Middle East Technical University
Department of Computer Engineering

CENG491
Computer Engineering Design II
Test Specification Report

ÖZGÜR YAZILIM

Özgür Özgür
Fırat Erdoğan
Onur Demircan
Abdulkerim Mızrak
Mehmet Emin Ulusoy

1. INTRODUCTION

1.1. Goals and Objectives

Being a 3D MMOG, Treasure Hunt consists of multiple modules. These modules are handled under two main titles, which are graphics and networking. In addition to these modules, Treasure Hunt brings the need of physics implementation, which is related to both of these main modules (i.e. graphics and networking) and the other submodules. Being a game, Treasure Hunt must be implemented in a logical way, and bugs and problems should be eliminated as much as possible to be able to serve users efficiently.

This brings the requirement of an intense testing on every module (like network, graphics, sounds, physics, etc.), and on the overall architecture as a whole to be able to see the problems derived from integration of the modules. Besides the small tests and debugging processes during development, the testing of the whole program will let us reach our goal of developing a bug-free game.

1.2. Scope of Document

This document involves the types of tests, and the methods of these testing processes. Since the testing issue is something that walks together with the development, we actually did some testing up to a point. The results and logs of these tests are also involved in this document.

But approaching to the end, a more intensive testing phase becomes compulsory, and this document gives information about the strategy to be used, and specifications of the testing process.

1.3. Statement of Testing Plan Scope

Treasure Hunt will be (and is being) tested by using multiple types of tests, and different strategies. Consisting of multiple modules, makes integration testing the most important one. All of the tests are going to be mentioned in next part of this document. They can be briefly explained as follows:

Type of test	**Description**
Unit Testing	This testing strategy involves testing process of modules as if they are standalone parts.
Integration Testing	The test type to check the consistency between different modules.
Validation testing	This test is applied to see if the requirement and design goals are achieved.
Performance testing	The tests to see the current performance of the game, and to improve the performance.
Stress testing	The test to see the maximum capability of the game by means of embedded characters & objects, and the networking.
Alpha testing	The test applied by non-programmers of the game, but with all of the requirements supplied.
Beta Testing	The test applied by end-users to reach a more stable version.

1.4. Major Constraints

Besides the testing & debugging process during the implementation, testing is a main phase on its own in the development process.

Taking into account the deadline of the project, time becomes the most important of the major constraints in testing. Still having work to be done, the remaining time should be

carefully organized to be able to achieve the test phase and get rid of the existing, (and also possible) bugs and problems.

Another important constraint is the usefulness of the testing code. Before testing a module or the overall system, we should take into account the trade-offs about writing testing codes for the modules. It should be done if it is compulsory, but generally the problems arise at the integration phase -which cancels the need of testing code-.

Hardware is another constraint about testing phase. Although this becomes important at beta testing, it also shows the actual hardware requirement of the game for the end-user.

The last constraint about testing is the staff. Our team consists of 5 members, and each member should try to come up with bug-free codes. The limitation on the number of people makes the staff another constraint, since the testing should go hand-in-hand with the development.

2. TESTING STRATEGY AND PROCEDURES

The procedures and strategies that we will follow will be stated in this part.

2.1. Unit Testing

In this project unit testing is essential in three areas: the game engine, game network, and game data. Since the time-cost and hardware resource of testing are very critical for our project, we will try to do the best of us while doing testing. For example we can not do a test with more than 10 computers.

2.1.1. Game Engine Tests

Since our game is played with multiplayer, there are two main game engines: Client-side game engine and Server-side game engine. Obviously, to test both engines together in a complete way is not possible for us but we will test at least the minimum requirement to be satisfied in our game. Since the client-side game engine modules can be tested easier than the server-side game engine, probably we will do better tests on the client-side game engine.

Black-box testing is our main test method while doing engine tests, because in a real-time system it is sometimes impossible to satisfy the test condition properly.

Since now, we have applied unit testing on these modules:

1. Character
2. Map
3. Sound
4. Collision Detection

To test the character, some cameras viewing the character are created and from different perspective the behaviors of the character are observed.

To test the map again we used some different cameras with different perspective.

To test the sound we use different sounds and attach each sound to a specific event or buttons and then by pressing a key or button we expect the correct sounds.

To test collision detection, we played with our character in the game map. We try to hit the walls of the architecture.

The above modules are now working correctly and they integrated to each other.

After now we will do the unit test on puzzle manager module, treasure-hunt objects manager module and treasure-hunt food manager module. Since these three modules are related with both server-side and client-side game engine it will be more difficult to

test these modules than the others. Also, since the Character module is related to the objects and foods manager modules, the Character module will be updated soon.

