JSTanks - Test Plan

Jiahao Li	Pavithran Pathmarajah	Viren Patel
LI577	PATHMAP	PATELVH3
001416646	001410729	001419057

December 8, 2016

Contents

List of Tables

List of Figures

1 Revision History

1.1 Revision 0

Table 1: Revision 1

Date	Developer	Change
October 31	Jiahao Li	Initial Draft
October 31	Pavithran Pathmarajah	Initial Draft
October 31	Viren Patel	Initial Draft

1.2 Revision 1

Table 2: Revision 2

Date	Developer	Change
December 6	Pavithran Pathmarajah	Update Tests Performed
October 31	Viren Patel	Spelling and Grammer

2 General Information

2.1 Purpose

This document is a plan for the testing, validation and verification process that are to be followed by JSTanks after the build process. These test cases were designed prior to the development of the final product; this test plan should be used as guidelines and for reference in the final product testing.

2.2 Scope

This project is being designed to move from running local applets that require compilation to an on-line script interpreted by browsers. Therefore the testing will mainly consists of the teams ability to port java coding methodologies and object oriented styles, to Javascript. The testing will cover, algorithms, data structures, visuals and the system.

2.3 Acronyms, Abbreviations, and Symbols

Acronym/Abbreviation	Meaning
JSTanks	Team Name
JSTanks	Project Name
JS	JavaScript
HTML	Hypertext mark up language
CSS	Cascading style sheets
git	Git Lab
API	Application program interface
GUI	Graphical user interface
AI	Artificial intelligence
PC	Personal computer

Figure 1: Acronyms

2.4 Overview of Document

This document is divided into sections and within each section subsections for different test types. There is the general information section covering information for the document. The plan for how the product will be tested the schedule and the tools used, The system tests to be performed functional and non-functional. Followed by the tests for the proof of concept demonstration, comparisons to the original Java code. Unit testing to be performed as well as the appendix.

3 Plan

3.1 Software Description

JSTanks is a game which allows users enjoy it on a website without downloading it. This game let the user control a tank to fire and move on the map in order to protect its home base and itself from the damage of other tanks'.

3.2 Test Team

The test team will consist of Jiahao Li, Pavithran Pathmarajah, Viren Patel and some random players as testers. Jiahao Li, Pavithran Pathmarajah, Viren Patel will split the entire testing which cover all different types of tests. The random player will test the performance of the game and give feedbacks to the development team.

3.3 Automated Testing Approach

For automated testing we plan to approach the topic through, the use of a secondary Script so that we may brute force many scenarios in a short period of time, and then be able to debug and run the same set of tests on the product again, until all tests are passed and debugging is completed. To save time as compared to having to manually enter the same tests each time.

3.4 Testing Tools

For testing we plan to use web-browsers and a testing script. We plan to use Mozilla Firefox, Google Chrome, and Apple Safari browsers to ensure that our tests pass on the three main browsers used today. We plan to do manual testing as well as automated testing via a custom made script tailored to our testing plan.

3.5 Testing Schedule

Below is our planned testing schedule broken down into milestones.

Table 3: Milestone 1 - Proof of Concept

Test #	Team Member	Comments	Date
4.1.1	Pavithran Pathmarajah	All Scripts load up	10\15 \2016
5.1	Pavithran Pathmarajah	HTML-5 Canvas functional	10\15 \2016
5.2	Pavithran Pathmarajah	Keyboard interface functional	10 \15\2016
5.3	Pavithran Pathmarajah	Update scripts functional	10\16\2016

Table 4: Milestone 2 - WebPage

Test #	Team Member	Comments	Date
4.1.2	Automated Script	SUCCESSFUL	12\6\2016
4.1.3	Automated Script	SUCCESSFUL	12\6\2016
4.1.4	Automated Script	SUCCESSFUL	12\6\2016
4.1.6	Automated Script	SUCCESSFUL	12\6\2016

