Platform Perils

Test Report

Steven Palmer
⟨palmes4⟩
Chao Ye
⟨yec6⟩

April 26, 2016

Contents

List of Tables

List of Figures

Revision History

Date	**Version**	**Notes**
March 24, 2016	1.0	Created document skeleton
March 28, 2016	1.1	Final version for rev 0

1 Introduction

This document provides a report of the results of the testing performed on the Platform Perils application. Both system testing and quality testing are covered. Traceability between testing and both requirements and modules is given in the final section.

Note that the tests that were carried out in this report are all described in the Test Plan document. Please refer to this document for further information.

2 Proof of Concept Test

A proof of concept test was carried out prior to the development of the game to demonstrate that risks associated with the project could be overcome.

Test #1:	**Proof of Concept**
Description:	Tests whether significant risks to the completion of the project can be overcome
Type:	Proof of Concept (manual)
Tester(s):	Game developers
Pass:	Successful development of a small demonstration which makes use of the Chipmunk2D physics engine and runs on Windows 7, Mac OS X, and Ubuntu
Result:	PASS

3 System Testing

3.1 Game Mechanics Testing

3.1.1 Automated Testing

A suite of automated unit tests has been created to test the game for basic functionality. The unit tests cover input response as well as collision physics.

These tests are useful to ensure that modifications to the game code do not break the fundamentals of the game. A description of the unit tests that were carried out and their results are given in the remainder of this section.

Test #2:	**Move left**
Description:	Tests if the hero moves left when the corresponding input is received when the hero is initially stationary
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with a hero object having x-velocity of zero
Input:	Keyboard function called with simulated left key down stroke
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is less than zero
Result:	PASS

Test #3:	**Move right**
Description:	Tests if the hero moves right when the corresponding input is received when the hero is initially stationary
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object having x-velocity of zero
Input:	Keyboard function called with simulated right key down stroke
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is greater than zero
Result:	PASS

Test #4:	**Stop moving left**
Description:	Tests if hero stops moving left when corresponding input is stopped
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object having x-velocity less than zero
Input:	Keyboard function called with simulated left key up stroke
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is zero
Result:	PASS

Test #5:	**Stop moving right**
Description:	Tests if hero stops moving right when corresponding input is stopped
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object having x-velocity greater than zero
Input:	Keyboard function called with simulated right key up stroke
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is zero
Result:	PASS

Test #6:	**Jump from static object**
Description:	Tests if hero jumps off a static object when corresponding input is received
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object having y-velocity of zero and a bottom edge in contact with a static object
Input:	Keyboard function called with simulated space bar key down stroke
Output:	Hero object y-velocity
Expected:	Hero object y-velocity is greater than zero
Result:	PASS

Test #7:	**Wall obstructs hero moving left**
Description:	Tests whether the hero is stopped by a wall object while moving left
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object having x-velocity less than zero situated directly to the right of a wall object
Input:	The chipmunk cpSpaceStep function is called
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is 0
Result:	PASS

Test #8:	**Wall obstructs hero moving right**
Description:	Tests whether the hero is stopped by a wall object while moving right
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object having x-velocity greater than zero situated directly to the left of a wall object
Input:	The chipmunk cpSpaceStep function is called
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is 0
Result:	PASS

Test #9:	**Floor supports stationary hero**
Description:	Tests whether the hero is supported by a floor object
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with stationary hero object situated directly on top of a floor object
Input:	The chipmunk cpSpaceStep function is called
Output:	Hero object y-velocity
Expected:	Hero object y-velocity is 0
Result:	PASS

Test #10:	**Floor stops hero in free fall**
Description:	Tests whether the hero in free fall is stopped by a floor object
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with hero object with y-velocity less than zero situated directly on top of a floor object
Input:	The chipmunk cpSpaceStep function is called
Output:	Hero object y-velocity
Expected:	Hero object y-velocity is 0
Result:	PASS

3.1.2 Manual Testing

The following manual tests were carried out for game mechanics related requirements that were not covered by unit tests.

