SE 3XA3: Test Report
Title of Project

Team #15, Dev$^{\text{enthusiasts}}$
Zihao Du (duz12)
Andrew Balmakund (balmakua)
Namit Chopra (choprn9)

April 9, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
08/04/2021	1.0	Initial Draft

This document provides an overview of the testing done for the implementation of T-Rex Acceleration. This document contains an evaluation of if the functional and non-functional requirements were implemented correctly, a comparison to the existing implementation of the software, an overview of the unit testing that was done, and changes made that were due to testing, and an overview of the automated testing that was done. It also contains a trace to the requirements and modules, as well as code coverage metrics.

All the tests below are from the test plan which can be viewed here.

1 Functional Requirements Evaluation

1.1 User Input

Mouse Input

1. test-UI1: Testing change settings

   Type: Functional, Dynamic, Manual

   Initial State: The game environment is loaded with the default settings.

   Input: The user clicks on various option settings.

   Output: The game system responds to the change of each setting.

   How test will be performed: A visual test is used to confirm the different game settings are reflected in the game as intended (theme color, text size). An auditory test is used to confirm the changes in the volume settings are working correctly.

   Results: PASS. A manual test was used to changed different setting volumes to verify that the sound volume settings increase and decrease ranging from mute to max volume.

Keyboard Input

1. test-UI2: Testing main menu play option

   Type: Functional, Dynamic, Manual

Initial State: The main menu is loaded and displayed in the game window.

Input: The user presses KEYBOARD_SPACE to play.

Output: The main menu disappears from the screen and the game starts to run. All the game assets, including the character, environment, and sound, are loaded and displayed on the screen.

How test will be performed: A member of the testing team will confirm that the game starts with all assets successfully loaded after the "Play" button is pressed.

Results: PASS. A manual test was used to press KEYBOARD_SPACE to verify that the game starts and all assets are loaded successfully.

2. test-UI3: Testing main menu quit option

   Type: Functional, Dynamic, Manual

   Initial State: The main menu is loaded and can be seen in the game window.

   Input: The user presses KEYBOARD_Q to quit.

   Output: The application gracefully stops running and the game window is closed.

   How test will be performed: A member of the testing team will confirm that the game window closes without errors after the "Quit" button is pressed.

   Results: PASS. When the user presses KEYBOARD_Q on the main menu, the game ends and the window closes .

3. test-UI4: Testing main menu settings option
   Type: Functional, Dynamic, Manual

   Initial State: The main menu is loaded and can be seen in the game window.

   Input: The user presses KEYBOARD_S to go to the "Settings" menu.

   Output: The settings screen is displayed with GIVEN_OPTIONS.

How test will be performed: A member of the testing team will confirm that the GIVEN_OPTIONS are displayed once the "Settings" button is pressed.

Results: PASS. When the user presses KEYBOARD_S on the main menu, the settings menu is displayed on the screen.

4. test-UI5: Test jump movement

   Type: Functional, Dynamic, Manual

   Initial State: The character is running on the platform.

   Input: The KEYBOARD_UP will be pressed once.

   Output: Dino character moves at PIXEL_JUMP up at a constant RATE on the Y-Axis and PIXEL_JUMP down at a constant RATE on the Y-Axis.

   How test will be performed: A visual test is used to verify that the Dino character jumps when KEYBOARD_UP is pressed, the Dino character position moves up along the Y-Axis at a constant RATE to a specific height PIXEL_JUMP and moves back down PIXEL_JUMP pixels on the Y-Axis at a constant RATE to the original standing position.

   Results: PASS. When the user presses KEYBOARD_UP, the dino character jumps and the image is translated up and down as intended.

5. test-UI6: Test duck movement

   Type: Functional, Dynamic, Manual

   Initial State: The character is running on the platform.

   Input: The KEYBOARD_DOWN will be pressed once.

   Output: Dino character moves at PIXEL_DUCK down at a constant RATE on the Y-Axis and PIXEL_DUCK up at a constant RATE on the Y-Axis.

   How test will be performed: A visual test is used to verify that the Dino character ducks when KEYBOARD_DOWN is pressed, the Dino character position moves down along the Y-Axis at a constant rate to a specific height PIXEL_JUMP and moves back up PIXEL_JUMP pixels on the Y-Axis at a constant RATE to the original standing position.

