

# SE 3XA3: Test Plan T-Rex Acceleration

Team 15, Dev<sup>enthusiasts</sup>

Zihao Du (duz12)

Andrew Balmakund (balmakua)

Namit Chopra (choprn9)

April 9, 2021

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>General Information</b>                                 | <b>1</b>  |
| 1.1      | Purpose . . . . .  | 1         |
| 1.2      | Scope . . . . .  | 1         |
| 1.3      | Acronyms, Abbreviations, and Symbols . . . . .             | 1         |
| 1.4      | Overview of Document . . . . .                             | 2         |
| <b>2</b> | <b>Plan</b>  | <b>2</b>  |
| 2.1      | Software Description . . . . .                             | 2         |
| 2.2      | Test Team . . . . .  | 2         |
| 2.3      | Automated Testing Approach . . . . .                       | 2         |
| 2.4      | Testing Tools . . . . .                                    | 2         |
| 2.5      | Testing Schedule . . . . .                                 | 2         |
| <b>3</b> | <b>System Test Description</b>                             | <b>3</b>  |
| 3.1      | Tests for Functional Requirements . . . . .                | 3         |
| 3.1.1    | User Input . . . . .                                       | 3         |
| 3.1.2    | Game Environment . . . . .                                 | 8         |
| 3.1.3    | Game Mechanics . . . . .                                   | 10        |
| 3.2      | Tests for Nonfunctional Requirements . . . . .             | 13        |
| 3.2.1    | Look and Feel . . . . .                                    | 13        |
| 3.2.2    | Usability . . . . .  | 14        |
| 3.2.3    | Performance . . . . .                                      | 14        |
| 3.2.4    | Maintainability and Support . . . . .                      | 16        |
| 3.3      | Traceability Between Test Cases and Requirements . . . . . | 17        |
| <b>4</b> | <b>Tests for Proof of Concept</b>                          | <b>18</b> |
| <b>5</b> | <b>Comparison to Existing Implementation</b>               | <b>19</b> |
| <b>6</b> | <b>Unit Testing Plan</b>                                   | <b>19</b> |
| 6.1      | Unit testing of internal functions . . . . .               | 19        |
| 6.2      | Unit testing of output files . . . . .                     | 19        |
| <b>7</b> | <b>Appendix</b>  | <b>20</b> |
| 7.1      | Symbolic Parameters . . . . .                              | 20        |
| 7.2      | Usability Survey Questions? . . . . .                      | 22        |
| 7.3      | Figures . . . . .  | 22        |

## List of Tables

|   |   |    |
|---|---|----|
| 1 | <b>Revision History</b>   | ii |
| 2 | <b>Table of Abbreviations</b>                                   | 1  |
| 3 | <b>Table of Definitions</b>                                     | 1  |
| 4 | Traceability Between Test Cases and Functional Requirements     | 17 |
| 5 | Traceability Between Test Cases and Non-Functional Requirements | 17 |
| 6 | <b>Symbolic Parameter Table</b>                                 | 21 |

## List of Figures

|   |   |    |
|---|---|----|
| 1 | Visual representation of platform in the game.                            | 22 |
| 2 | Visual representation of character in the game.                           | 22 |
| 3 | Visual representation of a kind of power-ups (Invincibility) in the game. | 22 |
| 4 | Visual representation of an obstacles in the game.                        | 23 |

Table 1: **Revision History**

| Date       | Version | Notes              |
|------------|---------|--------------------|
| 02/23/2021 | 1.0     | Initial Draft      |
| 03/04/2021 | 1.1     | Finish First Draft |
| 04/07/2021 | 2.0     | Revision 1         |

This document outlines and describes the plan for testing the implementation of the game T-Rex Acceleration.

# 1 General Information

## 1.1 Purpose

The purpose of this document is to provide a means to ensure that all requirements specified in the Software Requirements Document are met by the system being built.

## 1.2 Scope

This document outlines a plan for all the test cases related to the project, including tests for requirements, proof of concept, and the unit testing plan.