Test #11:	**No mid-air jumps**
Description:	Tests that the hero cannot jump when not in contact with a surface
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Hero cannot jump when not in contact with a surface
Result:	PASS

Test #12:	**Zoom in test**
Description:	Tests that the stage zooming works properly (in)
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Stage view can be zoomed in
Result:	PASS

Test #13:	**Zoom out test**
Description:	Tests that the stage zooming works properly (out)
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Stage view can be zoomed out
Result:	PASS

Test #14:	**Hero death**
Description:	Tests that the hero is killed when coming into contact with a fatal hazard
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Hero is killed when contacting fatal hazards
Result:	PASS

Test #15:	**Stage win**
Description:	Tests that the stage is won when hero comes in contact with the checkered goal
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Stage is won and user is returned to the main menu
Result:	PASS

Test #16:	**General physics behaviour**
Description:	Tests that the physics behaves as expected from a qualitative perspective
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Physics appears to function correctly
Result:	PASS

Test #17:	**General collision behaviour**
Description:	Tests that collisions behave as expected from a qualitative perspective
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Collisions appear to behave correctly
Result:	PASS

3.2 Game Design Testing

3.2.1 Game World Testing

Test #18:	**All areas reachable**
Description:	Tests that all areas of the game world that are intended to be reachable by the hero are in fact reachable by the hero
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	No areas are unreachable based on a thorough playthrough testing of the game
Result:	PASS

Test #19:	**No "points of no return"**
Description:	Tests that there are no areas of the game world that will cause the hero to become stuck (e.g. inescapable pits)
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	There are no inescapable areas detected on a thorough playthrough testing of the game
Result:	PASS

3.2.2 Graphics Testing

Test #20:	**Textures**
Description:	Tests if textures are properly implemented
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	In-game textures appear correct by inspection
Result:	PASS (small texturing problem detected on arch)

Test #21:	**Lighting**
Description:	Tests if lighting effects are properly implemented
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Lighting effects appear correct by inspection
Result:	PASS

3.2.3 Audio Testing

Test #22:	**Background music**
Description:	Tests if background music is properly implemented
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Background music plays while in game
Result:	PASS

Test #23:	**Sound effects**
Description:	Tests if sound effects are properly implemented
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Appropriate sounds play when events take place (e.g. hero death, complete stage)
Result:	PASS

3.2.4 Miscellaneous Testing

The following tests will be carried out

Test #24:	**Menu system**
Description:	The menu system works as intended
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	All menu options work correctly
Result:	PASS

Test #25:	**General look and feel**
Description:	The game has the intended look and feel
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	The game is a 2.5-D platformer with an Indiana Jones adventure theme
Result:	PASS

Test #26:	**Operating system support**
Description:	The game runs on Windows 7, Mac OS X, and Ubuntu
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Game can be compiled and system tests all pass on each platform
Result:	PASS

Test #27:	**Spelling and grammar check**
Description:	The game uses proper English and is free of any spelling or grammatical errors
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	No spelling or grammatical errors are detected (or all detected errors are corrected)
Result:	PASS

4 Quality Testing

4.1 Performance Testing

Stress testing was used to evaluate performance. In these tests, different types of game objects were continuously introduced to a stage while measuring changes in framerate. A high performance and low performance system were used in these tests. The specifications of each system is given in Table 1.

The objects used by the game can be broken down into two main categories: dynamic and static. Dynamic objects are free moving and subject

Table 1: Systems used in performance testing

System	Hardware
High performance	i7 4770K @ 4.4 GHz AMD Radeon HD 7970
Low performance	i5 2430M @ 2.4 GHz nVidia GT 540M

to all physics calculations, while static objects do not move and are only involved in collision calculations. All dynamic objects used in the game are essentially the same: they consist of a loaded mesh, textures, and a single shader. Most static objects also fit this description, with one notable exception: platform objects were implemented using 3 separate shaders.

Plots showing the variation in framerate with respect to the number of objects present in a stage are given in Figure 1 (dynamic) and Figure 2 (static). The results show that a large number of objects can be incorporated into a stage before performance begins to be affected: the high performance system maintained a framerate of 60 fps until roughly 1500 dynamic objects or 3000 static objects were introduced. Even the low performance system was able to maintain a framerate above 60 until around 300 dynamics objects. This means that the game will likely be able to run at 60 fps on most systems since the number of objects in a single stage is unlikely to approach 300.

A plot showing the variation in framerate with respect to the number of platforms was also produced and is shown in Figure 3. The results of this test show a drastic drop in framerate with very few objects, and this framerate drop appears to be consistent between the high end and low end systems. This suggests that the problem is in using multiple shaders: switching shaders is a resource intensive procedure and the consistencies in framerate between the two systems can be explained by the fact that the amount of time required for switching a shader would be approximately constant between the two systems. This problem was fixed by combining the three platform shaders into one shader. This change required only minor minor changes to the rendering code.

