Test Specification

CENG492
Spring 2005

Karavela Bilişim

Ali Galip Bayrak        1249986
Burçin Sapaz           1298173
Murat Yükselen         1250976
Özgür Gülderen         1250323
Sevgi Yaşar            1298447
# TABLE OF CONTENTS

1. INTRODUCTION ............................................................................................................. 2  
   1.1. Goals and Objectives .............................................................................................. 2  
   1.2. Scope of Document .................................................................................................. 2  
   1.3. Statement of Testing Plan Scope ............................................................................. 2  
   1.4. Major Constraints...................................................................................................... 3  
      1.4.1. Time .................................................................................................................. 3  
      1.4.2. Usefulness of Obtained Test Results/Cost Ratio .............................................. 3  
      1.4.3. Hardware........................................................................................................... 3  
      1.4.4. Staff................................................................................................................... 3  
      1.4.5. Platform Independency ..................................................................................... 3  
2. TESTING STRATEGY AND PROCEDURES ................................................................ 3  
   2.1. Unit Testing .............................................................................................................. 3  
      2.1.1. Engine Tests...................................................................................................... 3  
      2.1.2. Data Tests.......................................................................................................... 4  
   2.2. Integration Testing.................................................................................................... 5  
   2.3. Validation Testing..................................................................................................... 5  
      2.3.1. Requirements Validation .................................................................................. 6  
      2.3.2. Design Validation ............................................................................................. 6  
   2.4. High-Order Testing................................................................................................... 7  
      2.4.1. Performance Tests............................................................................................. 7  
      2.4.2. Stress Tests........................................................................................................ 7  
      2.4.3. Alpha and Beta Tests ........................................................................................ 7  
3. TEST RECORD KEEPING AND LOG............................................................................. 7  
   3.1. Log4cpp ................................................................................................................... 7  
   3.2. Test Report Forms..................................................................................................... 8  
4. TESTING TOOLS AND ENVIRONMENT ..................................................................... 8  
5. REPRODUCIBILITY: FOR DEBUGGING ....................................................................... 8  
6. TESTING RESOURCES AND STAFFING .................................................................... 9  
7. TEST SCHEDULE .......................................................................................................... 9  
8. APPENDIX...................................................................................................................... 10  
   8.1. An example of Log4cpp configuration file “log4cpp.properties” .......................... 10  
   8.2. Bug Tracking Report Form..................................................................................... 11  
   8.3. Alpha and Beta Tester Report Form ....................................................................... 12
1. INTRODUCTION

1.1. Goals and Objectives

The project “InterDimension” is a 3D arcade adventure game project which has to be implemented in a short period of time. In this short time interval, all the modules that a 3D arcade adventure game should have must be implemented: Render, AI, Scripting, GUI, Sound ... To make all modules work in a harmony, first of all one should verify duties of modules that are critical for the game and verify whether these modules communicate correctly when they are integrated. Therefore testing is very critical for our project and with being aware of this from the beginning of the project we are executing small tests.

Our goals for testing the process of the project “InterDimension” are: achieving a bug-free(as much as we can achieve), logically correct, high performance, well instrumented product.

1.2. Scope of Document

This document briefly describes the testing process of the project “InterDimension”. As in fact the testing process is continuous from the starting date of the project to the end date, this document also includes some information about how the testing has been done up till now in the project. However the main aim and scope of this document is to present the testing plan for the remaining part of the project in which the implementation is considered to be nearly complete.

At this point it is important to emphasize that every detail in our test plan is decided by considering the specific properties of our project and the major constraints that we have for testing(i.e. mostly time). We tried to develop a testing plan which is adequate for a 3D Game Project.

1.3. Statement of Testing Plan Scope

Testing process of the project “InterDimension” includes unit testing, integration testing, validation testing, performance testing, stress testing, alpha and beta testing.

**Unit testing:** Only for some specific modules.

**Integration testing:** Most important part during implementation phase because there are too many modules in game development and their harmony is very critical.

**Validation testing:** In two categories: requirement validation and design validation.

**Performance testing:** Includes testing of methods used to improve speed and quality of graphics rendering of the game.

**Stress testing:** Since the project is a game project it includes too much data(objects, characters), and consequently the control of each data.

**Alpha testing:** Testing of Anafor by someone other than the group members(customer side), at the environment that the game is developed with all the required conditions for the game(development environment).

