Module Interface Specification for Breaking Effect

Marshall Xiaoye Ma

December 7, 2017

1 Revision History

Date	Version	Notes
Date 2017-11-17	1.0	New doc

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/MaXiaoye/cas741/blob/master/Doc/SRS/SRS.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for Breaking Effect. [It is usually a good idea to avoid one sentence paragraphs. —SS]

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.

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.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/MaXiaoye/cas741.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Breaking Effect.

Data Type	Notation	Description
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
String	String	represents sequences of characters.
Object	Object	A data structure to store attributes of input target object that provided by Unity3D.
PieceObject	PieceObj	A data structure to store attributes of pieces that generated as intermediate steps.

[As far as I can tell, you don't actually define Object or PieceObj anywhere. You probably haven't done this because they are implemented elsewhere. (I'm assuming they are

implemented in Unit.) However, you still need a spec of Object and PieceObj for your MIS to make sense. You should create a simplified interface for these two ADTs. You only need to document those parts that you actually need for your specification. —SS]

The specification of Breaking Effect uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Breaking Effect uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module Piece Object Module Pieces initialization module Displacement calculation module
Software Decision Module	Target Object Module Collision with ground detection Module Output Module Camera controlling Module

Table 1: Module Hierarchy

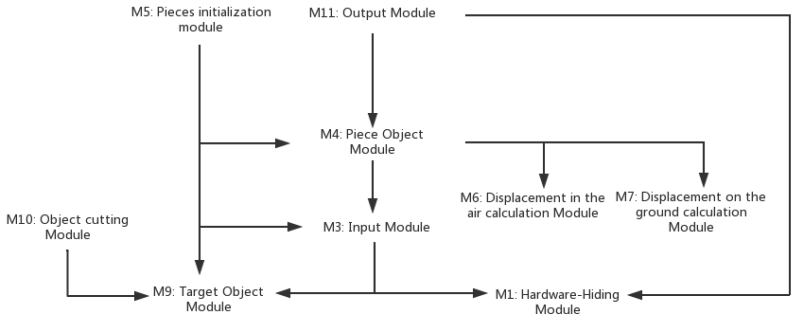

Figure 1: Use hierarchy among modules

[This figure is difficult to read. You could make it bigger, or save it as a pdf, which avoids the rasterization of png. —SS]

6 MIS of Input Module(M3)

This module collect verifies input from user and store in corresponding variables. Include position of target object, explosion level, coefficient of ground friction.

6.1 Module

InputModule

6.2 Uses

Hardware-Hiding Module (M1)

6.3 Syntax

6.3.1 Exported Access Programs

Name	In	Out	Exceptions
InputVerify()	$\mathbb{R}^2; TargetObj$	-	InvalidInput

6.4 Semantics

6.4.1 Environment Variables

$\mu_k : \mathbb{R}$
$E : \mathbb{R}$
$TargetObj : Object$

[This isn't how environment variables are used. For a project like yours the only environment variables that would apply are external files and the screen. —SS] [Why do you have TargetObj as both an environment variable and an input. If it is an input, it isn't going to also appear as a state (or environment) variable. —SS]

6.4.2 State Variables

None

6.4.3 Assumptions

- Object is a 3D model in Unity3D, which contains its position (X, Y, Z)
- User needs provide a 3D model and attach the program to it. [What does this mean? If this is a Unity step, you should provide a pointer to an external resource that describes what this means. —SS]

6.4.4 Access Routine Semantics

InputVerifiy(μ_k, E, $TargetObj$): [The type in the syntax section does not match what is written here? $\mathbb{R}^2$ means a sequence of two reals. That means your syntax calls for two inputs, but you have 3. On the next page μ_k could be null. null is not a float. —SS]

- transition: N/A
- output: Exceptions or None.
- exception:
  exc := (μ_k = null $\Rightarrow$ NoMuException)
  exc := (E = null $\Rightarrow$ NoELvException)
  exc := ($X \notin \mathbb{R} \vee (X \leq -1000) \vee (X \geq 1000) \Rightarrow$ InvalidCoorXException)
  exc := ($Z \notin \mathbb{R} \vee (Z \leq -1000) \vee (Z \geq 1000) \Rightarrow$ InvalidCoorZException)
  exc := ($Y! = 0 \Rightarrow$ InvalidCoorYException)
  exc := ($E \notin \mathbb{R} \vee (E \leq 0) \vee (E \geq 10) \Rightarrow$ InvalidELvException)
  exc := ($\mu_k \notin \mathbb{R} \vee (\mu_k \leq 0) \vee (\mu_k \geq 1) \Rightarrow$ InvalidMuException)

7 MIS of piece object module(M5)

Customize class for pieces. Pieces are generated after explosion happens to replace original target object from input.