Additional testing was performed to assess whether the framerate was predominantly affected by procedures involving physics calculations or rendering. The results of these tests are given in Figure 4 and Figure 5 for the

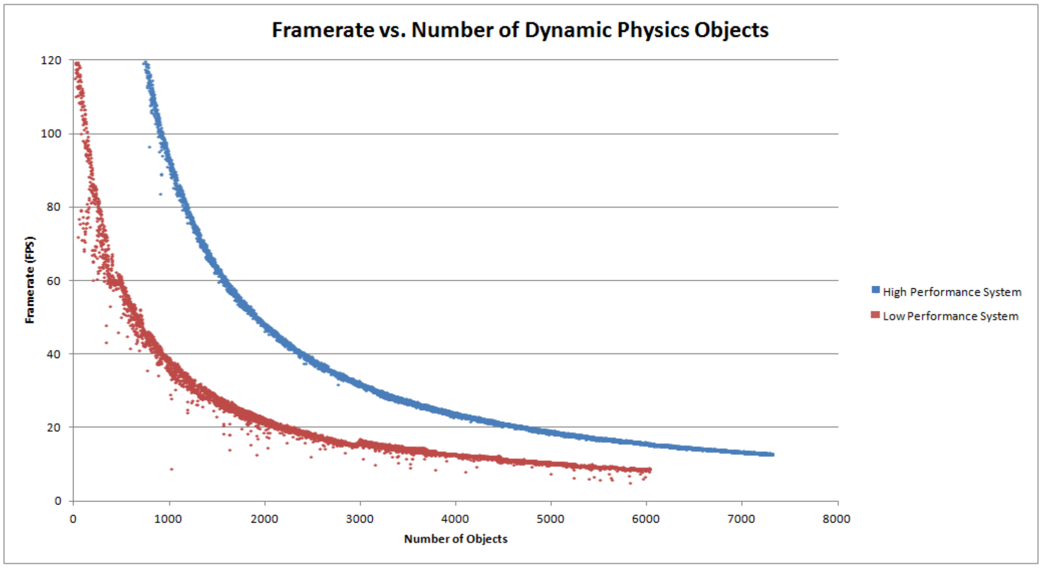

Figure 1: Framerate vs. number of dynamic objects

high and low performance systems, respectively. Interestingly, from the plot of the high performance system it appears that neither is a dominant factor: both make a roughly equal contribution to the overall framerate. This does not appear to be the case in the low performance system, where it is clear that the rendering step limits the framerate.

Test #28:	**Hardware requirements**
Description:	Tests for the minimum hardware requirements required to maintain an average frame rate of at least σ frames per second
Type:	Functional (dynamic, manual)
Tester(s):	Development team
Pass:	Game maintains frame rate of at least σ frames per second in a stage that contains Φ dynamic objects when tested on a low-performance system
Result:	Borderline PASS due to platform shader performance

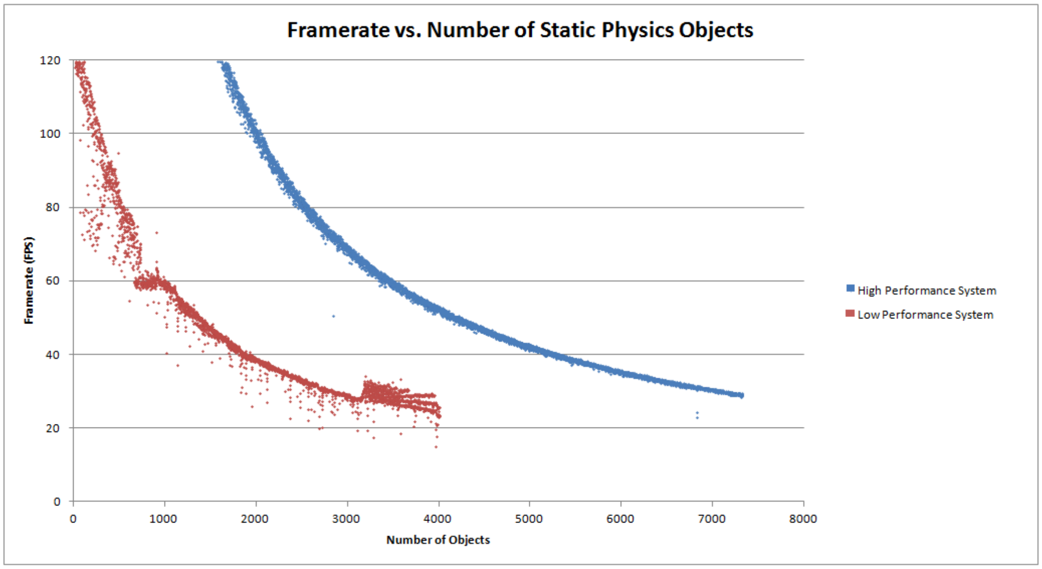

Figure 2: Framerate vs. number of static objects

4.2 Usability Testing

A user experience survey will be used to assess useability. This survey will be administered in the future and will not .

