Tamias2D: System Verification and Validation Plan

Oluwaseun Owojaiye

December 24, 2018

1 Revision History

Date	Version	Notes
2018-10-15	1.0	Initial draft
2018-11-01	1.1	Updates based on document review and issue tracker
2018-12-22	1.2	Updated testcases

2 Symbols, Abbreviations and Acronyms

2.1 Table of Symbols

Symbol	Unit	Description
$\mathbf{a}$	$\mathrm{m\,s^{-2}}$	Acceleration
α	$\mathrm{rad\,s^{-2}}$	Angular acceleration
C_R	unitless	Coefficient of restitution
$\mathbf{F}$	N	Force
g	$\mathrm{m\,s^{-2}}$	Gravitational acceleration ($9.81\ \mathrm{m\,s^{-2}}$)
G	$\mathrm{m^3\,kg^{-1}\,s^{-2}}$	Gravitational constant ($6.673 \times 10^{-11}\ \mathrm{m^3\,kg^{-1}\,s^{-2}}$)
$\mathbf{I}$	$\mathrm{kg\,m^2}$	Moment of inertia
$\hat{\mathbf{i}}$	m	Horizontal unit vector
$\hat{\mathbf{j}}$	m	Vertical unit vector
j	N s	Impulse (scalar)
$\mathbf{J}$	N s	Impulse (vector)
L	m	Length
m	kg	Mass
n	unitless	Number of particles in a rigid body
$\mathbf{n}$	m	Collision normal vector
$\boldsymbol{\omega}$	$\mathrm{rad\,s^{-1}}$	Angular velocity
$\mathbf{p}$	m	Position
$\boldsymbol{\phi}$	rad	Orientation
r	m	Distance
$\mathbf{r}$	m	Displacement
t	s	Time
τ	N m	Torque
$\boldsymbol{\theta}$	rad	Angular displacement
$\mathbf{v}$	$\mathrm{m\,s^{-1}}$	Velocity

2.2 Abbreviations and Acronyms

Symbol	Description
IM	Instance Model
R	Requirement
T	Test
TBD	To be determined
2D	Two-dimensional
SRS	System Requirement Specification

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**
- 2.1 Table of Symbols . . . ii
- 2.2 Abbreviations and Acronyms . . . iii

3 General Information **1**
- 3.1 Summary . . . 1
- 3.2 Objectives . . . 1
- 3.3 References . . . 2

4 Plan **2**
- 4.1 Verification and Validation Team . . . 2
- 4.2 SRS Verification Plan . . . 2
- 4.3 Design Verification Plan . . . 2
- 4.4 Implementation Verification Plan . . . 3
- 4.5 Software Validation Plan . . . 3

5 System Test Description **4**
- 5.1 Tests for Functional Requirements . . . 4
  - 5.1.1 Translational Motion Testing . . . 4
  - 5.1.2 Rotation of 2D Rigid Body Simulation . . . 9
  - 5.1.3 Rotation of 2D Rigid Body Simulation . . . 9
- 5.2 Tests for Nonfunctional Requirements . . . 11
  - 5.2.1 Usability Test . . . 11
  - 5.2.2 Correctness/Performance . . . 12
  - 5.2.3 Reusability . . . 12
  - 5.2.4 Understandability/Maintainability . . . 13
- 5.3 Traceability Between Test Cases and Requirements . . . 13

6 Static Verification Techniques **14**

7 Appendix **15**
- 7.1 Symbolic Parameters . . . 15
- 7.2 Usability Survey Questions? . . . 15

List of Tables

1 Requirements Traceability Matrix 13

List of Figures

This document provides a high-level verification and validation plan for Tamias2D- a 2D Rigid Body Game Physics Library. This document is based on the System Requirement Specification(SRS) document located in the following project repository link: https://github.com/smiths/caseStudies/tree/master/CaseStudies/gamephys. It discusses the verification and validation requirements for Tamias2D, and describes the test strategies and methods that will be used to evaluate the software. The verification and validation of the software utilizes review, analysis, and testing method to determine whether the software product complies with the specifed requirements. These requirements include both functional and non-functional.