Table 5: Milestone 3- Menus

Test #	Team Member	Comments	Date
4.1.5	Automated Script	SUCCESSFUL	12\6\2016
4.1.7	Automated Script	SUCCESSFUL	12\6\2016
4.1.8	Automated Script	SUCCESSFUL	12\6\2016
4.1.9	Automated Script	SUCCESSFUL	12\6\2016
4.1.10	Automated Script	SUCCESSFUL	12\6\2016
4.1.11	Automated Script	SUCCESSFUL	12\6\2016
4.1.15	Automated Script	SUCCESSFUL	12\6\2016

Table 6: Milestone 4 - Game

Test #	Team Member	Comments	Date
4.1.13	Automated Script	SUCCESSFUL	12\6\2016
4.1.14	Automated Script	SUCCESSFUL	12\6\2016
4.1.16	Pavithran Pathmarajah	Tank continuously moves in specified directoon	12\6\2016
4.1.17	Automated Script	SUCCESSFUL	12\6\2016
4.1.18	Automated Script	SUCCESSFUL	12\6\2016
4.1.19	Automated Script	SUCCESSFUL	12\6\2016
4.1.20	Automated Script	SUCCESSFUL	12\6\2016
4.1.21	Automated Script	SUCCESSFUL	12\6\2016
4.1.22	Automated Script	SUCCESSFUL	12\6\2016
4.1.23	Automated Script	SUCCESSFUL	12\6\2016
4.1.24	Automated Script	SUCCESSFUL	12\6\2016
4.1.25	Automated Script	SUCCESSFUL	12\6\2016
4.1.26	Automated Script	SUCCESSFUL	12\6\2016
4.1.27	Automated Script	SUCCESSFUL	12\6\2016
4.1.28	Automated Script	SUCCESSFUL	12\6\2016

Table 7: Unit Tests

Test #	Team Member	Comments	Date
7.1.1	Automated Script	SUCCESSFUL	12\6\2016
7.1.2	Automated Script	SUCCESSFUL	12\6\2016
7.1.3	Automated Script	SUCCESSFUL	12\6\2016
7.1.4	Pavithran Pathmarajah	Wall on screen	12\6\2016
7.1.5	Pavithran Pathmarajah	Steel screen	12\6\2016
7.1.6	Pavithran Pathmarajah	Base on screen	12\6\2016
7.1.7	Automated Script	SUCCESSFUL	12\6\2016
7.1.8	Automated Script	SUCCESSFUL	12\6\2016
7.1.9	Automated Script	SUCCESSFUL	12\6\2016
7.1.10	Automated Script	SUCCESSFUL	12\6\2016
7.1.11	Automated Script	SUCCESSFUL	12\6\2016
7.1.12	Automated Script	SUCCESSFUL	12\6\2016
7.1.13	Automated Script	SUCCESSFUL	12\6\2016
7.1.14	Pavithran Pathmarajah	Visible on screen	12\6\2016
7.1.15	Automated Script	SUCCESSFUL	12\6\2016
7.1.16	Automated Script	SUCCESSFUL	12\6\2016

4 System Test Description

4.1 Tests for Functional Requirements

4.1.1 HTML file test

Name: Loading the game
Description: Test if the game dependencies load in browser
Type: Unit test (dynamic, automatic, black box)
Initial State: Testing Script Loaded in
Input: Click the Run Script button
Output: Script is successful, Loads Dependencies
Pass: All files are able to be loaded up.

4.1.2 Standby state test

Name: Standby state
Description: Test if the game automatically runs or waits for user initialization.
Type: Unit test (static, manual, black box)
Initial State: A new browser
Input: Go to home page of game
Output: The home page of the game
Pass: The browser remains on the home page.

4.1.3 The game section of the menu test

Name: Menu of game section
Description: Test if the sub menu of new game section which has choices of level selection and map selection show up.
Type: Unit test (dynamic, automatic, black box)
Initial State: Menu in the standby state
Input: Script selects new game
Output: Menu Tests successfully
Pass: The sub menu with choice of starting a new game and quit.

