**Pookas ETventure**

**TEST PLAN**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Date | Primary Author(s) | Comments |
| 1.0 | 07 Oct 2009 | Ivan Loh, Sarah Lam, Goh Li-Xian | First Version |
| 1.1 | 20 Oct 2009 | Sarah Lam | Revised Test Plan |
| 1.2 | 27 Oct 2009 | Sarah Lam | Revised Test Plan |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc244882614)

[2. Test Plan 5](#_Toc244882615)

[2.1 Requirement Analysis 5](#_Toc244882616)

[2.1.1 Review of Requirement Specifications 5](#_Toc244882617)

[2.2 Design Analysis 6](#_Toc244882618)

[2.2.1 QuizBot Class 7](#_Toc244882619)

[2.2.2 LecturerBot Class 12](#_Toc244882620)

[2.2.3 StudentBot class 17](#_Toc244882621)

[2.3 Implementation 20](#_Toc244882622)

[2.3.1 Code inspection 20](#_Toc244882623)

[2.3.2 Unit Testing 21](#_Toc244882624)

[2.4 Integration Testing 22](#_Toc244882625)

[2.4.1System Testing 22](#_Toc244882626)

[Appendix A 24](#_Toc244882627)

[Appendix B 25](#_Toc244882628)

[Appendix C 26](#_Toc244882629)

# Introduction

It is crucial that bugs and errors are detected in the early stages of the software life cycle and also very essential that all software requirements and specifications are met before proceeding to another stage. This is to prevent incurring of high costs and time needed to make changes in future. Therefore Pooka’s Edventure will strictly adhere to the conditions of this document to achieve 100% code, ensuring that the functional and design requirements are implemented as specified earlier in the Project Planning documentation.

There are three main parts of preparation for the test plan:

1. **Test Strategy**: Pooka’s Edventure will be approaching testing by firstly defining the scope of system testing. We will begin by identifying the test environment (tools to be used, hardware and software requirements) and the strategies (test cases, followed by individual unit testing and lastly, integration system testing).
2. **Test Planning**: List out the activities, dependencies and effort required to conduct the System Test as mentioned in Project Planning documentation. Test risk analysis will also be discussed to prevent any negative events or risks from happening.
3. **Test Classes**: This is the identification and documentation of the test cases for each class which will show the description, details and the expected output for each test case.

By doing the test plan and implementing what is documented, we hope to achieve software of high quality, performing what is required with little or absence of bugs and errors.

# Test Plan

Waterfall model with feedback was chosen to be the life cycle model for the project, so the test plan will go according to the different stages in the Waterfall life cycle model.

## Requirement Analysis

Test planning should begin in the requirements phase of the software life cycle, in which requirements are analyzed for their testability. It is also to detect and report requirements errors that may have surfaced during the software requirements and design process.

Inspections and walkthroughs during the requirement analysis phase with the team are conducted to ensure correctness, completeness and quality of the requirement specification and project plan.

Walkthrough of the requirement specification with the client will be conducted to ensure that the specification of the project is fulfilled. It also allows communication and feedback between the team and the client.

Regular review meetings of the requirement specifications and project plan will be conducted at the start of every phase to monitor the progress of the project.

Specific roles will be assigned to the team members to perform the inspections for the requirement specification and project plan:

* Moderator
* Author
* Inspector
* Reader
* Recorder

A checklist will be given before the start to facilitate the execution of the inspection. This together with the inspection minutes will be recorded. Any error found will be taken down and highlighted for correction.