Test #29:	**Entertainment**
Description:	Tests that the game is entertaining
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	Average survey score of at least Θ
Result:	Planned for future (TBD)

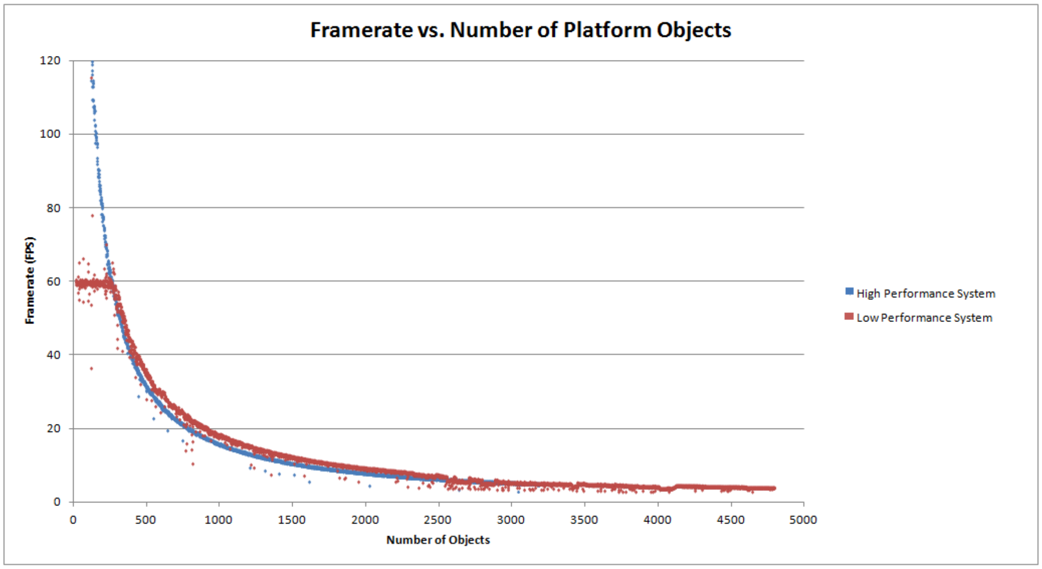

Figure 3: Framerate vs. number of platform objects

Test #30:	**Challenge**
Description:	Tests that the game is adequately challenging (not too easy or too hard)
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	Average survey score within Ψ of 5 result
Result:	Planned for future (TBD)

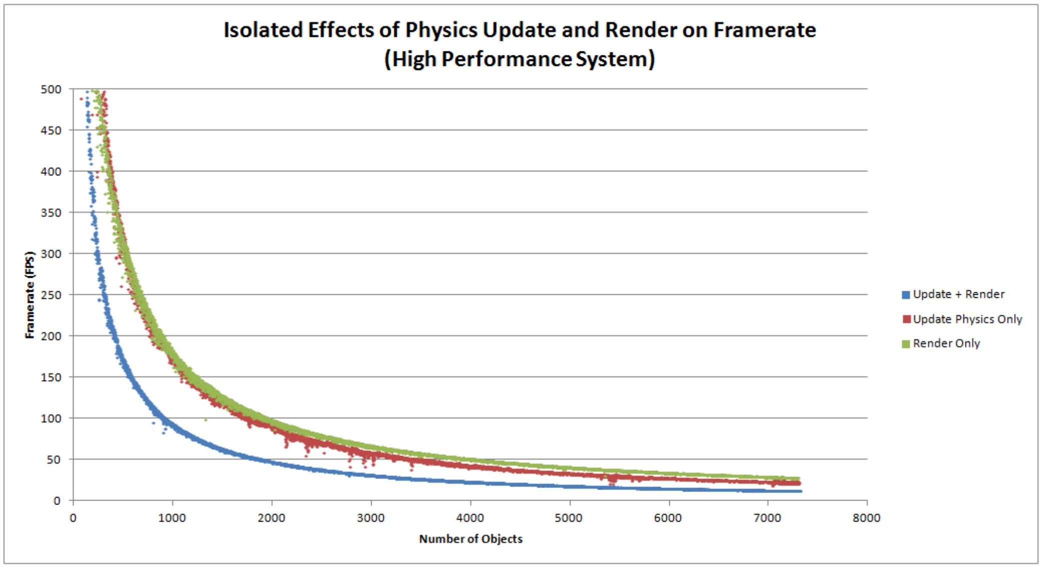

Figure 4: Effects of physics update and render on framerate on high performance system