4.1.4 The pause section of the menu test

Name: Menu representation
Description: check if the menu contains Home Page/Continue/ Instructions/New Game/Quit
Type: Unit test (dynamic, automatic, black box)
Initial State: A new browser
Input: Click the Run Script button
Output: Menu Tests successfully
Pass: The menu with five sections is represented in the standby state.

4.1.5 The level section of the menu test

Name: Menu of level
Description: Test if the sub menu of level section which has choices of level 1, level 2, and level 3 shows up when the level section is clicked.
Type: Unit test (dynamic, automatic, black box)
Initial State: Menu in the new game state
Input: Script runs through levels
Output: New Game Success
Pass: The sub menu with choice of level 1, level 2, level 3, level 4 and level 5.

4.1.6 Game start test

Name: Start the game
Description: Test if the game shall be reset and start when [starting a new game] is clicked.
Type: Unit test (dynamic, automatic, black box)
Initial State: The sub menu of the game section
Input: Click the choice of starting a new game
Output: New Game Success
Pass: The standby state of the game with all objects on their initial position on

the map.

4.1.7 Game quit test

Name: Quit the game
Description: Test if the game comes to ends and the user is redirected to the repo if the quit is clicked
Type: Unit test (dynamic, automatic, black box)
Initial State: The running state of the game
Input: Click the Quit option
Output: Menu Tests successfully
Pass: The game is redirected to the gitlab repository.

4.1.8 Game pause test

Name: Pause the game
Description: Test if the game comes to the pause state when the letter 'p' is pressed on the keybaord.
Type: Unit test (dynamic, automatic, black box)
Initial State: The running state of the game
Input: Script triggers a keyboard event 'p'
Output: The game pauses and the pause menu shows up
Pass: The game comes to a pause state. Game content freezes and stays in the temporal positions whilst the Pause menu is overlayed .

4.1.9 Game continue test 0

Name: Continue the game
Description: Test if the game comes back to the running state state when the letter 'p' is pressed on the keyboard.
Type: Unit test (dynamic, automatic, black box)
Initial State: The pause state of the game
Input: Script triggers a keyboard event 'p'
Output: The game resumes
Pass: All stuff frozen in the pause state are activated and back into the routine.

4.1.10 Game continue test 1

Name: Continue the game in the running state
Description: Test if there is any effect on the game when the choice of continue is clicked in the running state.
Type: Unit test (dynamic, automatic, black box)
Initial State: The running state of the game

Input: Script triggers a click on continue
Output: No effect
Pass: No effect on the running state.

4.1.11 AI test

Name: The routine of AI
Description: Test if the AI controls tanks to move and fire randomly.
Type: Unit test (dynamic, automated, white box)
Initial State: Single AI in centre of empty game board
Input: The script is run
Output: Ai Test succesful
Pass: The script tracks the AI movement over the course of 2 seconds, and if the AI has a net travel of more then 2 tiles, then it passes.

4.1.12 Level test

Name: Levels of the game
Description: Test if the moving speed of tanks controlled by the AI change when level 1, level 2, level 3, level 4 or level 5 is clicked.
Type: Unit test (dynamic, automated, black box)
Initial State: The running state of the game
Input:The script is run
Output: New game test is succesful
Pass: The script checks the AI movement delay value of each game, and if they correspond high to low with the level chosen, then it passes.

4.1.13 Default level test

Name: The default level of the game
Description: Test if level one is chosen if no level is selected. Since the game pareses the level form the URL.
Type: Unit test (static, automated, white box)
Initial State: The new browser windwo
Input: Script redirects to the /JSTanks.Html page directly
Output: The game starts with tanks controlled by the AI moving in the lowest speed
Pass: The script checks the AI movement delay value is at the lowest normal value it can be.

4.1.14 Instructions test

Name: The Instructions of the game
Description: Test if The window with the Instructions of the game in it pops up when the section of Instructions is clicked.
Type: Unit test (dynamic, automated, black box)
Initial State: New browser window
Input: Script opens instructions page
Output: Pages Section of script is succseful
Pass: The instructions modal opens with the instructions displayed.

