Test Plan for Breaking Effect (BE)

Marshall Xiaoye Ma

December 16, 2017

1 Revision History

Date	Version	Notes
2017-10-21	1.0	New Document

2 Symbols, Abbreviations and Acronyms

symbol	unit	description
S	unit of length	displacement
t	s	the time in seconds since the start of the game
v_0	m/s	initial speed
a	m/s^2	acceleration
g	m/s^2	gravity acceleration
ΔE_k	J	variation of kinetic energy
ΔE_p	J	variation of potential energy
W_f	J	work done by kinetic friction
x_n	m	x coordinates of gravity center of piece n, $n \in N$
y_n	m	y coordinates of gravity center of piece n
z_n	m	z coordinates of gravity center of piece n
θ_1	degree	angle between initial speed and horizontal
θ_2	degree	angle between x axiom and projection on horizontal of initial speed
S_x	m	displacement on direction of x axiom
S_y	m	displacement on direction of y axiom
S_z	m	displacement on direction of z axiom
μ_k		coefficient of friction

symbol	**description**
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
PS	Physical System Description
R	Requirement
SRS	Software Requirements Specification
BE	Breaking Effect
T	Theoretical Model

Contents

List of Tables

[If you don't have a list of figures, you can remove this heading. —SS]
[Removed ! —Author]

3 General Information

3.1 Purpose

The purpose for this document is to build test plan for project Breaking Effect.Test cases in the document are designed based on SRS of the project and aim to cover both functional and non-functional requirements described in SRS. [You don't need to end paragraphs with an explicit character. This practice has the unfortunate consequence of sometimes leaving an extra blank line in the generated document. —SS] [Thank you professor for this advice ! —Author] This document is used as a guide that need to be followed exactly in test stage before release of this project.

3.2 Scope

This test plan covers verification and validation for the project Breaking Effect to make sure the program is implemented as requirement specification, including automated testing, system test and unit test. The project relies on Unity3D as platform and uses function provided by Unity3D for object cutting. So test for object cutting is not included in this test plan. This test plan is designed based on SRS so that it doesn't cover test cases for requirements not in SRS. SRS and other documents for this project can be found on `https://github.com/MaXiaoye/cas741`

[An explicit web-link to your GitHub repo would be nice. —SS] [url added ! —Author]

3.3 Overview of Document

This test plan firstly describes environment and platform for Breaking Effect. Purpose and scope of outlined the document generally in previous sections. Detailed test plan and test cases are covered in following sections including test team, automated testing approach, verification tools and test cases for both functional requirements and non-functional requirements.

4 Plan

4.1 Software Description

Breaking effect presents how the pieces of an object move after it separates into parts with suddenness or violence.

This project implements running time breaking effect in codes for 3-D models in unity3D without help from any similar plug-in. Including different shapes 3-D objects breaking based on physics and pieces interacting with the momentum provided by the breaking force. The breaking effect program simulates 3-D objects destruction process in vision by implementing scientific computing functions.

[The text is better for version control, and for reading in other editors, if you use a hard-wrap at 80 characters —SS]

This project concentrates on calculation while HCI or GUI are not important parts. Applied force is decided in codes in advance as input and trace of motion is the output after calculation.

4.2 Test Team

The team that is responsible for all tests is Xiaoye Ma.

4.3 Automated Testing Approach

Most testing work for Breaking Effect does not rely on automated testing. Because the final output is visualization and experience from users. Verification of inputs will be done automatically through help from Unit Test Generator provided by IDE Visual Studio 2017. Test cases are covered in following sections. Verification of correctness and output from intermediate steps will be tested manually.

Automated testing also helps check correctness of C# codes including bugs and syntax errors detecting.

4.4 Verification Tools

The project is implemented by C# programming language that is supported by platform Unity3D. The program is written through IDE Visual Studio 2017 which supports following tools:

- Unit Test Generator

  Unit Test Generator will be used to decrease workload involved in creating new unit tests. It helps the routine test creation tasks. It also provides the ability to generate and configure test project and test class.

