cannot be parsed. All the functionality and edge tests are made during the development of the module. But the stress test will be done after integration of modules.

2.3.4. Sound

Only functionality test can be applied to sound manager in our project. The test scenarios are created by Mustafa Özpınar.

The scenario includes testing the unit for playing more than one sound at a moment, stopping, pausing, playing and looping them in a specific time interval (required for GUI sounds). Also it contains lowering and increasing the volume of each sound whenever needed. Not so much vital but it is better to test the module for flushing the sounds after stopping to make performance better.

2.3.5. Game Play

Game Play unit stands for the game logic. Namely they are serving passenger and its effects such as earning money, fuel consumption, traffic punishments, taxes, buying/selling car, car accessories, repairing car etc. Game should pass from state to state without any error so as to have consistent game play. This part will be tested by Yusuf OSMANLIOGLU and Osman ŞAHİN.

As functionality test, we need to test if all game states are working as they are intended to work. Namely they should follow each other in a predefined manner. For this purpose we need to take care of the cases such as:

- Does game enter to buying fuel state when handbrake is hold in the gas station? In the same manner does it enter to buy car state, repair car state or buy accessories state when handbrake is hold in car market, car repair shop and car accessories shop respectively?
- Does game enter to serve passenger state when vehicle is entered to the area near the passenger?
- Does car stop when fuel tank is empty?
- Does car stop when damage meter shows 100%?

- Do related menus fired when game state is changed? (i.e. buy fuel GUI opened when entered to gas station and hold handbrake and so on.)

Since game will be in one state at any time, performance test due to game play is not applicable. The thing that we must take care of at this point is not to have different states at the same time which will result to inconsistencies.

Stress test may be applied so as to see if our implementation allows player to be in different states at the same time. For this purpose handbrake should be hold in unrelated places such as in the middle of the road so as to see if game state changes. Since this is hard to implement by white box testing, it will be ideal and better to make this test with black box testing. By this way, it will be possible to test various points on the whole map.

Edge test may be applied to cases where two distinct states are very close to each other. For example we will need to test the case where fuel tank is completely empty but vehicle managed to reach the gas station somehow. In such a case buy fuel menu should be fired. But if player ran out of money, another menu indicating that s/he is ran out money should be fired instead of buy fuel menu.

2.3.6. AI

Functionality Test and Edge Test can be performed to test the AI module. The testing scenario is created by Osman Şahin.

The AI parts of our project are traffic control, passenger control, traffic light control, collision control (such as damage rate, guilty etc.) and etc. In traffic control, whether the traffic vehicles are created correctly, whether the vehicles drive along the roads correctly and whether they collide are tested. Considering the passenger control, the randomly generation of a passenger with correct destination points, behavior of the passenger towards the act of the driver (pleasured or disappointed) are tested. Traffic light control is tested by checking the transition from one state to another correctly, whether the game can understand that the player passed under red, yellow or green lights and thus punishing him. AI in collision control is tested by comparing the expected results of the collision with the collision of a user's car, determination of the car damage, the guilty and innocent users and punishment of them according to these results.

Edge Test can be performed by controlling the game status when there occurs an error while generating, loading cars, passengers and etc. For example a corrupted 3D model is placed and then the result can be checked. After all these tests it is vital that game is in a stable state.

2.3.7. GUI

Graphical user interface is the most attractive part of the games since it includes visual effects as well as sound effects. There are mainly two types of GUI : static and dynamic. Game entrance menus are static types for example. Speed, career and map menus are dynamic types which means they will change while playing. Their tests will also be different. For instance, while testing static menus we will not run the game completely in order not to spend time on compiling all project, but testing dynamic will need database module. This part will be tested by Osman ŞAHİN.

As functionality test, we need to test whether if GUI module works properly. For this purpose we need to check the answers of the following questions:

- Does it go to correct submenus when a button pressed?
- Does it synchronize with sounds if any?
- Are the parent/child relationships of the menus correct?
- Are the hidden menus are opening with according action?
- Does the resizing property of windows function correctly?

As performance test, we need to test whether the performance is reducing if we loads high quality images and sounds to the menus. Besides, we need to test if we destroy some static menus which will be not used after first time is the frame number increasing.

2.3.8. Training

Training menu is an offline menu. Environmental behaviors should be generated by the client side in all training scenarios. So scenarios must be consistent in their own scope and also must be consistent with the online game play. This module will be tested by Adem Ali YILMAZ.

As functionality test, we need to test each scenario one by one. And see whether if they operate according to their generation purpose. For example:

- Does serve passenger scenario works properly?
- Can player serve the passenger without any problem?
- Is it easy to learn and manipulate?

These questions should be answered for all scenarios.

Since other parts of this test are done in other modules performance test, stress test and edge test are not applicable for this module.

2.3.9. Database Connection

This module is only on the server side of our project so it is mainly discussed with network module. Test scenarios for this unit are created by Mustafa Özpınar. Functionality Test, Stress Test and Edge Test are applicable to this database module.

Functionality tests are mainly made away with black-box testing. Insertion, deletion and update operations are checked for the given input parameters.

Stress test is applied by connecting multiple clients to same database and making operations continuously.

Edge test can be carried out by giving wrong parameters having different types then expected and checking the stability and output of the program. The resulting bugs are fixed after testing. Mostly these test functions are considered and executed during development of the module.

2.3.10.File control

In this module, mostly the file operations required to save and load the game settings are tested. Only the Functionality Test is applied to this unit, which the scenarios are created by Osman ŞAHİN.

In Functionality Test, read, write and update operations of the module is tested. This is important for our game. This module is used to save the user options in the game such as keyboard configuration, video settings, audio settings, radio settings and etc. If this does not work properly, user will have to set and apply his own settings every time he starts the game, which is not acceptable by him. By giving some inputs to the module functions, the produced options file is checked. Also reading options is controlled by creating some options file manually.

2.4. INTEGRATION TESTING

2.5. SYSTEM TESTING

At the end we need to test the overall game working on a PC that is running Windows XP. We need to test all the functionalities altogether. First of all if inputs are taken right with both keyboard and mouse, display is made right. And internet connection works fine, no back door left open. Then whether sound is being played properly meanwhile. After that whether menu selections resulting to right game states. Then physics calculations are done right. And so on. We need to look at the system as a whole, and test our software on that system.

3. CONCLUSION

Testing process is the heart part of all software projects. So it is vital that all the modules in the product function without any bug. Thus, we are going to do a number of testing to see whether our modules are error-free. At the same time, our testing will go on iteratively since in this one-month period time we will be adding new codes to our modules which will be need retesting. As a result the quality of the project will increase.