Test Plan

Table of Contents

Test Plan Identifier

The test plan identifier for this test plan is “Test Plan – 1”.

References

The list of documents, which have been used as references are given below:

1. Project Plan
2. http://www.computing.dcu.ie/~davids/courses/CA267/ieee829mtp.pdf
3. http://en.wikipedia.org/wiki/Test_plan
4. http://management.simplicable.com/management/new/130-project-risks
5. http://groups.engin.umd.umich.edu/CIS/course.des/cis375/projects/risk table/risks.htm

Introduction

This is the test plan for our Hobbit project, created in Java. This specific test plan has been created for some specific functions, which have been stated below (in the “Test Items (Functions)” section). This test plan shall contribute of the following items:

1. Test items & Features to be tested
2. Software risk issues
3. Approach
4. Item pass/fail criteria
5. Test deliverables

Test Items (Functions) & Features To Be Tested

The following functions will be tested in this test plan:

1. The “Leave” class, which should leave the entity it has taken from the “Take” class
2. The “Goblin” class
3. The “Pony” class
4. The “Saddlebag” class

The acceptance criteria for the above classes have been provided in the “Item Pass/Fail Criteria”.

Software Risk Issues

The following risks must be considered for this current test plan:

1. Game may not function properly
2. Bugs could drive players away

Item Pass/Fail Criteria

The following criteria must be met to ensure that the test plan has been completed successfully:

1. “Leave” should make the actor leave the object that it has taken using the “Take” affordance
2. Goblin has a home base
3. Goblin starts from the home base
4. Goblins can leave home base to patrol
5. Goblin patrols in square paths of various sizes, after which it returns to its home base
6. When the Goblin reaches its home base (or upon initiation of the simulation), it should wait for a random number of turns and then randomly chooses an initial direction and the length of the side of the square for its patrol.
7. The random number of turns must be between 1 and 5
8. The “act()” method must be called to randomly find the number of turns.

9. The length of the side of the square must be 1 and 4.
10. Each time the “act” method is called, the goblin must move one step in its current direction. This is until it cannot move further (no path remaining).
11. After not being able to move further, the goblin must retrace its steps to its home base, one step at a time.
12. If goblin completes one side of the square, then it changes its direction to 90º anticlockwise. Then it continues to patrol.
13. It continues to do the above, until it reaches the home base.
14. The goblin should be a part of the “EVIL” team.
15. When a goblin encounters an entity from a different team, and the goblin has an attack affordance, it should attack the entity.
16. Pony class helps players overcome limitations of only being able to carry one object
17. Pony owner must be any “HobbitActor”.
18. Pony “Act” method tells if the owner is in the same location.
19. If owner is present, Pony should stay.
20. If owner is in an adjacent location, then the Pony moves to that location.
21. If owner is not nearby, or it doesn’t have an owner, then it should take one step in a random direction
22. A pony can carry up to two “Saddlebags”.
23. Store method stores items in the Saddlebag
24. Retrieve method removes items from the Saddlebag
25. Each Saddlebag can store up to 10 items
26. At least two Saddlebags are added to the map
27. Saddlebags can be picked up using Take
28. Saddlebags can be dropped using Leave.
29. If Saddlebag is empty, then non-Pony HobbitActors can pick it up.
30. If Saddlebag is fully-laden, then the HobbitActor cannot pick it up as it is too heavy
31. A pony is attackable
32. 30% chance of destroying a Saddlebag if Pony is killed.
33. 50% chance of item being destroyed if Saddlebag is destroyed

Approach (Strategy)

For this test plan, we plan to use a combination of Whitebox testing and Blackbox testing. (If the software was updated, then regression testing would be needed). The reason we will be using Whitebox testing is to ensure that each unit completes the functions they are meant to complete, properly. We will be testing the specific units stated above, using a library named “JUnit”. The reason we will be using Blackbox testing is to ensure that all the specifications are met and are easily testable.

Test Deliverables

The following documents are the documents that must be delivered as a result of this test plan:

1. Test plan
2. Test cases