Tamias2D: Unit Verification and Validation Plan for a 2D Rigid Body Game Physics Library

Oluwaseun Owojaiye

December 16, 2018

1 Revision History

Date	Version	Notes
Nov. 29, 2018	1.0	Initial Draft
Dec. 1, 2018	1.1	Updates from github feedback

2 Symbols, Abbreviations and Acronyms

See MIS documentation at https://github.com/smiths/caseStudies/blob/gamephy_MIS/CaseStudies/gamephys/docs/Design/MIS/GamePhysicsMIS.pdf

symbol	description
T	Test

Contents

List of Tables

List of Figures

This document describes the unit verification and validation (V&V) plan for Tamias2D, a 2D Physics Game Library. The purpose of this document is to specify the details of testing each unit of software by providing test cases based on the modules in the library's module interface specification (MIS) document. The MIS, with all other documentation relating to Tamias2D, can be found at: https://github.com/smiths/caseStudies/tree/master/CaseStudies/gamephys.

3 General Information

3.1 Purpose

This section details the summary of what is being tested in this document, the objectives of this document and references for this document. Tamias2Dis a 2D rigid body physics library that can be used to simulate the interaction between rigid bodies.This software is able to simulate movement, rotation and collision of rigid bodies.

3.2 Scope

All modules will be verified using the testcases listed in this document.

4 Plan

4.1 Verification and Validation Team

Member(s) of the verification and validation team will include myself, Olu.

4.2 Automated Testing and Verification Tools

PyTest framework will be used for automated unit testing. A script with all the testcases covering all Tamias2D modules will be created by me.

4.3 Non-Testing Based Verification

Not applicable for Tamias2D

5 Unit Test Description

The test cases discussed in this section are based on the Module Interface Specification(MIS) which can be found at https://github.com/smiths/caseStudies/blob/gamephy_MIS/CaseStudies/gamephys/docs/Design/MIS/GamePhysicsMIS.pdf. Each test case listed covers for all applicable functions that need to be tested in each module to ensure that Tamias2D functions as intended based on the requirements of the software.

5.1 Tests for Functional Requirements

5.1.1 Body Module

Body module is responsible for storing the physical properties of an object such as mass, position, rotation properties, velocity, e.t.c and provides operations on rigid bodies such as setting the mass, moment, applying force e.t.c.

1. UTC1 : Validate apply_force1

   Type: Automatic.

   Initial State: Initial force on a body is Vec2(0.0, 0.0)

   Input: b.apply_force(Vec2(10.0, 6.0))

   Output: New total force applied to body is Vec2(10.0, 6.0)

   How test will be performed: PyTest
   Ref. source: https://onlinemschool.com/math/assistance/vector/calc/

2. UTC2: Validate apply_force2

   Type: Automatic

   Initial State: Vec2(10.0, 6.0)

   Input: b.apply_force (Vec2(-20.0, 2.0))

   Output: New total force is Vec2(-10.0, 8.0) Ref. source: https://onlinemschool.com/math/assistance/vector/calc/

3. UTC3: Validate apply_force3

   Type: Automatic

   Initial State: Vec2(-2.0, -1.0)

   Input: b.apply_force (Vec2(-5.0, -8.0))

   Output: New total force is Vec2(-7.0, -9.0)

   How test will be performed: PyTest
   Ref. source: https://onlinemschool.com/math/assistance/vector/calc/

4. UTC4: Validate apply_torque

   Type: Automatic

   Initial State: 0.0

   Input: apply_torque(100.0)

   Output: self.torque=100.0

   How test will be performed: PyTest

5. UTC5: Validate set_space

   Type: Automatic

   Initial State:

   Input: body.set_space(space)

   Output: Expected result of(space is body.space): True

   How test will be performed: PyTest

6. UTC6: Validate - Initialize dynamic body
   This testcase is to initialize a body object. A body object is initialized with these parameters:
   body = Body(position_, mass_, inertia_, shape_, angle=0, bodyType_= DYNAMIC, restitution_ = 0) to create and initialize a body.

Initial State:

Input: space.add_body(body)
body = Body(Vec2(250,0), 1000, 100, shape, 0, Body.DYNAMIC,1)

Output: body = Body(Vec2(250,0), 1000, 100, shape, 0, Body.DYNAMIC,1),
1 body created

How test will be performed: PyTest

7. UTC7: Validate - Initialize static body
   This testcase is to initialize a body object. A body object is initialized with these parameters:
   body = Body(position_, mass_, inertia_, shape_, angle=0, bodyType_= STATIC, restitution_ = 0) to create and initialize a body.