Test #31:	**Controls**
Description:	Tests that the game controls are intuitive
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	Average survey score of at least Ω result
Result:	Planned for future (TBD)

4.3 Robustness Testing

Robustness of the application with regard to erroneous input was not formally tested. User inputs are only accepted as defined keystrokes and mouse clicks. The way these inputs map to setting/modifying variables is entirely controlled by the game code and thus the user is never able to directly change any variables via inputs. This means that all variables are maintained within their expected ranges and no explicit testing is required.

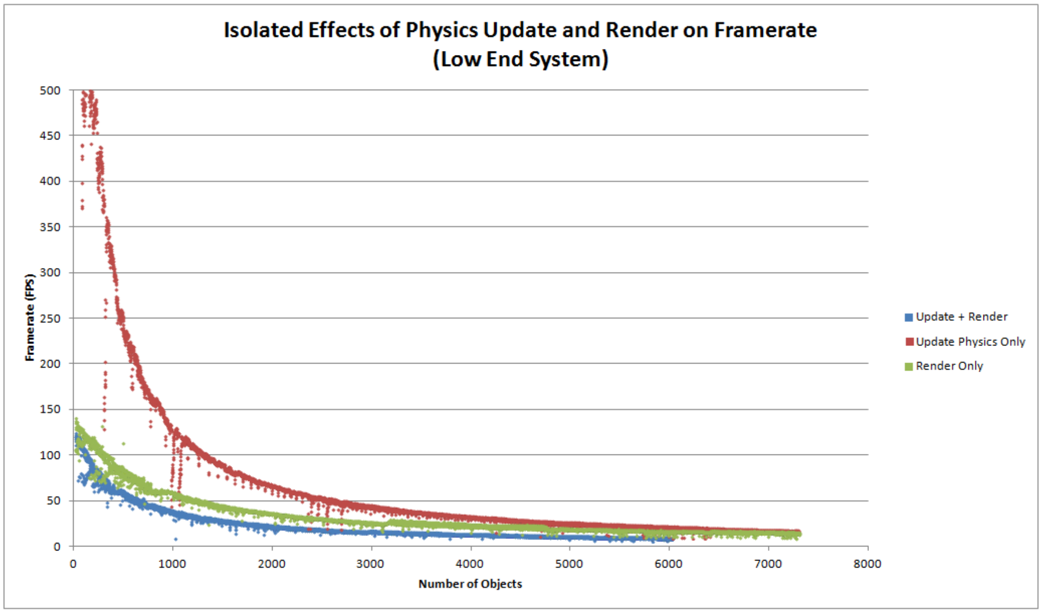

Figure 5: Effects of physics update and render on framerate on low performance system

Robustness tests in the form of stress testing were covered under performance testing.

5 Summary of Changes

Few changes were made to the game as a result of the tests described in this document. This is due to the fact that informal testing was performed constantly as the game was being developed. Every time changes were made to the game code or new features were added, the game was compiled and played by the development team to ensure that everything was working as intended.

A summary of the changes made in response to testing are given in Table 2.

Table 2: Changes made in response to testing

Test	Changes Made

6 Traceability

6.1 Trace of Testing to Requirements

A trace between requirements and testing is given in Table 3.

6.2 Trace of Testing to Modules

A trace between modules and testing is given in Table 4.

Table 3: Requirements Traceability

Requirement	Test(s)
1	24
2	24
3	24
4	24
5	2, 4
6	3, 5
7	6
8	11
9	16
10	19
11	18
12	12
13	13
14	9, 10
15	7, 8
16	14, 17
17	14
18	15
19	24
20	25
21	25
22	20, 21, 25
23	22, 23
24	29
25	31
26	30
27	28
28	26
29	27

Table 4: Trace between modules and tests

Module	**Test(s)**
1	28
2	20, 21
3	20, 21
4	20, 21
5	16, 17
6	17
7	17
8	9, 10, 17
9	7, 8, 17
10	17
11	15
12	16, 17
13	14, 16, 17
14	14, 16, 17
15	16, 17
16	16, 17
17	2-17
18	16, 17
19	20, 21
20	20, 21
21	20, 21
22	2-15, 24
23	24
24	2-15
25	24
26	12, 13
27	22, 23
28	22, 23
29	20, 21
30	20, 21