7.1 Module

ObjCutModule

7.2 Uses

Input Module(M3)
Displacement in the air calculation Module(M8)
[This reference did not work when I clicked on it. —SS]

7.3 Syntax

7.3.1 Exported Access Programs

Name	In	Out	Exceptions
PieceObj()	$\mathbb{R}^2$; $Object$	-	-
MoveInAir()	-	-	-
MoveOnGround()	-	-	-
Translate()	$\mathbb{R}^3$	-	-

[These don't look like access programs. Isn't PieceObj a type? If it is a constructor for PieceObj, then you should document this in the MIS for the PieceObj ADT. —SS]

7.4 Semantics

7.4.1 Environment Variables

$PieceObj.obj : Object$
$PieceObj.obj.transform.x : \mathbb{R}$
$PieceObj.obj.transform.y : \mathbb{R}$
$PieceObj.obj.transform.z : \mathbb{R}$
$PieceObj.onGround : Boolean$
$PieceObj.stop : Boolean$
$PieceObj.\theta_1 : \mathbb{R}$
$PieceObj.\theta_2 : \mathbb{R}$
$PieceObj.initSpeed : \mathbb{R}$
$PieceObj.speedThisFrameX : \mathbb{R}$
$PieceObj.speedLastFrameX : \mathbb{R}$
$PieceObj.speedThisFrameZ : \mathbb{R}$
$PieceObj.speedLastFrameZ : \mathbb{R}$
[Please review what environment variables are used for. —SS]

7.4.2 State Variables

None

7.4.3 Assumptions

- obj is the 3D model of PieceObj in scene.
- x, y, z are coordinates of object.
- onGround indicates if the object is on the ground.
- stop indicates if the speed of object already equals to 0.
- θ_1 is the angle between initial speed v_0 and horizontal.
- θ_2 is the angle between x axiom and projection on horizontal of initial speed
- initSpeed is the initial speed the object has when explosion happens.
- PieceObj() is constructor that use input from M3.
- Translate() controls motion of the object.

- MoveInAir() and MoveOnGround() controls motion of the object by calling Translate(). It check onGround firstly to make sure the object is in the air or on the ground. Based on value of bool variable onGround, that call and provide corresponding destination as input to Translate(). Destination to Translate() is calculated by M8 and M9.

7.4.4 Access Routine Semantics

PieceObj():

- transition: Initialize PieceObj
- output: None
- exception: None

thetaOneCalc(): [This (and other access programs) do not appear in the syntax section. —SS]
Calculate the angle between initial speed v_0 and horizontal θ_1. [θ_1 and θ_2 are values of each PieceObj —Author]
Equation: $\theta_1 = arctan\frac{y_n}{\sqrt{(x_n-X)^2+(z_n-Y)^2}}$
Convert equation to codes:
Mathf.Atan(PieceObj.y / Mathf.Sqrt(Mathf.Pow(PieceObj.x - TargetObj.x,2) + Mathf.Pow(PieceObj.z - TargetObj.z,2))); [Redefining the equation in code isn't usually done in the MIS. If you instead defined a local function with parameters, and then in your spec called the local function with the parameters set to the code object variables, that would help explain how the parts of your implementation are connected. —SS]

- transition: $\theta_1 : null \rightarrow \mathbb{R}$
- output: None
- exception: None

thetaTwoCalc():
Calculate the angle between x axiom and projection on horizontal of initial speed θ_2.
Equation: $\theta_2 = arctan\frac{x_n-X}{z_n-Z}$
Convert equation to codes:
Mathf.Atan2(PieceObj.x - TargetObj.x, PieceObj.z - TargetObj.z)