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

3 General Information

3.1 Summary

The software being tested is Tamias2D. It is a 2D rigid body game physics library designed to simulate the interaction between rigid bodies in the game space. It will simulate the movement of objects in space, the behavoiur of objects when there is a collision and it tracks a history of the velocity, position and orientation of each object. Since physics libraries are an important part of video game development, game developers would be able to make use of Tamias2Din their products.

3.2 Objectives

The purpose of verification and validation activitiy is to find bugs and defects in the TamiasMini2D software and also to determine if it has met all the required functionality. It is also to verify that the software meets the required standard and the end product conforms with the software requirements based on the SRS. The objectives of System VnV activities for Tamias2Dare to:

- Build confidence in software correctness and performance.
- Verify the maintainability of the software, based on the product's ability to be easily enhanced, modified and reused.
- Verify and demonstrate the ease of use and learning of the software.

3.3 References

[You should introduce the references, not just include a link. —SS] [updated —OO]

1. Software Requirement Specification for Tamias2D: https://github.com/smiths/caseStudies/blob/gamephy_finaldoc/CaseStudies/gamephys/docs/SRS/GamePhysicsSRS.pdf

4 Plan

4.1 Verification and Validation Team

The verification and validation team consists of a one member team: Olu Owojaiye

4.2 SRS Verification Plan

The SRS for the project will be reviewed by Dr. Smith [LaTeX has a rule that it inserts two spaces at the end of a sentence. It detects a sentence as a period followed by a capital letter. This comes up, for instance, with Dr. Smith. Since the period after Dr. isn't actually the end of a sentence, you need to tell LaTeX to insert one space. You do this either by Dr. Smith (if you don't mind a line-break between Dr. and Smith), or Dr. Spencer Smith (to force LaTeX to not insert a line break). —SS] and coursemate Karol Serkis; feedback will be provided. Some SRS feedback for this project have already been provided and addressed using github issue tracker. Once the software has been implemented, the SRS will be reviewed to ensure that software meets all the specified requirements and more feedback will be provided via github issue tracker.

[You can be specific about which classmates are going to review your documents; the specific assignments are in Repos.xlsx. —SS] [updated —OO]

4.3 Design Verification Plan

To ensure that the Design Specification has been properly specified and meets software requirements, Dr. Smith and coursemates Robert White and Hanae

Zlitni will be verifying the software design. The Module Guide and Module Interface Specification located at https://github.com/smiths/caseStudies/tree/gamephy_MG/CaseStudies/gamephys/docs/Design/MG and https://github.com/smiths/caseStudies/tree/gamephy_MG/CaseStudies/gamephys/docs/Design/MIS respectively will contain information about the software design. Feedback is expected to be provided by reviewers via github issue tracker. [You should have links to MG and MIS. —SS] [updated with links —OO]

4.4 Implementation Verification Plan

The implementation of Tamias2D will involve inspection of the software to ensure that all the required features have been implemented successfully and are functional. Once the development activities are completed, Dr. Smith and one of my CAS741 coursemate will be assigned to perform the implementation verification activities. The software will be installed by the reviewer and system test cases specified in Section 5 will be executed as well as unit test cases located at https://github.com/smiths/caseStudies/blob/gamephy_UnitVnV/CaseStudies/gamephys/docs/VnVPlan/UnitVnVPlan/UnitVnVPlan.pdf. Reviewers are expected to verify both functional and non-functional requirements specified below except otherwise stated. Exploratory testing can also be performed by testers. Any implementation verification issues will be reported and tracked via gitHub issue tracker and these issues will be resolved in order of severity by myself. After the issues raised have been fixed, they will be sent back to the reviewer(s) for re-verification.