Results: PASS. When the KEYBOARD_DOWN key is pressed, the duck image of the character is displayed on the game screen.

6. test-UI7: Test game pause

   Type: Functional, Dynamic, Manual

   Initial State: The game is running with all assets loaded successfully.

   Input: The KEYBOARD_P will be pressed once on the user's keyboard.

   Output: The game session freezes - all visual elements of the game stop moving, and a window displaying the pause menu appears.

   How test will be performed: A visual test is used to verify that the gameplay stops when KEYBOARD_P is pressed, the character stops moving, the background stops changing and the sound effect also paused. Also, the character does not reply to any more keyboard input.

   Results: PASS. When the user presses KEYBOARD_P, the game freezes and all visual elements stop moving with a pause menu appearing on the screen.

7. test-UI8: Test game resume

   Type: Functional, Dynamic, Manual

   Initial State: The game is paused.

   Input: The KEYBOARD_R will be pressed once on the user's keyboard.

   Output: The paused menu disappears revealing the current game state; however, the game does not start for another after RESUME_TIME_DELAY. The game continues once the countdown from RESUME_TIME_DELAY. is over.

   How test will be performed: A visual test is used to verify that when KEYBOARD_R is pressed, the pause menu starts to count down for RESUME_TIME_DELAY. The pause menu disappears and the user can control the character again. The character, background, obstacles, and sound effects resume as well.

Results: PASS. When the KEYBOARD_R is pressed, the pause menu disappears, revealing the current game with a countdown from RESUME_TIME_DELAY.

8. test-UI9: Test end menu restart

   Type: Functional, Dynamic, Manual

   Initial State: The game ends and the end menu appears.

   Input: The KEYBOARD_R will be pressed once on the user's keyboard.

   Output: The game should turn to the main menu page.

   How test will be performed: A visual test is used to verify that when KEYBOARD_R is pressed, the end menu disappears. The current game session ends and the user goes back to the main menu.

   Results: PASS. When the KEYBOARD_R is pressed, the end menu disappears and the game returns to the main menu.

9. test-UI10: Test end menu quit
   Type: Functional, Dynamic, Manual

   Initial State: The game ends and the end menu appears.

   Input: The KEYBOARD_Q will be pressed once on the user's keyboard.

   Output: The application gracefully stops running and the game window is closed

   How test will be performed: A visual test is used to verify that when KEYBOARD_Q is pressed the game window closes without errors

   Results: PASS. When the KEYBOARD_Q is pressed on the end menu, the game window closed without errors.

10. test-UI11: Test pause quit

    Type: Functional, Dynamic, Manual

    Initial State: The player is in a current game session and the pause menu has been accessed.

Input: The user presses KEYBOARD_Q to quit.

Output: The game should turn to the main menu page.

How test will be performed: A visual test is used to verify that when KEYBOARD_Q is pressed, the current game session ends. The user goes back to the main menu.

Results: PASS. When the KEYBOARD_Q is pressed during the pause menu, the current game ends and displays the main menu.

11. test-UI12: Test instruction menu back

    Type: Functional, Dynamic, Manual

    Initial State: The game is in the instruction menu.

    Input: The KEYBOARD_B will be pressed once on the user's keyboard.

    Output: The game should turn to the main menu page.

    How test will be performed: A visual test is used to verify that when KEYBOARD_B is pressed, the instruction menu disappears. The current game session ends and the user goes back to the main menu.

    Results: PASS. A visual test is used to verify that when the KEYBOARD_B is press on the instruction menu, the game will return to the main menu.

1.2 Game Environment

1. test-GE1: Test current score

   Type: Functional, Dynamic, Manual

   Initial State: The game is running with all assets loaded successfully.

   Input: None.

   Output: A score is shown on the top left corner of the screen and increases as the character runs.

   How test will be performed: A visual test is used to verify that a score is shown on the top left corner when the test starts and when the character moves forward, the score increases and the score stops increasing when the game is paused.

Results: PASS. When the user is playing the game, the score is continuously increasing until the pause menu has been accessed. The score before and after the pause menu stays the same. The score also starts from 0 every new game.

