Ceng492 Graduation Project

The Bride Project

Test Specification Report

Presented by Meda

Ankara, 2005

Table of Contents

1. Introduction

In the context of this document the information about testing specifications of group *Meda*'s "The Bride" project is aimed to be represented. Proper testing of a project is an indispensable issue in the manner of verifying accuracy & usability of the application produced. In this respect, the test methods used (how?), tester assigned to that test (who?), the time and duration tests take place (when?) for each module of the project -separately- and release of the integrated project -as a whole- (what?) are going to be identified. We should also underline that test process includes finding the errors and defects and correcting them and after the test down along this document performed, required actions will be taken to correct found bugs.

2. Testing of the Modules

"The Bride" consists of 7 distinct modules; game engine, GUI, graphics, physics, scripting, AI, multimedia, input. We are planning to conduct both white-box (low level) and black-box (high level) testing of the modules. Taking the size of the project and number of project members into account the testing considerations adopted for each module separately are documented below.

Some notes about testing that is common to all the modules is worth mentioning here. The white-box testes are planned to be performed by the developers who implemented the module and includes examining source code with respect to the code standards. The developer is expected to revise his code check for the arguments & return types of the functions, possible used without initialized objects and unused objects. He is also responsible for adjusting source code so that user can be warned if an error occurs identifying the type of the error.

The idea lies behind the low level test specifications is based on the assumption that developer has provided -almost perfectly- error free functions and have tested the functionality of each key function (functions that are critical for the implementation of that module) in the module by writing his "main" routine. On the other hand, the black-box tests of the modules will be performed by developers who had not involved in the process of implementation of that module.

3. Game Engine

The game engine is responsible for the game progression. This is a heavy process due to the fact that all the flow between the modules is handled by the game engine. The game initialization and game finalization have to be done both effectively and efficiently so that neither unnecessary source are kept nor any memory leakage occurs. The game inputs are distributed & pre-game setup is done correctly. In order to satisfy these needs the team is going to test the game engine module both in a white-box & black-box testing manner, individually.

3.1. High Level Testing

The team is going to write high level tester functions in order to control the outputs produced by the game engine are correct with the given inputs, in a black-box manner. Since the engine is the core of the game and all the manipulations and calculations needed for the environment variables are done by this module the testing of the module have to be done rigorously.

The testing of the module will be done with functions which are given the specified conditions and the necessary expected results that start the module and looks for the same results from the game engine.

After running the tester functions with the module we will analyze the results and look for solutions when the expected results and the original results that the engine ends with are not equal.

3.2. Low Level Testing

In order to control the pre-game setup is done correctly, the developer have to do control of the variables written from the files are done correctly so that the other modules wont be able to write over them.
The developer shall look for one-to-one correspondence between the design and the modification or calculation of the environment variables in the game.

Sample High Level Test Case:

Test Case 1:

Aim: Control if the health of the hero is updated according to the formula specified in the design.
Condition: Health=80, Experience=20, Strength=45,
MonsterType=RANGED,
MonsterAttackDamage=15,
Distance=10,

Expected Result: Health=70 for any shot of the monster.

Use The Formula:

```
if (MonsterType==RANGED && Distance)
{
      Health -=
      (MonsterAttackDamage/Distance)*1k/(Experience*Strength)
}
else if (!Distance)
{
      Health –= (MonsterAttackDamage)*1k/(Experience*Strength)
}
```

k:coefficient to keep the value in a reasonable amount

4. GUI

The Graphical user interface is the entry to or exit point from the game. The users will be able to communicate with the game engine through GUI, however, since the responsibility of the module is to spawn a new game or load a new one from the saved-game lists or save an unfinished one it is possible to test the module by the help of tree diagrams corresponding to the use cases & paths in the design.

4.1. High Level Testing

The tester will control the activities, changes done are correctly & in the correct order specified by the design with the help of use cases and tree diagrams.

4.2. Low Level Testing

The developer shall control that the paths (use cases) specified for the module in the design exist in the implementation with the specified order and coded accordingly.

