BLACK BOX TESTING DOCUMENT

for

Virtual tour based serious Game

April 22, 2018

Prepared by
Inderpreet Singh Chera-160101035
Shubhendu Patidar-160101068
Siddharth Sharma-160101071

IIT GUWAHATI

Contents

1 Introduction

Black box testing is a software testing techniques in which functionality of the software under test (SUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.
In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

1.1 Purpose of Black Box Testing

Unit testing is undertaken after a module has been coded and successfully reviewed.
Generally Black Box Testing attempts to find errors in the external behavior of the code in the following categories:

- Incorrect or missing functions
- Performance Problems
- Concurrency and timing errors
- Initialization and termination errors
- Etc.

Through this testing we can determine if the functions appear to work according to specification. It is suggested that the black box testing is performed by an actor who is not a developer as the tests are made to ensure the functions perform as needed by the customer.

2 Test cases for Black Box Testing

In the black-box testing, test cases are designed from an examination of the input/output values only and no knowledge of design, or code is required. The following are the two main approaches to designing black box test cases.

- Equivalence Class Partitioning
- Boundary Value Analysis

To perform black box testing, a tester who has some experience with unity and C# was asked to test this document. He was asked to pick up any value from the range and test it. To perform black box testing he was given code which only had that particular unit that was required to be tested. Rest all code was commented.

2.1 Equivalence Class Partitioning

In this approach, the domain of input values to a program is partitioned into a set of equivalence classes. This partitioning is done such that the behavior of the program is similar for every input data belonging to the same equivalence class.

2.1.1 Player Movement

When a particular key is pressed, movement of player is observed. So, equivalence classes is based on whether or not a particular key is pressed.

1. Input : 'w', 's', 'a', 'd', '↑', '→', '←', '↓', 'space' key is pressed.
   Output : Movement of player is observed.
   Equivalence Classes (based on keys present on the keyboard):
   - C1 : {'w', 's', 'a', 'd', '↑', '→', '←', '↓', 'space'}
   - C2 : { 'key' | 'key' $\notin$ C1 $\wedge$ 'key' $\in$ ASCII}
   - C3 : { 'key' | 'key' $\notin$ C1 $\wedge$ 'key' $\notin$ ASCII $\wedge$ 'key' $\in$ keyboard}
     Note : 3rd invalid class is made because they vary from computer to computer.

Test Case	Class Status	Input Value	Expected Output	Actual Output
C1	Valid Class	'w' key	Player Moves	Moves Forward
C2	Invalid Class	'z' key	Do Nothing	Does Nothing
C2	Invalid Class	'F1' key	Do Nothing	Mutes Audio

2.1.2 Camera Movement

When the mouse pointer is moved, a change in the player's orientation is observed. So, equivalence classes is based on whether or not the mouse pointer moves.

1. Input : The mouse pointer is moved.
   Output : Player's orientation changes.
   Equivalence Classes (based on pointer movement on the screen):
   - C1 : $\{|\delta x| + |\delta y| > 0 \mid \delta x, \delta y$ are change in x and y coordinates of the pointer$\}$
   - C2 : $\{|\delta x| + |\delta y| = 0 \mid \delta x, \delta y$ are change in x and y coordinates of the pointer$\}$

Test Case	Class Status	Input Value	Expected Output	Actual Output
C1	Valid Class	$\delta x < 0$, $\delta y = 0$	camera orientation changes	camera turns left
C2	Invalid Class	$\delta x = 0$, $\delta y = 0$	Do Nothing	Does Nothing

2.1.3 Number of questions (and options)

Questions and answers are written in a specific order in 3 different files.
File 1 : Questions - with 1 question in each line.
File 2 : Options A - corresponds to correct option at same line number in question file.
File 3 : Options B - corresponds to incorrect option at same line number in question file.
Therefore, number of lines in all 3 files will be same.
5 Questions should be randomly selected from the file and will be displayed
To check if number of questions and options are increased or decreased beyond a range, will it still work.
File Valid Range : $\{ x \mid x \in [5, \infty) \wedge x$ is integer $\}$

