SE 3XA3: Test Report
Super Refactored Mario Bros.

203, Abstract Connoisseurs
Alexander Samaha, samahaa
Dan Noorduyn, noorduyd
David Jandric, jandricd

April 6, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
April 6, 2020	1.0	Finished Test Reporting

This document provides an overview of the testing done for the implementation of Super Mario Refactored Bros. 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.

1 Functional Requirements Evaluation

The Function Requirements tests below are from the test plan released earlier. You can view the test plan here.

1.1 User Input

1. Standard 180 degree plane movement

   - **test-UI1**: Test Standard Left Movement

     Type: Functional, Dynamic, Manual

     Initial State: Mario is standing still on the ground level.

     Input: The left-arrow will be pressed and held on the user's keyboard.

     Output: Mario moves left, along the 180 degree line, at speed 1.

     How test will be performed: A visual test will testify that Mario moves left, along the 180 degree line, at speed 1 while the left-arrow is pressed

   - **test-UI2**: Test Standard Right Movement

     Type: Functional, Dynamic, Manual

     Initial State: Mario is standing still on the ground level.

     Input: The right-arrow will be pressed and held on the user's keyboard.

     Output: Mario moves right, along the 0 degree line, at speed 1.

How test will be performed: A visual test will confirm that Mario moves right at speed 1, along the 0 degree line, while the right-arrow is pressed and held.

Result: PASS. When the right-arrow/left-arrow button is pressed on a User's keyboard, Mario moves right/left as expected, unless Mario runs into an obstacle and can no longer proceed. This is the expected behaviour.

2. **test-UI3**: Test Standard Jump Movement

   Type: Functional, Dynamic, Manual

   Initial State: Mario is standing still on the ground level.

   Input: The up-arrow/spacebar will be pressed on the user's keyboard.

   Output: Mario moves up, along the 90 degree line, at speed -12, until he reaches height 120. Mario then falls back to the ground level at the speed of gravity.

   How test will be performed: A visual test will confirm that when the up-arrow/spacebar is pressed, Mario jumps up at speed -12, along the 90 degree line, until height Y, before falling back down to ground level at the speed of gravity.

   Result: PASS. When the up-arrow/spacebar button is pressed on a User's keyboard. Mario performs a jump movement up to a maximum of height Y.

3. **test-UI4**: Test Duck Movement

   Test was removed from Revision 1.

4. Sprint Tests

   - **test-UI5**: Test Sprinting Right Movement

     Type: Functional, Dynamic, Manual

     Initial State: Mario is standing still on the ground level.

     Input: The right-arrow and the shift-key will be pressed and held simultaneously on the user's keyboard.

Output: Mario moves right, along the 0 degree line, at speed 1 + 5.

How test will be performed: A visual test will confirm that Mario moves right faster than normal, at speed 1+5, along the 0 degree line, while the right-arrow and the shift-key is pressed and held.

- **test-UI6**: Test Sprinting left Movement

  Type: Functional, Dynamic, Manual

  Initial State: Mario is standing still on the ground level.

  Input: The left-arrow and the shift-key will be pressed and held simultaneously on the user's keyboard.

  Output: Mario moves right, along the 180 degree line, at speed 1 + 5.

  How test will be performed: A visual test will confirm that Mario moves right faster than normal, at speed 1+5, along the 180 degree line, while the left-arrow and the shift-key is pressed and held.

Result: PASS. When the right-arrow/left-arrow and the shift-key buttons are pressed and held simultaneously. Mario moves left/right faster than normal, at speed 1+5, until the character reaches an obstacle.

5. Jumping and holding directional keys simultaneously.

   - **test-UI7**: Test Jumping Up and Right Movement

     Type: Functional, Dynamic, Manual

     Initial State: Mario is standing still on the ground level.

     Input: The right-arrow will be pressed and held and the up-arrow/spacebar will be pressed simultaneously on the user's keyboard.

     Output: Mario moves right and upwards, at a vertical speed (90 degree line) of -12 and a horizontal speed (0 degree line) of 1, resulting in a direction between the 0 and 90 degree lines.

     How test will be performed: A visual test will confirm that Mario moves upwards and right, at a vertical speed (90 degree line) of -12 and a horizontal speed (0 degree line) of 1, while simultaneously the right-arrow is pressed and held and the up-arrow/spacebar is pressed.

