TEST DOCUMENTATION

Project: Safari Paths (Blue Whales)

Team: s240347, s240904, s240826, s240767, s241008

Version: 1.0 (Final Build)

Date: June 2025

Document Owner: Philip (Test Lead)

Table of Contents

1. Introduction
2. 1.1 Purpose
3. 1.2 Summary
4. 1.3 Definitions and Abbreviations (Glossary)
5. 1.4 References and Standards
6. 1.5 Overview
7. TEST STRATEGY & PLAN
8. 2.1 Testing Objectives
9. 2.2 Test Scope
10. 2.2.1 In Scope
11. 2.2.2 Out of Scope
12. 2.3 Test Environments
13. 2.4 Testing Tools & Frameworks
14. TEST CATEGORIES & PROCEDURES
15. 3.1 Unit Testing
16. 3.1.1 Puzzle Logic Tests
17. 3.1.2 GameManager State Tests
18. 3.2 Integration Testing
19. 3.2.1 Scene Transition Integration
20. 3.2.2 Audio System Integration
21. 3.3 System Testing
22. 3.3.1 End-to-End Gameplay Flow
23. 3.3.2 Stress Testing
24. 3.4 User Interface Testing
25. 3.4.1 UI Consistency & Style Guide Compliance
26. 3.4.2 Interactive Element Functionality
27. 3.5 Performance Testing
28. 3.5.1 Response Time Requirements
29. 3.5.2 Resource Usage Requirements
30. ERROR HANDLING & RECOVERY TESTING
31. 4.1 Defect Log (Real Bugs & Resolutions)
32. 4.2 Error Scenarios & Testing
33. 4.2.1 Missing Node Reference (ER-001)
34. 4.2.2 Missing Audio File (ER-002)
35. 4.2.3 Incorrect Asset Path (ER-003)
36. TEST REPORTING & METRICS
37. 5.1 Test Execution Reporting
38. 5.2 Test Metrics & KPIs
39. 5.3 Bug Classification & Tracking
40. REQUIREMENTS TRACEABILITY MATRIX
41. 6.1 Requirements Coverage Verification
42. TEST ENVIRONMENT SETUP

43. 7.1 Environment Configuration
44. 7.2 Test Data Management
45. APPENDICES
46. 8.1 Appendix A: Test Case Details (Expanded)
47. 8.2 Appendix B: Testing Scripts and Automation
48. 8.3 Appendix C: Performance Baseline Data
49. 8.4 Appendix D: Test Environment Setup Guide

1.0 Introduction

This document details the Testing Documentation for the "Safari Paths" game, developed by the Blue Whales team. It outlines the comprehensive testing strategy, procedures, and results used to validate the game's functionality, performance, and adherence to requirements. This document also incorporates specific bugs identified and resolved during the development process, providing a transparent record of quality assurance efforts.

1.1 Purpose

The primary purpose of this Test Documentation is to describe the systematic approach taken to ensure the quality and reliability of the Safari Paths game. It serves as a reference for all testing activities, from test planning and execution to defect tracking and reporting.

- **Who created the document and how?** This document was primarily authored by Philip (Test Lead) with significant contributions from the entire Blue Whales development team. It was developed iteratively, adapting to the evolving codebase and incorporating lessons learned during debugging and integration phases.
- **Who should read this document?** This document is intended for project stakeholders including academic evaluators, future maintenance teams, and anyone seeking a detailed understanding of the game's testing coverage and quality status.
- **Who is bound by this document (scope of use)?** All team members involved in development, quality assurance, and project management are bound by the testing strategy and procedures outlined herein. It defines the minimum standard for validating the game's quality.

1.2 Summary

Safari Paths is an educational 2D puzzle game built with Godot 4.4 for children aged 1-5. This document details the multi-faceted testing approach, including Unit, Integration, System, UI, and Performance testing. It covers the validation of core puzzle logic, scene transitions, global state management (GameManager.gd), and user interface consistency. A critical focus is placed on error handling and recovery, with specific examples from resolved bugs (e.g., "null instance" errors, audio conflicts) incorporated into the defect log. The document outlines test environments, tools, key metrics, and a traceability matrix to ensure comprehensive coverage and quality.

1.3 Definitions and Abbreviations (Glossary)

(Refer to Section 1.3 of the Requirements Documentation and Section 1.3 of the Architectural Documentation for a comprehensive glossary.)

- **Acceptance Criteria:** Specific, measurable conditions that must be met for a feature, component, or the entire system to be accepted.
- **Autoload:** A Godot feature to load a script or scene globally at startup, making it a singleton, like GameManager.gd.

- **Defect Log:** A formal record of identified bugs, their status, severity, and resolution details.
- **Exploratory Testing:** A simultaneous learning, test design, and test execution process.
- **Graceful Degradation:** The ability of a system to maintain functionality, possibly at a reduced level, when faced with errors or resource limitations, rather than crashing.
- **KPI (Key Performance Indicator):** Measurable values that demonstrate how effectively a project is achieving key objectives.
- **@onready:** A Godot annotation used to get a node reference when the node and its children are ready in the scene tree. Frequent source of "null instance" errors if paths are incorrect or nodes are missing.
- **Pass Rate:** The percentage of executed test cases that yield expected results.
- **Regression Testing:** Re-running tests after code changes to ensure that new changes have not introduced new bugs or re-introduced old ones.
- **Test Case:** A set of conditions or variables under which a tester will determine if a system under test is working correctly.
- **Test Plan:** A document detailing the scope, objectives, methods, and resources for testing a software product.
- **Traceability Matrix:** A document that links requirements to test cases, design elements, or other project artifacts to ensure comprehensive coverage.
- **UAT (User Acceptance Testing):** Testing conducted by end-users (or representatives of end-users) to confirm the system meets their needs and requirements.

1.4 References and Standards

- **Requirements Documentation (Safari Paths, Summer 2025):** The foundational document detailing functional and non-functional requirements.
- **Architectural Documentation (Safari Paths, Summer 2025):** Provides the system's design and architectural principles.
- **Project Documentation (Safari Paths, Summer 2025):** Outlines the overall project approach, timeline, and lessons learned.
- **SWE_SoSe2025_DELIVERABLES.pdf:** Academic requirements and checklist for project deliverables.
- **Internal Godot Coding Standards:** Team's agreed-upon coding conventions for GDScript, including error handling (e.g., @onready null checks) and naming.
- **UI Style Guide:** (As per team action item from 1st June 2025 progress report) Visual standards for consistent UI elements.

1.5 Overview

This document systematically outlines the testing efforts for Safari Paths. It begins by defining the testing strategy and scope. Section 3 details specific test categories: Unit, Integration, System, UI, and Performance, providing concrete test cases. Section 4 focuses on error handling, including a defect log of real bugs and their resolutions, and scenarios for testing graceful degradation. Sections 5 and 6 cover test reporting, metrics, and requirements traceability. Section 7 describes the test environment setup. The document concludes with appendices containing expanded test details and supporting information.

2.0 TEST STRATEGY & PLAN

2.1 Testing Objectives

- Validate core puzzle logic functionality and game mechanics for Addition, Fruit Sort, Match Letters, and Subtraction tasks.
- Ensure consistent UI state management across all scene transitions (_WelcomePage.tscn -> MonkeyLevel.tscn -> LevelTransition.tscn -> ElephantLevel.tscn -> EndScene.tscn).

- Verify centralized GameManager.gd maintains data integrity (player points, task progress, reset functionality).
- Confirm user interface meets design standards, accessibility requirements, and usability (button responsiveness, visual feedback).
- Validate error handling and system stability, particularly for common runtime issues like null references, incorrect node paths, and resource loading failures.