- Automatic code checking

  Visual Studio automatically help check code error when developer is programming to pick out errors including syntax error, grammar error.

4.5 Non-Testing Based Verification

Code walkthrough will be done by developer and a peer reviewer to identify any defects. They both review codes and do logic analysis to see if the program satisfies all requirements in SRS. The walkthrough should also find and fix possible bugs and peer reviewer is expected to provide any comment to help developer improve the program.

[Great to see a code walkthrough! However, could you please give some more detail on how the walkthrough will be done. How many participants? How will they be recruited? What documentation will they be given? etc. What specific guidelines will there be for the meeting and the process? If you google "code walkthrough" you will find guidance on how to prepare for and structure the meeting. How will the results of the code walkthrough be summarized? —SS] [Actually development team only has one member Marshall :). I assume to ask a classmate to help review my code and make comments about possible errors. Perhaps I should call this peer review because my program doesn't have so many lines of code. —Author]

5 System Test Description

Section 5.1.1 focuses on verifying correctness of users inputs that covers R1 and R2. Testing cases in this section ensure Breaking Effect can handle both valid inputs and invalid inputs. Corresponding error messages for different situations of invalid inputs including incomplete inputs, inputs in incorrect data types and inputs do not satisfy data constraints.
Sections from 5.1.2 focus on verifying outputs from intermediate steps and

make sure all modules in program work correctly that cover other functional requirements including R3, R4, R5.

[Very nice to have explicit cross-references between documents. To keep the information up to date, consider using make to build your documentation. —SS]

5.1 Tests for Functional Requirements

Assumption

- In all test cases, $(X, Y, Z) = (a, b, c)$ means a target object with with explosion point (a, b, c). The target object for all test cases named "targetObj" consists of 32 cubes with size 1 x 1 x 1. It locates under projectDir/Assets/prefab/targetObj.prefab. The script with codes are attached on the prefab already with $E = 5$ and $\mu_k = 0.2$
- $\approx$ consider equal with a delta of 0.01

5.1.1 Getting input from user

This test suite is designed to ensure the program can handle both valid and invalid inputs from user that cover R1 and R2. Inputs provided by user include target object, explosion level (also known as initial momentum after explosion) and coefficient of friction on the ground.

1. testInput

   Type: Functional, Dynamic, Automated

   Initial State: New session

   Input: Please refer to table below

   Output: Please refer to table below.

   How test will be performed: Automated unit test

Name	In	Out	Performed by
CorrectIn	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	"Input verified"	Automated unit test
NoMu	$E = 5$, $(X, Y, Z) = (0, 0, 0)$	InvalidMuException	Automated unit test
NoObj	$E = 5$, $\mu_k = 0.2$	NoObjException	Automated unit test
NoExpLv	$(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	NoELvException	Automated unit test
NonNumE	$E = a$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	InvalidELvException	Automated unit test
NonNumMu	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = a$	InvalidMuException	Automated unit test
OutScoE	$E = -1$ or $E = 11$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	InvalidELvException	Automated unit test
OutScoC	$E = 5$, $(X, Y, Z) = (0, 99999, 0)$, $\mu_k = 0.2$	InvalidCoorYException	Automated unit test
OutScoM	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 99$	InvalidMuException	Automated unit test

5.1.2 Acquire pieces

This section covers R3, R4 in SRS that class PieceObj is defined as required. Instance of PieceObj can be created by acquiring each piece in the scene correctly.There is an assumption for this section that all tests in 5.1.1 are passed.

Acquire all pieces

1. pieceInit

   Type: Functional, Dynamic, Automated

   Initial State: New Session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$

   Output: A list of piece objects: pieceObj that pieceObj.length = 32. Piece shape is random that is decided by cutting method.

   How test will be performed: Automated unit test.

