SE 3XA3: Test Plan
Spaceshooter Remix

Team #4, IRS Development
Ibrahim Malik maliki2
Ryan Schnekenburger schneker
Saad Khan khans126

December 4, 2018

Contents

List of Tables

List of Figures

Table 1: Revision History

Date	Version	Notes
October 3, 2018	Rev0	Authored by Ibrahim, Saad, Ryan
November 28, 2018	Rev1	Updated document and added the changes recommended by TA.

This document describes the test plan that we have developed for Spaceshooter Remix written by IRS Development.

1 General Information

1.1 Purpose

The purpose of this project is to reimplement a version of the spaceshooter game into a modern remix. This will provide a sense of nostalgia for people who enjoy casual gaming and would like to play a popular, classic game. It was created for an early 90's era handheld console which is rare to find today. This game will provide the retro feel and be executed on their computer when the program is called using Python which is installed on most people's computers. There will be several types of testing to ensure the program runs correctly according to the requirements. We will test collision detection, user inputs and also game menus. There will be ~~mutation~~ manual testing to check for optimal sensitivity for collisions between asteroids and the user's sprite and for movements to ensure all movements operate as specified.

1.2 Scope

Currently, the game will run on Python 2 on many computers if pygame is installed but it will not be very stable and crash or show several bugs. Many features are lacking and it is primitive in design. It needs to be converted into a version that will work on every computer with Python 3 and increased stability to ensure it does not shut down instantly. Most of our testing will involve using validation to ensure that we are building the correct system meeting the requirements of the original game and before the programming is finally set, a series of verification processes to ensure the system will run error free. In the future, there will be a focus on creating levels for the user where the game increases in difficulty and there will be a menu screen with instructions.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
OS	Operating System
product	The game that we are creating
program	The code that our game uses to function
ex.	example
etc.	et cetera
client	who we are creating the game for
customer	who will be consuming our game
gitlab	Gitlab repository
IDLE	The integrated development environment for python
repo	The repository that our product will be stored
sprite	The spaceship displayed on the screen representing the user's character
Git repo	Gitlab repository
player	The human playing the game or using the software
user	The human playing the game or using the software
tester	The human who is assigned the role of tester to play the game or using the software
README	Instruction document provided in the root directory

Table 3: **Table of Definitions**

Term	**Definition**
Integration Testing	Testing conducted over a period of time where new modules are incorporated into existing modules and the program is run again
Regression Testing	Testing conducted after integrating new functions or modules based on previously run tests
Manual Testing	Testing conducted by humans
Unit Testing	Each piece of code is testing individually
Automated Testing	Testing conducted by software which initializes it and runs specific tests through a series of assert statements
Static Testing	Testing that does not involve program execution
Dynamic Testing	Testing which includes having test cases run during execution

1.4 Overview of Document

This document will break down the test plan, the software description, the schedule of testing, the various types of testing tasks that need to be accomplished for functional and non-functional requirements.

2 Plan

2.1 Software Description

The software is intended to be for entertainment in the form of a game that is played on the computer. It requires a computer with a keyboard and Python 3.5 or higher installed. The game is implemented in Python and there is a user sprite that is controlled by the arrow keys on the keyboard. The objective is to avoid as many asteroids as possible or shoot down any of them for as long as possible. The game is over when the user's spacecraft runs out of health or crashes into a lot of asteroids. There are no other inputs needed from the user other than those keyboard options.

2.2 Test Team

The testing team consists of the three team members who will act as developers. These members are Ibrahim, Saad, and Ryan who will split the testing evenly to cover the different tools of testing.

2.3 Automated Testing Approach

There will be an opportunity to perform automated testing in addition to numerous code inspections and static testing methods we shall employ throughout the implementation of the project. A lot of the code has functions that create animations or menus for example, and the simplest way to test for that would rather be a manual, visual inspection of the output. That is an example of white box testing. There will be integration testing constantly to check if all the modules work in cohesion with each other. Every time a new module is created, integration testing will be used to reassure to the proper basic functionality of the game where it can begin and end. In the future, there is a possibility of adding more features to the game. In that situation, we will use regression tests to re-run some of the earlier automated tests checking if they also successfully run on the newer code.

It is a challenge to create automated testing for GUI for this pygame based python script. Most of the modules do not contain any functions with return values that can be checked with various assert statements. Instead, we need to be more creative in incorporating automated testing. As mentioned before, pytest will be set up to test functions of the code where certain outputs will be predetermined and hopefully the code reproduces the exact same output as expected. Moreover, pygame comes with a built-in unit test suite package that allows testing of certain individual parts of the code. These parts of the code will be tagged and appropriate test cases will be generated to see if the resultant output is as expected by a predetermined state or value. This is an example of black box testing.