4.1.15 Tank test

Name: The movement of the tank controlled by the user
Description: Test if the tank controlled by the user moves left, right, up or down when the left, right, up or down key on the keyboard is pressed and fires when the f key is pressed.
Type: Unit test (dynamic, automated, black box)
Initial State: The running state of the game
Input: Script triggers left, right, up, down or 'f' keyboard events
Output: Script tank is succesful
Pass: The tank moves accordingly with the key board input and fires when "F" is pressed.

4.1.16 Continuous movement test

Name: The continuous movement of the tank controlled by the user
Description: Test if the tank controlled by the user keeps moving in the direction of left, right, up or down when the left, right, up or down key on the keyboard is held.
Type: Unit test (dynamic, manual, black box)
Initial State: The running state of the game
Input: Hold the left, right, up or down key on the keyboard
Output: The continuous movement of the tank controlled by the user
Pass: The tank keeps moving in the correct direction according to the key held by the user until the user release the key.

4.1.17 Bullet launch test

Name: Launch the bullet
Description: Test if a bullet is correctly launched by the fire commands
Type: Unit test (dynamic, automatic, black box)
Initial State: Blank game Screen
Input: Script calls the fire functionality of the game

Output: Script projectiles is successful
Pass: A bullet is fired.

4.1.18 Bullet movement test

Name: The movement the bullet
Description: Test if the bullet keep moving in the same direction after being launched.
Type: Unit test (dynamic, automated, black box)
Initial State: The bullet is launched
Input: –
Output: Script projectiles is successful
Pass: A bullet moves continuously in the same direction.

4.1.19 Bullet disappearance test

Name: The bullet disappearance
Description: Test if the bullet disappears when it hits the tank, wall, steel, home base or the boundary of the map.
Type: Unit test (dynamic, automated, black box)
Initial State: The bullet is moving in a specific direction
Input: –
Output: Script projectiles is successful
Pass: The bullet disappears when it hits the tank, wall, steel, home base or the boundary of the map.

4.1.20 Wall hit test

Name: The wall hit by the bullet
Description: Test if the wall disappears when it is hit by the bullet.
Type: Unit test (dynamic, automated, black box)
Initial State: Bullet fired toward a wall
Input: –
Output: Script projectiles is successful
Pass: The wall disappears immediately when it is hit by the bullet.

4.1.21 Steel hit test 0

Name: The steel hit by the bullet twice
Description: Test if the steel stays the same when it is hit by the bullet twice.
Type: Unit test (dynamic, automated, black box)
Initial State: Bullet fired toward a steel wall
Input: –

Output: Script projectiles is successful
Pass: The steel tile remains on screen but its strength is decreased.

4.1.22 Enemy tanks hit test

Name: The tank controlled by the AI hit by the bullet
Description: Test if the tank controlled by the AI disappears when it is hit by the bullet.
Type: Unit test (dynamic, automated, black box)
Initial State: The tank controlled by the AI is on screen
Input: Bullets fired at tank
Output: Script end game is succesful
Pass: The tank controlled by the AI disappears when it is hit by the bullets.

4.1.23 Home base hit test 0

Name: The home base hit by the bullet for first four times
Description: Test if the home base stays the same when it is hit by the bullet for first four times.
Type: Unit test (dynamic, automated, black box)
Initial State: The home base is on screen
Input: Bullets fired at base
Output: Script projectiles is successful
Pass: The home base stays the same when it is hit by the bullet for the first four times.

4.1.24 Home base hit test 1

Name: The home base hit by the bullet at the fifth time
Description: Test if the home base disappears when it is hit by the bullet at the fifth time.
Type: Unit test (dynamic, automated, black box)
Initial State: The damaged home base is on screen
Input: Bullets fired at tank
Output: Script end game is successful
Pass: The home base disappears when it is hit by the bullet a fifth time.