- **test-UI8**: Test Jumping Up and Left Movement

  Type: Functional, Dynamic, Manual

  Initial State: Mario is standing still on the ground level.

  Input: The left-arrow will be pressed and held and the up-arrow/spacebar will be pressed simultaneously on the user's keyboard.

  Output: Mario moves left and upwards, at a vertical speed (90 degree line) of -12 and a horizontal speed (180 degree line) of 1, resulting in a direction between the 90 and 1800 degree lines.

  How test will be performed: A visual test will confirm that Mario moves upwards and left, at a vertical speed (90 degree line) of -12 and a horizontal speed (180 degree line) of 1, while simultaneously the left-arrow is pressed and held and the up-arrow/spacebar pressed.

Result: PASS. When the left-arrow/right-arrow and the up-arrow/spacebar button are pressed simultaneously, Mario jumps vertically while leaning right/left at a vertical speed (90 degrees) of -12 and a horizontal speed of 1. This results in Mario having a horizontal and vertical displacement

6. Falling and holding directional keys simultaneously.

   - **test-UI9**: Test Standard Right Falling Movement

     Type: Functional, Dynamic, Manual

     Initial State: Mario is standing still on a platform above the ground level (see Figure 4 for a visual representation).

     Input: The right-arrow is pressed and held on the user's keyboard.

     Output: Mario moves right along the 0 degree line at a speed of 1, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move right, resulting in a direction between 270 degrees and 0 degrees, until he lands on the ground level.

How test will be performed: A visual test will confirm that when the right-arrow is pressed and held, Mario moves right along the 0 degree line at a speed of 1, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move right, resulting in a direction between 270 degrees and 0 degrees, until he lands on the ground level.

- **test-UI10**: Test Standard Left Falling Movement

  Type: Functional, Dynamic, Manual

  Initial State: Mario is standing still on a platform above the ground level (see Figure 4 for a visual representation).

  Input: The left-arrow is pressed and held on the user's keyboard.

  Output: Mario moves left along the 180 degree line at a speed of 1, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move left, resulting in a direction between 180 degrees and 270 degrees, until he lands on the ground level.

  How test will be performed: A visual test will confirm that when the left-arrow is pressed and held, Mario moves left along the 180 degree line at a speed of 1, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move left, resulting in a direction between 180 degrees and 270 degrees, until he lands on the ground level.

Result: PASS. When the left-arrow/right-arrow button is pressed while Mario falls off a platform parallel with the 180 degree axis, he is displaced between the left/right side of the 270 degree axis with gravity acting on the character.

7. Sprinting and falling off a platform while holding directional keys simultaneously.

   - **test-UI11**: Test Sprinting Left Falling Movement

     Type: Functional, Dynamic, Manual

Initial State: Mario is standing still on a platform above the ground level (see Figure 4 for a visual representation).

Input: The left-arrow and shift-key are pressed and held on the user's keyboard.

Output: Mario moves quickly left along the 180 degree line at a speed of 1 + 5, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move left, resulting in a direction between 180 degrees and 270 degrees, until he lands on the ground level.

How test will be performed: A visual test will confirm that when the left-arrow and shift-key is pressed and held, Mario moves quickly left along the 180 degree line at a speed of 1 + 5, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move left, resulting in a direction between 180 degrees and 270 degrees, until he lands on the ground level.

- **test-UI12**: Test Sprinting Right Falling Movement

  Type: Functional, Dynamic, Manual

  Initial State: Mario is standing still on a platform above the ground level (see Figure 4 for a visual representation).

  Input: The right-arrow and shift-key are pressed and held on the user's keyboard.

  Output: Mario moves quickly right along the 0 degree line at a speed of 1 + 5, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move right, resulting in a direction between 270 degrees and 0 degrees, until he lands on the ground level.

  How test will be performed: A visual test will confirm that when the right-arrow is pressed, Mario moves quickly right along the 0 degree line at a speed of 1 + 5, until he reaches the edge of the platform. Mario then falls downwards along the 270 degree line at the speed of gravity while also continuing to move right, resulting in a direction between 270 degrees and 0 degrees, until he lands on the ground level.

Result: PASS. Result: PASS. When the left-arrow/right-arrow and the shift-key button is pressed and held until Mario falls off a platform at speed 1+5 parallel with the 180 degree axis, he is displaced between the left/right side of the 270 degree axis with gravity acting on the character.