2.4 Testing Tools

1. pytest - Ideally we will make use of the built-in pytest suite from Python to ensure the functionality of all the functions. This will help us in ensuring that every function behaves accordingly to how it should run. The pytest framework will recheck some of the outputs of these functions against a predetermined output. It will be the first line of defense against identifying functions that break and produce wrong outputs.

2. Humans - Manual testing will be conducted by people for structured walkthroughs, code inspections, syntax checking, and hand test cases.

3. IDLE - This tool on the computer will quickly identify initial syntax errors when writing the Python code. If it fails to comply due to a missing syntax symbol, typically IDLE provides a description of the error in red that can be investigated.

4. Python pygame Unit Test Suite package - This is a test suite package that is a testing framework developed by pygame that will allow us to test individual parts of pygame.

2.5 Testing Schedule

See Gantt Chart at the following URL : `https://gitlab.cas.mcmaster.ca/khans126/IRS_Space_Shooter/blob/master/ProjectSchedule/IRSGanttChart.pdf`

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Starting up The Game

Opening The Window

1. F-1

   Type: Structural, Manual, Dynamic

   Initial State: Command Line

   Input: python IRS_Space_Shooter.py

   Output: A box that is ~~640 by 520~~ HEIGHT by WIDTH will appear on the desktop.

   How test will be performed: We will be running our code from the command line and take ruler measurements of the window that appears on the screen and we will compare what is on the window to the graphic of the main menu.

3.1.2 On the Main Menu

Executing the prompts

1. F-2

   Type: Structural, Manual, Dynamic

   Initial State: Main menu

   Input: A keyboard press of the 'enter' key.

   Output: A loading screen is displayed.

   How test will be performed: When on the main menu the 'enter' key will be pressed and we will compare the screen that is displayed to our loading screen graphic file.

2. F-3

Type: Structural, Manual, Dynamic

Initial State: Main menu

Input: A keyboard press of the ‘q’ key.

Output: The window where the main menu appeared on is closed.

How test will be performed: ~~On the main menu we will take a screenshot of the desktop and then hit the ‘q’ key. We will then compare what is now on the desktop to the screenshot to see if the window is still there.~~ Tester will press ‘q’ on the main screen and observe the behavior of the entire game. The tester will see what happens to the screen and if the game exits or terminates.

3.1.3 Game Screen

Moving the Ship

1. F-4

   Type: Structural, Manual, Dynamic

   Initial State: Game screen

   Input: A keyboard press of the ’ ‘D’ key.

   Output: The ship moves towards the right boundary of the screen at the speed of 1 cm/s at least.

   How test will be performed: ~~On the game screen we will take a screenshot of the desktop and then hit the right arrow key~~ The tester will press the‘D’ key and compare the speed of movement of the player sprite. Using a ruler to measure on the screen, an approximation can be made on the speed of the movement of the sprite across the screen along with the direction.

2. F-5

   Type: Structural, Manual, Dynamic

   Initial State: Game screen

   Input: A keyboard press of the ‘A’ key.

Output: The ship moves toward the left boundary of the screen at the speed of 1 cm/s at least.

How test will be performed: ~~On the game screen we will take a screenshot of the desktop and then hit the left arrow key~~ The tester will press the 'A' key and compare the speed of movement of the player sprite. Using a ruler to measure on the screen, an approximation can be made on the speed of the movement of the sprite across the screen along with the direction.

3. F-13

   Type: Structural, Manual, Dynamic

   Initial State: Game screen

   Input: The user score hits 800 after avoiding and destroying many asteroid objects.

   Output: Increase in the number of asteroids generated by at least 2 times the previous level.

   How test will be performed: The number of asteroids appearing on the screen per minute will be counted and tallied. A stopwatch externally will take the time and a manual check of the frequency of asteroids falling or generated from the screen will be performed.

Shooting

1. F-6

   Type: Structural, Manual, Dynamic

   Initial State: Game screen

   Input: ~~A keyboard press of the 'spacebar'~~ Left-mouse click will produce bullets.

   Output: ~~A bullet is fired, originating at the ship towards the top of the game screen.~~ The bullet will appear from the y-coordinate of HEIGHT - PLAYER_HEIGHT at the x coordinate of the player sprite. It will travel in a negative direction towards the top of the screen.