2. test-GE2: Test game instructions

   Type: Functional, Dynamic, Manual

   Initial State: The game is on main menu and loaded with assets successfully.

   Input: KEYBOARD_H is pressed to show "How to play" menu.

   Output: An instruction page is shown on the screen.

   How test will be performed: A member of the testing team will perform a visual test to confirm the game instructions appear on the screen after KEYBOARD_H is pressed on main menu.

   Results: PASS. A visual test is used to verify that an instruction page appears after KEYBOARD_H is pressed on the user keyboard.

3. test-GE3: Test sound effects

   Type: Functional, Dynamic, Manual

   Initial State: The game is running successfully with all assets loaded.

   Input: The user moves the character using the SPECIFIED_KEYS.

   Output: For each specified keyboard input, the corresponding sound effect is played.

   How test will be performed: A member of the testing team will perform a visual and auditory test to confirm that appropriate sound effects are played when the user presses SPECIFIED_KEYS.

   Results: PASS. When the user performs a jump action and jump sound effect occurs, duck action and duck sound effect occurs.

4. test-GE4: Test end menu appearance

   Type: Functional, Dynamic, Manual

Initial State: The character hits an obstacle.

Input: None.

Output: The game state ends and an end menu appears.

How test will be performed: A visual test will be used to confirm when the user hits an obstacle. The game ends and the end menu appears.

Results: PASS. When the character collides with an obstacle, the end menu is displayed on the game screen.

1.3 Game Mechanics

1. test-GM1: Test obstacle collision - Jump

   Type: Functional, Dynamic, Manual

   Initial State: The character is running on the platform and one obstacle spawns moving towards the Dino character.

   Input: The KEYBOARD_UP will be pressed once to jump.

   Output: When the character collides with an obstacle, the current game session ends and the end menu appears.

   How test will be performed: A visual test will be used to confirm when the character collides with an obstacle. The character is unsuccessful to jump over the obstacle (i.e jumps too early/late) and the game session is over.

   Results: PASS. When the KEYBOARD_UP is pressed as obstacles are appearing, user is able to successfully jump over an obstacle without colliding. Jumping too early or late, a correct collision detection is still determined.

2. test-GM2: Test obstacle collision - Duck

   Type: Functional, Dynamic, Manual

   Initial State: The character is running on the platform and one obstacle spawns moving towards the Dino character.

   Input: The KEYBOARD_DOWN to duck will be pressed down indefinitely.

Output: When the character collides with an obstacle, the current game session ends and the end menu appears.

How test will be performed: A visual test will be used to confirm when the character collides with an obstacle. The character is unsuccessful to duck underneath the obstacle and the game session is over.

Results: PASS. When the KEYBOARD_DOWN is pressed as obstacles are appearing, user is able to successfully duck under an obstacle without colliding. Ducking too early or late, a correct collision detection is still determined.

3. test-GM3: Test obstacle collision - No movement

   Type: Functional, Dynamic, Manual

   Initial State: The character is running on the platform and one obstacle spawns moving towards the Dino character.

   Input: No character movement.

   Output: When the character collides with an obstacle, the current game session ends and the end menu appears.

   How test will be performed: A visual test will be used to confirm when the character collides with the obstacle. During this test, the user does not perform any character movement and the game session ends.

   Results: PASS. When no keyboard button is pressed as obstacles are appearing, the character collides with an obstacle.

4. test-GM4: Test obstacle spawn time

   Type: Functional, Dynamic, Automated

   Initial State: The game environment is loaded without the character model and the first obstacle is spawned.

   Input: A second obstacle spawned at a random time delay after the first obstacle spawn time.

   Output: The second obstacle should spawn at MINIMUM_SPAWN_TIME.

   How test will be performed: A automated test is used to confirm that the obstacle spawn randomly that is at least the MINIMUM_SPAWN_TIME.

Results: PASS. Ensuring obstacles are spawning no less than the specific minimum time and no more than the specific maximum time to spawn.

5. test-GM5: Test random obstacle spawn order

   Type: Functional, Dynamic, manual

   Initial State: The game environment is loaded without the character model and obstacles.