8. **test-UI13**: Test Duck Overriding Other Controls

   Test was removed from Revision 1.

9. **test-UI14**: Test Game Pause

   Type: Functional, Dynamic, Manual

   Initial State: Any point during game play: Mario is standing still; Mario is jumping; Mario is falling.

   Input: The escape-key is pressed down on the user's keyboard.

   Output: The game-play freezes (nothing in the game-environment moves) and a pop-up window will display the paused menu.

   How test will be performed: A visual test will confirm that when the escape-key is pressed, the game-play freezes and a pop-up window displays the paused menu.

   Result: PASS. All testers in the team manually confirmed that when the escape-key button is pressed inside levels 1 to 4 (Demonstrated in the Revision 1 demo), the game level is immobilized and a blur appears along with a pop-up menu. This menu displays a continue game option, or exit game option.

10. **test-UI15**: Test Select Input

   Type: Functional, Dynamic, Manual

   Initial State: Home screen of the game with the focus on the button 'Levels'.

   Input: The return-key is pressed down on the user's keyboard.

Output: The main window changes from the home screen to display the level selection menu.

How test will be performed: A visual test will confirm that when the return-key is pressed the main window displays the levels selection menu instead of the home screen.

Result: PASS. All testers in the team manually confirmed that from the level selector display, any available level can be picked using directional arrows, and upon pressing the return-key button, a level is selected and loaded.

11. **test-UI16**: Test Program Robustness

    Type: Structural, Dynamic, Manual

    Initial State: Mario is standing still on the ground level.

    Input: One of the developers will play the game simulating a random user's interactions. This means all the viable input keys will be pressed in a unique combination and some will be held down for a long time.

    Output: The game responds accordingly to each input in the sequence corresponding to the keys being pressed.

    How test will be performed: A visual test will confirm that when the input keys are pressed, the game responds accordingly and in the correct sequence.

    Result: PASS. All of the members of the test team manually confirmed that pressing directional keys or up-arrow/spacebar/shift-key buttons activate in sequential order. All members attempted to press the left/right movement keys at the same time, whichever button was initiated on the physical keyboard first occurred in game. This was done for combinations of directional keys, with the up-arrow/spacebar and shift-key buttons.

12. **test-UI17**: Test Illegal Double Jump

    Type: Functional, Dynamic, Manual

    Initial State: Mario is standing still on the ground level.

Input: The up-arrow/spacebar is pressed one time on the user's keyboard. After Mario leaves the ground level, The up-arrow/spacebar is pressed once more.

Output: Mario jumps upwards (degree line 90) at speed -12, reaches height Y, and then falls at the speed of gravity till he lands on the ground level.

How test will be performed: A visual test will confirm that when the up-arrow/spacebar is pressed one time, and again after Mario leaves the ground level, that Mario jumps upwards (degree line 90) at speed -12, reaches height Y, and then falls at the speed of gravity till he lands on the ground level.

Result: PASS. There are boolean variables to determine if a character is in the air. During an attempt, when the up-arrow/spacebar is pressed the in air variable was printed to the screen. Mario was unable to jump until the in air boolean variable returned to false.

test-UI18: Test Illegal Run Past Camera Left

Type: Functional, Dynamic, Manual

Initial State: Mario has moved right 20 ground blocks which causes the camera to also move to the right. After this movement Mario is standing still on the ground level.

Input: The left-arrow is pressed and held on the user's keyboard.

Output: Mario moves left (degree line 180) at speed 1, reaches the edge of the camera, and then stops with 1 equal to zero (though the sprinting animation continues). The camera also doesn't move to the left.

How test will be performed: A visual test will confirm that when the The left-arrow is pressed and held on the user's keyboard, and all the initial state conditions are met, that Mario moves left (degree line 180) at speed 1, reaches the edge of the camera, and then stops with 1 equal to zero. It will also be visually confirmed that the camera doesn't move.

Result: PASS. All testers played each level and attempted to run past the leftmost point of the game window. It was visually confirmed that Mario cannot run past this point and the game camera no longer scrolls once the character moves left.

1.2 Game Environment

1. **test-GE1**: Test Proper Game Load

   Type: Dynamic, Manual

   Initial State: A running computer with Macintosh, Windows or Linux operating system.

   Input: The executable file for Super-Refactored-Mario-Bros is double clicked by the users right mouse button.

   Output: The game's code is loaded by the CPU's integrated graphics processor and the all the levels, characters, and enemies will be created.