[You can be more specific here. What is the shape. What is the centre of gravity? Why isn't this test automated? —SS] [Actually shape is decided by cutting method. Now I use external software to cut the object. Each piece is also considered as an object in Unity. —Author]

Calculation of angle between initial speed and horizontal(θ_1) as well as the angle between x axiom and projection on horizontal of initial speed(θ_2)

1. testAngles1

   Type: Functional, Dynamic, Automated

   Initial State: New Session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$, $(x_n, y_n, z_n) = (1.5, 2.0, 0.5)$; targetObj.transform.childCount = 32

   Output: $pieceObj[0].\theta_1 = arctan\frac{y_n - Y}{\sqrt{(z_n - Z)^2 + (x_n - X)^2}} \approx 0.90$ and $pieceObj[0].\theta_2 = arctan\frac{x_n - X}{z_n - Z} \approx 1.25$

   How test will be performed: Automated unit test. [Here we take $(X, Y, Z) = (0, 0, 0)$ as input case for System test. Test cases for different values will be covered in unit test part section 6 —Author]

   [Rather than be general, you should give specific input values and specific output values. This comment applies elsewhere in this document as well. Again, it isn't clear why this has to be a manual test.Include this test in automated test cases —SS] [Give specific input and output. It is able to do the test automatically —Author]

2. testAngles2

   Type: Functional, Dynamic, Automated

   Initial State: New Session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$, $(1.5, 2.0, 1.5)$

   Output: $\theta_1 \approx 0.76$ and $\theta_2 \approx 0.79$

   How test will be performed: Automated unit test.

3. testAngles3

   This test case is designed to make sure thwe workaround for "divide by 0" case.

   Type: Functional, Dynamic, Automated

   Initial State: New Session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$, $(0, y_n, 0)$ and $0y_n$ can't be zero

   Output: $\theta_1 = 90° \approx 1.57$ and $\theta_2 = 90° \approx 1.57$

   How test will be performed: Automated unit test.

5.1.3 Calculation of displacement for each piece

This section covers R5 that displacement of each piece in the air should be calculated correctly. Testing for edge cases and error handling will be covered in Section 6.
There is an assumption for this section that all tests in 5.1.1, 5.1.2 are passed.

1. testMotionAir

   Type: Functional, Dynamic, Automated

   Initial State: New Session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$, $\Delta t = 0.02$, $t = 2.86$

   Output: $S_x = v_0 \cdot cos\theta_1 \cdot cos\theta_2 \cdot \Delta t \approx 0.59$, $S_z = v_0 \cdot cos\theta_1 \cdot sin\theta_2 \cdot \Delta t \approx 0.20$ and $S_y = (v_0 \cdot sin\theta_1 - gt) \cdot \Delta t - \frac{1}{2}g \cdot \Delta t^2 \approx 0.40$. Above output should be shown both in text in console and in vision on the screen.

   How test will be performed: Automated unit test.

2. testMotionGround

   Type: Functional, Dynamic, Automated

   Initial State: New Session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$, $\Delta t = 0.02$, $t = 2.86$

   Output: $a = \mu_k g$, $v_x = v_x^{'} + a\Delta t$, $v_z = v_z^{'} + a\Delta t$, $S_x = v_x \cdot \Delta t - \frac{1}{2}a\Delta t^2 \approx 0.58$, $S_z = v_z \cdot \Delta t - \frac{1}{2}a\Delta t^2 \approx 0.19$

   How test will be performed: Automated unit test.

5.2 Tests for Nonfunctional Requirements

5.2.1 Usability

Installation of Unity3D is not in the scope of Breaking Effect's usability test.

1. Participants
   Classmate: Jason (RenJie)
   Other participants: My 2 roommates. [Hope it is acceptable:) —Author]

2. Document given
   Readme.doc under project folder.

3. Task for participants
   Run test case successfully by reading Readme.doc individually . Without help from others.

4. How test will be measured and performed
   According to content of Readme.doc and complexity of the program. If all participants can run test case successfully by only reading Readme.doc individually, then Usability of the program is good.