Sample Test Cases:

Test Case 1:
- Click the “New Game” button
- See a new game starts
- Press the ‘Esc’ key in the keyboard
- See the “Exit Menu” asking for “Save” or “Quit”
- Click the “Quit” button
- See the entry point (GUI)

Test Case 2:
- Click the “New Game” button
- See a new game starts
- Press the ‘Esc’ key in the keyboard
- See the “Exit Menu” asking for “Save” or “Quit”
- Click the “Save” button
- See the “Save Game Menu”
- Enter a new name to the list
- Click the “OK” button
- See the entry point (GUI)
- Look that the the file with the specified game contains the required attributes with the necessary values.

5. Physics Module

The physics module is used in the simulation of dynamic objects in some scenes requiring interaction between objects and hero or other objects. Since we are using a manufactured physics simulator namely, ODE, in our calculations the validation of the physics module is an effort to address whether the walls and objects are properly identified to ODE and the methods calling ODE's routines produce the desired results. In this perspective, a number of black-box tests will be performed on the physics module. As stated earlier white-box tests are left to the developer.

The information about objects are red from an object file that stores type, model name, position, state and many other characteristic of the object during initialization of the game. This file will be manipulated (add / remove objects, edit properties of an object) and outcomes will be observed (by means of either new positions of the objects in the world red from a file or graphical user interface -regarding integration of the physics & graphics engine is successful this is more desirable-)during the game.

Test Case 1:
- Remove a defined box from the file.
- The box should also be removed from the world.

Test Case 2:
- Change a defined box's type from dynamic to static in the file.
- The defined box should not be allowed to move by the user or other objects.

Test Case 3:
- Move a box towards a direction.
- The object should change its position and physical properties with respect to the applied action.

Test Case 4:
- Collide two objects.
- The objects should both continue to their movements sensibly (following basic physic rules).

6. AI Module

AI module is responsible for controlling behavior of the agents and includes routines for changing state of an agent or calculating a path between two points in the game. In this perspective, a number of black-box tests will be performed on the multimedia module. As stated earlier white-box tests are left to the developer.

Test Case 1:
- Move closer to an agent.
- The agent should come nearby if he is in patrolling state and if the hero is out of the range following possibly shortest path and afterwards he should start attacking.

Test Case 2:
- Move closer to an agent.
- The agent should start attacking if he is in patrolling state and if the hero is in his range.

Test Case 3:
- Wound the agent till his health reduces to a predefined value.
- Agent should escape from that scene following the possible shortest path to a predefined place.

7. Multimedia Module

The multimedia module includes routines for loading audio files into the game. Since we are using a manufactured sound library namely, OpenAL, in our multimedia routines (loading, playing, stopping, moving, sound and etc...) the validation of the multimedia module is an effort to address whether the sound objects are properly identified to OpenAL and the methods calling OpenAL's routines produce the desired results. In this perspective, a number of black-box tests will be performed on the multimedia module. As stated earlier white-box tests are left to the developer.

The information about sounds are red from an sound file that stores file name, position, type, strength and many other characteristic of the sound during initialization of the game. This file will be manipulated (add / remove sounds, edit properties of an sound) and outcomes will be observed during the game.

Test Case 1:
- Add a new sound to the game by entering new properties of the sound in the sound file.
- An extra sound with the specified characteristic will be heard.

Test Case 2:
- Change the sound position of a defined sound.
- The defined sound should be heard from its updated place in the game

Test Case 3:
- Move closer to a sound.
- The sound should be heard more loudly.

8. Input Module

The input module includes routines for handling the keyboard & mouse events during the game. Since we are window's library to handle these types of events the validation of the input module is an effort to address whether the inputs from both keyboard and mouse are handled accurately and produce desired action. In this perspective, a number of black-box tests will be performed on the multimedia module. As stated earlier white-box tests are left to the developer.

The tester will try possible key combinations that are recognized by the program and observe whether expected consequence is yielded.

Test Case 1:
- Press keys 'W', 'A', 'S', 'D'.
- The position of the hero should be updated in forward, left, right and backward directions respectively during the game.

Test Case 2:
- Move mouse in any direction.
- The direction of the hero should be updated in the specified direction.

Test Case 3:
- Press key 'Space'.
- The hero should perform a jump action.