   Input: The obstacles are spawned.

   Output: The order in which the different obstacles are spawned will be random.

   How test will be performed: A visual test is used to confirm that different obstacles are spawning in random order.

   Results: PASS. When playing the game, the order of obstacles is different every new game.

6. test-GM6: Test acquire power-up appearance

   Type: Functional, Dynamic, Manual

   Initial State: The character is running on the platform and a power-up is spawned.

   Input: The user moves the character using keyboard inputs to collide with the power-up icon.

   Output: When the character collides with a power-up, the character's appearance changes to indicate the power-up is acquired.

   How test will be performed: A member of the testing team will perform a visual test to confirm that the character's appearance changes for different power-ups acquired.

   Results: PASS. When the user is playing the game and acquiring different powerups, there are visual changes when different powerups have been acquired.

7. test-GM7: Test acquire power-up statistics

Type: Functional, Dynamic, Manual

Initial State: The character is running on the platform and a power-up is spawned.

Input: The user moves the character using keyboard inputs to collide with the power-up icon.

Output: When the character collides with a specific power-up icon, updated properties of the character's statistics are displayed on the screen.

How test will be performed: A member of the testing team will perform a visual test to confirm that the character's properties (which are displayed on the screen) have been altered based on the power-up acquired. For different power-ups, certain properties will be focused more than others.

Results: PASS. Before the release of the final version of the game, character statistics were displayed on the screen. Whenever the character acquired a powerup, updated character statistics, with the powerup acquired, appeared on the screen.

8. test-GM8: Test power-ups spawn

   Type: Functional, Dynamic, Automated

   Initial State: The game environment is loaded without the character model and one powerup spawning.

   Input: A second powerup is spawned at a random time delay after the first powerup spawn time.

   Output: The second powerup should spawn at MINIMUM_SPAWN_TIME.

   How test will be performed: A automated test is used to confirm that the powerup spawn randomly that is at least the MINIMUM_SPAWN_TIME.

   Results: PASS. Ensuring powerups are spawning no less than the specific minimum time and no more than the specific maximum time to spawn.

9. test-GM9: Test updating highest score

Type: Functional, Dynamic, Manual

Initial State: The character is running on the platform and obstacles are spawning.

Input: The user moves the character using keyboard inputs to survive as long as they can.

Output: When the character surpasses the current high score, the game shall update the current high score with the current game score.

How test will be performed: The visual test will be used to confirm when that the high score is updated once the user achieves a current score higher than the previous high score.

Results: PASS. The highest score is updated when the current score is higher than previous high score during the game. The change in high score is also retained for new games in the same session.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

1. test-LF1: Test bright colour scheme

   Type: Functional, Dynamic, Manual

   Initial State: The game is loaded successfully.

   Input/Condition: The tester plays the game and record all colours appeared.

   Output/Result: All displayed colours have at least 50% brightness and saturation.

   How test will be performed: A tester will play the game and record all the colours that appear in the game and check if their brightness and saturation are over 50% in some photo editor.

   Results: PASS. A photo editor confirmed that the brightness and saturation of the main colours used in the application are at least 50% each.

2.2 Usability

1. test-UH1: The game shall have few and simple controls.
   Type: Functional, Dynamic, Manual, Static etc.

   Initial State: The game session has started.

   Input: The user starts playing the game.

   Output: The user gives an average rating of 9 for the controls of the game out of 10.

   How test will be performed: A survey will be taken amongst a group of individuals of MIN_AGE to determine how easily and quickly they learned the controls of the game.

   Results: PASS. From the survey we conducted amongst our friends and family, we collected a rating of 9.5/10 for simplistic controls.

2. test-UH2: The user shall be able to change the volume of the game.
   Type: Functional, Dynamic, Manual

   Initial State: The game sound assets are loaded successfully.

   Input: The user changes the volume in the setting menu.

   Output: The game volume changes accordingly to the user's modification.

   How test will be performed: A tester will change the volume in the setting menu and test the difference between sound effects before and after the change.

   Results: PASS. When the user changes the volume settings, they can hear the volume difference from the before and after settings.