- transition: $\theta_2 : null \rightarrow \mathbb{R}$
- output: None
- exception: None

MoveInAir(): [use M8 here —Author]

- transition: Move PieceObj in the air by updating x, y, z
- output: None
- exception: None

MoveOnGround(): [use M8 here —Author]

- transition: Move PieceObj on the ground by updating x, y, z
- output: None
- exception: None

8 MIS of pieces initialization module (M6)

8.1 Module

PieceInitModule

8.2 Uses

Input Module(M3)
Piece Object Module(M5)
target object module(M4)

8.3 Syntax

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
targetObj	-	-	-
subObj[]	-	-	-
pieceObj[]	-	-	-
PieceObj()	$\mathbb{R}^2$; *Object*	PieceObj	-

8.4 Semantics

8.4.1 Environment Variables

targetObj: Object
subObj[]: list of Object
pieceObj[]: list of Piece Object

8.4.2 State Variables

None

8.4.3 Assumptions

- targetObj is the 3D model of target object in scene.
- subObj[] is a list of sub objects under target Object. Since all pieces make up the whole target object, all pieces object are considered as sub objects of the target object in Unity3D. We need to get all sub objects firstly and then use these sub objects to construct piece objects defined by myself.
- pieceObj[] is a list of piece objects defined by myself.
- PieceObj() is constructor of piece object that defined in M5. [use M5 here. —Author]

8.4.4 Access Routine Semantics

Do traversal to initialize all pieces. [Each piece is stored as an instance of class PieceObj defined in M5. Gravity center is position value in PieceObj —Author]
targetObj = GameObject.Find(”targetObj”);
subObj = targetObj.GetComponentsInChildren¡Transform¿();
pieceObj = new PieceObj[targetObj.transform.childCount];
for (int i = 1; i ¡ subObj.Length; i++) pieceObj[i - 1] = new PieceObj(subObj[i].gameObject, initSpeed, g);

9 MIS of Displacement calculation module(M8)

Calculate and output trace of motion for each piece in the air by using follow equations.

9.1 Module

DisAirCalModule

9.2 Uses

Input Module(M3)

9.3 Syntax

9.3.1 Exported Access Programs

Name	In	Out	Exceptions
DisAirCalX()	$\mathbb{R}$; PieceObj; TargetObject	$\mathbb{R}$	-
DisAirCalY()	$\mathbb{R}$; PieceObj; TargetObject	$\mathbb{R}$	-
DisAirCalZ()	$\mathbb{R}$; PieceObj; TargetObject	$\mathbb{R}$	-
DisGroCalX()	$\mathbb{R}$; PieceObj; TargetObject	$\mathbb{R}$	-
DisGroCalZ()	$\mathbb{R}$; PieceObj; TargetObject	$\mathbb{R}$	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Access Routine Semantics

DisAirCalX():
Equation: $v_0 = 10 * E, S_x = v_0 \cdot cos\theta_1 \cdot sin\theta_2 \cdot \Delta t$ [Rather than hard code in the value 10, you should use a symbolic constant. —SS] [Based on A**??** [This cross-reference didn't seem to work —SS] in SRS that value of initial velocity given by explosion is ten times input E unit length in unity per second. Δt is the gap between each frame that input from unity3D —Author]
Convert equation to codes:
initSpeed * Mathf.Cos(PieceObj.theta1) * Mathf.Sin(PieceObj.theta2) * Time.deltaTime

- transition: None
- output: $S_x : \mathbb{R}$
- exception: None

DisAirCalY():
Equation: $S_y = (v_0 \cdot sin\theta_1 - g \cdot t) \cdot \Delta t - \frac{1}{2} g \cdot \Delta t^2$ [t is real time since the explosion happens. So that $v_0 \cdot sin\theta_1 - g \cdot t$ means the initial speed on vertical direction at the beginning of each frame —Author]
Convert equation to codes:
(initSpeed * Mathf.Sin(PieceObj.theta1) + g * Time.realtimeSinceStartup) * Time.deltaTime + 1 / 2 * g * Time.deltaTime * Time.deltaTime