Test Case 4:
- Left click the mouse.
- The hero should fire.

Test Case 5:
- Press key 'E'
- The hero should use the item in his hands.

9. Graphics Engine

This module is responsible for rendering the scene with the current environment variables in the game. The rendering is a complex process and it is the programs job to send as less polygons as possible to the graphics card. Thus, as we have identified in the design we use BSP trees to draw the level and frustum culling to cull the unnecessary faces that needs not to be drawn.

9.1. High Level Testing

In order to test the mentioned module the tester will look for the following points:

- No jerky or blurred screens while playing due to the the constraint, constant frame rate specified in the design.
- The game, agents and objects take place in the game as specified in the design,
- The implementation and the functionality of the in game menu meets the requirements identified,
- The modes are thoroughly handled,
- The Inventory and the specified functionalities are exist in the game

9.2. Low Level Testing

In order to control the implementation is coded according to the design the user has to control BSP trees, frustum culling and Axis Aligned Bounding Box (AABB) algorithms exist and are implemented thoroughly.

10. Scripting Engine

The team uses scripts in the implementation of the effects, shades and the events that depend on the behaviors of the agents. So a script can represent an effect corresponding to the mask that is shown when the user wants to see the health of the hero or a puzzle showing in between relations of the objects and the agents. The aim is to change the puzzles and the effects on the fly without waiting for additional compile times. In order to test the scripts one will look for the events or effects if they are correctly represented in the scripts and correctly transferred to the game.

10.1. High Level Testing

The tester will look for the required effects and the puzzles with the identical paths given in the puzzle design exist exactly in the game.

10.2. Low Level Testing

The developer will look for correspondences between the puzzles or the effects that are specified are in the design of puzzles & effects, in the scripts.

Sample High Level Test Cases:

Test Case 1 – Scripting Test for Water Effect:

```
{
      map textures/effects/water.jpg
      blendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA
      alphafunc GL_LEQUAL
      alphagen custom 0.3
      depthfunc equal
}
```

Test Case 2 - Scripting Test for PuzzleL1P01:

```
{
      if (door closed AND has ball AND has Experience==45 in
room001)
      door opened
}
```

11. Integration Tests

In order to control all the modules work properly in the system when they are integrated under the same project, the team will have to make scenario based tests that control the paths specified in the use case diagrams, within the design. We will cover these tests under the topic "Integration Tests". These will be high level, Black-Box tests with scenario specific inputs and expected results given and look for the specified outputs for the test case.

Sample Test Cases

Test Case 1- Integration test for Puzzles, Game Engine, GUI, Graphics & Input Libraries:

- Click on the "New Game" button
- See a new game opened with the first level specified in the design data
- Finish the identified tasks for the first level in the game
- See the game loads the next level

Test Case 2 - Integration test for Graphics & Input Libraries:

- Click on the up, down, left & right keys on the keyboard
- See the hero moves accordingly

Test Case 3 – Integration test for Sound, Graphics & Input Libraries:

- Move towards to a monster
- Look for the sound if it increases when the distance between decreases

12. Testing Schedule

Module Name	Tester(s)	Start Date	End Date	Duration(days)
Game Engine	Samet Akpınar	27/05/2005	06/06/2005	9
GUI	İbrahim Alız	27/05/2005	06/06/2005	9
Physics Engine	İbrahim Alız	30/05/2005	08/06/2005	9
Graphics Engine	Mustafa Çelenk	30/05/2005	08/06/2005	9
Scripting Engine	Emre Güney	30/05/2005	08/06/2005	9
AI Engine	Emre Güney	30/05/2005	08/06/2005	9
Multimedia Engine	İlker Çıkrıkçılı	30/05/2005	08/06/2005	9
Input Engine	Mustafa Çelenk	25/05/2005	01/06/2005	6
Integration Tests	Whole Team	08/06/2005	11/06/2005	3

13. Conclusion

Within this document we only presented the ways we are going to test our modules and the final product, and we also present sample test cases. The number of test cases will change according to the module being tested. The aim is to recover as much error as possible. Regarding to the necessities of the modules individually test cases are outlined for each. Developers are assigned to these tests. We believe that this document will be a useful guide in generating an error free project.