5.2.2 Performance test

1. testPerformance

   Type: Nonfunctional,Dynamic, Manual

   Initial State: New session

   Input/Condition: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$ The range of the number of pieces is 4 - 64.

   Output/Result: Motion of each piece.

   How test will be performed: Adjust amount of pieces to see the performance that if program can run smoothly in huge amount of pieces.

[Be more specific. What will the range of the number of pieces be? How will you judge smoothness? You might need to also summarize the target hardware. Can you summarize the total time of calculation and plot it? —SS] [The range will be 4-64. Will judge smoothness by number of frame per second. —Author]

5.2.3 Maintainability

1. testMaintain

   Type: Nonfunctional,Dynamic, Manual

   Initial State: New session

   How test will be performed: Extend the program with additional physics effect rebound (not perfectly).

5.2.4 Reusability

1. testReuse

   Type: Nonfunctional,Dynamic, Manual

   Initial State: New session

   How test will be performed: Combine Breaking Effect with existing Unity3D project.

5.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4	R5
5.1.1	X	X			
5.1.2			X	X	
5.1.3					X

Table 1: Traceability Matrix Showing the Connections Between Test Cases and Requirements

6 Unit Testing Plan

Unit Test Generator provided by Visual Studio 2017 will be used to implement automated unit testing for this project.
For each function, a test class will be created through Unit Test Generator. Developer provides inputs, runs automated test to see if it is passed.
Inputs will be provided by different target objects, initial locations, initial momentum, coefficient of friction.
Unit tests for this project focuses on edge cases checking and error handling. Unit tests for input handling module M3 are covered in 5.1.1.Unit tests for Acquire Pieces module M5 are covered in 5.1.2. Unit tests for Displacement calculation Module M6 are covered in 5.1.3.Test cases related to functional requirements covered in previous sections will not be included in this section.

6.1 Unit test for Piece Object Module (M4)

1. ConstructorPieceObj

   Type: Functional, Dynamic, Automated

   Initial State: New session.

   Input: Refer to table below

   Output: Refer to table below

   How test will be performed: Automated unit testing.

Name	In	Out	Exception
UTPieceClass1	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].obj.name="Cube1"	-
UTPieceClass2	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].onGround = false	-
UTPieceClass3	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].initSpeed = 50	-
UTPieceClass4	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].gravity = -9.8	-
UTPieceClass5	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].stop = false	-
UTPieceClass6	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].speedLastFrameX = 1	-
UTPieceClass7	$E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$	pieceObj[0].speedLastFrameZ = 1	-

6.2 Unit test for Target Object Module (M8)

1. UTTarObj1

   Type: Functional, Dynamic, Automated

   Initial State: New session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$

   Output: targetObj.transform.childCount = 32

   How test will be performed: Automated unit testing.

6.3 Unit test for Collision with ground detection Module (M9)

1. UTCollision1

   Type: Functional, Dynamic, Manually

   Initial State: New session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$;

Output: It is clear to see that pieces drop on the ground.(Don't go through the ground). pieceObj[1].onGround = true

How test will be performed: Manually test by developer

6.4 Unit test for Output Module (M10)

1. UTOutput1

   Type: Functional, Dynamic, Manually

   Initial State: New session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$

   Output: Visualization on screen works well.

   How test will be performed: Manually test by developer

6.5 Unit test for Camera controlling Module (M11)

1. UTOutput1

   Type: Functional, Dynamic, Automated

   Initial State: New session.

   Input: $E = 5$, $(X, Y, Z) = (0, 0, 0)$, $\mu_k = 0.2$

   Output: User can control camera moving by keyboard and mouse.

   How test will be performed: Manually test by developer

References

7 Appendix

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

symbol	**value**	**description**
g	$9.8 m/s^2$	gravity acceleration

7.2 Usability Survey Questions?

Usability survey will not be done for this project.