2.1.2. Game Network Tests

As mentioned above our game is a multiplayer game so that the network is one of the most important parts of the game. Testing network is the most costly testing part of our game. Lack of both time and resource limit us to test network properly. However so far we construct a network infrastructure for our game. For example, chat module is working very well. Sending and capturing the position of the character is also finished. After now we will focus on to adopt the network to the futures that will be added.

2.1.3. Game Data Tests

Since our game is a 3D game the importance of 3d models, textures and audio resources is increasing. Beside the difficulties to create such resources, the game performance is highly affected by them. To create the resources we use some tools and programs such as 3ds Max 7, Photoshop, and Sound Effect Maker. Also these tools are very efficient for us to test each data that we created. Before using a data we test them in these programs and try to get rid of possible problems of that data.

The production of data is easily managed by group members but the efficiency is a serious problem. Because of that, sometimes we do not use some well designed resources that need high system requirements.

2.2. Integration Testing

We integrate a module in to the game when we test this module so that, we are sure that this module have enough maturity for the project. After the integration, we test the overall game to see the effects of newly added module to the game. This process is still used by us.

The method used for integration testing is sandwich testing method. Because all project members can not do his duty on time and in a complete manner, and each task ends without any importance or priority based. So that, bottom-up or top-down methods are impossible to use for us during the implementation. We integrate each module that is finished by the member who is responsible for.

We sometimes use black-box and white-box testing methods for integration of some specific modules also, but the main method that is used for integration testing is sandwich testing method.

2.3. Validation Testing

Validation test is a testing procedure which we have to do before any release. The validation tests will be performed into two main groups: Requirement validation and Design validation.

2.3.1. Requirements Validation

Since we have to supply all the requirements that we accept in the design report, all the tests for requirement validation will be done with black-box testing method. Below we will list the functional requirements of our game and the way how we will test them.

Game Menus:

The place of the menus, the place of menu items, the easy-to-use capability of menus will be taken into consideration. Each button will be clicked, each text will be written with some random texts. Each item will be tested whether it provides the expected functionality.

Game Flow:

The game will be played by some number of players and it will be tested during this run. The connection of a player will be blocked by cable disconnection so that testing the response of the server.

Testing the object taken by some player whether that object disappear from the other player game environment.

Whether the player's location in the game is correct or not will be tested during test runs of our game. One player will go to a specified location in the environment and the others also will be said to go there.

Character:

The game character will be tested by a test run by which we will do some experiments with that character. We will try to hit the walls, try to collect objects and foods.

All Engines:

We will do some experiments on all the modules during the test run. For example, we will send long string of message to the network engine during the chat, we will go to the options menu and try to lower the sound, we will not collect food so that we will expect some warning, etc.

2.4. High-Order Testing

To provide security for player's machines we have to test some extreme cases also.

2.4.1. Performance Tests

Our game frame rate have to be at least 24 fps, otherwise the physics of the game goes unexpectedly. We will find maximum number of objects, characters, and models that can be managed by a computer easily.

2.4.2. Stress Tests

We will add lots of characters, models and objects until a crash is occurred. So that, we will prevent a client machine to crash.

To test the network we will try to find as much computer as we can find and try to find how much client the server can serve.

2.4.3. Alpha and Beta Tests

Since our project is multiplayer, it is difficult to find tester who can test the game exactly at the same time, but we will try to do alpha and beta tests by ourselves and with some close friends and relatives. We will note the bugs during the test runs. And we will want our friends' and relatives' opinion about the game.

3.Test Record Keeping and Log

Since we are using as graphics engine Delta3d it has its own logging procedure.These are classes dtUtil::LogFile class dtUtil::Log class which the engine uses for all of its logging needs. The log file is formatted using html tags; therefore, any browser should display the log without any problems. There are different types of log messages that help us to find the reason of the problem easily and more quickly.

Log is a simple utility class for managing log messages in our code. It's the Delta3D way to put debug/warning/error logging behavior in our application without using printf or cout or std::cerr. dtUtil::Log lets us add permanent logging behavior that we can turn on and off.

The different types of log messages.

Member Enumeration Documentation

Enumerator:

LOG_DEBUG

LOG_INFO

LOG_WARNING

LOG_ERROR

LOG_ALWAYS

The log level is adjusted according to the member enumaration of log messages.We can assign level of logging that will be logged by logmessagetypes.If we set the lowest level to the LOG_DEBUG, all messages will be sent.On the other hand if we set the level to error only errors will be sent.