   How test will be performed: An early version of the game will be created with one simplistic level that can be tested. An initial visual test will confirm that when the game is loaded, the correct level and the corresponding details such as platforms and coins are all created properly. After this is successful, a secondary visual test will be run. One by one, different types of enemies will be added to the level and the game will be reloaded with each new enemy type. Each time, there will be a visual confirmation that the new enemy type has be correctly generated.

   Result: PASS. All Developers created one level each. Each level was developed sequentially, each asset was added individually and visually checked by rerunning the game. After development, the full level was tested to ensure all actual features of the level were expected from the level file.

2. **test-GE2**: Test Fault JSON File Handling

   Type: Dynamic, Manual, Functional

   Initial State: A running computer with Macintosh, Windows or Linux operating system and a purposeful faulty JSON file in a load location.

   Input: The executable file for Super-Refactored-Mario-Bros is double clicked by the users right mouse button.

   Output: The game's code tries to be loaded by the CPU's integrated graphics processor but because of the faulty JSON file it doesn't complete its loading. A pop-up message will inform the user of the issue.

How test will be performed: An early version of the game will be created with one simplistic level that can be tested. An initial visual test will confirm that when the game is trying to load up, a pop-up message appears that informs the user of the fault JSON file.

Result: PASS. A working level was modified and faulty code was added. The game continues to run, however, when an attempt is made to load the faulty level, a message appears on the user's terminal alerting them of an error in the level code and the game safely terminates.
Note: this is not the expected behaviour from the test case, however, the team agrees this is a safe alternative.

1.3 Game Mechanics

1. **test-GM1**: Test Killing Standard Enemies

   Type: Functional, Dynamic, Manual

   Initial State: Mario is standing still on the ground level with an enemy near.

   Input: Mario is controlled by the input keys to jump on top of the enemy.

   Output: When Mario lands on top of the enemy, the enemy will die and disappear from the level. The score of the user will be increased accordingly.

   How test will be performed: A visual test will confirm that when Mario lands on top of the enemy, the enemy dies and disappears from the level and the user's score increased accordingly.

   Result: PASS. When Mario lands on top of a Goomba enemy, the enemy alive state variable is printed to the screen and manually confirmed to show it is updating to a death animation. There was visual confirmation that the death animation was performed and the enemy disappears from the level, the appropriate points are awarded for this action.

2. **test-GM2**: Test Killing 'Koopas'

Type: Functional, Dynamic, Manual

Initial State: Mario is standing still on the ground level with a Koopa near.

Input: Mario is controlled by the input keys to jump on top of the Koopa.

Output: When Mario lands on top of the Koopa, the Koopa will retract into its shell and Mario will be left standing on top of the shell. The user's score will be increased accordingly (see **Figure 5** for a visual representation)

How test will be performed: A visual test will confirm that when Mario lands on top of the Koopa, the Koopa will retract into its shell and Mario will be left standing on top of the shell. Further, the user's score shall be confirmed as increasing accordingly.

Result: PASS. When Mario lands on top of a Koopa enemy, the enemy alive state variable is printed to the screen and manually confirmed to show it is updating to a hidden shell animation. There was visual confirmation that the hidden shell animation was performed and the enemy is immobile, the appropriate points are awarded for this action.

3. **test-GM3**: Test Koopa's Shell Movement

   Type: Functional, Dynamic, Manual

   Initial State: Mario is standing still on the ground level with a Koopa's shell motionless on his right, also on the ground level .

   Input: Mario is controlled by the input keys (right-arrow is held down on the user's keyboard) to contact the Koopa's shell.

   Output: When Mario hits the Koopa's shell, the shell will begin moving to the right on the ground level. If it comes into contact with a wall or pipe, the shell will bounce off it and move back in the direction it came from.

   How test will be performed: A visual test will confirm that when Mario moves right and contacts the Koopa's shell, the shell begins to move to the right. Further, if the shell contacts a wall or pipe, it will be visually confirmed that is bounces off and moves back in the direction it came from.

Result: PASS. When Mario lands on top of a Koopa enemy, the enemy alive state variable is printed to the screen and manually confirmed to show it is updating to a spinning shell animation. There was visual confirmation that the spinning shell animation was performed and the enemy is moving in the direction opposite to the side it was hit, the appropriate points are awarded for this action.