**Beta testing:** Testing of Anafor by many people(customer side) at different environments(customer environment).
1.4. Major Constraints

1.4.1. Time

Since Anafor is a run-time application, to reach the same state at which an error had occurred is time consuming even you have record of the actions of the player that can be repeated. We have nearly 1 month to complete the project InterDimension, including removing the bugs found and improving the project. As a result of this, time is the greatest constraint for testing of the project “InterDimension”.

1.4.2. Usefulness of Obtained Test Results/Cost Ratio

Amount and method of different test procedures will be decided by this ratio. For example we have to write test codes for unit testing of some modules and we will get their results. Writing of such code also brings both human resource requirement and time cost. Another example is that, during integration tests we will manually test the project by playing and we will obtain their results. These result may even be less useful with respect to other testing procedures. Therefore, the amount of usefulness of test results per the amount of time it takes and per the amount of human resource requirement is one of the most important factors limiting the testing of the project.

1.4.3. Hardware

Beta testers should have “state of the art” computers: memory usage, performance of the graphics card are the other factors limiting tests of the project.

1.4.4. Staff

The project group is composed of 5 people. Since the group will also be working on debugging and development, the staff is also a major constraint for the tests of our project.

1.4.5. Platform Independency

The project “InterDimension” is planned to be platform independent. In other words, the project can work on both Windows and Linux platforms. Therefore the proposed tests are planned to be done in both platforms as much as we have the required time.

2. TESTING STRATEGY AND PROCEDURES

It is important to remind here that this report describes the whole testing process of the project “InterDimension”. Past actions are described briefly and future plans are presented clearly. The following subsections includes different testing procedures that are being used or that will be used in the project “InterDimension” and each testing procedure is described in this past-future manner.

2.1. Unit Testing

In project “InterDimension” unit testing is considered for both of the two main development areas: the game engine and the game data. Also, it is important to point that unit testing is used in this project only when its results will be really useful and also when it does not have much cost for the team.

2.1.1. Engine Tests

Being a game development project, “InterDimension” has 11 main module and lots of sub-modules working under the main modules. Being realistic, it is very difficult and very time consuming to have unit tests for each module. The reason is that each module requires some extra code to be written. This brings some extra work which can not be underestimated. For
this reason, in “InterDimension” when engine is considered, we apply unit tests only to the modules:
- Which have critical requirements for their interior processing
- Whose processing results can be observed easily also while working independent of the other modules.

All the engine related unit tests are done in a black-box fashion thus only the input data and related output is checked.

Till now, five modules have been applied unit tests on their specific requirements before their integration. These modules are the character, the map, the model, the sound and the video module. Some extra code have been written for these and some input data from the game have been provided. For example video module includes smpeg library and before adopting the module to the game, a simple code whose job is only rendering a mpeg video to the screen using this library was written. Then this code is compiled and executed in both of the Windows and Linux platforms as the “InterDimension” game is platform independent. During execution sample mpeg videos are provided to the test executable. After observing probable adjustments, existing stand-alone bugs from the test results and making proper changes accordingly in the code, the module is integrated to the project. The other modules also had similar procedures.

After now, we are planning to do unit testing on two other modules; the resource manager and the collision processor. Mainly resource manager is expected to provide correct data to the game without duplicates and redundancies and also provide efficient management of these data within the game. This job is critical since for example the same model can exist more than once in the game and the related model must only be loaded once in the game. Also another example is that some models can have the same texture, however the game must not load the texture more than once. Collision processor is also critical since it is somehow a bridge between the physics module and all the remaining modules. Moreover nearly all the interaction in the game is based on this module. This module must provide correct collision results for expected interaction.

Also unit testing of the sound module is not finished yet and will continue in the future. The reason for such a situation is that the sound module uses OpenAL library which is still under development. Thus this library is not stable and everyday some bugs are revealed and corrected related to this library. This situation forces us to execute the unit tests again in order to avoid any problems.

### 2.1.2. Data Tests

Being a 3D game project, “InterDimension” includes lots of data production and these data must be verified by unit tests. As specific programs are used in the production of each data item, such unit tests does not bring us any extra work and thus cost of executing unit tests for each data is very low for us. During the unit tests of data, correctness and properness of each and every detail is checked. By the use of the related programs and some of their functionalities, we are able to check the data and the result of any change on the data without using the game engine. For example when a new animation is modeled for a character, without using the game engine and executing the game, we can observe the animation and test it. We make these unit tests at each creation step of each data.