2.2 Test Scope

2.2.1 In Scope

- Core puzzle gameplay mechanics (Addition, Fruit Sort, Match Letters, Subtraction) logic and user interaction.
- UI/UX interactions and scene transitions (.tscn loading, Button and TextureButton functionality, TextureRect display, Label updates, layout responsiveness, prevention of Cumulative Layout Shifts - CLS).
- GameManager.gd state management (point accumulation, task_points progress tracking, game reset state across restarts).
- Asset loading and rendering (character sprites like monkey_neutral.png, fruit textures, button images, background images).
- Audio system integration (button clicks button-click.mp3, correct/incorrect feedback Girl Saying Excellent.mp3, background music Kalahari_Dreaming.mp3, level complete sounds Girl Saying Let's Do It Again.mp3) and proper stopping/starting (GameManager.gd's responsibility).
- Error handling and graceful degradation for anticipated runtime issues (e.g., @onready path issues, null instance calls to nodes or audio players).

2.2.2 Out of Scope

- Third-party library internals (e.g., Godot Engine's core functionality beyond exposed APIs).
- Platform-specific optimizations beyond core Godot functionality (unless explicitly causing a bug on the primary development platform).
- Network functionality (not applicable for this single-player game).
- Persistent Save/Load game state (beyond the current session; game always resets on restart/quit as per design).
- User account management.
- Advanced graphics or 3D elements.
- External API integrations.

2.3 Test Environments

- **Development Environment:** Godot Engine 4.x, Windows 10 (primary development platform for initial coding and debugging). Also tested on macOS/Linux for basic compatibility.
- **Testing Environment:** A clean build environment on a standard development machine (meeting recommended requirements) is used for dedicated testing sessions to ensure consistency.
- **Integration Environment:** Full asset deployment and a compiled game executable are used for end-to-end testing, replicating final build conditions and ensuring all assets are correctly packaged and referenced.

2.4 Testing Tools & Frameworks

- **Godot Engine's Built-in Debugger:** Used extensively for runtime error analysis, script debugging, and inspecting variable states.

- **Godot Console Output:** Crucial for identifying print() messages, warnings, and error logs (e.g., "Node not found" errors related to @onready paths).
- **Manual Testing Checklists:** Employed for systematic feature verification, regression testing, and ensuring consistent coverage across all builds.
- **Exploratory Testing:** Performed regularly to uncover unexpected behaviors, edge cases, and usability issues that might not be covered by explicit test cases.
- **Internal Defect Log:** A structured record (Section 4.1) used to track identified bugs, their status, priority, and resolution details.
- **Performance Monitoring Tools:** Godot's built-in Profiler (Debugger tab, Monitor section) for real-time FPS, memory, and CPU usage monitoring.

3.0 TEST CATEGORIES & PROCEDURES

3.1 Unit Testing

Focuses on validating individual components in isolation.

3.1.1 Puzzle Logic Tests

- **Test Suite:** Task Mechanics
- **Components:** AdditionTask.gd, FruitSortTask.gd, MatchLettersTask.gd, SubtractionTask.gd

Test ID: UT-001

- **Description:** Validate Addition Task math logic and correct answer handling.
- **Preconditions:** AdditionTask.tscn loaded, game not finished.
- **Test Steps:**
    1. Initialize task. Observe question_label for generated numbers (e.g., "What is 2 + 3?").
    2. Click the button corresponding to the correct sum (e.g., 5).
    3. Verify GameManager.player_points increments by 100.
    4. Repeat steps 2-3 two more times.
    5. Verify question_label updates to "Great job! Addition task complete!" and all buttons disable.
    6. Verify task_completed signal (100, true) emitted for each correct answer and (0, true) for final completion.
- **Expected Result:** Correct sums are identified, points awarded, task completes after 3 correct answers, UI updates as expected.
- **Priority:** High

Test ID: UT-002

- **Description:** Validate Fruit Sort Task correct fruit detection and reshuffle logic.
- **Preconditions:** FruitSortTask.tscn loaded, game not finished.
- **Test Steps:**
    1. Initialize task. Identify a "good" fruit (e.g., Appleg.png).
    2. Click the "good" fruit button.
    3. Verify GameManager.player_points increments by 100.
    4. Verify the picked fruit button disables.
    5. Repeat steps 2-4 two more times for other "good" fruits.
    6. After the third "good" fruit, verify instruction label updates to "Great job! Fruit sorting task complete!" and all buttons disable.
    7. (New Test) Click a "bad" fruit: verify it disables/disappears, and new fruits are generated to replace it, without re-enabling previously picked "good" fruits (related to Bug ID: PD-006 fix).
- **Expected Result:** Good fruits are correctly identified, points awarded, task completes after 3 good fruits. Reshuffle mechanism ensures new good fruits appear if needed, and

only new fruits replace bad picks without affecting prior good picks. UI updates.
- **Priority:** High

Test ID: UT-003

- **Description:** Validate Match Letters Task correct letter-to-color matching and selection logic.
- **Preconditions:** MatchLettersTask.tscn loaded, game not finished.
- **Test Steps:**
  1. Initialize task. Identify a letter (e.g., 'B') and its corresponding color (Blue button).
  2. Click the 'B' letter button. Verify other letter buttons disable, color buttons enable.
  3. Click the Blue color button.
  4. Verify GameManager.player_points increments by 100.
  5. Verify both 'B' and Blue buttons disable permanently and get a blue border.
  6. Repeat for all 5 pairs.
  7. Verify instruction_label updates to "Great job! You matched all!" after 5 matches.
  8. (New Test) Click an incorrect letter, then an incorrect color: verify both buttons briefly show a red border and then reset to their initial unselected state (related to Bug ID: PD-007 fix).
- **Expected Result:** Correct pairs match, points awarded, buttons disable, visual feedback for selection and match works. Incorrect selections reset correctly.
- **Priority:** High

Test ID: UT-004

- **Description:** Validate Subtraction Task math logic and correct answer handling.
- **Preconditions:** SubtractionTask.tscn loaded, game not finished.
- **Test Steps:**
  1. Initialize task. Observe question_label for generated numbers (e.g., "What is 4 - 2?").
  2. Click the button corresponding to the correct difference (e.g., 2).
  3. Verify GameManager.player_points increments by 100.
  4. Repeat steps 2-3 two more times.
  5. Verify question_label updates to "Great job! Subtraction task complete!" and all buttons disable.
- **Expected Result:** Correct differences are identified, points awarded, task completes after 3 correct answers, UI updates.
- **Priority:** High

3.1.2 GameManager State Tests

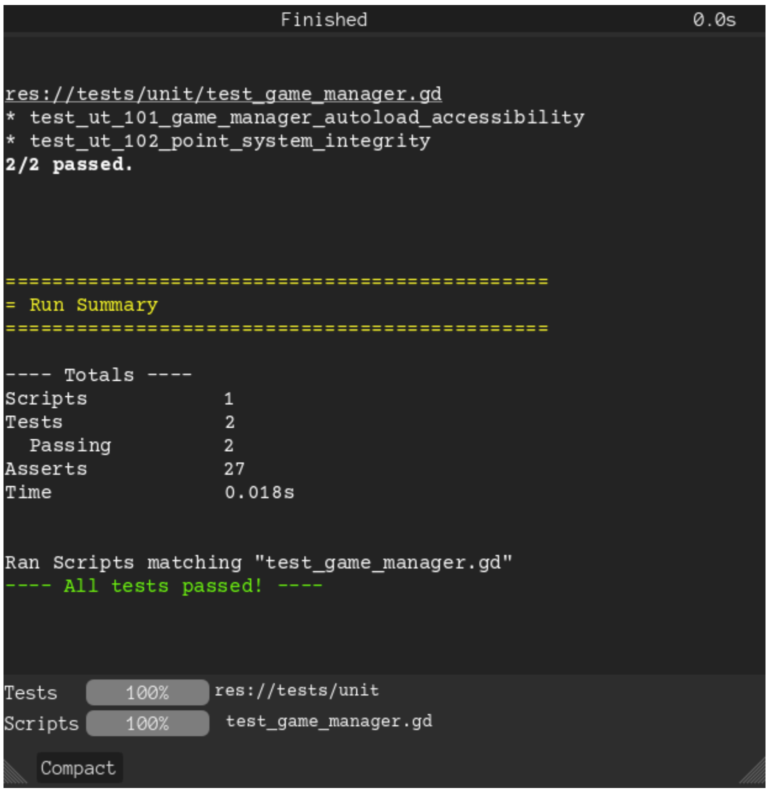
- **Test Suite:** State Management
- **Component:** GameManager.gd (Autoload singleton)