4. **test-GM4**: Test Koopa's Shell Killing

   Type: Functional, Dynamic, Manual

   Initial State: There is a Koopa's shell motionless on the ground level to the right of Mario. To the right of the shell are two enemies and then further along a wall.

   Input: Mario is controlled with the inputs from a user's keyboard to move right to hit the shell.

   Output: When the shell is hit, it moves right and collides with the first enemy who dies and disappears. The shell continues right and hits the second enemy, who also dies and disappears. The shell continues right until it hits the pipe and bounces off, now moving left towards Mario. The shell hits Mario and Mario dies.

   How test will be performed: A visual test will confirm that when Mario moves right and hits the shell, the shell will move right, killing both enemies, before bouncing off the pipe and hitting Mario, who also dies.

   Result: PASS. The enemy's alive state variables were printed to the screen and there is manual confirmation that these variables are changed and the character is updating to its death animation. There is visual confirmation that the enemy is hit and dies.

5. **test-GM5**: Test Player's Character's General Death

   Type: Functional, Dynamic, Manual

   Initial State: Mario is standing still on the ground level with an enemy near (Mario is small - no power-up was achieved).

   Input: Mario is controlled to move right (right-arrow is held down on user's keyboard) and collides with an enemy.

Output: When Mario collides with the enemy, he dies and is sent to the beginning of the level which has been reset (all enemies are regenerated).

How test will be performed: A visual test will confirm that when Mario moves right and collides with the enemy, he dies and is sent back to the beginning of the level, which is also visually confirmed to be reset.

Result: PASS. A manual test was performed printing Mario's alive state variable. It is confirmed that when Mario hits an enemy character while in a small Mario state, the death procedure is performed and the game level is restarted.

6. **test-GM6**: Test Player's Character's lives

   Type: Functional, Dynamic, Manual

   Initial State: Mario is standing still on the ground level with an enemy near (Mario is small - no power-up was achieved). The player has 3 of their lives left.

   Input: Mario is controlled to move right (right-arrow is held down on user's keyboard).

   Output: When Mario collides with the enemy he dies and one of the player's three lives is lost (visually represented by mushrooms at the top of the screen).

   How test will be performed: A visual test will confirm that when Mario moves right and collides with the enemy, he dies and one of the player's lives is lost.

   Result: PASS. A manual test was performed to confirm that when Mario dies, the level is restarted and a life is removed from the top of the screen.

7. **test-GM7**: Test Player's Character's Final Death

   Type: Functional, Dynamic, Manual

   Initial State: Mario is standing still on the ground level with an enemy near (Mario is small - no power-up was achieved). The player has one of their three lives left.

Input: Mario is controlled to move right (right-arrow is held down on user's keyboard).

Output: When Mario collides with the enemy he dies and the level is exited. The player is then returned to the main menu and their high-score is saved.

How test will be performed: A visual test will confirm that when Mario moves right and collides with the enemy, he dies and the level exits, returning the player to the main menu.

Result: PASS. A manual test was performed at Mario's final life. When Mario collided with an enemy, the usual death procedure is played, however, an extra sequence is displayed afterwards which alerts the player they are out of lives. The game resets to the menu screen and records the high-score.

8. **test-GM8**: Test Beating Level

   Type: Functional, Dynamic, Manual

   Initial State: A user has completed an entire level and Mario is currently standing still on the ground level next to the end of level.

   Input: Mario is controlled to move right (right-arrow is held down on user's keyboard).

   Output: When Mario collides with the end of level, the time will be converted into points and the player will be transported to the start of the next level.

   How test will be performed: A visual test will confirm that when Mario moves right and collides with the end of level, the time will be converted into points and the player is transported to the start of the next level.

   Result: PASS. A Manual test was performed where Mario reaches the end of the level. The game is immobilized and the User's current time left is converted into points. the game proceeds to the next level.

9. **test-GM9**: Test Player Beating Game

   Type: Functional, Dynamic, Manual

Initial State: Every level has been completed in sequence and Mario is standing still on the ground level next to the end of level of the final level.

Input: Mario is controlled to move right (right-arrow is held down on user's keyboard).

Output: When Mario collides with the end of level, the player will be transported the main menu. The player's high-score will be saved.

How test will be performed: A visual test will confirm that when Mario moves right and collides with the end of level, the player will be transported the main menu. It will be further observed that the player's high-score is entered into the database.