For each of the data groups, we have related programs with some mainly used functionalities for their unit testing which are all listed below:
### 2.2. Integration Testing

During the implementation, a module is integrated to the project when it is mature enough. As soon as the integration is completed this module’s requirements and its interaction with the rest of the project are tested. This test procedure has been done for each part of the project till now and will be done till the end of the implementation phase. However as time is a big constraint for testing, we do not plan to do integration tests after the end of the implementation. Because such a process will require the integration of each module starting from the beginning. Therefore ratio of usefulness/cost is very low for such a process as cost is really high.

The integration tests during the implementation are being done with sandwich testing method. The reason for this decision is that neither top-down integration nor bottom-up integration is adequate for the needs of our project. Top-down integration is not adequate, because although inputs for higher level modules can be generated theoretically, for some modules it is difficult to generate logical input data. For example AI behaviours of the characters in the game requires collision results from the map module and the related character’s behaviours adjusted accordingly. Preparing collision results without using map module is difficult and may have lots of errors. Bottom-up integration is not adequate, because it is not possible to test some modules without the processing results of some higher level modules. For example game engine core is a high level module which provides tick mechanism for lower level modules. Without this tick mechanism, animations can not be tested, therefore bottom-up testing is not possible.

As a result we are considering the sandwich testing method and at the beginning of the tests we prepared a framework on which we had the most crucial modules with their most crucial features for testing of other modules. Then we continued our integration and integration tests by adding higher and lower modules without a sorting between these two distinct groups.

The integration testing method can be white box testing or black box testing according to the specific module and its functionalities. The tester is free to choose between these two methods by considering the module properties.

### 2.3. Validation Testing

Validation testing is the developer side testing procedure that will give the greatest importance within the other tests those are performed after implementation. We have divided the validation tests those we will be performing into two main groups.
2.3.1. Requirements Validation

The tests for validation with respect to the requirements will be black box testing. Because during validation it is important to satisfy the functionalities stated in requirements. Below is the groups of functional requirements stated in analysis document and how we will test them.

Menu Requirements:
- General Requirements: Where the menu will be displayed how to reach its items. To test these requirements, we will do what has been stated in requirements. For example: display menu at the beginning of the game, during the game, click item, then its child item...
- Menu Items Requirements: What are the items of the menu, and their function. To test these requirements, we will check whether menu items accomplish their functionalities.

Game Flow Requirements:
- Game Logic Requirements: How many levels is the game composed of, the level’s functionalities. To test these requirements, the game will be played.
- Environment: What does the world of the game will be composed of, properties of the items in the game, functionalities of the game controlled characters. To test these requirements, the character will interact with environment, and we will observe results.

Main Character Requirements:
- The movement, attack and other abilities that the main character will own, play-game interaction requirements, capabilities of the player during the game. To test these requirements, we will make character move by the controller keys

Engine Requirements:
- The main functionalities each engine in the game will be responsible for. To test these requirements, we will play game, at some modules we will make unit testing and see whether the communication between modules are correct, whether ai behaves as expected, whether rendering is done properly, whether sound engine works properly and we get expected sound...

Finally the requirements that are not correctly implemented, validated, will be revised and corrected.

2.3.2. Design Validation

During the implementation, the most important consideration for us was to be consistent with the design. Moreover, generally our aim was to satisfy one to one correspondence between the class designs and implemented classes. Of course some changes have occurred in the class design but there is nothing designed and not implemented, however there are some useful classes added to the design and implemented.

For design validation we are proposing to check the class and module interactions those are designed in the detailed design document. For this purpose we will use the white box testing method and verify&validate each interface function. By this way, we will make sure that our implementation is consistent with our design.
2.4. High-Order Testing

Moreover we are planning the following high level tests for the project “InterDimension”:

2.4.1. Performance Tests

By instantiating the scripts we can load characters or objects to the game. We will load amount of characters and objects near to the upper bound we have decided that our game can accept and test the frame rate, the time it takes to get the response to an input.

At the worst case the frame rate is expected not to be lower than 24 fps.

2.4.2. Stress Tests

We will load as much as characters and objects we want and check whether the system crashes or continues running.