2.3 Performance

1. test-PF1: Test average FPS for a regular game session

   Type: Functional, Dynamic, Automated

   Initial State: The game environment is loaded with the character and obstacles spawning.

Input: The tester controls the character to dodge the obstacle and survive as long as possible.

Output: After the current game session has finished, the average FPS will be displayed and should be greater or equal to FPS_GOAL.

How test will be performed: A pygame method can be used to obtain the current FPS of the application. A visual test will be used to confirm the average FPS that is displayed and is around FPS_GOAL after the player's game session has ended.

Results: PASS. When the game is played, the all FPS readings was taken throughout the game and the average FPS meets the FPS_GOAL.

2. test-PF2: Test FPS of multiple objects displayed at the same time

   Type: Functional, Dynamic, Automated

   Initial State: The game environment is loaded with only the character model.

   Input: Varying number of obstacles and power-ups spawning in.

   Output: The FPS will be displayed to show the effect of increasing the number of obstacles and power-ups spawned.

   How test will be performed: A pygame method can be used to obtain the current FPS of the application. A visual test will be used to confirm the FPS for varying inputs of the number of obstacles and power-ups spawned simultaneously.

   Results: PASS. When the game is played, about 5 obstacles and 5 powerups (a total of 10 moving images) were able to be drawn on the screen while maintaining a comfortable FPS average for the systems we tested on.

3. test-PF3: Test input response time for character movement

   Type: Functional, Dynamic, Manual, Automated

   Initial State: The game environment is loaded with the character model, obstacles, and power-ups spawning.

   Input: The player uses KEYBOARD_UP and KEYBOARD_DOWN.

Output: The response time is displayed when the character has pressed one of the character movement buttons. The average time of the system's response to less than or equal to the RESPONSE_TIME.

How test will be performed: A visual test will be used to confirm the response times for the character movement buttons when obstacles and power-ups are also spawning. The response time average should correspond to RESPONSE_TIME.

Results: PASS. When the game is played, the time when the user makes a request for a character movement (jump or duck), the system responds back well below the RESPONSE_TIME.

4. test-PF4: Test loading in visual assets

   Type: Functional, Dynamic, Automated

   Initial State: Empty PyGame window.

   Input: All images from the visual assets are loaded into the PyGame window.

   Output: IO Exception error is generated if one of the images are not able to load into the PyGame window (does not exist or can not access file). And all the loading takes less than LOAD_ASSET_TIME.

   How test will be performed: A automated test will be used to verify that all the visual assets can be accessed and loaded into the PyGame window. However in the case there is one image not able, an IO Exception error will be generated.

   Results: PASS. All assets such as image and audio files were able to load successfully (no exception thrown) in less than LOAD_ASSET_TIME.

5. test-PF5: Test unknown keyboard input

   Type: Functional, Dynamic, Manual

   Initial State: The game is running successfully.

   Input: Series of unrecognized keyboard buttons are pressed one at a time.

   Output: Nothing should happen when there is an unrecognized keyboard input.

How test will be performed: A visual test will be used to verify that unrecognized keyboard input does not perform any action and/or change the current game state.

Results: PASS. For all different menus accessed and when the game is played, no unknown keyboard input affects the game.

2.4 Maintainability and Support

1. test-MS1: Test operating system supportability

   Type: Functional, Dynamic, Manual

   Initial State: The game application (all source code, visual and audio assets are downloaded) and all of its library dependencies are downloaded.

   Input: The player executes a command sequence to run the game.

   Output: The game shall run without any crashes or errors.

   How test will be performed: A visual test will be used to confirm the game is able to run on these OPERATING_SYSTEMS.

   Results: PASS. The game was able to successfully run on different OPERATING_SYSTEMS without any errors or modifications to the code.

3 Comparison to Existing Implementation

The original implementation has issues in the way it was modularized. Most of the modules are high coupling and all in the same file and it makes it pretty hard to update and modify the system. We improved the architecture of the game following the MVC design pattern and make ease of testing and maintenance of the system. The new architecture makes the code more readable, making new features easier to add and update more handy to the developers. Also, it makes the testing much more convenient. If you want to know more about the architecture of the new implementation, you can view our Module Guide here and our Module Interface Specification here.

