SE 3XA3: Test Plan
Super Refactored Mario Bros.

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

April 6, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Feb 28, 2020	1.0	Revision 0
April 6, 2020	2.0	Revision 1

This document will give an overview of the testing that will be done for the implementation of Super Refactored Mario Python throughout development.

1 General Information

1.1 Purpose

The purpose of this document is to provide a plan for verifying that the product satisfies the requirements through testing.

1.2 Scope

This document outlines the tests that will be done for testing the functional and non-functional requirements and proof of concept, as well as unit tests. This document also details the associated software products to achieve those goals.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirements Specification document.
JSON	JavaScript Object Notation, a readable file format to transmit data objects.
GUI	Graphical User Interface.
FPS	Frames Per Second calculated in Hertz.

1.4 Overview of Document

This document will walk through the test plan for Super Mario Refactored Python, which includes a plan for how the testing will be conducted, a system test description which goes over the tests for functional and non-functional requirements, a plan for testing the proof of concept, a unit testing plan and an appendix.

Table 3: **Table of Definitions**

Term	**Definition**
Mario	The in-game player controlled character
Emulator	A software product that ports a game originally intended for another system.
Koopa	A non-playable adversary in the Super Mario Bros. game.
Enemy	Refers to either of the Koopa or Goomba non-playable adversaries.
PyTest	Python unit testing suite.
PyGame	Python simple game development environment.

2 Plan

2.1 Software Description

This software is an extended re-implementation of a Super Mario Bros. clone, written in Python.

2.2 Test Team

The individuals on the test team are David Jandric, Dan Noorduyn, and Alexander Samaha.

2.3 Automated Testing Approach

~~We will use PyTest to test the core functionality of the project, excluding any GUI elements (which will be tested manually). Some examples of what could be tested automatically would be: the physics, level parsing, and collision. Automatic testing will ensure that the major functions of the project work correctly, and can be done quickly and efficiently. The automated testing will cover each function in each module separately. Testing functions that require user input will be done by automating the input.~~ The overwhelming majority of our project's functionality consists of user interaction. Therefore we determined that it would be costly to create a suite of unit tests of all the base logic. Instead, our testing technique will be a manual test suite that ensures the correct responses to the user's input. If each test passes it is implied that the underlying logical functionality of our program corresponds properly.

2.4 Testing Tools

The testing for this project will be done with PyTest, which is a testing suite framework for Python projects, which deals with unit testing and code coverage.

2.5 Testing Schedule

See Gantt Chart at the following url ...
[Link to Gantt chart and Resource chart]

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

Please observe the **Figures** section in the Appendix at the end of this paper for a better understanding of the player controlled Mario character's location in the game world and a definition of directions.

Keyboard Movement Inputs

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

   Type: Functional, Dynamic, Manual

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

   Input: The KEYBOARD_LEFT will be pressed and held on the user's keyboard.

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

   How test will be performed: A visual test will testify that Mario moves left, along the 180 degree line, at speed V while the KEYBOARD_LEFT is pressed and held.

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

   Type: Functional, Dynamic, Manual

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

   Input: The KEYBOARD_RIGHT will be pressed and held on the user's keyboard.

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

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

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

   Type: Functional, Dynamic, Manual

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

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

   Output: Mario moves up, along the 90 degree line, at speed C, until he reaches height Y. 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 KEYBOARD_JUMP is pressed, Mario jumps up at speed C, along the 90 degree line, until height Y, before falling back down to ground level at the speed of gravity.

4. ~~**test-UI4**: Test Duck Movement~~
   ~~Type: Functional, Dynamic, Manual Initial State: Mario is standing still on the ground level. Inp~~

3.1.2 Game Environment

15. **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.

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.

3.1.3 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.

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.

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 (KEYBOARD_RIGHT 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.

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.

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 (KEYBOARD_RIGHT 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.

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 (KEYBOARD_RIGHT 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 ~~hearts~~ 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.

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 (KEYBOARD_RIGHT 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.

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 ~~flagpole~~ end of level.

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

Output: When Mario collides with the ~~flagpole~~ end of level, ~~there will be a message stating they have completed the 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 ~~flagpole~~ end of level, ~~a congratulatory message appears~~ the time will be converted into points and the player is transported to the start of 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 ~~flagpole~~ end of level of the final level.

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

   Output: When Mario collides with the ~~flagpole~~ end of level, ~~there will be a "You Win" screen shown and~~ 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 ~~flagpole~~ end of level, ~~a "You Win" screen will be shown and~~ the player will be transported the main menu. It will be further observed that the player's high-score is entered into the database.

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 (KEYBOARD_UP 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.

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 (KEYBOARD_UP 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.