2.4.3. Alpha and Beta Tests

Being a game project, the project “InterDimension” does not have a specific customer. On the contrary its possible customers is in fact anybody who can play games. This situation makes alpha and beta testing very important(i.e. more than any other project) for “InterDimension”. Thus we are planning to do both alpha and beta testing. For both of these tests, in order to get clear feedback, we will be using our reproducibility method and the form that we have prepared for the testers. (Form can be found in the appendix) Clear feedback is very crucial for alpha and beta tests and we believe that by this way we will get this clear feedback.

We are proposing to assign 2-3 alpha testers and 5-6 beta testers. The numbers are decided according to the time constraint and the requirement of development environment for alpha testing.

3. TEST RECORD KEEPING AND LOG

3.1. Log4cpp

Log4cpp is library of C++ classes which helps to log to files, to standard output or to any other destination. We use “Category” class of log4cpp. The reason we have chosen this class is that it is the central class of the library, consequently it has more capabilities in logging option.

The main reason we used log4cpp is to log necessary information in a systematic way in unit testing and in integration testing. We used log4cpp to take log at different modules of the project; and make it easier to distinguish the logs. Consequently, this method helps finding the source of bugs easily. In addition to this by using log4cpp we are able to distinguish logs by their importance(logging level). For example: when logging level of a module is DEBUG, one can observe logs at debug, info, warn and fatal logging levels. When logging level of a module is fatal one can observe logs only at level fatal. The logging level can be adjusted by a configuration file: “log4cpp.properties”(an example of this file that we are currently using can be found at appendix). By using log4cpp the logs can also be directed to a file. Logging levels we used in our project and details about how we used them are as follows:

Debug: The writer of the module uses this level; to print values of variables, to indicate which part of the code passed, which part of the module is being executed... as it can be understood by its name, this level is used for logging debug related information. This mode helps us in both unit testing and integration testing. However a disadvantage of this mode is
if this is used in each module there will be too much logs because of the game loop. As a result of this execution would slow down.

**Info:** This logging level is used by someone other than writer of the module, to check whether the module works properly or not. As it can be understood from its name this level is used for informing.

**Warn:** This logging level is used to log the conditions at which something unexpected but not critical happens. This kind of logs can be a sign to a problem that will occur lately.

**Fatal:** This logging level is used to log the conditions at which the processes of the project would stop their execution; the conditions not expected and critical.

### 3.2. Test Report Forms

As we have mentioned above we use log4cpp for logging necessary information. As a result of this, the main resource that will inform us about the tests, test results execution processes, bugs, and that will helps us in debugging will be log4cpp.

We will have 2 kind of reports: one for alpha and beta testers, and the other one for us during the other tests we have mentioned before.

The form for beta testers will ask if any defect has occured, if so where has the defect occured, did the defect cause the crash of the system, is the defect reproducible if so the path to detect the defect. Whole group will read the form, and will try to decide on the source of the defect. The writer/s of the related parts of the defect source will debug that part.

The form that we will use during our tests will be a kind of log of the tests. At this form we will state whether the test successed or failed, if failed(and can be detected) why it failed, and the operations logged by log4cpp to help debugging.

The forms mentioned above can be found at the appendix.

### 4. TESTING TOOLS AND ENVIRONMENT

The testing tools and environments below will be used during the tests:

- `g++` (for Linux)
- Visual C++ (for Windows)
- Gdb (for linux)
- Glut library (for some parts of the unit testing)

### 5. REPRODUCIBILITY: FOR DEBUGGING

Up to this point, we have explained the approaches that will be used in testing the project. However debugging a failed test case is hard to accomplish.

Developing a game is actually working on a realtime application. In contrast with event based applications, realtime applications have more difficulty in debugging. The system doesn't stay in a steady state for a predictable time. We as humans always use the system with a different path. Therefore some bugs can only be seen with a great chance in a test drive. These kind of bugs are hard to find and even impossible to reproduce them with the path followed previously. We have solved this reproducibility problem in runtime system.

The runtime system is the first and the lowest level module initialized when the game is executed. The reproducibility functionality introduced the runtime module two new runtime modes in addition to normal mode: record and play.
Normal mode represents the release version of the runtime module, nothing related to debugging is performed. Time information is read from the operating system.

In record mode, runtime feeds the game engine with the events coming from the user. Furthermore, all the events fed are dumped in a binary file. This feature can be used by alpha and beta testers to properly define the error and give a precise feedback to us.

