2012

Space Shooter

Test Results

COP 4331: Processes for Object Oriented Software Development
Fall 2012

Team 2
11/30/2012

Table of Contents

Prefatory Information

Modification history:

Version	Date	Who	Comment
v0.1	11/29/2012	Joshua Thames	Template
v1.0	11/29/2012	Andre Meireles	Final

Team Name:

Team 2

Team Members:

Name	Email address
Andre Meireles	andre.meireles@knights.ucf.edu
Alex Banke	banke@knights.ucf.edu
Christopher Margol	margol_chris@knights.ucf.edu
Chris Lin	christophercklin@gmail.com
Josh Thames	jthames88@knights.ucf.edu
Thaddeus Latsa	tlatsa@knights.ucf.edu

Definitions

CTD = “Crash to desktop”, results when a program exits abnormally.

Introduction

It is critically important for entertainment software to be free of bugs and faults. The purpose of a video game, like Space Shooter, is to provide entertainment in a simulated environment, which would take the player out of their natural environment and place him or her into the world we, the programmers, have created on the PC. Bugs, from poorly written code, distract the user from our end purpose and may manifest themselves in a variety of ways. Some of the bugs which could arise from a video game environment might be fatal to the process (resulting in a CTD), user interface glitches, misrepresented graphics, and all round unintended gameplay distractions. All of these types of glitches remind the user that he or she is still in fact playing a simulated shooting game on a computer and on earth and the programmer's mission has failed.

To ensure the release of a stable product free from common maladies, the developers shall routinely test the code in such a way that is time-effective and provides the development team with meaningful information that will aid in the eradication of software failures.

Reference Documents:

- Concept of Operations
- Project Plan
- SRS

Description of Test Environment

Hardware: Since Space Shooter is being developed in Java, the development team will test Space Shooter in any hardware environment that supports the Java Runtime Environment (JRE). This includes, but is not limited to, name-brand desktop and laptop computers (Dell, Asus, Sony), as well as Apple desktops and laptops.

The results from the testing conclude that a 2.5 GHz+ with a nvidea GeForce 9600 graphics card is the minimum requirement to run smoothly.

Software: Space Shooter will be tested in the Windows, Macintosh, and possibly Android operating system environments. Linux support is not currently planned, so will not be tested.

Testers: The development team members will all be testing Space Shooter. Additionally, outside volunteers with different perspectives may be sought for testing purposes.

Final remarks: The testing environment in regards to both hardware and software is expected to be identical to the environment in which the software will operate after release.

Stopping Criteria

Software testing will be conducted as follows:

Method, function, or unit testing is to be conducted on each discrete component of the software as often as possible by the individual team member that is producing the component. For example, if Team Member A is tasked (by the software lead) to produce a Method B, Team Member A will be responsible for testing the method as far as the expected input and output bounds will allow. Once the component has been declared functional by the team member producing it, it will be sent to the software lead to be integrated into the final product.

Once the component has been integrated, the project itself will be tested for both the old functionality (before the new component was added) to ensure that nothing that was working before is broken, and new functionality (after new component). If testing is successful, further components (that have been tested) may be integrated into the project, and further testing may be completed.

Testing the software will be the responsibility of all group members. This may be as simple as playing the game and noting what bugs or errors are encountered. Once the software is in a playable state, every group member will have the same version of the software to take home and test prior to the next meeting OR software update.

The key idea here is to make sure that everyone is testing either their own components or the same version of the project at the same time, so that any reported bugs will be relevant to the current iteration of the project.

Once the time comes where everyone has to report what bugs they have encountered, serious issues such as CTDs take precedence over other issues, such as arithmetic errors. Fixing the known bugs shall be done in phases, with the higher precedence errors being fixed in the beginning phases, and culminating in minor bug fixes at the latter phases.

Test cases to use:

To be decided as development progresses, but generally: have broad test cases covering most common usage scenarios and specific test cases covering unusual usage scenarios. Test cases must also fall within preset bounds governing the input/output.