OutputStreamOptions

enum dtUtil::Log::OutputStreamOptions

Enumerator:

NO_OUTPUT	Log messages don't get written to any device.
TO_FILE	Log messages get sent to the output file.
TO_CONSOLE	Log messages get sent to the console.
STANDARD	The default setting.

We can define output stream options as shown in the table.We can write them in an output file or console or we don’t write anywhere.These options are available by output stream options.

3.1 Test Report Forms

We will use Delta3d's logging classes to get information about tests,test results execution progress and bugs.We have divided our log files for all modules.For example Mehmet Emin will be responsible for the log report of CEGUI which shows the defects about interfaces.On the other hand serverlogfile and clientlogfile will be interpreted by network team.
We can observe some test reports from the previous work:

CEGUI.txt:

03/05/2007 15:07:57 (InfL1) CEGUI::Logger singleton created.
03/05/2007 15:07:57 (InfL1) ---- Begining CEGUI System initialisation ----
03/05/2007 15:07:57 (InfL1) CEGUI::ImagesetManager singleton created
03/05/2007 15:07:57 (InfL1) CEGUI::FontManager singleton created.
03/05/2007 15:07:57 (InfL1) CEGUI::WindowFactoryManager singleton created
03/05/2007 15:07:57 (InfL1) CEGUI::WindowManager singleton created
03/05/2007 15:07:57 (InfL1) CEGUI::SchemeManager singleton created.
03/05/2007 15:07:57 (InfL1) CEGUI::MouseCursor singleton created.
03/05/2007 15:07:57 (InfL1) CEGUI::GlobalEventSet singleton created.
03/05/2007 15:07:57 (InfL1) CEGUI::WidgetLookManager singleton created.
03/05/2007 15:07:57 (InfL1) CEGUI::WindowRendererManager singleton created
03/05/2007 15:07:57 (InfL1) WindowFactory for 'DefaultWindow' windows added.
03/05/2007 15:07:57 (InfL1) WindowFactory for 'DragContainer' windows added.
03/05/2007 15:08:00 (InfL1) WindowRendererFactory 'Falagard/Editbox' failed to add.

On the other hand we will have forms for alpha and beta testers questioning whether any defect occured.If occured where and whether the defect is fatal or not.After getting this kind of reports team members will read the reports and find the responsible module for defect or crash.The developer of the module will debug and find out the responsible part.

4. REPRODUCIBILITY: FOR DEBUGGING

Debugging is done when our game doesn't do what it is meant to do and according to that we try to understand the problem and fix it.

We have 2 main debugging steps :
First one is pre-integration debugging
Second one is post-integration debugging

In the pre-integration debugging step every group member implements the part that he is responsible for and tests it apart from the whole project. This helps us to find the errors which are specific to one module. As our project consists of 5 main modules namely network, model, GUI, audio and physics each module is tested and debugged by each member separately in the pre-integration step. If the reproducibility can not be achieved then the member logs the error and shares it with the other group members in order to fix the problem. If there are still some problems with it then the decision to change or remove this part is taken.

In the post-integration debugging step as we integrated the modules into whole project we make our tests on the whole project. Some problems occur after integrating the modules. The testing is done by every member in this part and if an error occurs we classify the error and the debugging process is done by the group member who is responsible for this error. At this part the other group members try to help this member while debugging to understand other parts of the project.

Being a real-time application, testing is done by following different ways in the game or trying to accomplish some tasks in the game. For example hitting a wall or climbing onto a car or falling down from something. Most of the bugs are seen by chance in a test drive. These kind of bugs are hard to find and even impossible to reproduce them with the path followed previously. After all the tests and debugs we release a new version for the final stable state of the program.

5. TESTING RESOURCES AND STAFFING

For implementation, test and debug purposes we only need a PC with the required development environment set.
Who is responsible for what in the testing phase is as follows:

Testing and debugging of network module	Onur Demircan
Testing and debugging of models and environment	Abdulkerim Mızrak
Testing and debugging of main menu and in-game menu	Mehmet Emin Ulusoy
Testing and debugging of audio module	Fırat Erdoğan
Testing and debugging of physics module	Özgür Özgür

6. TEST SCHEDULE

Test plan delivery	06.05.2007
Unit Testing	15.05.2007 – 18.05.2007
Integration Testing	19.05.2007 – 21.05.2007
Validation Testing	22.05.2007 – 23.05.2007
Correction	23.05.2007 – 01. 06.2007
High-Order Testing	02.06.2007 – 09.06.2007