   Initial State:

   Input: space.add_body(body)
   body = Body(Vec2(250,0), 1000, 100, polyShape, 0, Body.STATIC,1)

   Output: body = Body(Vec2(250,0), 1000, 100, polyShape, 0, Body.STATIC,1);
   1 body created

   How test will be performed: PyTest

8. UTC8: Validate - Initialize multiple bodies
   This testcase is to initialize multiple body objects. A body object is initialized with these parameters:
   body = Body(position_, mass_, inertia_, shape_, angle=0, bodyType_= DYNAMIC, restitution_ = 0) to create and initialize a body.

   Initial State:

   Input: space.add_body(body)
   body = Body(Vec2(250,0), 1000, 100, polyShape, 0, Body.DYNAMIC,1)
   body = Body(Vec2(200,0), 1000, 100, polyShape, 0, Body.DYNAMIC,1)

Output: body = Body(Vec2(250,0), 1000, 100, polyShape, 0, Body.DYNAMIC,1)
body = Body(Vec2(200,0), 1000, 100, polyShape, 0, Body.DYNAMIC,1)
2 bodies created

How test will be performed: PyTest

9. UTC9: Validate - Update Body
   This testcase is to update body object(s), after a cycle/tick.

   Initial State:

   $$p_{initial} = (20, 20), v_{initial} = (0, 0)$$

   Input: apply_force(20, 20)
   Output: The table below displays the position and time of a rigid body when a force is applied. The result shows the output velocity and position from 0.5secs to 3secs. How test will be performed: PyTest

Position	Velocity	after time t(secs)
(20.025, 20.025)	(0.05, 0.05)	0.5sec
(20.175, 20.172)	(0.15, 0.15)	1secs
(20.625, 20.613)	(0.15, 0.15)	1.5secs
(21.625, 20.592)	(0.5, 0.5)	2.0secs
(23.500, 23.430)	(0.75, 0.74)	2.5secs
(26.650, 26.520)	(1.05, 1.03)	3.0secs

   Ref. source: https://calculator.tutorvista.com/physics/535/velocity-calculator.html

5.1.2 Vector Module

Vector module provides operations such as addition, scalar and vector multiplication, dot and cross products e.t.c.

1. UTC10: Validate vector addition

   Type: Automatic.

Initial State:

Input: Vec2(2.22,5.17) + Vec2(1.00,1.00)

Output: Vec2(3.22, 6.17)

How test will be performed: PyTest
Ref. source: https://onlinemschool.com/math/assistance/vector/calc/

2. UTC11: Validate vector subtraction1

   Type: Automatic

   Initial State:

   Input: Vec2(2.0 ,5.0) - Vec2(1.0 ,2.0)

   Output: Vec2(1.0, 3.0)
   How will test be performed: PyTest
   Ref. source: https://onlinemschool.com/math/assistance/vector/calc/

3. UTC12: Validate vector subtraction2

   Type: Automatic

   Initial State:

   Input: Vec2(-20.0,5.0) - Vec2(1.0,-10.0)

   Output: Vec2(-21.0, 15.0)
   How will test be performed: PyTest
   Ref. source: https://onlinemschool.com/math/assistance/vector/calc/

4. UTC13: Validate scalar multiplication1

   Type: Automatic

   Initial State:

   Input: Vec(2.0, 4.0) * 2.0

   Output: Vec2(4.0, 8.0)

How test will be performed: PyTest Ref. source: https://onlinemschool.com/math/assistance/vector/multiply3/

5. UTC14: Validate scalar multiplication2

   Type: Automatic

   Initial State:

   Input: Vec(-2.0, -4.0) * 2.0

   Output: Vec2(-4.0, -8.0)

   How test will be performed: PyTest Ref. source:https://onlinemschool.com/math/assistance/vector/multiply3/

6. UTC15: Validate scalar division1

   Type: Automatic

   Initial State:

   Input: apply force Vec(-10.2, 4.2) / 2.0

   Output: New total force is Vec2(-5.1, 2.1)

   How test will be performed: PyTest

7. UTC12: Validate scalar division2

   Type: Automatic

   Initial State:

   Input: Vec(22.24, 4.34) / 2.0

   Output: Vec2(11.12, 2.17)

   How test will be performed: PyTest

8. UTC16: Validate vector magnitude1

   Type: Automatic

   Initial State:

   Input: Velocity = Vec(2.0, 0.0)

Output: magnitude = V.mag() = 2.0

How test will be performed: PyTest
Ref: https://onlinemschool.com/math/assistance/vector/length/

9. UTC17: Validate vector magnitude2

   Type: Automatic

   Initial State:

   Input: Velocity = Vec(-2.0, -4.0)

   Output: magnitude = V.mag() = -5.65

   How test will be performed: PyTest
   Ref: https://onlinemschool.com/math/assistance/vector/length/

10. UTC18: Validate vector dot product

    Type: Automatic

    Initial State:

    Input: Vec2.dot(Vec2(1.0,2.0), Vec2(2.0,1.0))

    Output: magnitude = 4.0

    How test will be performed: PyTest
    Ref. source: https://onlinemschool.com/math/assistance/vector/multiply/

5.1.3 Shape Module

Shape module is responsible for storing the surface properties of an object such as restitution and provides operations on shapes, such as setting the coefficient of restitution. e.t.c.

1. UTC19 : Validate set_angle

   Type: Automatic.

   Initial State:

   Input: shape.set_angle(45)

Output: self.angle = 45

How test will be performed: PyTest

2. UTC20: Validate set_position

   Type: Automatic

   Initial State:

   Input: shape.set_pos(Vec2(2,2))

   Output: Vec2(2,2)

3. UTC21: Validate get_vertices

   Type: Automatic

   Initial State:

   Input: shape.get_verts(Box(Vec2(0,0), Vec2(100,100), 0))

   Output: 4

   How test will be performed: PyTest

5.1.4 Space Module

The space module is responsible for all the rigid bodies and shape interaction. It is the container for simulation.

1. UTC22 : Validate set_gravity

   Type: Automatic.

   Initial State:

   Input: space.set_gravity(Vec2(0, 9.8))

   Output: (Vec2(0, 9.8)

   How test will be performed: PyTest

2. UTC23: Validate add_body

   Type: Automatic

Initial State:

Input: space.add_body(b)

Output: len(self.bodies) = 1

5.1.5 Collision Solver Module

This module is responsible for shape to shape collision detection, response and resolution.Te new velocity,positions and orientation of bodies will be calculated and updated.

1. UTC24 : Validate _is_intersecting_with_True
   This testcase checks for collision i.e. if points on rigid bodies are intersecting.
   Type: Automatic.

   Initial State:

   Input:
   shape1 = Box(Vec2(0,0), Vec2(100,100), 0)
   body1 = Body(Vec2(250,0), 1000, 100, shape1, 0, Body.DYNAMIC,0)

   shape2 = Box(Vec2(0,0), Vec2(100,200), 0)
   body2 = Body(Vec2(300,50), 1000, 100, shape2, 0, Body.DYNAMIC,0)

   Output: _is_intersecting_with(body1, body2): True

   How test will be performed: PyTest

2. UTC25 : Validate _is_intersecting_with_False
   This testcase checks for collision i.e if points on rigid bodies are intersecting.
   Type: Automatic.

   Initial State:

   Input:
   shape1 = Box(Vec2(0,0), Vec2(100,100), 0)
   body1 = Body(Vec2(250,0), 1000, 100, shape1, 0, Body.DYNAMIC,0)

shape2 = Box(Vec2(10,0), Vec2(300,200), 0)
body2 = Body(Vec2(300,50), 1000, 100, shape2, 0, Body.DYNAMIC,0)

Output: _is_intersecting_with(body1, body2): False

How test will be performed: PyTest

5.2 Tests for Nonfunctional Requirements

Non-Functional testing will not be required for unit testing. This will be covered in section 5.2 of the System Verification and Validation document located at:
https://github.com/smiths/caseStudies/blob/gamephy_SysVnVPlan/CaseStudies/gamephys/docs/VnVPlan/SystVnVPlan/SystVnVPlan.pdf.pdf

5.3 Traceability Between Test Cases and Requirements

The table below shows the traceability map between test cases and modules.

Test Case ID	Module
UTC1-UTC9	Body Module
UTC10 - UTC18	Vector Module
UTC19 - UTC21	Shape Module
UTC22 - UTC23	Space
UTC24 - UTC25	Collision Solver

Table 1: Traceability Between Test Cases and Modules

References

6 Appendix

This section provides additional content related to this document.

6.1 Symbolic Parameters

There are no symbolic parameters used in this document.