Result: PASS. All testers played through the entire level suite, once reached the final level, a visual confirmation is made that the game performs the usual end of level animation, however the game redirects the User to the main screen and their high-score is recorded.

10. **test-GM10**: Test Mushroom Power-up Appear

    Type: Functional, Dynamic, Manual

    Initial State: Mario is standing beneath a mushroom power-up box (see Figure 3 for a visual).

    Input: Mario is controlled to jump up and collide with the bottom of the mushroom power-up box (up-arrow/spacebar is pressed on user's keyboard).

    Output: When Mario collides with the bottom of the mushroom power-up box, a mushroom pops up out of the mushroom power-up box, and moves left or right, resulting in it falling down towards the ground.

    How test will be performed: A visual test will confirm that when Mario is controlled to jump up and collide with the bottom of the mushroom power-up box a mushroom pops up out of the mushroom power-up, and moves left or right, resulting in it falling down towards the ground.

    Result: PASS. A manual test was performed 10 times where the game level files were modified to add the power-up box intentionally in. Each time Mario collides with the box from the bottom, the direction the mushroom headed was recorded visually. The mushroom moved 6 times to the left and 4 times to the right.

11. **test-GM11**: Test Coin From Random-Box Appear

    Type: Functional, Dynamic, Manual

    Initial State: Mario is standing beneath a random box (see Figure 3 for a visual).

    Input: Mario is controlled to jump up and collide with the bottom of the random box (up-arrow/spacebar is pressed on user's keyboard).

    Output: When Mario collides with the bottom of the random box, a coin pops up out of the random box, revolves and then disappears, resulting in an increase in score.

    How test will be performed: A visual test will confirm that when Mario is controlled to jump up and collide with the bottom of the random box a coin pops up out of the random box, revolves and then disappears, resulting in an increase in score.

    Result: PASS. A visual test was performed on each level. Each time Mario collides with the Random-box a coin is collected and it is visually confirmed that it was added to the coin bank and points bank.

12. **test-GM12**: Test Coin Collision

    Type: Functional, Dynamic, Manual

    Initial State: Mario is standing to the left of a coin.

    Input: The right-arrow key is pressed and held.

    Output: When Mario collides with the coin, it disappears resulting in an increase in score.

    How test will be performed: A visual test will confirm that when the right-arrow key is pressed and held and Mario collides with the coin, it disappears resulting in an increase in score.

    Result: PASS. The tester loaded a standard level with coins embedded. A visual test confirmed the coin disappears upon collision and the 100 points are added to the point counter.

13. **test-GM13**: Test Mushroom Collision

    Type: Functional, Dynamic, Manual

    Initial State: Mario is standing to the left of a mushroom.

    Input: The right-arrow key is pressed and held.

    Output: When Mario collides with the mushroom, it disappears resulting in an increase in score and small Mario becomes big Mario.

    How test will be performed: A visual test will confirm that when the right-arrow key is pressed and held and Mario collides with the mushroom, it disappears resulting in an increase in score and small Mario becoming big Mario.

    Result: PASS. The tester loaded a standard level with a power-up box embedded. Upon colision with the mushroom, a visual test was confirmed that the mushroom disappears and the 100 points are added to the player's score count.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

1. **test-LF1**: Test movement is similar to original game

   Type: Dynamic, Manual, Functional

   Initial State: The modified game is launched and an emulator version of the original game is loaded.

   Input/Condition: The tester will make all available movements (Right-Left, Jump, Crouch) in the modified game and the original game emulator.

   Output/Result: PASS. The tester will determine if both versions of the game have a similar feel through visual confirmation.

   How test will be performed: The tester will first play an original version of the game through an emulator service and go through the initial level. The tester will then play our version of the game and finish its initial level. A test that PASS.es will have visual confirmation that the two games are similar in movement feel.

Result: PASS. A visual test was performed, both versions of the game are loaded and each movement in the modified game is repeated in the emulated game.

2. **test-LF2**: Test ability to navigate through game menu.

   Type: Dynamic, Manual, Functional

   Initial State: Terminal or file explorer screen of Operating System.

   Input: Tester will boot the game and reach the menu screen.

   Output: Tester will navigate to a level and boot the level.

   Result: PASS. All testers were instructed to load the first level of the game. It was manually tested that each user was able to load the first level and continue playing the game.

2.2 Performance

1. **test-PF1**: Test the game runs at FRAMES_PER_SECOND.