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

    Type: Functional, Dynamic, Manual

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

Input: The KEYBOARD_RIGHT 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 KEYBOARD_RIGHT key is pressed and held and Mario collides with the coin, it disappears resulting in an increase in score.

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

    Type: Functional, Dynamic, Manual

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

    Input: The KEYBOARD_RIGHT 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 KEYBOARD_RIGHT 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.

14. **test-GM14**: Test Big Mario to Small Mario

    Type: Functional, Dynamic, Manual

    Initial State: Mario is standing still on the ground level with an enemy near (Mario is big - the mushroom power-up was achieved).

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

    Output: When Mario collides with the enemy, he turns back into small Mario, and is invincible for a split second.

    How test will be performed: A visual test will confirm that when Mario moves right and collides with the enemy, he turns back into small Mario, and is invincible for a split second.

3.2 Tests for Nonfunctional Requirements

3.2.1 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: 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 passes will have visual confirmation that the two games are similar in movement feel.

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.

3.2.2 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: 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.

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 100 Koopa characters to the level.

   Output/Result: 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.

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: 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.

3.3 Traceability Between Test Cases and Requirements

The following table details which test cases trace to which requirements (the ID's of the requirements are taken from the Requirements Document found here).

Table 4: **Requirements Traceability**

Requirement ID	**Requirement Description**	**Test ID(s)**
FR1 - FR5, PR4	These requirements relate to movement of the player character	UI1 - UI3, UI5 - ~~UI13~~ UI12
FR6 - FR9	These requirements detail how enemies can and should die.	GM1, GM2, GM3, GM4
FR10, FR11, FR12	These functional requirements detail how the players character can die.	GM5, GM6, GM7
FR13, FR14	These functional requirements detail how the player can beat levels and collectively the game.	GM8, GM9
FR15, FR16	These functional requirements details the mushroom power-up	GM10, GM13, GM14
FR17, FR18	These functional requirements details the collection of coins	GM11, GM12
LF1	This requirement relates to the feel of the game being similar to the original.	LF1
LF2	This requirement relates to the menu being modern and easy to navigate.	LF2
PR1, PR3, PR7 - PR9	These requirement relate to game performance when under load.	PF1, PF2
PR6, PR9	These requirements ensure the game can handle the overhead of one player.	PF3

4 Tests for Proof of Concept

4.1 Movement

During the Proof of Concept demonstration, our group demonstrated gameplay by testing a base level included in the original release. We showed how a character moves on screen in response to user input and also showed how alterations to the game files are reflected in the game.

Test that the game responds to user input.

1. **test-M1**

Type: Functional, Dynamic, Manual

Initial State: The game will be run and a base level will be loaded.

Input: Use the keyboard input KEYBOARD_LEFT, KEYBOARD_RIGHT and KEYBOARD_JUMP to move the onscreen character.

Output: The onscreen player character will be displaced according to user input.

How test will be performed: The user will push the keyboard input for the left, right and jump movements and visually confirm that the onscreen character has moved on the screen for the test to pass.

4.2 Level design

Test response of changes to level.

1. **test-LD1**

   Type: Functional, Dynamic, Manual

   Initial State: The game will be run and a base level will be loaded.

   Input: Use the game level files to modify the world length for a level to TEST_LEVEL_LENGTH.

   Output: The onscreen character will move past the end of the level from the base game.

   How test will be performed: By duplicating the game data for a base level and modifying the level length to one, a comparison can be made between both levels. The test passes if the character is able to move past the end of level area of the base game level.

5 Comparison to Existing Implementation

N/A

6 Unit Testing Plan

~~Super Refactored Mario Python will use PyTest to evaluate the functionality of each module and subsystem.~~

6.1 Unit testing of internal functions

~~The project will feature unit testing files for every module created. This will help ensure the behaviour of each function is traceable to the requirements specified in the SRS. The type of testing will be dynamic and functional in nature and will involve a white box approach. Code coverage will be tracked using a PyTest plugin called PyTest-cov. The aim of unit testing in this project is to ensure a minimum of CODE_COVERAGE_VALUE and to instill confidence in the correctness of the game logic. Further, this test suite can be easily extended as new modules are created and set up to support automated regression testing for each future iteration of the game.~~

6.2 Unit testing of output files

The game does not create output files, however it does take in JSON formatted files for level design and settings options. We will use visual confirmation methods to determine that faulty JSON files do not crash the game or cause any unwanted behaviour. The game features a graphic user interface using PyGame. This will have to be tested manually and dynamically in a black-box fashion. As such, the development team will play the game and test user interactions with the game according to the functional requirements specified in the SRS. Visual confirmation that the behaviour of the game matches the intended behaviour will signify a pass for the test.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