How test will be performed: On the game screen we ~~will tap the spacebar~~ will tap on the left-click. Once bullets are fired we will visually check for the accuracy of the origin of these bullets. Manual testing will be used for this purpose and we will observe using humans the behavior of these objects. ~~and take a screenshot as quickly as possible. We will then wait for a short time then take another screenshot while the bullet is still on the screen. We will compare the two screenshots and observe where the bullet originated from the first screenshot and the difference of positions of the bullet.~~

2. F-12

   Type: Structural, Manual

   Initial State: Game Screen

   Input: Left-mouse click will produce bullets.

   Output: The bullet that will be fired will travel at the pygame speed value of -10 upwards to the top of the screen. This should equate to roughly 1 cm/s traveling up.

   How the Test will be performed: The tester will attempt to fire bullets using the left click of their mouse. The rate of movement of the bullet will be then observed. A ruler can be used on the screen to estimate the total height of the screen, and a stopwatch can be used by a second tester to evaluate the time taken for the bullet to cross the screen.

3. F-7

   Type: Structural, Dynamic, Manual

   Initial State: On the game screen with an asteroid in the ship's line of sight.

   Input: ~~A keyboard press of the 'spacebar'~~ Left click press on mouse

   Output: After a period of time, the asteroid is no longer on the screen.

   How test will be performed: On the game screen we will tap the left click on a mouse ~~spacebar~~ while an asteroid is coming straight towards us. ~~We will then quickly take a screenshot. We will wait for a short time and take another screenshot making sure that the bullet crossed the~~

~~path of the asteroid. We will compare the two screenshots to observe if the asteroid that was hit is still present and observe the score.~~ Using manual testing, it will be visually observed if the asteroid is removed from the game screen after the user interacts with the left mouse click.

Taking Damage

1. F-8

   Type: Structural, Dynamic, Manual

   Initial State: Game screen with an asteroid about to make contact with the ship.

   Input: ~~Right or Left arrow keys to ensure contact is made with the asteroid~~ 'A' or 'D' to have the spaceship sprite come into contact with the asteroid object in the screen.

   Output: The health bar decreases after contact is made. This decrease in health bar should not be more than 1 cm.

   How the test will be performed: ~~On the game screen we will take a screenshot of the desktop. We will guide the ship into an asteroid using the right and left arrow keys. We will take a screenshot of the resulting game screen and compare the health bar of the two screenshots.~~We will observe using human testers to see if the health bar decreases at all after a collision between the user sprite and the asteroid object after guiding the spacecraft into the path of oncoming asteroids.

Dying

1. F-9

   Type: Structural, Dynamic, Manual

   Initial State: Game screen with a low health bar.

   Input: ~~Right or Left arrow keys~~ Use 'A' and 'D' for movement to ensure contact is made with an asteroid.

   Output: After contact is made with the asteroid the ship is no longer visible on the game screen.

How the Test will be performed: ~~On the game screen we will take a screenshot of the desktop. We will guide the ship into an asteroid using the right and left arrow keys. We will take a screenshot of the resulting game screen and compare the two screenshots to observe the ship and the number of lives of the two screenshots.~~ Using a human tester, we will drive the health bar to as low as possible by colliding with several asteroids. Ideally, the health bar will not be more than 0.5-1 cm in length as verified by the human eye. In this state of low health, the tester will purposely collide with one more asteroid to witness the interaction. Several iterations of this interaction will occur as this test will be repeated for many different sizes of asteroids to ensure consistency amongst behaviors.

2. F-10

   Type: Structural, Dynamic, Manual

   Initial State: Game screen with a low health bar and one life remaining

   Input: ~~Right or Left arrow keys to ensure contact is made with an asteroid~~ Use ‘A’ and ‘D’ for movement to ensure contact is made with an asteroid.

   Output: After contact is made with the asteroid, the game over menu is displayed.

   How the Test will be performed: ~~On the game screen, we will guide the ship into an asteroid using the right and left arrow keys. We will compare what is on the screen to the graphic file of the main menu.~~ The tester guides the sprite towards any of the oncoming asteroids. Then they observe what happens to the gameplay and if the game is still continuing in this mode after the collision.

Restarting

1. F-11

   Type: Structural, Dynamic, Manual

   Initial State: ~~Game Screen~~ Game over screen

   Input: A keyboard press of ‘R’

Output: The game returns to the main menu.

How the Test will be performed: ~~On the Game Screen we will take a screenshot of the desktop. Next we will hit the 'r' key. We will take a screenshot of the result. We will compare the scores of the two screenshots.~~ The tester will lose the game on purpose running out of lives. Once the game over screen is displayed, they will press the 'R' key on their keyboard and observe what happens to the screen of the game.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability

Testing of installation and maintainability

1. NF-1

   Type: Structural, Static, Manual

   Initial State: Program is downloaded into the computer from GitLab repository and installed on the computer.