- transition: None
- output: $S_y : \mathbb{R}$

- exception: None

DisAirCalZ():
Equation:$S_z = v_0 \cdot cos\theta_1 \cdot cos\theta_2 \cdot \Delta t$
Convert equation to codes:
initSpeed * Mathf.Cos(PieceObj.theta1) * Mathf.Cos(PieceObj.theta2) * Time.deltaTime)

- transition: None
- output: $S_z : \mathbb{R}$
- exception: None

DisGroCalX():
Euqation: $a = \mu_k g$; $S_x = (v_0 \cdot cos\theta_1 \cdot sin\theta_2 - at) \cdot \Delta t - \frac{1}{2} a \cdot \Delta t^2$
Convert equation to codes:
(initSpeed * Mathf.Sin(PieceObj.theta2) * Mathf.Cos(PieceObj.theta1) - a * Time.realtimeSinceStartup) * Time.deltaTime - 1 / 2 * a * Time.deltaTime * Time.deltaTime

- transition: None
- output: $S_x : \mathbb{R}$
- exception: None

DisGroCalZ():
Euqation: $a = \mu_k g$; $S_z = (v_0 \cdot cos\theta_1 \cdot cos\theta_2 - at) \cdot \Delta t - \frac{1}{2} a \cdot \Delta t^2$
Convert equation to codes:
(initSpeed * Mathf.Cos(PieceObj.theta2) * Mathf.Cos(PieceObj.theta1) - a * Time.realtimeSinceStartup) * Time.deltaTime - 1 / 2 * a * Time.deltaTime * Time.deltaTime

- transition: None
- output: $S_z : \mathbb{R}$
- exception: None

10 MIS of target object module(M4)

Object class provided by platform

10.1 Module

TarObjModule

10.2 Uses

Input Module(M3)

10.3 Syntax

10.3.1 Exported Access Programs

Name	In	Out	Exceptions
Find()	*String*	Object	-

10.4 Semantics

10.4.1 Environment Variables

$TargetObj.transform.X : \mathbb{R}$
$TargetObj.transform.Y : \mathbb{R}$
$TargetObj.transform.Z : \mathbb{R}$
position: $\mathbb{R}^3$

10.4.2 State Variables

KeyCode.Space: Boolean. This bool value indicates if key space is pressed on keyboard.

10.4.3 Assumptions

- X, Y, Z are coordinates of object. position is 3D vector that contains X, Y, Z while it is also considered as gravity center location of the object

10.4.4 Access Routine Semantics

Find($name : String$):

- transition: Initialize an instance of target object by searching name.
- output: None
- exception: None

11 MIS of Collision with ground detection Module (M11)

Detect if there is a collision between a piece and the ground. If so, set onGround value to true.

11.1 Module

ColDetectModule

11.2 Uses

Input Module(M3)

11.3 Syntax

11.3.1 Exported Access Programs

Name	In	Out	Exceptions
OnTriggerEnter()	-	-	-

11.4 Semantics

11.4.1 State Variables

onGround

11.4.2 Access Routine Semantics

OnTriggerEnter():

- transition: pieceObj.onGround: false → true;
- output: None
- exception: None

12 MIS of Output Module(M12)

Unity3D interface with codes by calling function update() each frame. Unity3D convert data into visualization. Provide free camera for people to control view.

12.1 Module

OutputModule

12.2 Uses

Displacement calculation module(M8)

12.3 Syntax

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
update()	codes to be run each frame	Visualization	-
start()	-	-	-
CameraControl	-	-	-

12.4 Semantics

12.4.1 State Variables

Scene;

12.4.2 Assumptions

- Start is called on the frame when a script is enabled just before any of the Update methods is called the first time.
- Fixedupdate is called every frame. In Fixedupdate(), we listen if space is pressed as start point of the explosion. It also keeps updating status of all objects in the scene to convert location of objects to visualization that can be seen on the screen.

12.4.3 Access Routine Semantics

start():

- transition: Initialization of scene.
- output: None
- exception: None

Fixedupdate():

- transition: Piece objects → Visualization
- output: None
- exception: None

[You seem to be on the right track, but not following the MIS template makes the design difficult to review. Please clarify your access programs, their input types, output types, your state variables, and your environment variables. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

13 Appendix

[If you don't use this section, you can remove it. —SS]