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

Table 5: **Symbolic Parameter Table**

Symbolic Parameter	**Description**	**Value**
KEYBOARD_LEFT	Keyboard key that moves the onscreen character sideways to the left.	Left Arrow
KEYBOARD_RIGHT	Keyboard key that moves the onscreen character sideways to the right.	Right Arrow
KEYBOARD_JUMP	Keyboard key that moves the onscreen character vertically upwards.	Up Arrow
KEYBOARD_CROUCH	Keyboard key that crouches the on-screen character.	Down Arrow
KEYBOARD_SPRINT	Keyboard key that makes the onscreen character sprint.	Shift
KEYBOARD_ENTER	Keyboard key that selects an option in a menu.	Enter
KEYBOARD_EXIT	Keyboard key that pauses a game mid-level.	ESC
TEST_LEVEL_LENGTH	The exact length of a game level.	60
CODE_COVERAGE_VALUE	The test coverage goal expressed in percentage.	100%
TIME_TO_NAVIGATE	Time to navigate through a menu to reach an action.	30 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

7.2 Usability Survey Questions?

Usability is defined by ISO9241-11 as

"the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use".

As such, we want to poll users on their experience playing the game, particularly in the areas of effectiveness and satisfaction. This will help determine if we are reaching our goal of providing a better way to play the classic Super Mario Game.

1. Do the controls of the game make sense?
2. Was your experience bug free? if not, where did you encounter issues?
3. How much time do you usually play the game in one sitting? Please answer; Under 10 minutes, under 30 minutes, under 1 hour, more than 1 hour.
4. Was the game language easy to understand?
5. Did you enjoy your time with the game? Please rate from 1 (did not enjoy) to 5 (very much enjoyed).
6. Would you recommend this game to a friend?
7. Would you come back to this game instead of using a Super Mario Bros. emulator?

7.3 Figures

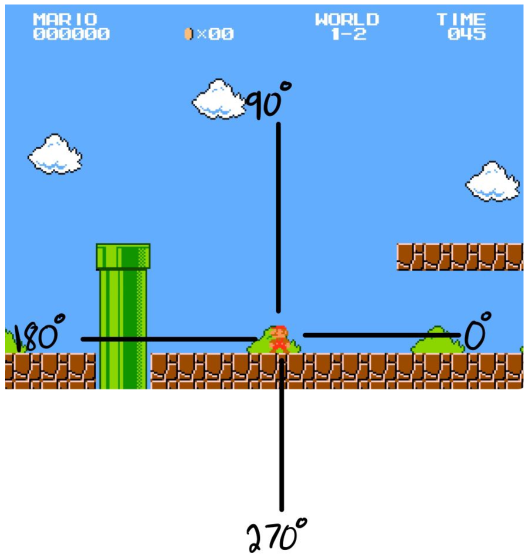

Figure 1: Visual representation of the directions of movement with respect to Mario.

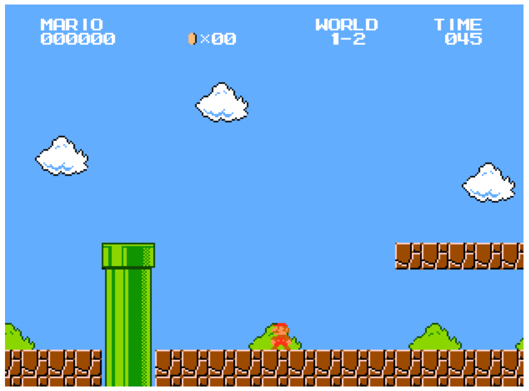

Figure 2: Visual representation of Mario on the ground level of the game.

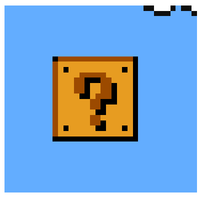

Figure 3: Visual representation of ~~Mario standing and then ducking.~~ a mushroom power-up box and random box.

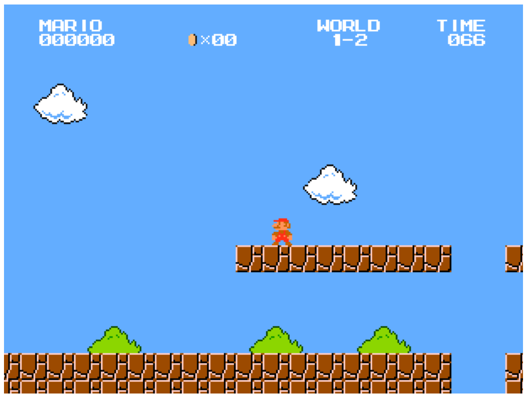

Figure 4: Visual representation of Mario on a platform in the game.

Figure 5: Visual representation of a Koopa and a Koopa retracted into its shell.