4 Unit Testing

4.1 Unit testing of internal functions

The module LoadAssets tested using unit testing. Each method is the module is testing for the time it takes to load the asset(s). All methods in the module successfully passed the test cases and loaded the assets in less than the target time. Using unit testing, the load time of the application was determined to be less than 1ms.

4.2 Unit testing of output files

N/A

5 Changes Due to Testing

5.1 User Input

There were no changes due to testing in user input.

5.2 Game Environment

Testing the game instructions revealed the need for an instructions menu. The time of the displayed instructions was too short and would interfere with the gameplay if they were displayed for longer. Taking this into account, the team removed displaying the instructions at the start of the game and displayed it in a separate menu. Another change that occurred due to testing was playing one character movement sound effect at a time. The user would be able to play the ducking sound effect despite the character being mid-air. The team had to ensure that only one character movement sound effect should be played at any one time despite multiple inputs to accurately reflect the appearance of the character on the screen.

5.3 Game Mechanics

Our team made some changes about powerups statics. All the powerup information was stored in the GameController, but the powerup time changes when pausing in the test and the powerup properties were moved to another module to pass the test. The team also made some changes about the character's appearance after acquiring a powerup. In the test, it was hard to distinguish if the appearance changes in some bright background, and the team redesigned this part by showing powerup icons on top of the character instead.

5.4 Look and Feel

There were no changes due to testing in look and feel.

5.5 Usability

There were no changes due to testing in usability.

5.6 Performance

Testing the performance of the game involved with testing the entire system. Many of the performance issues faced in the beginning dealt with the pygame visuals displayed on the screen. We also noticed there was a maximum amount of translating images we can have on a screen at a time before there was a noticeable decrease in FPS of the game. We also reduce the quality of some images (reducing the resolution) which also slightly increased performance in terms of FPS gain.

5.7 Maintainability and Support

Some of the assets had to be changed as they could not be loaded successfully on all operating systems. Sounds files and images were swapped for different ones for better compatibility across multiple operating systems.

6 Automated Testing

The team had agreed to perform automated testing on pytest, but the automated testing actually ended up with unit test. The team initially had only one automated testing on loading assets measuring the loading time, but in Revision 1, more automated tests were added to test the FPS and the response time. The input response time cannot be tested by automated testing only since the testing need user input. Therefore the team made the testing manual and automated by printing test assertions in the game. However, due to the nature of the game, it would be easier to detect faulty behaviour visually in contrast to inspecting test suites, and therefore there are few automated tests comparing to manual and exploratory testing.

7 Trace to Requirements

Table 2: Traceability Between Test Cases and Functional Requirements

Functional Requirement ID	**Test Cases**
1	test-UI5
2	test-UI6
3	test-GM1, test-GM2, test-GM3
4	test-GM4, test-GM5
5	test-UI7
6	test-UI11
7	test-UI11
8	test-UI7
9	test-GE1
10	test-GM9
11	test-GM8
12	test-GM6, test-GM7
13	test-GM7
14	test-GE3
15	test-UI9, test-UI10
16	test-UI1, test-UI2, test-UI4
17	test-GE2

Table 3: Traceability Between Test Cases and Non-Functional Requirements

Non-Functional Requirement	**Test Cases**
NFR 3.1.2 LF1	test-LF1
NFR 3.2.1 UH1	test-UH1
NFR 3.2.2 UH2	test-UH2
NFR 3.3.1 PR2	test-PF1, test-PF2
NFR 3.3.1 PR1	test-PF3, test-PF4
NFR 3.5.2 MS1	test-MS1