In play mode, runtime opens the binary file containing the events and timing information and uses the contents for driving the game. Time information is not read from the operating system. If this feature is not present, debugging 2 consecutive frames will be impossible because a normal frame time of 30 millisecond will jump to 10 of seconds which will result in unstable physics simulations and realtime effects. Only one key enables the runtime to finalize the application before the record file ends. If the record file is corresponding to a successful test drive, this can be played for successful testing cases.

This feature of the project greatly helps and simplifies reproducing and debugging the errors seen.

6. TESTING RESOURCES AND STAFFING

No specific resources other than a PC which is also required for development is needed.

The duties of the staff for testing phase are as follows:

| The Person in Charge with Reproducibility | Murat Yukselen |
| Bug Tracer and Status Controller         | Özgür Gülderen |
| Test Coordinator                        | Burçin Sapaz  |
| The Person in Charge with Tracibility of Requirements to Tests | Sevgi Yaşar |

7. TEST SCHEDULE

The following is the schedule for the testing plan that is presented in this report:

Test Plan Delivery: (Deadline) 08.05.2005
Unit Test and Integration Tests: (Deadline) 23.05.2005
Validation Tests: (Start) 23.05.2005 - (Deadline) 30.05.2005
Performance and Stress Tests: (Start) 26.05.2005 - (Deadline) 30.05.2005
Beta and Alpha Tests: (Start) 28.05.2005 - (Deadline) 06.06.2005
Results (i.e. Bugs) Tracing and Correction: (Deadline) 11.06.2005
8. APPENDIX

8.1. An example of Log4cpp configuration file “log4cpp.properties”

```properties
# a simple test config
log4j.rootCategory=FATAL, rootAppender
log4j.category.Engine=DEBUG
log4j.category.EventMgr=FATAL
log4j.category.Script=INFO
log4j.category.ScriptCX=INFO
log4j.category.Physics=FATAL
log4j.category.CollProc=FATAL
log4j.category.CmdProc=FATAL
log4j.category.Command=FATAL
log4j.category.CollCommand=FATAL
log4j.category.PlayerCtl=WARN
log4j.category.CameraCtl=WARN
log4j.category.DebugCtl=DEBUG
log4j.category.Map=INFO
log4j.category.Camera=INFO
log4j.category.Character=FATAL
log4j.category.AgentActionWalk=FATAL

#log4j.category.Character=FATAL
#log4j.additivity.sub1.sub2=false

log4j.appender.rootAppender=org.apache.log4j.ConsoleAppender
log4j.appender.rootAppender.layout=org.apache.log4j.BasicLayout

#log4j.appender.A1=org.apache.log4j.FileAppender
#log4j.appender.A1.fileName=A1.log

#log4j.appender.A2=org.apache.log4j.ConsoleAppender
#log4j.appender.A2.layout=org.apache.log4j.PatternLayout
#log4j.appender.A2.layout.ConversionPattern=The message '%m' at time %d%n
```
### 8.2. Bug Tracking Report Form

First three lines of the excel sheet are filled as an example:

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Bug Definition</th>
<th>Assigned to</th>
<th>Bug Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Map Collision detects collision correctly, however returns wrong position</td>
<td>Burcin</td>
<td>Fixed</td>
</tr>
<tr>
<td>1</td>
<td>Same model data is loaded twice</td>
<td>Sevgi</td>
<td>Working on</td>
</tr>
<tr>
<td>2</td>
<td>Tab key event not processed by the game engine</td>
<td>Murat</td>
<td>Debugging</td>
</tr>
</tbody>
</table>
# 8.3. Alpha and Beta Tester Report Form

<table>
<thead>
<tr>
<th>Alpha and Beta Tester Report Form</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name of the Tester:</strong></td>
</tr>
<tr>
<td>➢ During your test process did any bugs occurred and observed? Please list.</td>
</tr>
</tbody>
</table>

If the answer is yes, answer the below questions. (If more than one answer exist(i.e. more than one bug), please list them)

- Where did it occur; in the interface or during the game?

  - Exactly when and how did the bug occur? Please explain your last actions before the bug occurred in their correct order.

  - Did the bug made the system crash?

- Did you observe any unusual behaviors or any probable mistakes? Please give as much detail as you can.

- Do you have any additional comments?