1. Input : file questions,file optionsA, file optionsB.
   Output :List<String> such that if number of questions are greater than 5 then 5 questions are displayed else same number of questions are displayed as in questions file.
   Equivalence Classes (based on file length):
   - C1 : $\{$ file.Length $\mid file.Length \in [5, \infty) \}$
   - C2 :$\{$ file.Length $\mid file.Length \in [0, 5) \}$
     where file.Length is length of any file(all will have same length)

Test Case	Class Status	Input Value	Expected Output	Actual Output
C1	Valid Class	questions,optionsA,optionsB (questions.Length = 10)	5 random questions	5 random questions
C2	Invalid Class	questions,optionsA,optionsB (questions.Length = 3)	Display 3 questions	Loads some questions (2 in particular)

2.1.4 Score Calculator

Time taken to complete the task and number of correct questions answered are input to the unit.
Correct Answers range: Set S1 : { 0, 1, 2, 3, 4, 5 }
Time taken range : [0, ∞)

1. Input : float Tuple < TimeTaken, CorrectAnswers >.
   Output : Total score.
   Equivalence Classes (based on Time taken and correct answers):
   - C1 : { <TimeTaken, CorrectAnswers>|$TimeTaken \in$ [0, 10] $\wedge$ CorrectAnswers $\in$ S1 }
   - C2 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ (10, 60] $\wedge$ CorrectAnswers $\in$ S1 }
   - C3 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ (60, ∞) $\wedge$ CorrectAnswers $\in$ S1 }
   - C4 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ [0, 10] $\wedge$ CorrectAnswers $\notin$ S1 }
   - C5 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ (10, 60] $\wedge$ CorrectAnswers $\notin$ S1 }
   - C6 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ (60, ∞) $\wedge$ CorrectAnswers $\notin$ S1 }
   - C7 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ (-∞, 0) $\wedge$ CorrectAnswers $\in$ S1 }
   - C8 : { <TimeTaken, CorrectAnswers> |$TimeTaken \in$ (-∞, 0) $\wedge$ CorrectAnswers $\notin$ S1 }

Test Case	**Class Status**	**Input Value**	**Expected Output**	**Actual Output**
C1	Valid Class	<5, 1>	60	60
C2	Valid Class	<36, 3>	54	54
C3	Valid Class	<61, 0>	0	0
C4	Invalid Class	<7, -1>	"Error"	40
C5	Invalid Class	<54, 117>	"Error"	1176
C6	Invalid Class	<89, 8>	"Error"	0
C7	Invalid Class	<-100, 1>	"Error"	60
C8	Invalid Class	<-92, 23>	"Error"	280

2.1.5 Volume Change

To check when the slider moves, music volume changes.

1. Input : float volume
   Output : Volume changes to input.
   Volume range : [0.0f, 1.0f]
   Equivalence Classes (based on volume):
   - C1 : {volume |$volume \in$ [0.0f, 1.0f]}
   - C2 : {volume |$volume \in$ (-∞, 0.0f)}
   - C3 : {volume |$volume \in$ (1.0f, ∞)}

Test Case	Class Status	Input Value	Expected Output	Actual Output
C1	Valid Class	0.61f	0.61f	0.61f
C2	Invalid Class	-1.6f	0.0f	0.0f
C3	Invalid Class	11.6f	1.0f	1.0f

2.1.6 Get Labels

When the player is in vicinity of some room, he should get label telling information about that room.

1. Input : float Player <x1,y1,z1>, float DetailsCoordinates <x2,y2,z2>, List<string> details
   Output : If player near <x2,y2,z2>, then display label else display ""
   Coordinates range : Any coordinates on the map
   Equivalence Classes (based on coordinates):
   - C1 : {Player ,DetailsCoordinates , details |$\sqrt{(x1-x2)^2+(y1-y2)^2+(z1-z2)^2} < 10.0f$}
   - C2 : {Player ,DetailsCoordinates , details |$\sqrt{(x1-x2)^2+(y1-y2)^2+(z1-z2)^2} >= 10.0f$}

Test Case	Class Status	Input Value	Expected Output	Actual Output
C1	Valid Class	$< 0,0,0 >, < 0,1,0 >, display$	display	display
C2	Invalid Class	$< 1,0,0 >, < 10,11,0 >, display$	""	""

2.1.7 Random Initial And Final Coordinates

When the game is played then player should be placed at random initial position and should be asked to go to random final position.

