**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 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 Pookas ETventure 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**: Pookas ETventure 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.

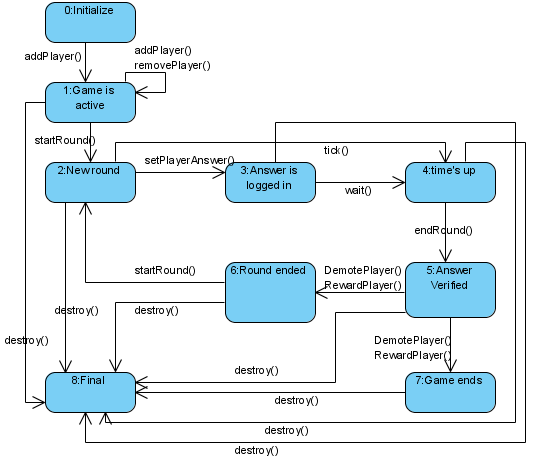
By doing the test plan and implementing what is documented, we hope to achieve software of high quality, performing what is required with little or absence of bugs and errors.

# 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.

## 3 FSM: QuizBot



## 2.2 Transition Tree: QuizBot

## 2.3 Test Cases: QuizBot

|  |  |
| --- | --- |
| Test Case 1 | 0->1->8 |
| Description | To put the game quiz in the active state |
| Detail | * A player is added using addPlayer() * There must be at least 1 player in the game for the game to start |
| Expected Output | The game quiz can be started or more players can be added to the game |

|  |  |
| --- | --- |
| Test Case 2 | 0->1->1 |
| Description | To add or remove the number of players in a game |
| Detail | * 1 player can be added to the game using addPlayer() each time * There is no limit to the number of players in the game * 1 player can be removed from the game using removePlayer() each time |
| Expected Output | The game quiz can be started or more players can be added to the game |

|  |  |
| --- | --- |
| Test Case 3 | 0->1->2->4->8 |
| Description | Timer expires |
| Detail | * The timer expires with no answer logged in |
| Expected Output | “Time’s up!” is displayed |

|  |  |
| --- | --- |
| Test Case 4 | 0->1->2->8 |
| Description | To have the game in operation |
| Detail | * The game is started using startRound() * getQuestion() is called from the Quiz Question class |
| Expected Output | The question is displayed on the screen |

|  |  |
| --- | --- |
| Test Case 5 | 0->1->2->4->5->8 |
| Description | Answer is verified |
| Detail | * After the timer has timed out, endRound() calls checkAnswer() from Quis Question class which verifies that there’s no answer * One round of the game has ended |
| Expected Output | “There is no answer given. The correct answer is {answer}” will be displayed |

|  |  |
| --- | --- |
| Test Case 6 | 0->1->2->4->5->6->8 |
| Description | Leveling the player |
| Detail | * Since no answer is given, the player got the answer wrong * DemotePlayer() will decrease the y-position of the player |
| Expected Output | “You are at level {level}” will be displayed |

|  |  |
| --- | --- |
| Test Case 7 | 0->1->2->4->5->6->2 |
| Description | A new round proceeds if it’s not the round where = last |
| Detail | * startRound() calls getQuestion() from QuizQuestion class. |
| Expected Output | A new question will be displayed |

|  |  |
| --- | --- |
| Test Case 8 | 0->1->2->3->8 |
| Description | Player chooses their answer |
| Detail | * Players move to one of the 4 tiles to choose their answer * OnBump() from the QuizPanel class is called * OnBump() detects the event and answer is set in setPlayerAnswer() |
| Expected Output | “You have chosen answer {answer}” will be displayed on the screen |

|  |  |
| --- | --- |
| Test Case 9 | 0->1->2->3->4->8 |
| Description | Timer expires |
| Detail | * Timer expires with answer logged in |
| Expected Output | “Time’s Up!” is displayed |

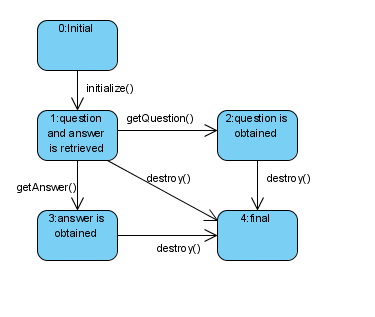
|  |  |
| --- | --- |
| Test Case 10 | 0->1->2->3->5->8 |
| Description | Answer is verified |
| Detail | * endRound() calls checkAnswer() will be called from the Quiz class * player’s answer will then be verified with the answer of the question |
| Expected Output | “Congratulations! You have the correct answer!” or “You got the answer wrong! The answer is {answer}” will be displayed |

|  |  |
| --- | --- |
| Test Case 11 | 0->1->2->3->5->6->8 |
| Description | A new round proceeds if it’s not the round where = last |
| Detail | * startRound() calls getQuestion() from QuizQuestion class. |
| Expected Output | A new question will be displayed |

|  |  |
| --- | --- |
| Test Case 12 | 0->1->2->3->5->6->2 |
| Description | Leveling the player |
| Detail | * if the player got the answer correct then RewardPlayer() increases y-position of player * if the player got the answer wrong then DemotePlayer() decreases y-position of player |
| Expected Output | “Your new level is {level}” will be displayed |

|  |  |
| --- | --- |
| Test Case 13 | 0->1->2->3->5->7->8 |
| Description | The game ends |
| Detail | * destroy() is called |
| Expected Output | “Thank you for playing the game quiz has ended!” will be displayed |

# 3 FSM: QuizQuestion



## 3.2 Transition Tree: QuizQuestion

## 3.3 Test Case: QuizQuestion

|  |  |
| --- | --- |
| Test Case 1 | 0->1->4 |
| Description | Retrieval of quiz question and answer |
| Detail | * Data from QuizPage is initialized |
| Expected Output | - |

|  |  |
| --- | --- |
| Test Case 2 | 0->1->2->4 |
| Description | The quiz question is obtained |
| Detail | * getQuestion() is called |
| Expected Output | The question is displayed |

|  |  |
| --- | --- |
| Test Case 3 | 0->1->3->4 |
| Description | The quiz answer is obtained |
| Detail | * getAnswer() is called |
| Expected Output | The answer is displayed |