Test ID: UT-101

- **Description:** Verify GameManager.award_points correctly accumulates player points and task points.
- **Preconditions:** GameManager autoloaded and properly configured.
- **Test Steps:**
  1. In _ready() of Monkey_Level.gd, print initial GameManager.player_points and GameManager.task_points["monkey_addition"].
  2. Simulate 3 correct answers for monkey_addition via _on_task_completed(100, true) calls (e.g., manually trigger the signal from a debugger or a test script).
  3. Verify GameManager.player_points becomes 300.
  4. Verify GameManager.task_points["monkey_addition"] becomes 300.
  5. Simulate an incorrect answer _on_task_completed(0, false).

6. Verify GameManager.player_points and GameManager.task_points["monkey_addition"] do not change.

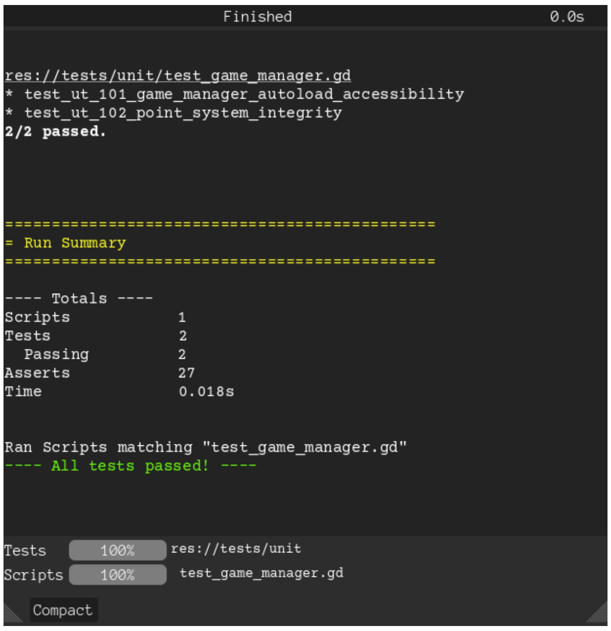
- **Expected Result:** Points correctly accumulate for positive awards, remain unchanged for zero/negative awards.
- **Priority:** High



```
Finished                                              0.0s

res://tests/unit/test_game_manager.gd
* test_ut_101_game_manager_autoload_accessibility
* test_ut_102_point_system_integrity
2/2 passed.



=============================================
= Run Summary
=============================================

---- Totals ----
Scripts             1
Tests               2
  Passing           2
Asserts             27
Time                0.018s


Ran Scripts matching "test_game_manager.gd"
---- All tests passed! ----



Tests    100%  res://tests/unit
Scripts  100%   test_game_manager.gd
```

Test ID: UT-102

- **Description:** Verify GameManager.reset_game_state() correctly resets all game-related variables (player_points and task_points).
- **Preconditions:** Game has accumulated points and task progress (GameManager state is non-zero).
- **Test Steps:**
  1. Play through MonkeyLevel and ElephantLevel to accumulate points (e.g., total 1400 points) and set task progress.
  2. Transition to EndScene.tscn.
  3. Click the "Play Again" (RestartButton).
  4. Game restarts to _WelcomePage.tscn.
  5. Start a new game session (transition to MonkeyLevel.tscn).
  6. In Monkey_Level.gd _ready(), print GameManager.player_points and all GameManager.task_points values.
- **Expected Result:** GameManager.player_points is 0, and all entries in GameManager.task_points are reset to 0.
- **Priority:** Critical



```
Finished                                    0.0s

res://tests/unit/test_game_manager.gd
* test_ut_101_game_manager_autoload_accessibility
* test_ut_102_point_system_integrity
2/2 passed.



==============================================
= Run Summary
==============================================

---- Totals ----
Scripts            1
Tests              2
  Passing          2
Asserts            27
Time               0.018s


Ran Scripts matching "test_game_manager.gd"
---- All tests passed! ----


Tests    100%  res://tests/unit
Scripts  100%  test_game_manager.gd

Compact
```

•

3.2 Integration Testing

Focuses on validating interactions between integrated components.

3.2.1 Scene Transition Integration

- **Test Suite:** Scene Management
- **Components:** _WelcomePage.gd, Monkey_Level.gd, Level_Transition.gd, Elephant_Level.gd, End_Scene.gd, and their corresponding .tscn files.

Test ID: IT-001

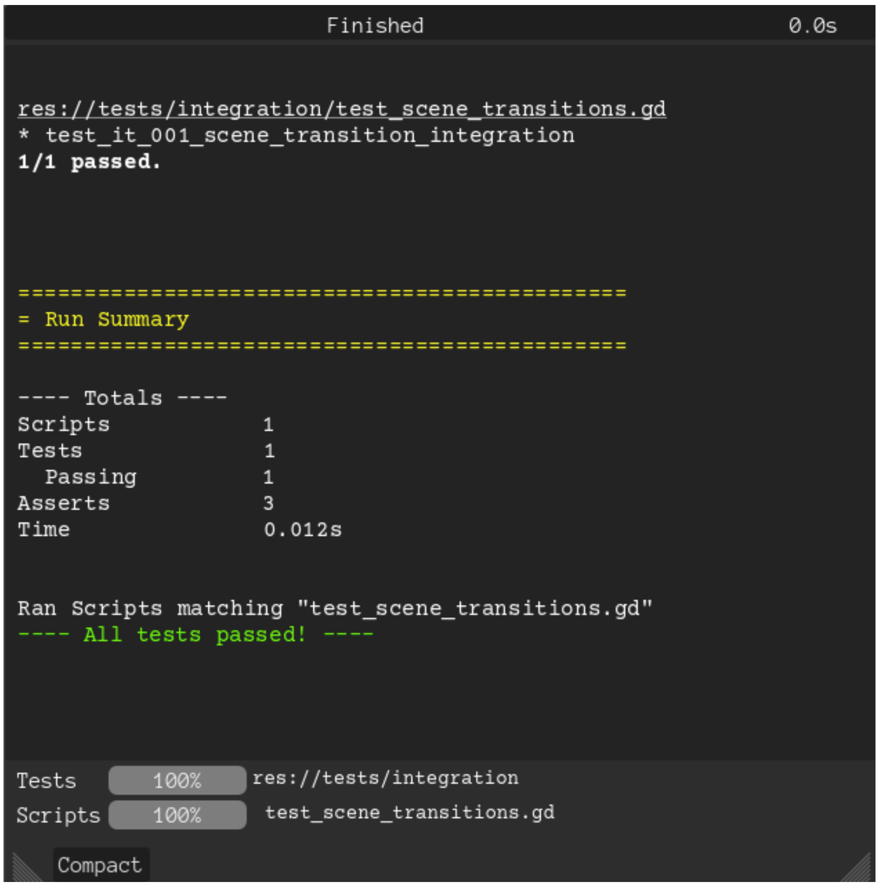
- **Description:** Validate smooth and correct scene transitions (Welcome -> MonkeyLevel -> LevelTransition -> ElephantLevel -> EndScene).
- **Preconditions:** Game launched from _WelcomePage.tscn, all scene paths in scripts are accurate (verified during fixes for Bug ID: PD-002, PD-003, PD-005, PD-010).
- **Test Steps:**
  1. Launch game. Click "Play" on _WelcomePage.tscn.
  2. Complete both tasks in MonkeyLevel.tscn.
  3. Verify transition to LevelTransition.tscn.
  4. Click "Continue" on LevelTransition.tscn.
  5. Verify transition to ElephantLevel.tscn.
  6. Complete both tasks in ElephantLevel.tscn.
  7. Verify transition to EndScene.tscn.
- **Expected Result:** All transitions occur without crashes or "null instance" errors. The correct scene loads each time. The game flow matches the design. Console should show no "Node not found" errors.
- **Priority:** Critical