8 Trace to Modules

Table 4: Traceability Between Test cases and Modules

Test Cases	**Modules**
test-UI1	M11, M13, M14, M17
test-UI2	M1, M2, M3, M4, M5, M6, M8, M9, M10, M11, M12, M13, M15, M16
test-UI3	M11, M14, M16, M17
test-UI4	M11, M13, M14, M17
test-UI5	M1, M11, M12, M13, M14, M15
test-UI6	M1, M11, M12, M13, M14, M15
test-UI7	all modules were used for this test
test-UI8	all modules were used for this test
test-UI9	all modules were used for this test
test-UI10	M11, M14, M15, M17
test-UI11	all modules were used for this test
test-UI12	M14, M15, M16, M17
test-GE1	M5, M6, M10, M11, M14, M16, M17
test-GE2	M11, M14, M15, M17
test-GE3	all modules were used for this test
test-GE4	all modules were used for this test
test-GM1	M1, M3, M4, M5, M8, M10, M11, M12, M15
test-GM2	M1, M3, M4, M5, M8, M10, M11, M12, M15
test-GM3	M1, M3, M4, M5, M8, M10, M11, M12, M15
test-GM4	M3, M5, M8, M10, M11, M15, M16
test-GM5	M3, M5, M8, M10, M11, M15, M16
test-GM6	M2, M5, M9, M10, M11, M15, M16
test-GM7	M1, M2, M4, M5, M6, M9, M10, M11, M12, M15, M16
test-GM8	M2, M5, M9, M10, M11, M15, M16
test-GM9	M1, M2, M3, M4, M5, M6, M8, M9, M10, M12, M14, M15, M16, M17
test-LF1	M1, M2, M3, M4, M5, M6, M8, M9, M10, M12, M14, M15, M16, M17
test-UH1	M1, M2, M3, M4, M5, M6, M8, M9, M10, M12, M14, M15, M16, M17
test-UH2	M11, M13, M14, M15, M16, M17
test-PF1	all modules were used for this test
test-PF2	all modules were used for this test
test-PF3	all modules were used for this test
test-PF4	M15
test-PF5	all modules were used for this test
test-MS1	all modules were used for this test

9 Code Coverage Metrics

Coverage Metrics are not used to evaluate the testing for T-Rex Acceleration. This is due in part to the fact that the game does not have enough automated testing component for coverage metrics. But upon all the test results from the manual testing and the traceability between the modules and the test cases, the team builds a confidence in reliability, robustness, and consistency of the game.

10 Appendix

Additional information of the document.

10.1 Symbolic Parameters

The definition of the requirements calls for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Table 5: Symbolic Parameter Table

Symbolic Parameter	**Description**	**Value**
GIVEN_OPTIONS	Changes we can make in settings menu	Background Colour, Background Music, Sound effect
PIXEL_JUMP	The height of the dinosaur when it jumps.	20
PIXEL_DUCK	The height of the dinosaur when it ducks.	10
RATE	The "gravity" of the game at which the dinosaur is moving up and down the y-axis at a constant rate	1
KEYBOARD_UP	Keyboard key that moves the onscreen vertically up.	Up Arrow
KEYBOARD_DOWN	Keyboard key that moves the onscreen character duck.	Down Arrow
KEYBOARD_P	Keyboard key that pauses the game when the game is running.	P
KEYBOARD_R	Keyboard key that restarts the game in the end menu or resumes the game in pause menu.	R
KEYBOARD_Q	Keyboard key that quit the game in the main and end menu.	Q
KEYBOARD_S	Keyboard key that accesses the settings menu.	S
KEYBOARD_SPACE	Keyboard key to start playing the game from the main menu.	SPACE
KEYBOARD_B	Keyboard key that gets the user back to the main menu in the setting/instruction menu.	B
KEYBOARD_H	Keyboard key that shows the instruction in the main menu.	H
INSTRUCTION_DISPLAY_TIME	The time that instructions appears on the screen.	4 second
SPECIFIED_KEYS	All the keys we used in the game system.	UP Arrow, Down Arrow P, B, R, Q, S, SPACE, H
MINIMUM_SPAWN_TIME	The minimum time between when one object is spawned, the subsequent object should be spawned at a minimum time after its previous object was spawned.	2 seconds
RESPONSE_TIME	The maximum time delay between when a user provides a keyboard input, the system shall recognize this input within a certain time frame.	5 milliseconds
LOAD_ASSETS_TIME	The maximum time delay for loading all assets.	5 seconds
FPS_GOAL	The desired FPS of the game.	45
OPERATING_SYSTEM	The list of operating systems.	Windows, MAC OS Ubuntu
MIN_AGE	Children younger than this age may have difficulty playing the game	8 years old
RESUME_TIME_DELAY	Time between the resume input and the game actually resuming	3 seconds