### 2.1.1 Review of Requirement Specifications

|  |  |  |
| --- | --- | --- |
| **Date** | **Description** | **Inspection Document** |
| 20/02/2009 | Review Requirement Specifications | [Refer to Appendix A](#_Appendix_A) |

## 

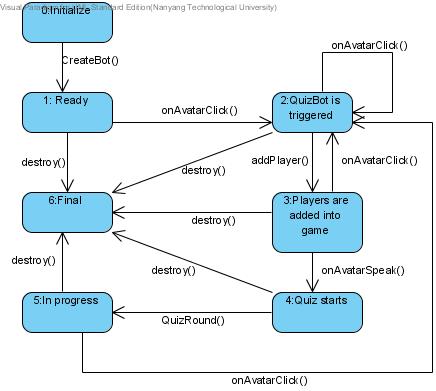
## 2.2 Design Analysis

Use Case diagrams are designed based on initial requirements specification. After which, Class and Sequence diagrams will be checked and discussed by all the project team members. After the verification, the project team leader will ensure that the diagrams match with the requirement specification. Finite State Machine (FSM) Testing will be used so as to ensure the correctness and completeness of the workflow of the Sequence diagrams.

FSM diagrams are designed based on how the class is intended to behave in Active Worlds. The transition tress, which listed down the valid transitions, will be generated based on the FSM diagrams. The test cases will be constructed according to the full or partial branch in the transition tree. Then, the program is tested according to the test cases so as to ensure the correctness of the project.

### 2.2.1 QuizBot Class

#### 2.2.1.1 FSM: QuizBot



#### 2.2.1.2 Transition Tree: QuizBot

#### Test Cases: QuizBot

|  |  |
| --- | --- |
| Test Case 1 | 0->1->6 |
| Description | To initialize the location and set name of the QuizBot |
| Detail | * The name of the quizbot is set in CreateBot() * The 3 dimensional coordinates x,y,z are initialized n CreateBot() |
| Expected Output | The QuizBot should be located in position (x,y,z) in active world and it’s name should be the same as the one set in CreateBot() |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 2 | 0->1->2->6 |
| Description | The QuizBot is triggered |
| Detail | * A user clicks on the Quizbot and this event is handled by onAvatarClick() * The user is prompted to join the quiz |
| Expected Output | “Hello. Up for some quizzing today? (y/n)” |
| Exception | When the user has already triggered the QuizBot, clicking the QuizBot again will not elicit a response from it. |

|  |  |
| --- | --- |
| Test Case 3 | 0->1->2->2 |
| Description | The QuizBot can be triggered multiple times |
| Detail | * Mutiple users can click on the Quizbot and this event is handled by onAvatarClick() * The user is prompted to join the quiz |
| Expected Output | “Hello. Up for some quizzing today? (y/n)” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 4 | 0->1->2->3->6 |
| Description | Adding players to the Quiz |
| Detail | * If a user whispers ‘y’ or ‘Y’ to QuizBot when being prompted to play the quiz, the user is being added to the quiz game. * If a user whispers ‘n’ or ‘N’ to QuizBot when being prompted to play the quiz, the user is not added to the quiz game. |
| Expected Output | If ‘y’ or ‘Y’  “You have signed up for the quiz.”  If ‘n’ or ‘N’  “Alrighty then.” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 5 | 0->1->2->3->2 |
| Description | When the quiz is not in operation and the user clicks on the QuizBot |
| Detail | * When the quiz is not in operation and the user who has already been added into the quiz clicks on the QuizBot will be prompted by it. |
| Expected Output" | “Please wait for the game to start” |
| Exception |  |