4.5 Software Validation Plan

Tamias2D will be validated at a future phase with existing physics library Pymunk.

[You should the reason why there is no software validation plan. —SS] [updated —OO]

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 Translational Motion Testing

1. TC1: test_system_horizontal_translation_noforce()
   Description: This test computes the position and velocity of a 2D rigid body that is static over 4 secs. Gravity is not applied. The object does not move, hence velocity and position do not change. Varying positive input values can be used in TC1 for each of the parameters

   Control: Automatic

   Initial State: NA

   Input:
   $\mathbf{p_i(t_0)}$ = (50, 50) ;This is the initial position of body - (x,y) coordinate position
   $\mathbf{v_i(t_0)}$ = 0, 0
   $\mathbf{F}$ = 0, 0
   $\mathbf{m}$ = 10
   $\mathbf{g}$ = 0 (acceleration due to gravity does not apply on a static body)
   Output:
   $\mathbf{p_i(t_1)}$ = (50, 50); $\mathbf{v_i(t_1)}$ = (0, 0)
   $\mathbf{p_i(t_2)}$ = (50, 50); $\mathbf{v_i(t_2)}$ = (0, 0);
   $\mathbf{p_i(t_3)}$ = (50, 50); $\mathbf{v_i(t_3)}$ = (0, 0);
   $\mathbf{p_i(t_4)}$ = (50, 50); $\mathbf{v_i(t_4)}$ = (0, 0);

   How test will be performed: Unit testing with Pytest Ref. source: https://www.calculatorsoup.com/calculators/physics/displacement_v_a_t.php

2. TC2: test_system_horizontal_translation_right()

   Description: This test computes the position and velocity of a 2D rigid body moving at constant acceleration to the right over 4 secs. Gravity does not apply.

Control: Automatic

Initial State: NA

Input:
$\mathbf{p_i}(\mathbf{t_0}) = (20, 20)$ this is the (x,y) coordinate position
$\mathbf{v_i}(\mathbf{t_0}) = 0$
$\mathbf{F} = (100, 0)$
$\mathbf{m} = 10$

Output:
$\mathbf{p_i}(\mathbf{t_1}) = (22.84\ 20.00)$; $\mathbf{v_i}(\mathbf{t_1}) = (10.50, 0.0)$
$\mathbf{p_i}(\mathbf{t_2}) = (31.02, 20.0)$; $\mathbf{v_i}(\mathbf{t_2}) = (20.83, 0.0)$
$\mathbf{p_i}(\mathbf{t_3}) = (44.54, 20.00)$; $\mathbf{v_i}(\mathbf{t_3}) = (31.16, 0.0)$
$\mathbf{p_i}(\mathbf{t_4}) = (60.66, 20.00)$; $\mathbf{v_i}(\mathbf{t_4}) = (40.16, 0.0)$

How test will be performed: Unit testing with pytest

[Where did the answers come from? The reader won't be able to verify what you are saying. You are verifying for one point in time. You can accomplish more by verifying the full history of the change in position. You should provide the closed form solutions for position change under constant acceleration. You then get can run the simulation until the position is zero and verify that the positions are correct. To get one number for your test you can use the Euclidean (or other) norm of your vector and then divide by the norm of the expected result. —SS] [updated result with history of velocity and position over a period of time —OO]

3. TC3: test_system_horizontal_translation_left()

Description: This test computes the position and velocity of a 2D rigid body moving at constant acceleration to the left over 4 secs. Gravity does not apply.