   Type: Dynamic, Manual, Structural

   Initial State: The tester has launched the game.

   Input/Condition: The tester will navigate the menu and start a level.

   Output/Result: PASS. The tester will finish the level and exit the game.

   How test will be performed: The tester will start the game and play through one level with an FPS counter such as FRAPS recording the gameplay, then will exit the game. The test will PASS. if after the recording is finished, the FPS does not drop lower than FRAMES_PER_SECOND for more than FPS_MARGIN tolerates.

   Result: PASS. the test was performed instead by printing the framerate to the terminal. It was visually confirmed that the game ran FRAMES_PER_SECOND and did not drop below that number for more than FPS_MARGIN.

2. **test-PF2**: Test the game does not slow down under stress.

   Type: Dynamic, Manual, Structural

   Initial State: The tester has launched the game.

   Input/Condition: The tester will navigate the menu and start a level. Tester will then add ENTITIES to the level.

   Output/Result: PASS. The tester will finish the level and exit the game.

   How test will be performed: The tester will start the game and play through one level with an FPS counter such as FRAPS recording the gameplay, then will exit the game. The test will PASS. if after the recording is finished, the FPS does not drop lower than FRAMES_PER_SECOND for FPS_MARGIN.

   Result: PASS. The tester created a game file with a few less than ENTITIES. This simulates having to load and update many entities. The game was monitored by printing the FPS when it dropped below FRAMES_PER_SECOND and printing the time elapsed since it returned.

3. **test-PF3**: Test the game can run for at least GAME_AVAILABILITY.

   Type: Dynamic, Manual, Structural

   Initial State: The tester has launched the game.

   Input/Condition: The tester will navigate the menu and start a level.

   Output/Result: PASS. The tester will finish the game after GAME_AVAILABILITY elapsed.

   How test will be performed: The tester will start the game and play through one level with a stopwatch recording time elapsed, then will exit the game. The test will PASS. if after GAME_AVAILABILITY has passed, the game has not crashed and is still in a stable state.

   Result: PASS. The game was tested manually, a timer was set to GAME_AVAILABILITY and the game was left alone. The game has an internal countdown that ends the level gracefully before the GAME_AVAILABILITY has elapsed. The game is stable and did not crash.

3 Comparison to Existing Implementation

The existing implementation had issues in the way it was modularized. Most of the modules were highly coupled and depended on each other to an extent that it was difficult to modify and import errors would appear often. As such, the new version of the game features a more defined architecture. We followed a Main/Subroutine architecture where we maintained the original main program from the existing implementation. Further, features were brought together into files and modules were removed if they did not have a secret worth separating. Certain features needed to be moved from a module to accommodate for this lower coupling and avoid import errors. The second most important change to the original implementation were the features that were missing. The team added new sprites, soundbytes and gameplay to the existing implementation. These include the life system, the highscore system, the level timer, power-ups, fixed enemy interaction and an end of level system. These new features often took a lower priority than the original goal of refactoring the current implementation of code. It was important to focus on this aspect to make it easier to add new features in the future. This newer implementation is much easier to understand architecturally and is a higher quality training tool for those looking to learn pygame and pyhthon in general. If you would like to learn more about the modularization of the new implementation, you can view the Module Guide here and the Module Interface Specification here.

4 Unit Testing

The team made the decision to forego unit testing due to the higher cost with little reward in this particular context. Unit testing provides a minor amount of confidence or further information when testing such a visual product. Our team instead opted for a suite of manual and visual tests to ensure the game was behaving properly. This is consistent with how the games industry tests their products, usually through thorough manual testing and exploratory testing.

5 Changes Due to Testing

5.1 User Interaction

No changes were necessary made due to testing.

5.2 Game Environment

Out team made some changes for when a faulty JSON file is detected. The game now safely exits if the faulty file is loaded. This is deemed to be a safe alternative as the game should not attempt to continue if the game files are corrupted in any way. Initially, we wished to alert the user that the JSON file is faulty and continue running the game.

5.3 Game Mechanics

No changes were necessary due to testing

5.4 Look and Feel

No changes were necessary due to testing

5.5 Performance

No changes were necessary due to testing

6 Automated Testing

The team had initially agreed to perform automated testing on modules with pyTest. It was later assessed that this testing would not be feasible. This is due to the highly visual nature of the game. The time investment would be better spent performing exploratory testing and manual testing which would assess the behaviour of the game when a User interacts with it. It would also be easier to detect faulty behaviour visually in contrast to inspecting coordinate values through a unit test suite.