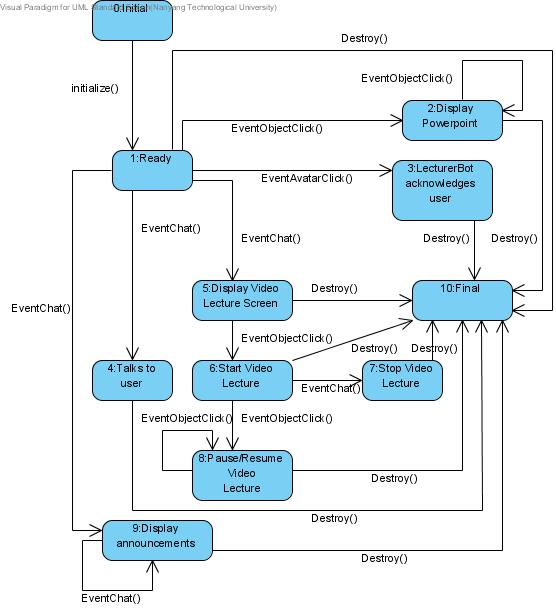
|  |  |
| --- | --- |
| Test Case 6 | 0->1->2->3->4->6 |
| Description | Announcement and start of quiz |
| Detail | * After the first player has been added into the quiz game and the quiz game hasn’t started, the QuizBot is in IDLE state and prompts the user who clicks on it first * The user who clicks on the QuizBot first is considered the starting player and will be the one to start the quiz. * Following the prompt to the starting user will be a broadcast of an announcement |
| Expected Output | The prompt to the user is: “You are the starting player. Click on me when you would like to begin the quiz for everyone.”  The announcement is: “Join the quiz game starting now at Level 2” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 7 | 0->1->2->3->4->5->2 |
| Description | The QuizBot is clicked when the quiz is in operation |
| Detail | * When the quiz is in progress and a user clicks the Quizbot, the QuizBot will prompt the user that the game is in progress. |
| Expected Output | “A quiz is in progress, please wait for the current game to finish.” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 8 | 0->1->2->3->4->5->6 |
| Description | Quiz game in operation |
| Detail | * The QuizBot will broadcast a question to all the players * It will wait for 10 seconds for players to run to either one of the 4 tiles (A,B,C or D) which will represent their answer * The QuizBot will indicate whether the answer is right or wrong and compute the score of each player * This will go on until all the questions have been completed. * The winner of the quiz will be broadcasted. * The QuizBot will indicate that the quiz is over * All scores will be reset after the end of the quiz |
| Expected Output | Example of a wrong answer  “Time’s up!”  **3. What software is used for to control source code revision?**  **[QuizBot]: 1 A: Tortoise SVN**  **[QuizBot]: 2 B: Apache**  **[QuizBot]: 3 C: Microsoft Project**  [QuizBot] Wrong answer, boo. Your score is: 0 (whispered)  **[QuizBot]The correct answer was A: Tortoise SVN (broadcasted)**  **Example of a right answer**  **“Time’s up!”**  **2. What is software quality management?**  **[QuizBot]: 1 A: Ensuring that the required level of quality is achieved in a product**  **[QuizBot]: 2 B: Hoping it works**  **[QuizBot]: 3 C: Shipping the product**  [QuizBot]: Time's up!  [QuizBot]: You got it right! Your score is: 1 (whispered)  **[QuizBot]: The correct answer was A: Ensuring that the required level of quality is achieved in a product (broadcasted)**  **When the quiz is over**  "The quiz is over! Thank you for playing." |
| Exception |  |