## 1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

| Abbreviation | Definition  |
|--------------|---|
| FPS          | Frames Per Second; used to refer to the number of images, called frames, drawn on the screen per second |

Table 3: **Table of Definitions**

| Term         | Definition  |
|--------------|---|
| Pytest       | A testing framework based on python   |
| T-Rex Runner | A game that can be played on Google Chrome when the user is disconnected from the internet. |
| Game state   | Refers to the current game session the user is playing in.                                  |
| Platform     | Ground level for the Dino character and obstacles to be travelling on                       |

## 1.4 Overview of Document

This document will cover the test plan for the T-Rex Acceleration game, including a brief plan overview of how testing will be conducted, system test description for functional requirements, non-functional requirements, traceability between test cases and requirements, and unit test planning.

# 2 Plan

## 2.1 Software Description

This project is a re-development of the Google game T-Rex Runner with additional features.

## 2.2 Test Team

The test team will consist of all developers involved with T-Rex Acceleration: Zihao Du, Andrew Balmakund, and Namit Chopra.

## 2.3 Automated Testing Approach

Since T-Rex Acceleration is an endless runner game, a large part of testing would involve the user's involvement. It will be costly to make a large number of automated test cases for the entire system. Therefore, minimal automated testing will be used and manual testing is used for the majority of the project development.

## 2.4 Testing Tools

The testing tool used for this project is Pytest. Pytest is a python testing framework that is easy to use and write automated test cases. And we want to achieve least 100% branch coverage after all unit, integration and system testing.

## 2.5 Testing Schedule

See Gantt Chart at the following [url](#).

## 3 System Test Description

### 3.1 Tests for Functional Requirements

#### 3.1.1 User Input

##### Mouse Input

1. ~~test-UI1: 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 clicks on “Play” button.~~

~~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 clicked.~~

2. ~~test-UI2: 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 clicks on “Quit” button.~~

~~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 clicked.~~

3. test-UI3**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.

#### 4. ~~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 clicks on the “Settings” option.~~

~~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 clicked.~~

## 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.

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.

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.

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.

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.

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.

7. test-UI8: Test game resume

Type: Functional, Dynamic, Manual

Initial State: The game is paused.

Input: The KEYBOARD\_Ⓟ 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\_Ⓟ 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.

#### 8. test-UI9: Test end menu restart

Type: Functional, Dynamic, Manual

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

Input: The KEYBOARD\_Ⓟ 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\_Ⓟ is pressed, the end menu disappears. The current game session ends and the user goes back 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\_Ⓡ will be pressed once on the user's keyboard.

Output: ~~A new game should start after the key is pressed.~~ 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\_Ⓡ is pressed the end menu disappears and the user enters a new game. KEYBOARD\_Ⓡ is pressed the game window closes 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 player selects “Quit” button in the pause menu.

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

How test will be performed: A visual test is used to verify that when the ‘Quit” button is selected, the current game session ends. The user goes back to 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.

### 3.1.2 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.

2. test-GE2: Test game instructions

Type: Functional, Dynamic, Manual

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

Input: ~~The “Play” is pressed to start the game.~~ KEYBOARD\_H is pressed to show “How to play” menu.

Output: ~~The controls to play the game are displayed on the screen for INSTRUCTION\_DISPLAY\_TIME.~~ 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 for INSTRUCTION\_DISPLAY\_TIME.~~ 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.

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.

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.

### 3.1.3 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.

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.

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.

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.

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.

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.

#### 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.

#### 8. test-GM8: Test power-ups spawn

Type: Functional, Dynamic, ~~Manual~~ Automated

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

Input: ~~Random power-ups spawning~~ A second powerup is spawned at a random time delay after the first powerup spawn time.

Output: ~~Random order of power-ups generated on the platform moving to the left.~~ The second powerup should spawn at MINIMUM.SPAWN.TIME.

How test will be performed: ~~A visual test will be used to confirm the spawning of different power-ups occurs in random order.~~ A automated test is used to confirm that the powerup spawn randomly that is at least the MINIMUM\_SPAWN\_TIME.

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.

## 3.2 Tests for Nonfunctional Requirements

### 3.2.1 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.



### 3.2.2 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. (see Section 7.2)  
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.
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.

### 3.2.3 Performance

1. test-PF1: Test average FPS for a regular game session  
  
Type: Functional, Dynamic, ~~Manual~~ 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.