Control: Automatic

Initial State: NA

Input:
$\mathbf{p_i}(\mathbf{t_0}) = (950, 50)$ this is the (x,y) coordinate position

$\mathbf{v_i}(\mathbf{t_0}) = 0$
$\mathbf{F}$ = (-100, 0)
$\mathbf{m} = 10$

Output:
$\mathbf{p_i}(\mathbf{t_1})$ = (947.16 50.00); $\mathbf{v_i}(\mathbf{t_1})$ = (-10.50, 0.0)
$\mathbf{p_i}(\mathbf{t_2})$ = (938.98, 50.0); $\mathbf{v_i}(\mathbf{t_2})$ = (-20.83, 0.0)
$\mathbf{p_i}(\mathbf{t_3})$ = (925.46, 50.00); $\mathbf{v_i}(\mathbf{t_3})$ = (-31.16, 0.0)
$\mathbf{p_i}(\mathbf{t_4})$ = (909.66, 50.00); $\mathbf{v_i}(\mathbf{t_4})$ = (-40.16, 0.0)

How test will be performed: Unit testing with pytest

4. TC4: test_system_free_falling

Description: This test computes the position and velocity of bodies falling from height over time. Force of gravity is in effect. The position and velocity is calculated over time.

Control: Automatic

Initial State: NA

Input:
Body1: $\mathbf{p_i}(\mathbf{t_0})$ = (100.0, 50.0); $\mathbf{v_i}(\mathbf{t_0}) = 0$
Body2: $\mathbf{p_i}(\mathbf{t_0})$ = (300.0, 50.0); $\mathbf{v_i}(\mathbf{t_0}) = 0$
Body3: $\mathbf{p_i}(\mathbf{t_0})$ = (600.0, 50.0); $\mathbf{v_i}(\mathbf{t_0}) = 0$
$\mathbf{m}$ = 100.0; $\mathbf{g}$ = 9.8 (acceleration due to gravity in space)
Mass(m) for each body is 100

Output:
Body1:
$\mathbf{p_i}(\mathbf{t_1})$ = (100.0, 52.78); $\mathbf{v_i}(\mathbf{t_1})$ = (0.0, 10.289)
$\mathbf{p_i}(\mathbf{t_2})$ = (100.0, 60.80); $\mathbf{v_i}(\mathbf{t_2})$ = (0.0, 20.41)
$\mathbf{p_i}(\mathbf{t_3})$ = (100.0, 74.05); $\mathbf{v_i}(\mathbf{t_3})$ = (0.0, 30.54)
$\mathbf{p_i}(\mathbf{t_4})$ = (100.0, 74.05); $\mathbf{v_i}(\mathbf{t_4})$ = (0.0, 39.54)

Body2:
$\mathbf{p_i}(\mathbf{t_1})$ = (300.0, 52.78); $\mathbf{v_i}(\mathbf{t_1})$ = (0.0, 10.289)

$\mathbf{p_i}(\mathbf{t_2})$ = (300.0, 60.80); $\mathbf{v_i}(\mathbf{t_2})$ = (0.0, 20.41)
$\mathbf{p_i}(\mathbf{t_3})$ = (300.0, 74.05); $\mathbf{v_i}(\mathbf{t_3})$ = (0.0, 30.54)
$\mathbf{p_i}(\mathbf{t_4})$ = (300.0, 89.85); $\mathbf{v_i}(\mathbf{t_4})$ = (0.0, 39.54)

Body3:
$\mathbf{p_i}(\mathbf{t_1})$ = (600.0, 52.78); $\mathbf{v_i}(\mathbf{t_1})$ = (0.0, 10.289)
$\mathbf{p_i}(\mathbf{t_2})$ = (600.0, 60.80); $\mathbf{v_i}(\mathbf{t_2})$ = (0.0, 20.41)
$\mathbf{p_i}(\mathbf{t_3})$ = (600.0, 74.05); $\mathbf{v_i}(\mathbf{t_3})$ = (0.0, 30.54)
$\mathbf{p_i}(\mathbf{t_4})$ = (600.0, 89.85); $\mathbf{v_i}(\mathbf{t_4})$ = (0.0, 39.54)

How test will be performed: Unit testing with pytest

5. TC5: test_system_free_bodies_forces

Description: This test computes the position and velocity of floating bodies over time by applying force (F) on each body. We assume that the force of gravity is not in effect in this case. The position and velocity is calculated over time.