### 2.2.2 LecturerBot Class

#### 2.2.2.1 FSM: LecturerBot



#### 2.2.2.2 Transition Tree: LecturerBot

#### 2.2.2.3 Test Case: LecturerBot

|  |  |
| --- | --- |
| Test Case 1 | 0->1->10 |
| Description | Initialize the LecturerBot |
| Detail | * AIML required by the LecturerBot is loaded * Location of LecturerBot in Pooka’s Active World is set * Name of LecturerBot is set * Events to be handled by the LecturerBot is set * Makes an announcement |
| Expected Output | * LecturerBot should located at the position it is set in Pooka’s Active World * “Hello everyone I’m in LT 1B” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 2 | 0->1->2->10 |
| Description | Display powerpoint slides on the screen |
| Detail | * Upon clicking the powerpoint slide screen, EventObjectClick() will detect and handle this event * getPowerpoint() will retrieve the powerpoint slide |
| Expected Output | * Powerpoint slide is displayed on the screen |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 3 | 0->1->2->2 |
| Description | Display subsequent powerpoint slides on the screen |
| Detail | * Upon clicking the powerpoint slide screen, EventObjectClick() will detect and handle this event * getPowerpoint() will retrieve the powerpoint slides one at a time on every click of the screen. * When there are no more new powerpoint slides to be displayed, the first slide is displayed again. |
| Expected Output | * Powerpoint slides will be displayed on the screen one at a time |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 4 | 0->1->3->10 |
| Description | LecturerBot acknowledges user |
| Detail | * Upon clicking the LecturerBot, EventAvatarClick() will detect and handle this event * LecturerBot will greet the Avatar |
| Expected Output | * “Yes, {avatar’s name}?” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 5 | 0->1->4->10 |
| Description | LecturerBot talks to user |
| Detail | * Whenever Lecturer detects certain keywords by the user, it will answer accordingly to the template in the AIML |
| Expected Output | * e.g. What \* CSC 303, Tell \* CSC 303 * “CSC303 is the code for the module software engineering 2.” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 6 | 0->1->5->10 |
| Description | Display video lectures |
| Detail | * When the words ‘**start**’ and ‘**lecture**’ are mentioned by the user, EventChat() will detect the event and handles it * E.g. can we **start** the **lecture**? * The lecturer replies the user * The video lecture screen is launched and appears beside the lecturer |
| Expected Output | * “Ah yes, the lecture will begin now.” * Video lecture screen is launched |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 7 | 0->1->5->6->10 |
| Description | Start the video lecture |
| Detail | * Upon clicking the video lecture screen, EventObjectClick() detects the event and handles it. * The video lecture will load and start |
| Expected Output | * The video lecture starts |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 8 | 0->1->5->6->7->10 |
| Description | Video lecture ends |
| Detail | * When the words ‘**end**’ and ‘**lecture**’ are mentioned by the user, EventChat() will detect the event and handles it * E.g. I want to **end** the **lecture**. |
| Expected Output | * The video lecture stops and the screen closes |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 9 | 0->1->5->6->7->8->10 |
| Description | Video lecture is paused |
| Detail | * Upon clicking the video lecture which has already started, EventObjectClick() detects the event and handles it |
| Expected Output | * The video lecture pauses at its current slide |
| Exception |  |

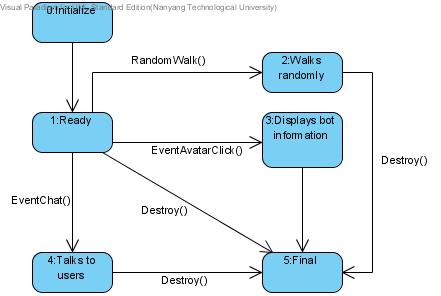
|  |  |
| --- | --- |
| Test Case 10 | 0->1->5->6->7->8->8 |
| Description | Video lecture is paused/resumed |
| Detail | * Upon clicking the video lecture which has already paused, EventObjectClick() detects the event and handles it * The video lecture will resume * The video can be clicked again which will result in a pause * The above 3 steps could be continuously repeated |
| Expected Output | * A paused video will resume playing while a video which is playing will be paused |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 11 | 0->1->9->10 |
| Description | Display announcements from Edventure |
| Detail | * When words such as ‘**news**’ and ‘**announcements**’ are mentioned by the user, EventChat() detects the event and handles it. * The latest announcement posted on CSC 303 announcement page will be displayed * When words such as “2nd” or “3rd” are mentioned by the user, EventChat() detects the event and handles it. * The 2nd latest or 3rd latest announcement posted on CSC303 announcement page will be displayed. |
| Expected Output | * E.g. The latest announcement * “Hey {avatar’s name}, Please bring your clickers to class. It's time to get into action!” * E.g. The 2nd announcement * “Hey {avatar’s name}, Dear TS1 members, Pls kindly be informed that the TS1 lab make up has been scheduled to tomorrow Wed 10:30am to 12:30pmPls let me know if any group are not able to make it. So far from the reply I got, this time slot is preferred. Regards, Chuny” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 12 | 0->1->9->9 |
| Description | Display announcements one after another |
| Detail | * RefreshEdventure() allows the LecturerBot to read the CSC303 announcement page every 30 seconds therefore the LecturerBot does not have to restart to read a new announcement * After displaying an announcement, another announcement can be displayed by using the keywords in Test case 11. * If there is a new announcement, the latest announcement can be displayed by using the keywords in Test case 11. |
| Expected Output |  |
| Exception |  |