1. Input : file Initial, file Final
   Output : float coordinates <x,y,z>, string finalDestination
   Equivalence Classes (based on coordinates):

- C1 : Initial ,Final
  Note : Same equivalence class will be give twice to be sure that we are getting random results each time.

Test Case	Class Status	Input Value	Expected Output	Actual Output
C1	Valid Class	Initial, Final	random<x,y,z>,randomstring	$< -51.625, 0.0, -98.397 >$,Hardware Lab
C2	Valid Class	Initial, Final	random<x,y,z>,randomstring	$< 9.28, 0.0, -95 >$,Seminar room

2.2 Boundary Value Analysis

A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs. Therefore, it is necessary to do boundary value analysis.

2.2.1 Number of questions (and options)

Questions and answers are written in a specific order in 3 different files.
File 1 : Questions - with 1 question in each line.
File 2 : Options A - corresponds to correct option at same line number in question file.
File 3 : Options B - corresponds to incorrect option at same line number in question file.
Therefore, number of lines in all 3 files will be same.
5 Questions should be randomly selected from the file and will be displayed
To check if number of questions and options are increased or decreased beyond a range, will it still work.
File Valid Range : { x |$x \in$ [5, ∞) $\wedge$ x is integer }

1. Input : file questions,file optionsA, file optionsB.
   Output :List<String> such that if number of questions are greater than 5 then 5 questions are displayed else same number of questions are displayed as in questions file.

Boundary Case	Boundary Value	Expected Output	Actual Output
B1	questions,optionsA,optionsB (questions.Length = 5)	5 random questions	5 random questions
B2	questions,optionsA,optionsB (questions.Length = 4)	4 random questions	Loads some questions (1 in particular)
B3	questions,optionsA,optionsB (questions.Length = 6)	5 random questions	5 random questions
B4	questions,optionsA,optionsB (questions.Length = 0)	0 random questions	Loads default text
B5	questions,optionsA,optionsB (questions.Length = 1)	1 random questions	Loads 1 question

2.2.2 Score Calculator

Time taken to complete the task and number of correct questions answered are input to the unit.
Correct Answers range: Set S1 : { 0, 1, 2, 3, 4, 5 }
Time taken range : [0, ∞)

1. Input : float Tuple < TimeTaken, CorrectAnswers >.
   Output : Total score.
   Note : Boundary values will be only on Time taken as correct answers data is discrete.

Boundary Case	Boundary Value	Expected Output	Actual Output
B1	<0, 1>	60	60
B2	<0.1, 2>	70	70
B3	<9.9, 2>	70	70
B4	<10, 3>	80	80
B5	<10.1, 2>	69.9	69.9
B6	<59.9, 4>	40.1	40.1
B7	<60, 5>	50	50
B8	<60.1, 0>	0	0
B9	<-0.1, 1>	60	60

2.2.3 Volume Change

To check when the slider moves, music volume changes.

1. Input : float volume
   Output : Volume changes to input.
   Volume range : [0.0f, 1.0f]

Test Case	Input Value	Expected Output	Actual Output
B1	0.0f	0.0f	0.0f
B2	-0.1f	0.0f	0.0f
B3	0.1f	0.1f	0.1f
B4	1.0f	1.0f	1.0f
B5	1.1f	1.0f	1.0f
B6	0.9f	0.9f	0.9f

3 Conclusions Drawn

Some bugs were identified during Black box testing, which needs to be fixed.
Some of them are written here :

1. When number of questions in the file are less then 5, then the questions are not correctly displayed.
2. If keyboard keys like Windows key are pressed, then our game gets paused and Windows start menu opens which shouldn't happen.
3. Boundary Cases were all covered carefully.