   Input/Condition: Program is launched via command line using the command 'python IRS_Space_Shooter' on operating systems Windows 10, Mac OS 11 and Linux.

   Output/Result: The program should instantly execute resulting in a small window with the game running.

   How test will be performed: Each team member has one distinct operating system and will clone the repository. The program will be run on each of the computers via command line and executed to ensure that it is running correctly on those particular systems.

2. NF-2

   Type: Structural, Static, Manual

   Initial State: The program will be downloaded but nothing will be installed on the computer.

   Input/Condition: The users will be asked to install the software and launch the game without assistance and reading the README file if necessary.

   Output: The majority of the users should be able to install the program in 5 minutes or less.

   How test will be performed: Beta-testers will be selected from a group of people who are outside the class or engineering stream with basic knowledge of computers. They will be asked to download the game on their computers. Once they are ready, a stopwatch will begin to record the time it takes for them to launch the programs on their computers (includes installation time).

Satisfaction Requirements (Look and feel)

1. NF-3

   Type: Structural, Static, Manual

   Initial State: The program is launched in the user's computer and the user is facing the opening screen of the program.

   Input/Condition: The users will be asked to play the game itself and rate the program based on the criteria of ease of installation, gameplay, graphics and overall satisfaction on a scale from 1 to 10 with 1 being lowest, and 10 being highest score.

   Output: The results of the survey with 4 categories should be an average score of 7 with no category scoring between 1-3.

   How the test will be performed: Beta-testers from the previous test will also be provided with a small page in the form of an anonymous questionnaire that will explain all the categories and how to grade them. The grades from 1-3 represent 'Poor', 4 represents 'Below average', 5-6 represent 'Adequate', 7 represents 'Slightly better than average', 8-9 represent 'Above expectations' while 10 should be 'Excellent' The average is calculated from the 4 categories.

3.2.2 Performance

Responsiveness

1. NF-4

   Type: Structural, Dynamic, Manual

   Initial State: The program is launched and the game begins with all the default settings.

   Input/Condition: The game is started and played using the keyboard

   Output/Result: Responses of the sprite are instant and under the EXEC-TIME.

   How the test will be performed: Beta-testers are asked to play the game on a computer with a keyboard. The reaction time of the game is noted as they use the controls of the game. It will also be measured by a stopwatch and by eye to ensure that the response of the interactions is instant to the human eye.

2. NF-5

   Type: Structural, Dynamic, Manual

   Initial State: The program is launched and the game begins with all the default settings.

   Input/Condition: The game is started and played using the keyboard but several keys on the keyboard are pressed at the same time.

   Output/Result: Responses of the sprite are instant and under the EXEC-TIME but only one interaction occurs per keyboard hit and this is non-deterministically chosen by the computer.

   How the test will be performed: Beta-testers are asked to play the game on a computer with a keyboard. The game will be sort of stress-tested with several inputs being simultaneously being thrown at it via the selection of multiple keys being hit by the user.

3.3 Traceability Between Test Cases and Requirements

Test Case #	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
F-1	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-
F-2	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-
F-3	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-
F-4	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-
F-5	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-
F-6	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
F-7	-	-	-	x	x	-	-	-	-	-	-	-	-	-	-	-
F-8	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-
F-9	-	-	-	-	-	-	-	-	-	-	-	x	-	x	-	-
F-10	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-
F-11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x
F-12	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
F-13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-

Test Case #	NF1	NF2	NF3	NF4	NF5	NF6	NF7	NF8	NF9	NF10	NF11	NF12	NF13	NF14	NF15	NF16	NF17	NF18	NF19	NF20	NF21
NF-1	-	-	-	-	-	-	-	x	-	-	x	-	x	x	x	-	-	-	-	-	-
NF-2	-	-	-	-	-	-	-	x	-	-	x	-	x	x	x	-	-	-	-	-	-
NF-3	x	x	x	-	x	x	-	-	-	x	-	x	-	-	-	-	x	x	-	-	x
NF-4	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NF-5	-	-	-	-	x	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-

4 Tests for Proof of Concept

Testing for proof of concept will be focused on ensuring that all animations behave as expected, as this also means all the relevant values are being updated correctly.

4.1 Testing for Animations

1. FC-1

   Type: Functional, Manual

   Initial State: Ship is in centre of the screen

   Input: Left or right directional button on the keyboard

   Output: The spaceship moves in the desired direction when the arrow key is pressed, and immediately stops when the key is released

   How test will be performed: The left and right arrow keys will be pressed once the game starts, and the movement of the ship will be observed.