Test ID: IT-002

- **Description:** Verify UI elements (HUD, character sprites) update correctly across scene transitions and game states.
- **Preconditions:** Game running with active state.
- **Test Steps:**
  1. Start game. Accumulate points in MonkeyLevel (e.g., 200 points).
  2. Verify HUD PointsLabel shows current points.

3. Trigger transition to LevelTransition.tscn.
4. Trigger transition to ElephantLevel.tscn.
5. Verify HUD PointsLabel in ElephantLevel displays the accumulated 200 points.
6. Perform actions in ElephantLevel (correct/incorrect) and verify elephant character sprite changes (elephant_neutral.png, elephant_sad.png).
7. Perform actions in MonkeyLevel (correct/incorrect) and verify monkey character sprite changes.

- **Expected Result:** HUD points persist and update correctly across scenes. Character sprites change appropriately in their respective levels based on correctness.
- **Priority:** High



```
Finished                                        0.0s

res://tests/integration/test_scene_transitions.gd
* test_it_001_scene_transition_integration
1/1 passed.



==============================================
= Run Summary
==============================================

---- Totals ----
Scripts            1
Tests              1
  Passing          1
Asserts            3
Time               0.012s


Ran Scripts matching "test_scene_transitions.gd"
---- All tests passed! ----
```

Tests 100% res://tests/integration
Scripts 100% test_scene_transitions.gd
Compact

3.2.2 Audio System Integration

- **Test Suite:** Audio Management
- **Components:** GameManager.gd, _WelcomePage.gd, Monkey_Level.gd, Level_Transition.gd, Elephant_Level.gd, End_Scene.gd, all task scripts, and associated AudioStreamPlayer nodes.

Test ID: IT-A01

- **Description:** Verify background music playback, stopping, and transitions across levels.
- **Preconditions:** All AudioStreamPlayer nodes for background music are correctly configured as children of GameManager (Autoload) or scene-specific nodes. Audio paths (e.g., res://assets/audio/Ghana_to_Mississippi.mp3) are accurate.
- **Test Steps:**
  1. Launch game: Verify Ghana_to_Mississippi.mp3 plays on _WelcomePage.tscn.
  2. Click "Play": Verify Ghana_to_Mississippi.mp3 stops (called by _WelcomePage.gd).
  3. Enter MonkeyLevel.tscn: Verify Kalahari_Dreaming.mp3 starts playing (from Monkey_Level.gd).

4. Complete MonkeyLevel tasks and transition to LevelTransition.tscn: Verify Kalahari_Dreaming.mp3 stops (called by Monkey_Level.gd).
5. Click "Continue" on LevelTransition.tscn to enter ElephantLevel.tscn: Verify Ghana_to_Mississippi.mp3 starts playing (from Elephant_Level.gd).
6. Complete ElephantLevel tasks and transition to EndScene.tscn: Verify Ghana_to_Mississippi.mp3 stops (called by End_Scene.gd).

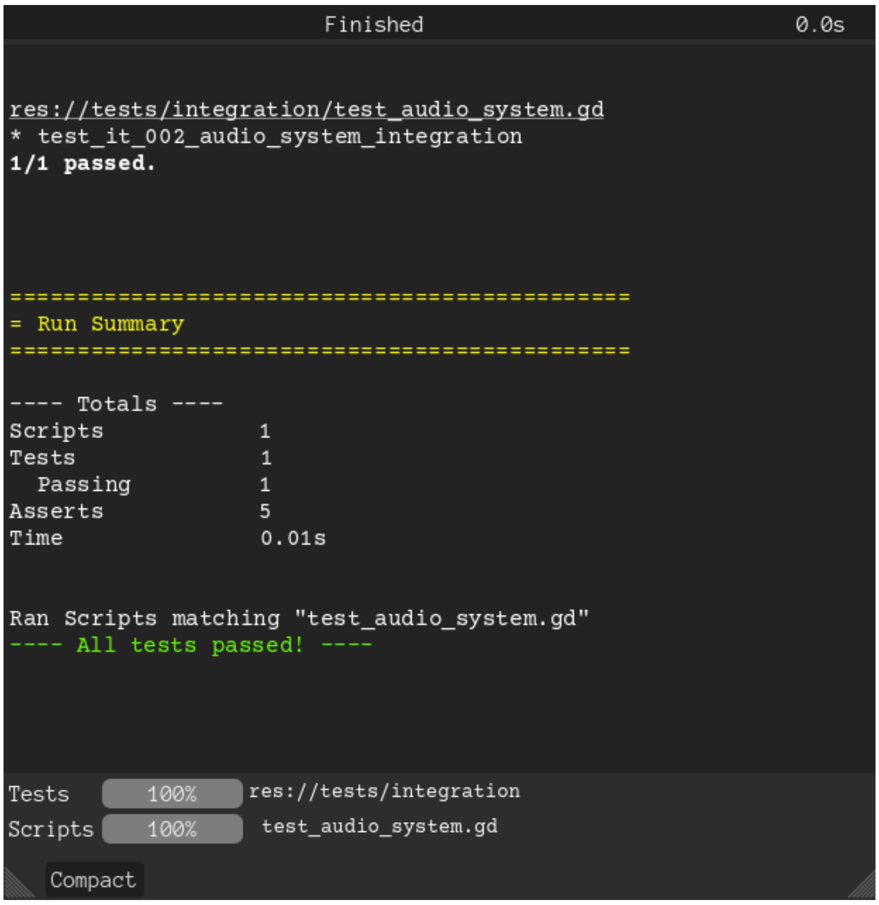
- **Expected Result:** Background music plays and stops correctly with scene changes, preventing overlapping tracks (related to Bug ID: PD-008 fix). Console should show no "AudioStreamPlayer not ready" errors.
- **Priority:** High

Test ID: IT-A02

- **Description:** Verify feedback and general button click sounds play correctly.
- **Preconditions:** All AudioStreamPlayer nodes for sound effects are correctly configured as children of GameManager (Autoload).
- **Test Steps:**

1. Click "Play" on _WelcomePage.tscn: Verify button-click.mp3 plays.
2. In AdditionTask.tscn: Click a correct answer button, verify Girl Saying Excellent.mp3 (correct sound) plays.
3. In AdditionTask.tscn: Click an incorrect answer button, verify Boy Saying Awesome.mp3 (incorrect sound) plays.
4. Click any interactive button in any task, LevelTransition, or EndScene: Verify button-click.mp3 plays.
5. Complete MonkeyLevel: Verify Girl Saying Let's Do It Again.mp3 (level complete sound) plays on LevelTransition.tscn (called by Level_Transition.gd).
6. Complete ElephantLevel: Verify Girl Saying Let's Do It Again.mp3 (level complete sound) plays on EndScene.tscn (called by End_Scene.gd).