7 Trace to Requirements

Table 2: **Requirements Traceability**

Requirement ID	**Test ID(s)**
FR1 - FR5, PR4	UI1 - UI3, UI5 - UI12, UI17, UI18
FR6 - FR9	GM1, GM2, GM3, GM4
FR10, FR11, FR12, FR17, FR18,	GM5, GM6, GM7
FR13	GM8, GM9
FR15, FR16	GM10, GM13, GM14
FR19	GM11, GM12
LF1	LF1, UI14, UI15
LF2	LF2
PR1, PR3, PR7 - PR9	PF1, PF2
PR6, PR9	PF3

8 Trace to Modules

Click here to see the latest Module Guide, which will provide a breakdown of all the modules in the game.

Tests	Modules
UI1	M1, M2, M6, M13, M19, M25, M27, M30
UI2	M1, M2, M6, M13, M19, M25, M27, M30
UI3	M1, M2, M6, M10, M13, M19, M25, M27, M28, M30
UI5	M1, M2, M6, M13, M19, M25, M27, M30
UI6	M1, M2, M6, M13, M19, M25, M27, M30
UI7	M1, M2, M6, M10, M13, M19, M25, M27, M28, M30
UI8	M1, M2, M6, M10, M13, M19, M25, M27, M28, M30
UI9	M1, M2, M6, M10, M13, M19, M25, M27, M30
UI10	M1, M2, M6, M10, M13, M19, M25, M27, M30
UI11	M1, M2, M6, M10, M13, M19, M25, M27, M30
UI12	M1, M2, M6, M10, M13, M19, M25, M27, M30
UI14	M1, M2, M4, M5, M6, M7, M8, M9, M11, M19, M30
UI15	M1, M2, M3, M14
UI16	every module is tested in this test
UI17	M1, M2, M6, M10, M13, M19, M25, M27, M28, M30
UI18	M1, M2, M6, M7, M13, M19, M25, M27, M30
GE1	all modules are tested in this test
GE2	M1, M3, M11
GM1	M1, M2, M4, M6, M7, M8, M10, M13, M18, M27, M28, M29
GM2	M1, M2, M4, M6, M7, M9, M10, M13, M18, M27, M28, M29

GM3	M1, M2, M4, M6, M7, M9, M10, M13, M18, M27, M28, M29
GM4	M1, M2, M4, M6, M7, M8, M9, M10, M13, M18, M27, M28, M29
GM5	M1, M2, M4, M6, M7, M8, M10, M13, M18, M27, M29
GM6	M1, M2, M4, M6, M7, M8, M10, M13, M18, M27, M29
GM7	M1, M2, M4, M6, M7, M8, M10, M13, M18, M27, M29
GM8	M1, M2, M4, M6, M7, M11, M13, M14
GM9	M1, M2, M3, M4, M6, M7, M11, M13, M14
GM10	M1, M2, M4, M6, M11, M18, M23, M24, M29
GM11	M1, M2, M4, M6, M11, M18, M21, M22
GM12	M1, M2, M4, M6, M11, M18, M20
GM13	M1, M2, M4, M6, M11, M18, M23, M29
LF1	M1, M2, M6, M10, M13, M19, M25, M27, M28, M30
LF2	M1, M2, M3, M14
PF1	tests all modules
PF2	tests all modules
PF3	tests all modules

Table 3: Trace Between Test Cases and Modules

9 Code Coverage Metrics

The team decided against using code coverage metrics to evaluate the Super Refactored Mario Bros. game. This is due in part to the fact that the game does not have any automated testing component, and no stub was created to purely invoke branches. Without a stub, it is infeasible to accurately estimate code coverage of the program. However, Upon examination of the manual tests and the traceability matrix between modules, it's clear that the important functionalities are tested The Abstract Connoisseurs team feels confident in the reliability and consistency of the game.

9.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in the table below.

Table 4: **Symbolic Parameter Table**

Symbolic Parameter	**Description**	**Value**
ENTITIES	Required Number of Koops Entities for stress test.	10 seconds
FRAMES_PER_SECOND	Framerate in Hz of the game.	60Hz
FPS_MARGIN	Time required spent at framerate or above through use of product.	95%
GAME_AVAILABILITY	Minimum time required of game to run continuously.	1 hour