### 2.2.3 StudentBot class

#### 2.2.3.1 FSM: StudentBot



#### 2.2.3.2 Transition Tree: StudentBot

#### 2.2.3.3 Test Cases: StudentBot

|  |  |
| --- | --- |
| Test Case 1 | 0->1->5 |
| Description | StudentBot is initialized |
| Detail | * AIML required by the StudentBot is loaded * Location of StudentBot in Pooka’s Active World is set * Located on level 2, carpet area * Name of StudentBot is set * Events to be handled by the StudentBot is set * Does a greeting |
| Expected Output | * “Hi, I am a student bot” |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 2 | 0->1->2->5 |
| Description | StudentBot displays its name and location in Pooka’s active world |
| Detail | * Upon clicking the StudentBot, EventAvatarClick() will detect and handle this event * StudentBot displays its (x,y,z) coordinates and Yaw |
| Expected Output | * E.g. [StudentBot] (X: 4700, Y: 805, Z: 600, Yaw: 2760) |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 3 | 0->1->3->5 |
| Description | StudentBot walks randomly |
| Detail | * StudentBot can walk in either one of the 4 directions (North, East, South and West) or stay in its current location * It has a 20% chance of doing any of the 5 actions * StudentBot must walk within the parameters or it may go through walls and objects |
| Expected Output |  |
| Exception |  |

|  |  |
| --- | --- |
| Test Case 4 | 0->1->4->5 |
| Description | StudentBot talks to users |
| Detail | * Using the student AIML, studentBot will have a 70% chance of talking randomly |
| Expected Output |  |
| Exception |  |

## 2.3 Implementation

During the implementation, **code inspection and unit testing** will be carried out. It is to test the correctness of the program in parts.

### 2.3.1 Code inspection

**Code inspection** will be carried out in sequential order of 6 different stages- planning, orientation, preparation, review meeting, rework and verification.

In the planning stage, the Moderator assembles the team and gathers the review package which includes checklist, work product, references and data sheets. He will ensure that any enhancements needed for the checklist will be made and also ensures the readiness of the work product. He plans dates for meetings and assists the Author in the preparation of the overview. The objectives of the planning stage are forming an inspection team, having all documents needed for the review package gathered and planning subsequent meetings.

In the orientation stage, the Moderator will distribute the review package that was compiled in the planning stage and the Author will give an overview of the review package. There after, duties required for the Review Meeting is assigned to the Recorder. Also, the Moderator will review the procedures needed for the preparation stage. The objectives of the orientation stage are having the review package to be distributed to reviewers, having the Author provide an overview of the package, establishing goals for the preparation stage and having reviewers commit to participate.

A recommended time frame is allocated to the preparation stage before it begins. After that, work products of individuals are reviewed using checklists and references. Critical, severe, moderate issues pertaining to work products are noted using Reviewer Data Form while minor issues and the Authors are being questioned on minor issues of the program. The objective of the preparation stage is to find the maximum number of non-minor issues.

In the review stage, the Moderator requests issues in a sequential fashion with reviewers raising the issues. The Recorder then notes these issues in the Recorder Data Sheet which is visible to everyone. The objectives of the review stage are to consolidate a list of non-minor issues and allow everyone to have shared knowledge of work product while reviewing others work.

In the rework stage, the Author assess all issues raised and work product in the Recorder Data Sheet and notes down the action taken for each issue in the Author Data Sheet. Each issue is identified with a type of defect (req/spec/design/imp, etc). There after, the Author reworks the product and passes it together with the Author Data Sheet for the Moderator to verify. The objectives of the rework stage are to assess issues and take appropriate action to remove them if necessary and also to take note of actions to resolve the issues.

In the verification stage which is also the last stage, the Author reviews the reworked product against the issues recorded by the reviewers and the Author. He will ensure that issues are resolved by the Author before performing a sign-off with the reviewers. The objectives of the verification stage are to assess the quality of the reworked product and determine whether it has made the mark and to assess the inspection process.

After completing these 6 stages, the code inspection process will conclude.