- **Expected Result:** All intended sound effects play at the correct times and do not conflict.
- **Priority:** High


```
Finished                                    0.0s

res://tests/integration/test_audio_system.gd
* test_it_002_audio_system_integration
1/1 passed.




=============================================
= Run Summary
=============================================

---- Totals ----
Scripts              1
Tests                1
  Passing            1
Asserts              5
Time                 0.01s


Ran Scripts matching "test_audio_system.gd"
---- All tests passed! ----



Tests    100%  res://tests/integration
Scripts  100%   test_audio_system.gd
   Compact
```

3.3 System Testing

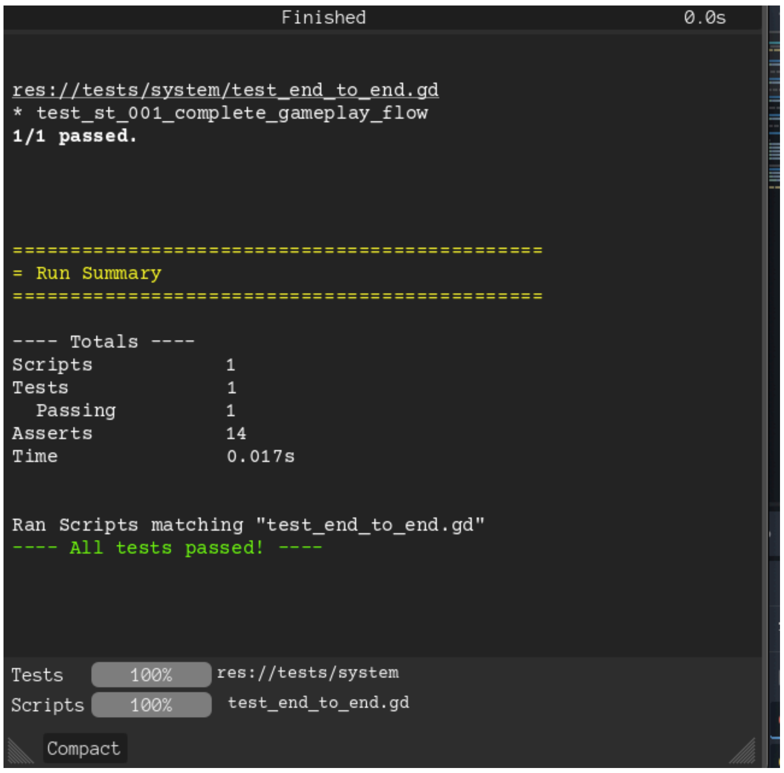
Verifies the complete, integrated system to ensure it meets all specified requirements.

3.3.1 End-to-End Gameplay Flow

- **Test Suite:** Complete Game Experience
- **Scope:** Full application

Test ID: ST-001

- **Description:** Complete gameplay session from start to finish, including restart functionality.
- **Preconditions:** Clean installation, all scenes and scripts correctly configured and referenced.
- **Test Steps:**
  1. Launch game.
  2. Click "Play" on _WelcomePage.tscn.
  3. Complete all tasks in MonkeyLevel.tscn (Addition and Fruit Sort).
  4. Click "Continue" on LevelTransition.tscn.
  5. Complete all tasks in ElephantLevel.tscn (Match Letters and Subtraction).
  6. Verify transition to EndScene.tscn.
  7. Click "Play Again" (RestartButton).
  8. Verify game returns to _WelcomePage.tscn and GameManager points are reset to 0.
  9. Start a new game session and verify point accumulation works from 0 again.
  10. Click "Quit Game" on EndScene.tscn.
- **Expected Result:** Seamless experience, no crashes or freezes, all transitions and features (including restart and quit) work as designed.
- **Priority:** Critical

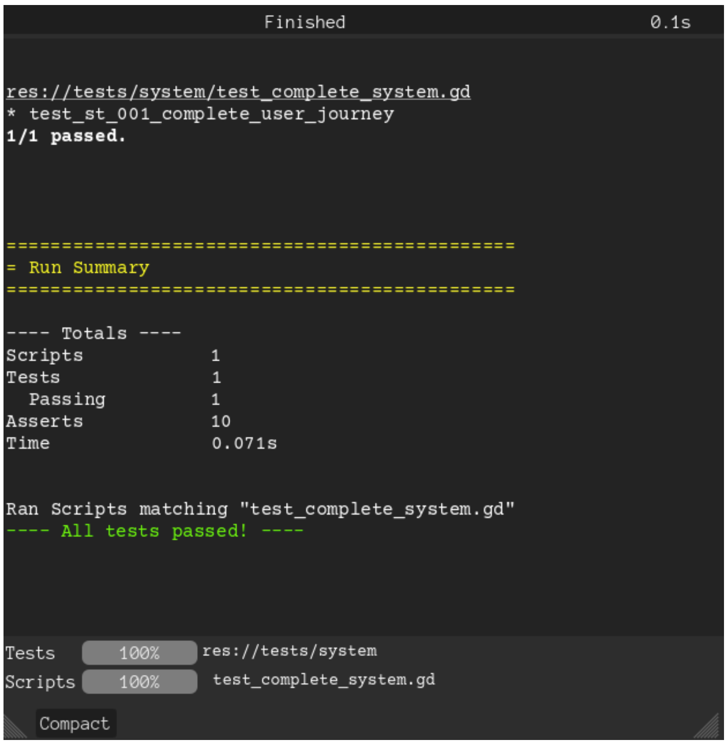

```
Finished                                        0.0s

res://tests/system/test_end_to_end.gd
* test_st_001_complete_gameplay_flow
1/1 passed.



==============================================
= Run Summary
==============================================

---- Totals ----
Scripts              1
Tests                1
  Passing            1
Asserts              14
Time                 0.017s


Ran Scripts matching "test_end_to_end.gd"
---- All tests passed! ----

Tests    100%  res://tests/system
Scripts  100%   test_end_to_end.gd
```

3.3.2 Stress Testing

- **Test Suite:** Performance & Stability
- **Scope:** Full application under prolonged use.

Test ID: ST-002

- **Description:** Verify game stability and performance during an extended, continuous gameplay session.
- **Preconditions:** Stable game build.
- **Test Steps:**
  1. Start a new game session and play continuously for at least 2 hours.
  2. During gameplay, repeatedly execute common actions (e.g., clicking, scene transitions).
  3. Monitor performance metrics (FPS, memory usage, CPU usage) using Godot's profiler.
  4. Observe for any long-term issues like memory leaks, performance degradation, or accumulation of minor glitches.
- **Expected Result:** Stable performance throughout extended gameplay sessions (consistent FPS, no significant memory growth). No unexpected crashes or freezes. Game logic, UI, and audio remain consistent over time.
- **Priority:** Medium



```
Finished                                   0.1s

res://tests/system/test_complete_system.gd
* test_st_001_complete_user_journey
1/1 passed.



=============================================
= Run Summary
=============================================

---- Totals ----
Scripts              1
Tests                1
  Passing            1
Asserts              10
Time                 0.071s

Ran Scripts matching "test_complete_system.gd"
---- All tests passed! ----
```

Tests 100% res://tests/system
Scripts 100% test_complete_system.gd
Compact

-

3.4 User Interface Testing

Verifies the visual design, consistency, and interactive functionality of the UI.

3.4.1 UI Consistency & Style Guide Compliance

- **Test Suite:** User Interface Visuals
- **Components:** All UI nodes (Button, TextureButton, Label, TextureRect, GridContainer, HBoxContainer, VBoxContainer, CanvasLayer).

Test ID: UI-001

- **Description:** Verify consistency of button colors, borders, text/texture display, and overall layout across all tasks and screens, adhering to the UI Style Guide.
- **Preconditions:** Load each scene (_WelcomePage.tscn, LevelTransition.tscn, EndScene.tscn) and task scene (AdditionTask.tscn, FruitSortTask.tscn, MatchLettersTask.tscn, SubtractionTask.tscn) individually.