4.1.25 User tank hit test 0

Name: The tank controlled by the user hit by the bullet for the first time
Description: Test if the tank controlled by the user stays the same when it is hit by the first bullet.
Type: Unit test (dynamic, automated, black box)

Initial State: The tank controlled by the user is on screen
Input: bullets fired at tank
Output: Script projectiles is successful
Pass: The tank controlled by the user stays the same when it is hit by the bullet for the first four times.

4.1.26 User tank hit test 1

Name: The tank controlled by the user hit by the bullet at the fifth time
Description: Test if the tank controlled by the user disappears when it is hit by the bullet at the fifth time.
Type: Unit test (dynamic, automated, black box)
Initial State: The damaged tank controlled by the user is on screen
Input: Bullets fired at tank
Output: Script end game is successful
Pass: The tank controlled by the user disappears when it is hit by the bullet a fifth time.

4.1.27 Game over test 0

Name: The player tank is destroyed
Description: Test if the game comes to the end state when the tank controlled by the user disappears.
Type: Unit test (dynamic, automated, black box)
Initial State: The tank controlled by the user disappears
Input: bullets fired at tank
Output: Script end game is successful
Pass: The player tank is destroyed and the game enters an End state.

4.1.28 Game over test 1

Name: The home base is destroyed
Description: Test if the game comes to the end state when the home base disappears.
Type: Unit test (dynamic, automated, black box)
Initial State: The damaged home base is on screen
Input: Bullets fired at base
Output: Script end game is successful
Pass: The base is destroyed and the game enters an End state

4.2 Tests for Nonfunctional Requirements

4.2.1 Appearance / Style

Description: A survey will be provided to classmates who will test the game and fill out the survey.
Questions:

- Can user Tank be distinguished from AI?
- Are all non-user controlled tanks the same colour?
- Can wooden walls be distinguished from Steel walls?
- Can the home base be distinguished?
- Is the overall colour scheme put any strain on the eyes?
- Does the menu cover everything needed to play the game?

Pass: If 95% of surveys are positive, then the test is considered as passed.

4.2.2 Ease of Use

Description: A survey will be provided to classmates who will test the game and fill out the survey.
Questions:

- Is the response time satisfying?
- Is the game straight forward to play?
- Are the instructions convoluted in nature?
- Are all menus understandable?

Pass: If 95% of surveys are positive, then the test is considered as passed.

4.2.3 Accessibility Requirements

Description: Test if the game functions on specified browsers. How: Running all manual unit tests from section 4.1 on the following browsers:

- Google Chrome
- Mozilla Firefox
- Apple Safari

Pass: If the game passes all manual unit tests on all browsers, the requirement is met.

4.2.4 Performance

Description: The game should run at equal speed across all platforms and with different hardware specifications.
How: Playing a standard game on the following systems:

- Late 2013 - MacBook Pro
- Lab - Virtual Computer
- Thode - Virtual Computer
- 2014 - Surface Pro 3

Pass: If game runs at similar speeds across all platforms, where relative similarity will be decided by the tester.

4.2.5 Maintainability

Description: The game's source code should be easy to read, maintain, and learn from.
How: Have a classmate whom does not work with JS to read over an object file, and have them tell us if they find the code easy to understand.
Pass: Classmate states that code is easy to read and learn from.

4.2.6 Security

Description: The game should not access and send user data to an external source.
How: Download the source files, then close all network connections.
Pass: If the game is able to run off-line, it is not sending any data back. Reasoning: If one method in JS fails the entire script usually crashes.

4.2.7 Cultural Requirements

4.2.8 Legal Requirements

Description: The game should follow Canadian Anti-Spam legislation.
How: Review legislation and look through code to ensure legislation is followed.
Pass: Legislation is not violated.

5 Tests for Proof of Concept

The proof of concept forJSTanks was to demonstrate a block moving across the screen in up, down, left, and right directions according to the user input from the keyboard. In addition, boundaries had to be set so that the block does not go off-screen. JSTanks was successfully able to present this during the proof of concept demonstration. The testing measures included functional dynamic testing as follows:

5.1 Display

Name: Display block
Description: Test if the block is successfully displayed on the web screen once the corresponding HTML file is launched.
Type: Unit Test (dynamic, manual, black box)
Initial State: Empty white screen
Input: Not Applicable
Output: A block
Pass: A block is displayed on the screen.