2.3.1.1 Code Review Checklist template

**Refer to appendix B**

2.3.1.2 Scheduled Code Review Inspections

|  |  |  |
| --- | --- | --- |
| **Date** | **Description** | **Inspection Document** |
| 20/10/2009 | Code Review for Lab 5 Implementation | Refer to Appendix B |

### 2.3.2 Unit Testing

**Unit testing** will be carried out to detect a large percentage of defects if any by performing the test cases coded already available. The test cases will be executed and any errors found will be fixed and recorded as part of the Test Report.

Each individual unit of smaller modules will be tested for correctness separately before integration into larger sets up to the entire system and also interfaces between modules using white box and black box testing. This is to check for correct integration and operation of the system. All testing procedures will have to meet functional (I/O), behavioral (event handlers) and performance requirements. Fixed units will be re-tested again and the cycle will continue until no more errors in each unit are found.

2.3.2.1 Unit Test Cases

**Refer to appendix C**

2.3.2.2 Scheduled Unit Test inspections

|  |  |  |
| --- | --- | --- |
| **Date** | **Description** | **Inspected?** |
| 21/10/2009 | Unit Test Inspection for Lab 5 Implementation | Yes |

## 2.4 Integration Testing

**Integration Testing** is a testing phase which focuses on interfacing different units of a program. Individual software units are combined and tested as a group. It occurs after unit testing and before system testing. Unit testing will ensure the correctness and usability of each module and therefore if a problem occurs during integration testing, it is highly likely due to the interfacing of the different units and rather than faults with the individual modules.

The advantage of this technique is that it reduces the number of possibilities to a far simpler level of analysis. Final Integration Test proves that system works as integrated unit when all the fixes are complete. Integration testing will be carried out to ensure that every function of the program works correctly which allows for system testing to take place.

There are 4 common approaches in integration testing:

1. top down approach

2. bottom up approach

3. big bang approach

4. incremental approach

In this project, we will be using the bottom up approach to integration. With incremental testing, drivers are coded first which will call units under test. The expected output will be verified and errors will be checked for if the sub program returns incorrect values. After the lowest level of units have been tested using the drivers, the next higher level of units will be tested with drivers. This will go on until the highest level has finished its testing.

### 2.4.1System Testing

#### 2.4.2.1 Regression Testing

After integration testing, regression testing will ensue. Regression testing includes rerunning previously run tests and checking whether previously fixed faults have re-emerged.

Some strategies and factors to consider during this process include the following:

* Test fixed bugs promptly. The programmer might have handled the symptoms but not have gotten to the underlying cause.
* Watch for side effects of fixes. The bug itself might be fixed but the fix might create other bugs.
* Write a regression test for each bug fixed.
* If two or more tests are similar, determine which is less effective and get rid of it.
* Identify tests that the program consistently passes and archive them.
* Focus on functional issues, not those related to design.
* Make changes (small and large) to data and find any resulting corruption.
* Trace the effects of the changes on program memory.