2. FC-2

   Type: Functional, Manual

   Initial State: Spaceship is not shooting

   Input: ~~The spacebar is pressed~~ The left mouse click is pressed

   Output: The spaceship shoots a projectile that will collide with an asteroid in its path and disappear.

   How the test will be performed: The left mouse click is pressed once the game starts, and the projectile will be observed leaving the ship and colliding with any asteroids in its path.

3. FC-3

   Type: Functional, Manual

   Initial State: Asteroids explode upon hitting the ship

   Input: The spaceship remains stationary and waits to collide with an asteroid

Output: When an asteroid hits the spaceship it is destroyed in an explosion.

How test will be performed: Once the game starts, the spaceship will be kept stationary until an asteroid hits

4. FC-4

   Type: Functional, Manual

   Initial State: The health bar decreases in size when the ship is hit by an asteroid

   Input: The spaceship remains stationary and waits to collide with an asteroid

   Output: When an asteroid hits the spaceship the health bar will decrease in size

   How test will be performed: Once the game starts, the spaceship will be kept stationary until hit by an asteroid.

5. FC-5

   Type: Functional, Manual

   Initial State: The ship explodes upon taking fatal damage from an asteroid

   Input: The spaceship remains stationary and waits to collide with an asteroid, multiple times until one hit is fatal

   Output: When an asteroid hits the spaceship and its health is lower than the damage the asteroid inflicts, the spaceship will explode.

   How test will be performed: Once the game starts, the spaceship will be kept stationary until hitting enough asteroids to cause fatal damage to it.

6. FC-6

   Type: Functional, Manual

   Initial State: The ship starts with three lives

   Input: The spaceship remains stationary and waits to collide with an asteroid, multiple times until one hit is fatal. This is repeated 3 times.

Output: When the ship takes fatal damage, it loses a life and the lives icon in the top right corner disappears. If this happens 3 times, the game finishes and returns you to the main menu.

How test will be performed: Once the game starts, the spaceship will be kept stationary until hit by enough asteroids to cause fatal damage to it, 3 times in one game.

5 Comparison to Existing Implementation

All of the tests for proof of concept will be used to compare our implementation with the existing implementation. This ensures that our project has all the same functionalities as the original implementation. If all the animations pass the test cases on both implementations we will know that our implementation is correct.

6 Unit Testing Plan

The unit testing of this project will be conducted using ~~screenshots~~ pytest on certain functions if appropriate. As it stands right now, there is not any plan for such testing on any functions. Most testing will be conducted as black box, manual testing instead.

6.1 Unit testing of internal functions

The unit tests for this project will be conducted using pytest. We will create unit tests that the different update functions that occur after the methods in each of the classes. We will input different keyboard inputs using pytest to test the results of each of the methods. We do not need to import anything as the modules would already have what is needed to them. The project will need to import pytest in order to test. We will be using code coverage metrics to see the code coverage. We will aim for 90% code coverage.

6.2 Unit testing of output files

~~To test the output on the window that the game creates we will use screenshots to compare different functions. We will screenshot a before and after of each~~

~~of the methods to see the changes that each of the methods caused. In order for a test case to be successful it will have to have significant changes that coincide to what each of the keyboard inputs do.~~ Our game does not produce any output files. Hence this section is irrelevant to the project.

7 Appendix

This is a sample usability survey quiz that can be given to users to score the application.

Rate the game based on your experience playing it, from a scale of 1 to 10, with 1 being lowest, and 10 being the highest score. Compare this game experience to other low-end or 2D games that you have played with, or the original spaceshooter game (if you have previously played that).

1. Ease of installation.
   Score : 1 2 3 4 5 6 7 8 9 10

2. Gameplay
   Score : 1 2 3 4 5 6 7 8 9 10

3. Graphics
   Score : 1 2 3 4 5 6 7 8 9 10

4. Overall Satisfaction
   Score : 1 2 3 4 5 6 7 8 9 10

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.

Table 4: **Table of symbolic constants**

Abbreviation	**Definition**
HEIGHT	600 pixel screen size for height
WIDTH	800 pixel screen size for height
PLAYER_HEIGHT	The y-coordinates at which the sprite of the player is located in the game screen
EXEC-TIME	0.3 seconds

7.2 Usability Survey Questions

Here is a list of questions to potential users of the program to better assess the usability of the game.

1. What are some aspects of the game that you like?
2. What are some changes you would like to see that will improve the game?
3. How likely are you to play this game again?
4. Were the instructions to install and execute the game difficult or did you need help?
5. What is your immediate experience after playing the game compared to other games you played in the past?

References