When is it “good enough to deliver”?:

> *“When the entire game can be played from start to finish - without encountering any game-breaking bugs.”*

Test Results

As development progressed, a decision was made to forego specific cumbersome case and unit testing in favor of more generalized integration testing. This decision was made primarily to economize development time, which was forecasted to exceed the due date. The rationale for this decision was as follows: since we are developing a video game and not a safety critical or performance critical deployment, precise functioning of sub-components was not a critical concern. Rather, what was important was that a user playing the game had a relatively smooth and, from what the user could perceive, an error-free experience.

User perception is the critical idea here. Somebody playing our game is likely not going to notice off-by-one errors, or floating-point versus integer precision, in general gameplay unless those errors cascade into an unplayable state. Therefore, testing priority was to ensure a **playable** and not necessarily a precisely engineered one.

That being said, care was taken to verify each subcomponent for desired functionality. Testing was by all group members, but not every group member necessarily tested every component. Testing requirements for individual test cases were looser than initially specified owing to the game-oriented testing philosophy. Specific testing runs are noted below for each individual release.

Testing Release 1

Test case	Who	When	Environment	Result	Comments
Ship movement	Chris Margol	10/26	MacBook Pro OSX 10.8.2 Core 2 Duo 2.8Ghz	Success	Worked but need to implement double buffering to prevent flickering
Shooting	Chris Margol	10/26	See above	Success	Ship shoots upwards
Asteroid spawning	Chris Margol	10/26	See above	Success	Asteroids appear on the screen
Java console	Chris Margol	10/26	See above	Success	Some warnings but no faults
Integration	Andre Meireles	10/28	Custom desktop, AMD Hexacore 3.4 Ghz	Success	All simple features passed

Testing Release 2

GUI framework	Andre Meireles	10/31	Custom desktop, AMD Hexacore 3.4 Ghz	Success	Simple gui functionality without backgrounds
Integrate ship graphic	Chris Lin	10/31	HP laptop Core 2 Duo 2.6Ghz	Success	No comment
Integrate graphics for upgrade	Josh Thames	10/31	ASUS laptop Intel i3 2.3 Ghz	Success	Images load successfully
Integration testing	Andre Meireles	11/1	Custom desktop, AMD Hexacore 3.4 Ghz	Success	Integration achieved

Testing Release 3

Ship shop GUI integration	Alex Banke	11/7	Dell laptop i5 3.2 Ghz	Success	GUI functional, functionality to be added
Save game feature	Chris Margol	11/7	MacBook Pro OSX 10.8.2 Core 2 Duo 2.8Ghz	Success	File IO functions correctly
Currency and score tracking	Josh Thames	11/7	ASUS laptop Intel i3 2.3 Ghz	Success	Tracker updates correctly
Integration testing	Andre Meireles	11/8	Custom desktop, AMD Hexacore 3.4 Ghz	Success	Features functional

Testing Release 4

Physics functions	Andre Meireles	11/14	Custom desktop, AMD Hexacore 3.4 Ghz	Success	Returns expected values

Level 1 script	Josh Thames	11/14	ASUS laptop Intel Core 2 Duo 1.7 Ghz	Success	Script is read correctly
Integration testing	Andre Meireles	11/14	Custom desktop, AMD Hexacore 3.4 Ghz	Success	No comment

Testing Release 5

Level 2	Alex Banke	11/21	Dell laptop i5 3.2 Ghz	Success	No comment
Level 3	Chris Margol	11/21	MacBook Pro OSX 10.8.2 Core 2 Duo 2.8Ghz	Success	No comment
Level 4	Chris Lin	11/21	Laptop, i3 2.4 Ghz	Success	No comment
Level 5	Thaddeus Latsa	11/21	HP laptop Core 2 Duo 2.2 Ghz	Success	No comment
Integration	Andre Meireles	11/22	Custom desktop, AMD Hexacore 3.4 Ghz	Success	Levels are read and displayed correctly