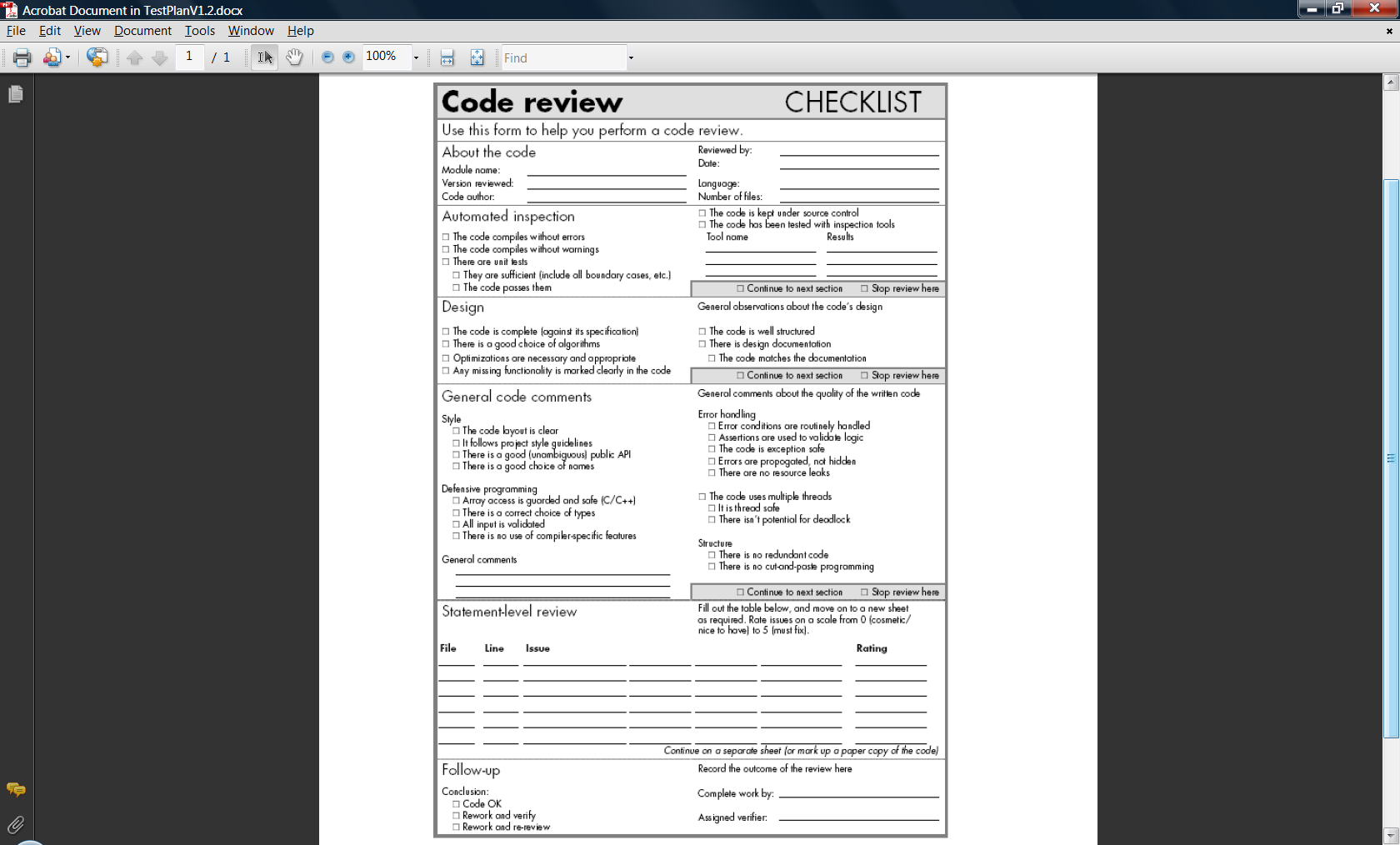
After all these testing phases are completed, Pookas Edventure will be ready to launch its demo, with all specifications met and its utmost quality assured.

# Appendix A

**Requirement Review Specifications**

1. Virtual Classroom system will be an integration of two existing platforms, ActiveWorld and Edventure.
2. ActiveWorld is a virtual reality community where the users are able to communicate and interact with each other.
3. Edventure is an internet-based course-management system which allows students access to course materials and other academic resources.
4. The system will allow the user to log in to his ActiveWorld account.
5. Once the user is logged in and is in Pooka’s world, the user can navigate around from one room to another.
6. The user is able to teleport from the entrance to various parts of Pooka’s world by teleporting stations placed at the entrance of Pooka’s world and vice versa.
7. The lecturer announces the Lecture theatre he’s at so that users will be able to find him.
8. The user is able to chat with the lecturer regarding CSC303 and ask the lecturer for the latest announcements or previous announcements using Artificial Intelligence Markup Language (AIML).
9. The user can watch video lectures on CSC303 in LT 1B
10. The user can view powerpoint slides on CSC303
11. There is a student bot who walks around randomly near the Quiz Area
12. The student bot can respond to a user and also talk randomly using AIML
13. The student bot can provide information on its current location and also its name.
14. There is a multiple choice Quiz game at the Quiz area to allow users to test their knowledge on CSC303.

# Appendix B



# Appendix C