Control: Automatic

Initial State: NA

Input:
Body1: $\mathbf{p_i}(\mathbf{t_0})$ = (0.0, 50.0); $\mathbf{v_i}(\mathbf{t_0})$ = 0; $(\mathbf{m_1})$ = 1; $\mathbf{F}$ =(-300, 0)
Body2: $\mathbf{p_i}(\mathbf{t_0})$ = (300.0, 50.0); $\mathbf{v_i}(\mathbf{t_0})$ = 0; $(\mathbf{m_2})$ = 4000; $\mathbf{F}$ = (0, 80000)
Body3: $\mathbf{p_i}(\mathbf{t_0})$ = (600.0, 50.0); $\mathbf{v_i}(\mathbf{t_0})$ = 0; $\mathbf{m_3}$ = 10.0, $\mathbf{F}$ = (1000, -2000)

Output:
Body1:
$\mathbf{p_i}(\mathbf{t_1})$ = (-85.233, 50.0); $\mathbf{v_i}(\mathbf{t_1})$ = (-314.97, 0)
$\mathbf{p_i}(\mathbf{t_2})$ = (-330.60, 50.0); $\mathbf{v_i}(\mathbf{t_2})$ = (-624.94, 0.0)
$\mathbf{p_i}(\mathbf{t_3})$ = (-736.10, 50.0); $\mathbf{v_i}(\mathbf{t_3})$ = (-934.91, 0.0)
$\mathbf{p_i}(\mathbf{t_4})$ = (-1219.76, 50.0); $\mathbf{v_i}(\mathbf{t_4})$ = (-1204.88, 0.0)

Body2:
$\mathbf{p_i}(\mathbf{t_1})$ = (300.0, 55.68); $\mathbf{v_i}(\mathbf{t_1})$ = (0.0, 21.00)
$\mathbf{p_i}(\mathbf{t_2})$ = (300.0, 72.04); $\mathbf{v_i}(\mathbf{t_2})$ = (0.0, 41.66)

$\mathbf{p_i}(\mathbf{t_3})$ = (300.0, 99.07); $\mathbf{v_i}(\mathbf{t_3})$ = (0.0, 62.33)
$\mathbf{p_i}(\mathbf{t_4})$ = (300.0, 131.32); $\mathbf{v_i}(\mathbf{t_4})$ = (0.0, 80.33)

Body3:
$\mathbf{p_i}(\mathbf{t_1})$ = (628.0, -6.82); $\mathbf{v_i}(\mathbf{t_1})$ = (104.99, -209.98)
$\mathbf{p_i}(\mathbf{t_2})$ = (710.20.0, -170.40); $\mathbf{v_i}(\mathbf{t_2})$ = (208.31, -416.63)
$\mathbf{p_i}(\mathbf{t_3})$ = (845.37, -440.74); $\mathbf{v_i}(\mathbf{t_3})$ = (311.64, -623.27)
$\mathbf{p_i}(\mathbf{t_4})$ = (1006.58, -763.17); $\mathbf{v_i}(\mathbf{t_4})$ = (401.63, -803.25)

How test will be performed: Unit testing with pytest

6. TC6: test_system_projectile_45deg

Description: "Projectile motion is a form of motion experienced by an object or particle (a projectile) that is thrown near the Earth's surface and moves along a curved path under the action of gravity only". The rigid object is falling from a height specified in input. In the horizontal direction, velocity is constant.