5.2 Movement

Name: Block movement
Description: Test if the blocks position is updated after the user input.
Type: Unit Test (dynamic, manual, black box)
Initial State: block displayed on screen
Input: Up, down, left, or right key on the keyboard
Output: The block moves accordingly
Pass: The tanks position is updated by the specified value in the correct direction.

5.3 Boundaries

Name: Screen Boundary
Description: Test if the block stays in bounds of the screen.
Type: Unit Test (dynamic, manual, black box)
Initial State: block displayed on screen
Input: Any one of up, down, left or right key on the keyboard until the block is at the edge of the screen in the according side.
Output: The block stays on the screen near the corresponding edge and does not go off-screen.
Pass: The block stays within the screen even though the input tries to force it off-screen.

6 Comparison to Existing Implementation

The open source game software we are working on is simply a Java application in its existent form. We are uploading the same game on a web browser which requires a different programming language altogether. As a result, we have not been able to use any existing code and have had to program the game completely from scratch. We have also not looked into the Java code to learn its implementation style or use any ideas for specific functions. Therefore, any similarities between our code and the existing code are coincidental.

The major difference between the two implementations is that we have HTML,

and CSS files in addition to the JavaScript source code which are required for any kind of web development. The existing code works with multiple classes representing different aspects of the game which can be seen in our code as well. However, it has more classes, three of them which are main method classes which can be explained by the programming language used. Our implementation has no main method or class, but instead the HTML file is used as the main class which drives the whole game. Another important difference is the style of programming; the existing implementation has made use of threads which we have not as we are still learning to work with JavaScript. The use of object oriented programming is evident in both implementations. For example, objects for tanks, bots, and barriers are included in both.

7 Unit Testing Plan

7.1 Unit testing for internal functions

7.1.1 Wall type test

Name: Ask for the type of the wall
Description: Test if the program returns the type of the wall when you ask for it.
Type: Unit test (dynamic, automated, white box)
Initial State: Barriers on screen made of specified wall typel
Input: wall.type()
Output: "BARRIER"
Pass: The program returns the type "BARRIER" when wall.type() is called.

7.1.2 Steel type test

Name: Ask for the type of the steel
Description: Test if the program returns the type of the steel when you ask for it.
Type: Unit test (dynamic, automated, white box)
Initial State: Barriers on screen made of type steel
Input: steel.type()
Output: "BARRIER"
Pass: The program returns the type "BARRIER" when wall.type() is called.

7.1.3 Home base type test

Name: Ask for the type of the home base
Description: Test if the program returns the type of the home base when you ask for it.
Type: Unit test (dynamic, automated, white box)

Initial State: Barriers on screen made of type home base
Input: homebase.type()
Output:""BARRIER
Pass: The program returns the type "BARRIER" when wall.type() is called.

7.1.4 Wall draw test

Name: Draw the wall on the game board
Description: Test if the image of the wall shows up on the position we set on the game board in the right size when we call this function.
Type: Unit test (dynamic, manual, white box)
Initial State: The game board with no image on the position (startX,startY)
Input: wall.draw(canvas,startX,startY,tileSize,t)
Output: The image of the wall shows up on the position (startX,startY) of the game board
Pass: The image of the wall shows up on the position (startX,startY) of the game board in the tileSize.

7.1.5 Steel draw test

Name: Draw the steel on the game board
Description: Test if the image of the steel shows up on the position we set on the game board in the right size when we call this function.
Type: Unit test (dynamic, manual, white box)
Initial State: The game board with no image on the position (startX,startY)
Input: steel.draw(canvas,startX,startY,tileSize,t)
Output: The image of the steel shows up on the position (startX,startY) of the game board
Pass: The image of the steel shows up on the position (startX,startY) of the game board in the tileSize.