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

Type: Functional, Dynamic, ~~Manual~~ 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.

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.

4. test-PF4: Test loading in visual assets

Type: Functional, Dynamic, ~~Manual~~ 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.

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.

### 3.2.4 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.

### 3.3 Traceability Between Test Cases and Requirements

Table 4: 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 5: 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           |

## 4 Tests for Proof of Concept

### Character Movement

1. test-C1: Test all character movement

Type: Functional, Dynamic, Manual

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

Input: The player uses `KEYBOARD_UP` to jump or `KEYBOARD_DOWN` to duck.

Output: The character performs a jump action or duck action depending on the corresponding keyboard button pressed.

How test will be performed: A visual test will be used to verify that the jump and duck movement of the character is working correctly.

### Game Mechanics

1. test-G1: Test obstacle collision

Type: Functional, Dynamic, Manual

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

Input: The player is controlling the character's movement.

Output: The game session will end if the user collides with the obstacle.

How test will be performed: A visual test will be used to verify that if a player is unable to successfully avoid the obstacle (jumps too early or too late), a collision will occur.

### Game Environment

1. test-E1: Test background changing

Type: Functional, Dynamic, Manual

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

Input: None.

Output: The background colour changes.

How test will be performed: A visual test will be used to verify that the colour of the background changes after a period of time.

## 5 Comparison to Existing Implementation

N/A The original implementation will be used to test various properties of the game, to give a better understanding of how playable the redevelopment is with respect to the original game. For example, testing for the minimum and maximum time of obstacles spawning, various points of collision detection between the character and obstacle, the difficulty of the game as the score increases (how manageable is it to play), and the similarity in character movements.

## 6 Unit Testing Plan

The project contains many modules that are highly coupled. This makes it difficult to create unit test cases for specific components. As the application does not produce any files, unit testing for output files is not needed.

### 6.1 Unit testing of internal functions

N/A For unit testing, many unit test will be done manually such as DisplayCharacter, DisplayPowerups, DisplayObstacles, etc. For automated tests cases such as LoadAssets, making sure each assets is able to load in properly (i.e not throw an exception). In addition for automated test cases for unit testing, they should have 100% code coverage.

### 6.2 Unit testing of output files

N/A Since the application does not create any output files, there is no need to test for output files. Even though the application uses image and sound assets as external/output files, this is tested in 6.1

## **7 Appendix**

Additional information of the document.

### **7.1 Symbolic Parameters**

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

Table 6: Symbolic Parameter Table

| Symbolic Parameter       | Description   | Value   |
|--------------------------|---|---|
| GIVEN_OPTIONS            | Changes we can make in settings menu  | Background Colour,<br>Background Music,<br>Sound effect |
| PIXEL_JUMP               | The height of the dinosaur when it jumps.   | UNDEFINED20   |
| PIXEL_DUCK               | The height of the dinosaur when it ducks.   | UNDEFINED10   |
| RATE                     | The “gravity” of the game at which the dinosaur is moving up and down the y-axis at a constant rate   | UNDEFINED1  |
| 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/resumes the game when the game is running/paused.  | 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 quits the game in the end menu 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<br>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<br>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   |



## 7.2 Usability Survey Questions?

Questions for user's to rate the usability and ease of play of the game. From a scale from 1-10, with 10 being the best/most, users give a rating for each question based on experience.

1. How easy are the controls to understand? (3.2.2 Usability: test-UH1)
2. How comfortable are you with the controls? (3.2.2 Usability: test-UH1)
3. How helpful are the instructions at the start of the game? (3.2.2 Usability: test-UH1)

## 7.3 Figures



Figure 1: Visual representation of platform in the game.



Figure 2: Visual representation of character in the game.



Figure 3: Visual representation of a kind of power-ups (Invincibility) in the game.

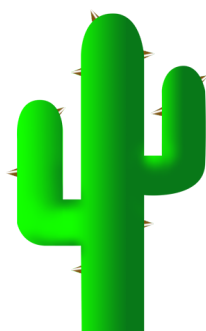


Figure 4: Visual representation of **an** obstacles in the game.