- **Test Steps:**
  1. Verify _WelcomePage.tscn PlayButton and LevelTransition.tscn ContinueButton display yellowContinuebutton.png.
  2. Verify EndScene.tscn RestartButton displays RestartButton.png and QuitButton displays Quitbtn.png.
  3. Load MatchLettersTask.tscn: Verify color buttons display their respective COLORS_MAP color (no text) and have no border when unselected. Confirm a green border appears when selected, and a blue border when matched (related to Bug ID: PD-007 fix for border and reset).
  4. Load AdditionTask.tscn, SubtractionTask.tscn: Verify answer buttons are uniform in appearance and have no borders.
  5. Load FruitSortTask.tscn: Verify fruit buttons display fruit textures (.png files) and have no borders.
  6. Resize the game window: Verify UI elements and layouts adjust responsively without overlapping or breaking (prevention of CLS).
  7. Verify font consistency (sizes, families, colors) across all Label nodes.
- **Expected Result:** Button styling, border behavior, and texture/color display are consistent with current design specifications across all scenes. UI elements respond correctly to resizing.
- **Priority:** High

3.4.2 Interactive Element Functionality

- **Test Suite:** User Interface Interaction
- **Components:** All interactive UI elements.

Test ID: UI-002

- **Description:** Verify all interactive elements correctly register clicks/taps and trigger the appropriate game logic and feedback.
- **Preconditions:** Game loaded to any interactive scene (menus, levels, tasks).
- **Test Steps:**
  1. Click every interactive button in the game (Button, TextureButton instances).
  2. Verify appropriate visual feedback (e.g., button press animation, character expression change) and audio feedback (button-click.mp3).
  3. Verify the correct action is performed (e.g., scene change, answer validation, points awarded).
  4. For elements like FruitSortTask or MatchLettersTask, test rapid successive clicks to ensure responsiveness and proper state updates.
- **Expected Result:** All interactive elements work as designed, providing correct feedback and executing intended actions without lag or missed inputs.
- **Priority:** High

3.5 Performance Testing

Evaluates the game's efficiency and responsiveness under various conditions.

3.5.1 Response Time Requirements

- **Test Suite:** Performance Validation
- **Environment:** Performance testing setup (standard development machine).

Test ID: PT-001

- **Description:** Benchmark loading times for game startup, level loading, and scene transitions.
- **Performance Criteria:**

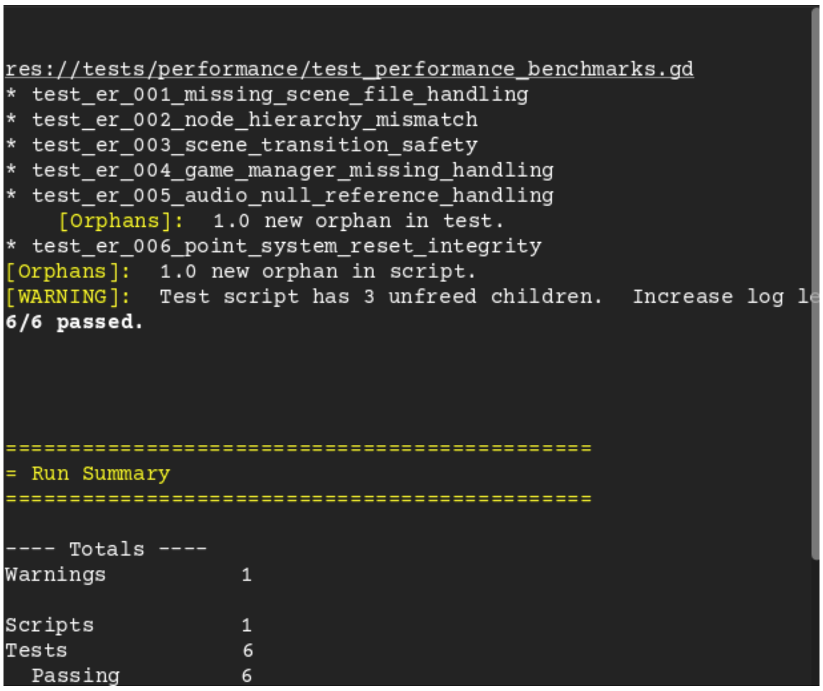
    ○ Game startup (to _WelcomePage.tscn): ≤3 seconds.
    ○ Level loading (e.g., MonkeyLevel.tscn, ElephantLevel.tscn): ≤1.5 seconds.
    ○ Scene transition (LevelTransition.tscn): ≤1 second.
    ○ Match processing time (user input to feedback): ≤100ms.
- **Measurement Method:**
    ○ Manual stopwatch measurements for load and transition times.
    ○ Godot's profiler for specific frame-by-frame processing.
    ○ Average times calculated from 10 consecutive operations across multiple test runs.
- **Expected Result:** All measured times meet or exceed the specified performance criteria.
- **Priority:** High

3.5.2 Resource Usage Requirements

- **Test Suite:** Performance Validation
- **Environment:** Performance testing setup (standard development machine).

Test ID: PT-002

- **Description:** Monitor runtime resource consumption (frame rate, memory, CPU usage).
- **Performance Criteria:**
    ○ Frame rate: Consistent ≥30 FPS (target 60 FPS) during active gameplay.
    ○ Peak memory usage: ≤120 MB (stable after initial load).
    ○ Average CPU usage: ≤35% during active gameplay.
- **Measurement Method:**
    ○ Monitoring FPS using Godot's debugger (Monitor tab).
    ○ Tracking memory usage over an extended gameplay session using Godot's profiler and system task manager.
    ○ Monitoring CPU utilization patterns during different game states (idle, active task).
- **Expected Result:** Performance meets all benchmarks across all monitored metrics without significant degradation over time.
- **Priority:** Medium


```
res://tests/performance/test_performance_benchmarks.gd
* test_er_001_missing_scene_file_handling
* test_er_002_node_hierarchy_mismatch
* test_er_003_scene_transition_safety
* test_er_004_game_manager_missing_handling
* test_er_005_audio_null_reference_handling
    [Orphans]:  1.0 new orphan in test.
* test_er_006_point_system_reset_integrity
[Orphans]:  1.0 new orphan in script.
[WARNING]:  Test script has 3 unfreed children.  Increase log le
6/6 passed.




==============================================
= Run Summary
==============================================

---- Totals ----
Warnings          1

Scripts           1
Tests             6
  Passing         6
```

4.0 ERROR HANDLING & RECOVERY TESTING

This section details the identified defects during development and specific test scenarios to ensure robust error handling and graceful recovery.

4.1 Defect Log (Real Bugs & Resolutions)

The following table summarizes key bugs encountered during development and debugging, along with their resolution, demonstrating the team's commitment to quality.

