SE 3XA3: Test Plan
Staroids

Team 20, Staroids
Eoin Lynagh, lynaghe
Jason Nagy, nagyj2
Moziah San Vicente, sanvicem

November 28, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 23	0.1	Added personal info to document
Nov 23	0.15	Code coverage, Automated testing, Changes due to testing
Nov 23	0.2	Added usability requirement results
Nov 25	0.25	Comparison to existing Implementation
Nov 28	0.3	Finished functional requirements
Nov 23	0.35	

This document is meant to show

1 Functional Requirements Evaluation

1.1 Loading into pre-game

To test this functional requirement, we must verify that the game always starts in pre-game state. This can e verified by manual code inspection, as upon startup, the code has no option but to begin in the pre-game phase. To see this, look at the "start" function in Staroids/src/gamestate.js. It sets the state to "pregame" on start, which cannot be avoided.

1.2 Starting game

To test this functional requirement, we must verify that for all Staroids game in the "pregame" state, they will be moved from the pregame state to the playing state within one second after pressing the spacebar. This can be verifed by code inspection as well as manual testing. The game is monitoring for the space input 30 times per second, so it cannot miss a human input. The pregame state is also restrictive to what states can be reached from the pregame state, so it has to move into the playing state within a second. This has also been the subject of extensive testing by many people, and all empirical evidence points to this behaviour occuring consistently.

1.3 Prohibit game start on pause

To test this functional requirement, we must verify that when the game is paused, all game functions must stop, all inputs must not be accepted (except unpause) and all game variables must be preserved. To test this, we can use a combination of code inspection, to verify that the paused state does not allow certain inputs and that it pauses the game state. This can also be verified by the teams testing, which shows that this behvaiour is consistent as well.

1.4 Permitting pause

To test this functional requirement, we must verify that the game will accept the pause input while the game is in a playing state. This can be verfied by code inspection and manual bug testing. We can inspect the code to make sure that when the game is in the running, nothing will obstruct the input or the game from changing state to paused. To test this, we tested how the game would respond to a series of pause commands during different times during the game execution, and from this testing, it seems like the pause commands works regardless of other situations.

1.5 Displaying player information above all

To test this functional requirement we must verify that the game always displays accurate user information, as well as verfying that the player info is drawn above all over graphicalm objects. To do this, we can use code inspection and manual testing. The code itself shows that the game info is displayed above all, as it drawn on top of all asteroids, players and bullets. Manual testing also confirms that this is held, as obeservation has shown that all objects are drawn under the player information.

1.6 Player firing and alien destruction

...

1.7 Asteroid collision with player

1.8 Asteroid separation

...

1.9 Witholding player spawn

To test this functional requirement, we must verify that the player will not respawn if their is an adversary within 100 pixels of the spawn point. This can be verfied by code inspection as well as manual testing. The code itself uses a modifed version of the collison to test for this, so as that has been extensively tested, we know that the functionality is okay, so to test the logic, the team inspected the code, and made sure that all flags are correct and placed in the right spots.

1.10 Post-game

...

2 Nonfunctional Requirements Evaluation

2.1 Usability

2.2 Player test

To test usability, playability and feel of the game, the Staroids team recruited ten outside people to play Staroids and give feedback. Several questions were asked of the test players, particularly about the feel of the game and how it played. The most common feedback we recieved related to the alien and the appearance of the ship. The alien was annoying and not a challenge to the play testers, so his spawn chance and times were reduced and he was made easier to hit. When the testers faced the new alien, there were much fewer complaints about him. The second large complaint from testers was the appearance of the ship. The ship, alien and asteroids used to be all white filled shapes with a black border. Testers often could not tell the direction the ship was facing, or even which object was their ship. To fix this, the ship was made narrower which helped show the front of the ship and the player ship and alien were made into solid black shapes so they are visible at a glance. The alien bullets and player bullets were more differentiated due to feedback as well.

2.3 Browser test

To test Staroids, the Staroids team used all major browsers for development and testing in an effort to notice Staroids bugs and issues that were specific to one browser. None were found.

2.4 Informational

2.4.1 State Change

The Staroids state transitions are programmed into the game, so through code inspection and testing, all state transitions were tested. An error involving the game starting while paused was found through this test.

2.5 Internal

3 Comparison to Existing Implementation

In comparing staroids to the original open source game it is based on, the first thing that jumps out is the change in styling on the home screen "Press Space to Play" as well as all of the other text in the game. It has been changed to more of a bubble font rather than the traditional square-ish letter stylings of it's predecessor. All the concepts and rules from the original however have been maintained, all that has been changed in that sence is just the shapes of things such as the asteroids, and speeds and movements of objects such as player, bullets, asteroids etc. This being said both games could be created from the same set of requirements, and should be able to pass the same test plan.

4 Unit Testing

5 Changes Due to Testing

So far there have been a couple of major changes due to testing: the alien movement, player movement, player colour and alien bullet. The alien changes came about during testing when it was found that the alien was moving in the pattern desired (sine wave) but the amplitude was too small, and did not give the player enough time to get out of the way. This lead to the changes in the alien's speed (slower) as well as its amplitude (higher) so it covered a larger path. The next change was player movement and this change simply involved limiting the players speed, due to the fact that during testing it was found to be volatile to the gameplay by just allowing the player to go unlimitedly fast. The max speed that was settled on was based on a trial and error method to get it to a max speed that seemed natural to all team members. The next change was a change to the actual player ships appearance. This change was made based on third party testing that envolved people with no prior experience playing the game, and it was relayed back that the player's ship color being changed to a different colour than the background would be helpful by making it easer to see. Another thing that was brough up during this was changing the ships shape slighly from an equilateral triangle to a isosceles so that the head of the ship would be easier to identify. The final change made due to testing was that of the alien bullet, this was another third party test and the feedback given lead the team to changing the alien bullet to look different from the player bullet, based on the confusion relayed by this tester.

6 Automated Testing

In this project it was found that automated testing was not feasable for a number of reasons. One being that the language chosen (javascript) does not have a testing framework that supported canvas integration which is how the program draws all of it's spites to the screen. Because of this, in order to do automated testing it would require creating another copy of the program that did not reference the canvas which would then need to be updated with every change that is made to the regular program. The second reason for this choice was that since the project is a game, the most complete, practical and appropriate way to test it is manually. Since there are so many moving pieces and states and it is so dependent on what is visually happening for the user rather than what's behind the scenes it seemed like the best testing approach because game's are all about feel and user interaction so the team wanted to make sure that was reflected in how it was tested.

7 Trace to Requirements

8 Trace to Modules

9 Code Coverage Metrics

No formal code coverage software was used for this project based on the decision made by the team to move away from automated testing as explained above and rely strictly on manual. However throughout the development of this project specifically during the MIS portion the team went through all individual functions, variables, objects and methods for this program and either documented them and what they do/how they are used or deleted them if they were not used. This has helped increase the percentage of code covered in during testing, as well as provided an easy way to just run throguh the description of the documented objects and functions to manually see if they are covered in our tests.