Test Plan

1. Introduction & Purpose

I have prepared this test plan to ensure the 10-pin bowling game prototype is thoroughly evaluated for correctness, reliability, and adherence to the bowling rules. My main goal is to verify that the application’s scoring logic accurately reflects the rules for strikes, spares, open frames, and the special conditions in the 10th frame.

2. Scope

This test plan covers:

* Core Scoring Logic: Regular rolls, strikes, spares, open frames.
* 10th Frame Special Rules: Extra rolls awarded in the case of strikes/spares.
* Edge Cases: Consecutive strikes, multiple spares in a row, zero-pin rolls.
* Data Validation: Ensuring the system handles out-of-range inputs well.

Out of scope for this project at the moment are any front-end interfaces, database interactions, or advanced performance testing. I will focus solely on the backend Python logic provided.

3. Test Objectives

1. Validate Basic Frame Scoring: Confirm that each frame’s pins are correctly tallied.
2. Ensure Correct Bonuses: Verify that strikes add the next two rolls to their frame total, and spares add the next single roll.
3. Handle 10th Frame Logic: Check scenarios where the bowler earns one or two extra rolls in the final frame.
4. Detect and Document Bugs: Record and fix any logic errors found in the scoring process.
5. Refactor and Optimize: After testing, ensure the final code is clean, maintainable, and efficient.

4. Test Approach & Strategy

I will primarily use unit testing (white-box approach) to verify individual functions and classes in the Python code. I plan to create a comprehensive test suite using Python’s unittest framework (though a framework like pytest would also be suitable).

* White-Box Testing: I'll look at the internal logic and ensure each function behaves correctly given a range of valid and invalid inputs.
* Manual Verification: For any unusual scenarios (e.g., multiple consecutive strikes), I might also run manual checks alongside the automated tests to confirm outcomes.
* Test-Driven Development (TDD) Elements: I intend to add tests incrementally, catching bugs early and then refactoring code without breaking existing functionality.

5. Test Environment

* Hardware/OS: I’ll be running the tests on my local machine (Windows), but the code should be platform-independent for Python 3.8+.
* Software & Tools:
  + Python 3
  + unittest module for creating and running test cases
  + coverage.py for measuring test coverage

No specialized hardware resources are required beyond what’s needed to run Python scripts.

6. Schedule & Milestones

1. Day 1:
   * Draft the test plan.
   * Identify testable units (functions/methods).
2. Day 2:
   * Develop initial unit tests for straightforward scoring scenarios (strikes, spares, open frames).
   * Execute tests, document any immediate bugs.
3. Day 3:
   * Expand test coverage to edge cases (multiple strikes, zero pins, 10th frame complexities).
   * Implement bug fixes and refactor code as needed.
4. Day 4:
   * Finalize all testing.
   * Generate coverage reports.
   * Complete documentation and summary report.

7. Risk Assessment

* Time Constraints: I only have a few days to complete testing and refactoring, so prioritizing critical features (like strike/spare logic) is essential.
* Incomplete Coverage: I risk missing edge cases if I rely solely on typical scenarios. I will mitigate this by methodically including test data for all possible combinations (e.g., continuous strikes, continuous spares, random sequences).
* Regression Issues: New bugs might appear after refactoring. I plan to run the entire test suite after each significant code change to catch these quickly.
* Overlooked Business Rules: If the original game specification changes or has ambiguities, tests might miss them. I will carefully revisit the specification whenever a new edge case arises.

8. Roles & Responsibilities

Because this is an individual project, I’m responsible for all tasks:

* Test Case Creation: I will design, write, and maintain all unit tests.
* Test Execution & Bug Tracking: I’ll run tests regularly, track issues in a simple spreadsheet or Git issue tracker, and verify the fixes.
* Refactoring & Documentation: I will ensure the code remains maintainable and fully documented with PythonDoc comments.