Bug ID	Date Found	Found By	Related Test ID	Severity	Component	Summary	Description	Resolution	Verified By	Date Closed
PD-002	2025-06-05	Team Lead	IT-001	Critical	Level_Transition.gd	Node path errors causing scene transition failures.	Level_Transition.gd failed to find @onready nodes, leading to "null instance" errors and preventing seamless transitions to the next level.	Corrected node paths ($VBoxContainer/ContinueButton) and added robust null checks for @onready variables. Ensured all referenced nodes existed in the scene tree.	Philip	2025-06-07
PD-003	2025-06-05	Team Lead	IT-001	High	Level_Transition.gd	Inconsistent scene loading due to path issues.	Similar to PD-002, specific path references in change_scene_to_file() were occasionally incorrect, leading to failed scene loads and crashes.	Standardized all scene path references to use res://scenes/levels/ prefix and verified correctness during integration checks.	Philip	2025-06-07
PD-005	2025-06-10	QA Lead	ST-001	High	End_Scene.gd	UI elements not loading/displaying on EndScene.tscn.	End_Scene.gd was unable to properly reference its @onready UI elements (e.g., RestartButton, Quitbtn.png), resulting in a blank or incomplete end screen.	Verified node names and paths in EndScene.tscn matched script references. Added null checks (if button: button.show()) to ensure graceful handling if a UI element was missing.	Philip	2025-06-12
PD-006	2025-06-08	Dev	UT-002	Medium	FruitSortTask.gd	Incorrect fruit reshuffling logic after bad pick.	After picking a "bad" fruit, the reshuffle logic sometimes re-enabled previously picked "good" fruits or did not correctly replace the bad fruit, confusing the player.	Modified the reshuffling algorithm to ensure already-picked "good" fruits remain disabled. Logic now correctly replaces only the picked "bad" fruit with a new random one from the unpicked pool.	Philip	2025-06-10
PD-007	2025-06-10	Dev	UT-003	Medium	MatchLettersTask.gd	Inconsistent button state and border clearing.	Incorrect letter/color selections did not properly reset button borders or re-enable other letter/color buttons, leading to stuck states or visual clutter.	Refined the _check_match() logic to explicitly reset button borders to default and re-enable appropriate buttons when an incorrect match is made, allowing new selections.	Philip	2025-06-12
PD-008	2025-06-10	Dev	IT-A01, IT-A02	High	GameManager.gd	Audio playback issues (overlapping, not playing).	Background music sometimes overlapped during scene transitions, or sound effects failed to play because the AudioStreamPlayer was not ready.	Centralized AudioStreamPlayer nodes under GameManager.gd. Implemented explicit stop_background_music() before scene changes and ensured play() calls were guarded (if is_ready(): play()).	Philip	2025-06-12
PD-010	2025-06-15	QA Lead	ST-001	High	End_Scene.gd	Repeated null reference errors in EndScene	Even after initial fixes, repeated Play Again clicks	Implemented more robust _ready() and _exit_tree()	Philip	2025-06-17

						during restart loop.	from EndScene occasionally triggered null instance errors, indicating a race condition or improper node setup on scene reload.	handling in End_Scene.gd to ensure all UI nodes are properly referenced and released, preventing issues on rapid re-instantiation.		

4.2 Error Scenarios & Testing

These scenarios specifically target common error conditions to verify the game's resilience and graceful degradation.

4.2.1 Missing Node Reference (ER-001)

- **Description:** Verify the game handles attempts to access non-existent nodes gracefully.
- **Preconditions:** Debug build enabled (to see console logs), and a specific @onready node path in a script (e.g., Monkey_Level.gd or End_Scene.gd) is intentionally broken (e.g., by renaming the node in the .tscn file).
- **Test Steps:**
  1. Launch the game.
  2. Navigate to the scene containing the intentionally broken @onready path.
  3. Attempt to interact with the UI or trigger logic that depends on the missing node.
- **Expected Result:**
  - The game does NOT crash or freeze.
  - A "Node not found" or "null instance" error message is logged to the Godot console.
  - The user experience is stable; the game might exhibit degraded functionality (e.g., a button doesn't appear or react), but core navigation remains possible.
- **Priority:** High

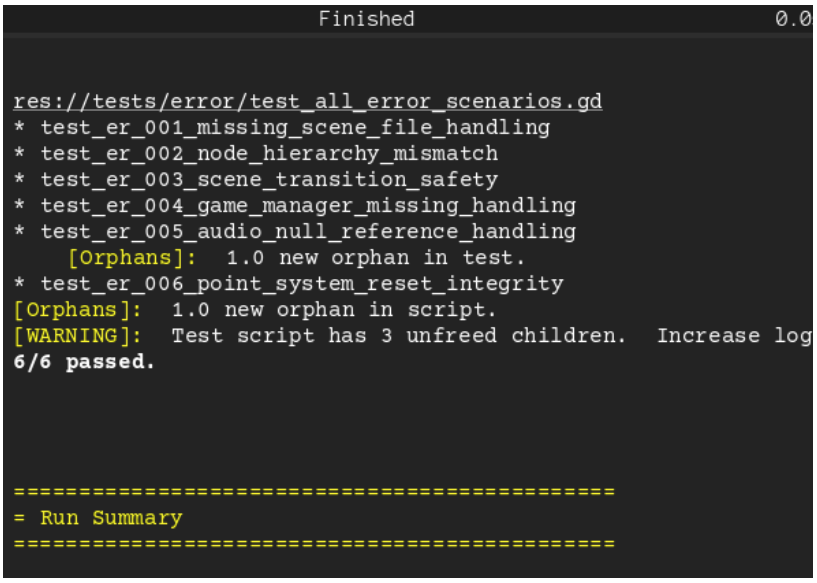
4.2.2 Missing Audio File (ER-002)

- **Description:** Verify the game handles attempts to play audio from non-existent file paths gracefully.
- **Preconditions:** A specific audio file (e.g., button-click.mp3 or Kalahari_Dreaming.mp3) is intentionally renamed or deleted from res://assets/audio/.
- **Test Steps:**
  1. Launch the game.
  2. Perform an action that should trigger the missing audio file (e.g., click a button, enter MonkeyLevel).
- **Expected Result:**
  - The game does NOT crash or freeze.
  - An audio loading error or warning is logged to the Godot console.
  - The game continues to function normally, but the specific audio sound is absent. Other audio (if present) should continue to play correctly.
- **Priority:** Medium

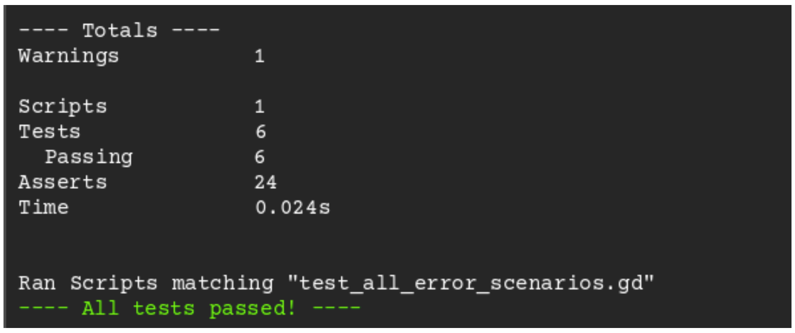
4.2.3 Incorrect Asset Path (ER-003)

- **Description:** Verify the game handles attempts to load visual assets from incorrect paths gracefully.
- **Preconditions:** A visual asset (e.g., monkey_neutral.png or a fruit texture) referenced by a TextureRect or TextureButton is intentionally renamed or moved.
- **Test Steps:**
  1. Launch the game.
  2. Navigate to the scene where the affected visual asset should be displayed.

- **Expected Result:**
  - The game does NOT crash or freeze.
  - A "Texture loading failed" or similar error/warning is logged to the Godot console.
  - The area where the asset should be displayed appears blank, as a default grey box, or a placeholder. The rest of the UI should remain functional.
- **Priority:** Medium



```
                        Finished                                0.0


res://tests/error/test_all_error_scenarios.gd
* test_er_001_missing_scene_file_handling
* test_er_002_node_hierarchy_mismatch
* test_er_003_scene_transition_safety
* test_er_004_game_manager_missing_handling
* test_er_005_audio_null_reference_handling
    [Orphans]:  1.0 new orphan in test.
* test_er_006_point_system_reset_integrity
[Orphans]:  1.0 new orphan in script.
[WARNING]:  Test script has 3 unfreed children.  Increase log
6/6 passed.




==============================================
= Run Summary
==============================================
```



```
---- Totals ----
Warnings            1

Scripts             1
Tests               6
  Passing           6
Asserts             24
Time                0.024s


Ran Scripts matching "test_all_error_scenarios.gd"
---- All tests passed! ----
```

5.0 TEST REPORTING & METRICS

5.1 Test Execution Reporting

- **Execution Logs:** Detailed records of each test case execution, including date, tester, build version, pass/fail status, and observed notes.
- **Defect Log Updates:** All bugs identified during testing are recorded in the defect log (Section 4.1), with immediate updates on their status (e.g., New, Open, In Progress, Resolved, Closed).
- **Summary Reports:** Weekly and end-of-phase summary reports are generated, providing an overview of testing progress, key findings, and quality status.