7.1.6 Home base draw test

Name: Draw the home base on the game board
Description: Test if the image of the home base shows up on the position we set on the game board in the right size when we call this function.
Type: Unit test (dynamic, manual, white box)
Initial State: The game board with no image on the position (startX,startY)
Input: homebase.draw(canvas,startX,startY,tileSize,t)
Output: The image of the home base shows up on the position (startX,startY) of the game board
Pass: The image of the home base shows up on the position (startX,startY) of the game board in the tileSize.

7.1.7 Wall hit test

Name: Hit the wall
Description: Test if the program decreases the points of strength of the wall after it has been hit.
Type: Unit test (dynamic, automated, white box)
Initial State: Wall on screen
Input: projectile hits wall
Output: Script projectiles is successful
Pass: The wall is destroyed and thus disappears from the game board.

7.1.8 Steel hit test

Name: Hit the steel
Description: Test if the program decreases the points of strength of the steel after it has been hit.
Type: Unit test (dynamic, automated, white box)
Initial State: Steel Walll on screen
Input: projectile hits steel wall
Output: Script projectiles is successful
Pass: One point of strength of the steel wall is decreased after a hit.

7.1.9 Home base hit test

Name: Hit the home base
Description: Test if the program decreases the points of strength of the home base after it has been hit.
Type: Unit test (dynamic, automated, white box)
Initial State: home base on screen
Input: projectile hits base
Output: Script projectiles is successful
Pass: One point of strength of the home base is decreased after a hit.

7.1.10 Wall health get test

Name: Get health of the wall
Description: Test if the program returns the remaining points of strength of the wall after calling this function.
Type: Unit test (dynamic, automated, white box)
Initial State: wall on screen
Input: projectile hits wall
Output: Script projectiles is successful
Pass: Return the remaining points of strength of the wall after calling this func-

tion.

7.1.11 Steel health get test

Name: Get health of the steel
Description: Test if the program return the remaining points of strength of the steel wall after calling this function.
Type: Unit test (dynamic, automated, white box)
Initial State: steel wall on screen
Input: projectile hits steel wall
Output: Script projectiles is successful
Pass: Return the remaining points of strength of the steel wall after calling this function.

7.1.12 Home base health get test

Name: Get health of the home base
Description: Test if the program return the remaining points of strength of the home base after calling this function.
Type: Unit test (dynamic, automated, white box)
Initial State: home base on screen
Input: projectile hits base
Output: Script projectiles is successful
Pass: Return the remaining points of strength of the home base after calling this function.

7.1.13 Tank Constructor Test

Name: Tank Constructor Testing
Description: Test if a tank object is created with the specified attributes when the constructor is called.
Type: Unit Test (dynamic, automated, white box)
Initial State: empty screen
Input: new tank object is created by script
Output: usertank succesful
Pass: The following methods create the specified tank object in position.

7.1.14 Tank Draw Test

Name: Tank Graphics Testing
Description: Test if the image of the tank shows up at the position set by the game board when the function is called.
Type: Unit Test (dynamic, manual, white box)

Initial State: The game board with no image on the position (startX, startY) newline Input: tankObject.draw(canvas, startX, startY, tileSize)
Output: The image of the tank appears on the position (startX, startY) of the game board.
Pass: The tank is successfully displayed on the correct position on the game board.

7.1.15 Tank Hit Test

Name: Projectile Impact Testing
Description: Test if the health points of the tank are reduced after the tank has been hit with a projectile.
Type: Unit Test (dynamic, automated, white box)
Initial State: Tank on screen projectile fried at tank
Input: projectile hits tank
Output: Script projectiles is successful
Pass: Specified points of the tanks health are reduced.

7.1.16 Tank Health Test

Name: Tank Health Getter Testing
Description: Test if the method returns the correct number of health points for the tank when called upon.
Type: Unit Test (dynamic, automated, white box)
Initial State: A tank object has been created, and hit by projectile
Input: projectile hits tank
Output: Script projectiles is successful
Pass: The method returns the correct number of health points remaining for the tank object.