Control: Automatic

Initial State: NA

Input: $\mathbf{p_i}(\mathbf{t_0})$ = (0, 125) this is the (x,y) coordinate position
$\mathbf{v_i}(\mathbf{t_0})$ = (10,0)
$\mathbf{F_i}(\mathbf{t_0})$ = 10
$\mathbf{m_i}$ = 10
$\mathbf{g}$ = 9.8 (acceleration due to gravity)
Output: $\mathbf{p_i}(\mathbf{t})$ = (50, 0); $\mathbf{v_i}(\mathbf{t})$ = (10, 50); after t secs, where t= 5

How test will be performed: Unit testing with pytest

[Same comment as for previous test. I believe all of the equations you will need for your closed form solutions are at: `https://en.wikipedia.org/wiki/Projectile_motion` —SS]

[You are using x and y in the conventional orientation, but you should also have tests where your projectile is thrown in the opposite direction and when your projectile has a non-zero velocity in the y direction. You want to make sure that there isn't an error in any of the coordinate directions. If

everything is down and to the left you won't notice errors with motion up or down. —SS]

5.1.2 Rotation of 2D Rigid Body Simulation

This test is to simulate the rotation of a 2D rigid body about its axis.

1. TC4: 2D Rigid body rotation about its axis

   Description: In rotational motion of 2D rigid bodies, Torque τ is the force which produces rotation. It has magnitude and direction.This test can also be used for multiple set of rigid bodies.(IM2)

   Control: Automatic

   Initial State: NA

   Input: $\mathbf{m_i} = 100$
   $\mathbf{g} = 0$
   $\phi_i(\mathbf{t_0}) = 50$
   $\omega_i(\mathbf{t_0}) = 0.3$
   $\tau = 1000$
   $\mathbf{I(i)} = 10000$

   Output: $\phi(\mathbf{t}) = 50.9$; $\omega(\mathbf{t}) = 0.3$; after t secs where t= 3

   How test will be performed: Unit testing with PyUnit

[Nice to see rotation tests not being forgotten. As before though, I would like to know how you come up with your output answers. —SS]

5.1.3 Rotation of 2D Rigid Body Simulation

This test is to simulate the collision of 2D rigid bodies.

1. TC5: Dynamic rigid body collision with static body test

Description: This is to test a set of rigid bodies that collide. This test case will test for collision of a dynamic object falling from a height with a static object. At collision the static object does not move, the dynamic object's velocity, position, angular velocity, orientation is calculated.Momentum is conserved.

Control: Automatic

Initial State: NA

Input: $\mathbf{m_k}$ = TBD (to be determined)
$\mathbf{p_k}(\mathbf{t_0})$ = TBD
$\mathbf{v_k}(\mathbf{t_0})$ = TBD
$\phi_k(\mathbf{t_0})$ = TBD
$\omega_k(\mathbf{t_0})$ = TBD
$\mathbf{C_R}$ = TBD

Output: $\mathbf{v_k}(\mathbf{t})$ = TBD
$\mathbf{p_k}(\mathbf{t})$ = TBD
$\phi_k(\mathbf{t})$ = TBD
$\omega_k(\mathbf{t})$ = TBD
after t secs(t, TBD)

How test will be performed: Unit testing with PyUnit

2. TC6: Dynamic rigid Body collision test(set of bodies)

Description: This is to test a set of rigid bodies that collide. This test case will test for collision of a set of dynamic object falling from a height with a static object. At collision bodies' velocity, position, angular velocity, orientation is calculated. Momentum is conserved. The input and output set generated is determined by the number of bodies added in space.Multiple objects will be simulated to fall at different times fro a height so we can simulate collision.

Control: Automatic

Initial State: NA

Input: $\mathbf{m_k}$ = TBD (to be determined)
$\mathbf{p_k}(\mathbf{t_0})$ = TBD
$\mathbf{v_k}(\mathbf{t_0})$ = TBD

$\phi_k(\mathbf{t_0})$ = TBD
$\omega_k(\mathbf{t_0})$ = TBD
$\mathbf{C_R}$ = TBD

Output: $\mathbf{v_k}(\mathbf{t})$ = TBD
$\mathbf{p_k}(\mathbf{t})$ = TBD
$\phi_k(\mathbf{t})$ = TBD
$\omega_k(\mathbf{t})$ = TBD
after t secs(t, TBD)