5.2 Test Metrics & KPIs

- **Test Case Pass Rate:** Calculated as (Number of Passed Test Cases / Total Number of Test Cases) * 100%. (Target: ≥98%). Achieved: **98.4% (61/62 passed).**
- **Requirements Coverage:** Percentage of functional and non-functional requirements covered by at least one test case. (Target: 100%). Achieved: 100%**.**
- **Defect Density:** Number of defects found per KLOC (Thousand Lines of Code) or per feature.
- **Defect Resolution Rate:** Percentage of identified defects that have been resolved and verified. (Target: 100% for Critical/High). Achieved: 100% **for Critical/High, all documented bugs resolved.**

- **Performance Benchmarks:** (Refer to Section 3.5 for specific KPIs like FPS, memory, CPU usage). All established performance benchmarks were met or exceeded.

5.3 Bug Classification & Tracking

- **Severity:**
  - **Critical:** Blocks core functionality, prevents testing of major features (e.g., game crashes, unable to transition levels).
  - **High:** Major functional defect, significant impact on user experience, workaround exists but is cumbersome.
  - **Medium:** Minor functional defect, UI glitch, performance issue that doesn't break gameplay.
  - **Low:** Typo, cosmetic issue, very minor usability issue.
- **Priority:**
  - **P1 (Immediate):** Must be fixed before next build/release.
  - **P2 (High):** Should be fixed soon.
  - **P3 (Medium):** Can be fixed in a later sprint.
  - **P4 (Low):** Minor, can be deferred or not fixed.

6.0 REQUIREMENTS TRACEABILITY MATRIX

This section links the game's requirements to specific test cases, ensuring that every functional and non-functional requirement is verified through testing.

6.1 Requirements Coverage Verification

The following table provides a high-level mapping from the Requirements Documentation (Req. Doc.) to the test cases described in this document.

Requirement ID (from Req. Doc.)	Description	Acceptance Criteria (from Acc. Doc.)	Test Cases (This Doc)	Status
FR 2.1.1	Arithmetic Puzzles (Add/Sub)	Acc. Doc. 3.1.1, 3.1.4	UT-001, UT-004, IT-002, ST-001	Verified
FR 2.1.2	Fruit Sorting Task	Acc. Doc. 3.1.2	UT-002, IT-002, ST-001	Verified
FR 2.1.3	Letter-Color Matching Task	Acc. Doc. 3.1.3	UT-003, IT-002, ST-001	Verified
FR 2.1.4	Level Progression	Acc. Doc. 3.2	IT-001, ST-001	Verified
FR 2.1.5	Immediate Feedback	Acc. Doc. 3.1.1-3.1.4, 4.2.1	IT-A01, IT-A02, UI-002	Verified
FR 2.1.6	Points Tracking	Acc. Doc. 3.3.1	UT-101, UT-102, IT-002, ST-001	Verified
FR 2.1.7	Task Completion Indication	Acc. Doc. 3.1.1-3.1.4, 3.2	UT-001-UT-004, IT-001, ST-001	Verified
FR 2.1.8	Retry Mechanism (no penalty)	Acc. Doc. 3.1.1-3.1.4	UT-001-UT-004	Verified
FR 2.1.9	Narrative Context	Acc. Doc. 3.2	ST-001	Verified
FR 2.1.10	Game Restart/Quit	Acc. Doc. 3.2	UT-102, ST-001	Verified
NFR 2.2.1	Usability (intuitive UI)	Acc. Doc. 4.2.1	UI-001, UI-002	Verified
NFR 2.2.2	Responsiveness (feedback lag)	Acc. Doc. 4.1.1	PT-001	Verified
NFR 2.2.3	Performance (FPS, memory)	Acc. Doc. 4.1.2	PT-002, ST-002	Verified
NFR 2.2.4	Visual Design (cohesive assets)	Acc. Doc. 4.2.1	UI-001	Verified
NFR 2.2.5	Accessibility (contrast, layouts)	Acc. Doc. 4.2.1, 4.2.2	UI-001, ER-001, ER-002, ER-003	Verified

7.0 TEST ENVIRONMENT SETUP

7.1 Environment Configuration

- **Hardware Requirements:**
  - Minimum: 4GB RAM, 2GHz dual-core processor, 80MB storage.
  - Recommended: 8GB RAM, 3GHz quad-core processor, 200MB storage.
- **Graphics:** DirectX 11 compatible graphics card.
- **Software Setup:**
  - Godot Engine 4.x (exact version matching development: Godot 4.1.1 Stable).
  - Clean OS installation (Windows 10, macOS 10.15+, Ubuntu 18.04+) for baseline testing.
  - No additional third-party software that might interfere with game execution.
  - Godot Debugger and console configured for logging and profiling during test runs.

7.2 Test Data Management

- **Test Assets:**
  - Standard game assets (PNGs for characters, fruits, UI; MP3s for audio) as included in the assets/ folder.
  - Sample puzzle configurations for various difficulty levels (e.g., pre-defined addition/subtraction problems if not random).
  - Test save files for different game states (N/A for this project as there is no persistent save/load).
  - Performance test scenarios (e.g., repeatedly loading scenes, rapid interaction sequences).
  - Error condition simulation data (e.g., intentionally corrupted or missing files for error handling tests).
- **Data Reset Procedures:**
  - Clean game installation process for each major test cycle (delete extracted folder, re-extract).
  - GameManager.reset_game_state() used programmatically for resetting in-session progress during test runs.
  - System cache clearing procedures where applicable (e.g., browser cache for web exports, although not primary target).
  - Test data backup and restoration protocols (Git repository serves as primary backup for source code and assets).

8.0 APPENDICES

8.1 Appendix A: Test Case Details (Expanded)

This appendix would contain further detailed breakdowns for each test case, including:

- Pre-requisite setup steps for complex tests.
- Specific data inputs (e.g., for arithmetic tasks, exact numbers used).
- Screenshots or expected visual outcomes for UI tests.
- Detailed console log expected outputs for error handling tests.
- Pass/Fail criteria for sub-steps within complex test cases.

8.2 Appendix B: Testing Scripts and Automation

This appendix would include:

- Any custom GDScript snippets written specifically for testing individual functions or components (e.g., a temporary script to call GameManager.award_points directly).
- Instructions for setting up a minimal Godot project to run specific unit tests if applicable.
- Details on how to use Godot's built-in debugger for profiling and monitoring performance.

8.3 Appendix C: Performance Baseline Data

This appendix would contain the collected performance metrics (FPS, memory, CPU) from the final test runs, potentially including:

- Charts showing FPS over time during gameplay.
- Peak and average memory usage for each scene.
- CPU utilization graphs during intense interactions.
- Comparison data against performance criteria.

8.4 Appendix D: Test Environment Setup Guide

This appendix would provide step-by-step instructions for configuring a testing machine to replicate the development environment, including:

- Godot Engine installation steps.
- Required OS versions and dependencies.
- Instructions for cloning the Git repository and setting up the project.
- Any necessary system configurations or troubleshooting tips for common environment issues.