How test will be performed: Unit testing with PyUnit

[The TBDs should be filled in. I get the impression that some of the physics is giving you trouble. The following resource looks pretty good `https://www.myphysicslab.com/engine2D/collision-en.html`. You could use this calculator `https://www.omnicalculator.com/physics/conservation-of-momentum` with your coefficient or restitution set to 1.0. You just need the objects to not rotate after their collision. Two spheres colliding should be fine. You can also play around with problems where one mass is so large that that object will essentially be stationary. I can also lend you a physics textbook, if that would be helpful. —SS]

5.2 Tests for Nonfunctional Requirements

5.2.1 Usability Test

Usability test

1. TC7

Type: Usability test

Initial State:

Input/Condition:

Output/Result:

How test will be performed: Users/reviewers of TamiasMini2D will be asked to install the library and use it. They will be asked to complete the survey in the Appendix section for Usability.

[Nice to see a usability test. It is a bit simplistic, but that is fine for right now. —SS]

5.2.2 Correctness/Performance

Correctness/Performance

1. TC8

   Type: Dynamic

   Initial State:

   Input:

   Output:

   How test will be performed: Correctness/performance will be measured by comparing output results with ODEs related to each requirement/function.

[This isn't really complete. For correctness, you can probably refer to your functional tests from the previous section. —SS]

5.2.3 Reusability

Reusability test

1. TC9

   Type: Dynamic

   Initial State:

   Input:

   Output:

   How test will be performed: Users/reviewers will be asked to see if they can extend the library and use for other purposes. ...

[A survey is an interesting way to measure this. —SS]

Table 1: Requirements Traceability Matrix

Testcase Number	Instance Models	CA Requirements
TC1	IM1	R1, R2, R4, R5
TC2	IM1	R1, R2, R4, R5
TC3	IM1	R1, R2, R4, R5
TC4	IM2	R1, R2, R4, R6
TC5	IM3	R1, R3, R4, R7, R8
TC6	IM3	R1, R3, R4, R7, R8
TC7		NFR3
TC8		NFR1, NFR2
TC9		NFR5
TC10		NFR4, NFR6

5.2.4 Understandability/Maintainability

Understandability/Maintainability test

1. TC9

   Type: Dynamic

   Initial State:

   Input:

   Output:

   How test will be performed: Users/reviewers will be asked to see check some see if they are able to find the space module and update the body parameters are desired. Users will be asked on the scale of 1 to 5 how easy it was to find the code, understand it and make changes. ...

5.3 Traceability Between Test Cases and Requirements

The purpose of the information in Table 1 below is to provide a mapping between the test cases and the requirements in the SRS for easy reference and verification.

6 Static Verification Techniques

Code review and inspection will be used as the method for implementation verification. [You can remove this section, since the details can be covered in Section 4.4. I've realized that I should remove this section from the template. —SS]

[You should also think about parallel testing. Calculate the answers for some more complex simulations with existing software and compare these results to your results. —SS]

References

- http://www.physicstutorials.org/home/mechanics/1d-kinematics/projectile-motion?start=1
- https://en.wikipedia.org/wiki/Projectile-motion
- R. A. BROUCKE. "Equations of motion of a rotating rigid body", Journal of Guidance, Control, and Dynamics, Vol. 13, No. 6 (1990), pp. 1150-1152.
- https://github.com/smiths/caseStudies/blob/master/CaseStudies/gamephys/docs/SRS/Ga

7 Appendix

This is where you can place additional information.

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.

7.2 Usability Survey Questions?

1. On the scale of 1 - 5, 1 being very difficult and 5 being very easy, How easy was it to install the program using the installation guide?

   Comment on what can be improved:

2. On a scale of 1 - 5, how easy were you able to update the parameters in a space? e.g change the velocity of a body

3. Did the program return the expected output based on the testcase and input values?

   If no, please add comments explaining issues encountered: