**Bowling Game Test Plan**

**1. Introduction**

This document outlines a detailed **test plan for the Bowling Game Assessment**, which focuses on verifying the correct implementation of a 10‑pin bowling game. The main goal is to ensure the BowlingGame class behaves according to standard bowling rules, including scoring, spares, strikes, 10th-frame bonuses, and error handling.

The plan aims to provide a **structured and thorough testing methodology**, highlight potential risks, and define a clear approach to ensure a high-quality assessment submission.

**2. Test Objectives**

The key objectives of this test plan are:

* To verify that the score is correctly calculated after each roll and frame.
* To confirm that special cases, such as **spares** and **strikes**, are scored according to the official rules.
* To ensure the **10th frame** correctly accounts for bonus rolls.
* To validate that invalid inputs (negative numbers, pins greater than 10, or rolling after the game is finished) are correctly rejected.
* To ensure the game stops accepting rolls once it is completed, preventing any inconsistencies.

These objectives ensure the game is both **functionally correct** and **robust against improper inputs**.

**3. Test Scope**

**In Scope**

* Core gameplay logic of the BowlingGame class (roll() and score()).
* Special scoring scenarios including **spares, strikes, and 10th-frame bonuses**.
* Handling of invalid inputs and exceptions.
* Full-game scenarios, such as gutter games, all ones, and perfect games.

**Out of Scope**

* Any graphical user interface (GUI) or visual representation.
* Integration with external systems or databases.
* Persistent game storage between sessions.

**4. Test Strategy**

Testing will primarily utilize **unit testing** with pytest. The strategy includes:

1. **Functional Testing**  
   Each feature of the BowlingGame class will be tested individually. This includes rolling pins, calculating scores per frame, and handling special conditions like spares and strikes.
2. **Boundary Testing**  
   Critical edge cases will be tested, such as 0 pins (gutter ball), 10 pins (strike), and bonus rolls in the 10th frame.
3. **Negative Testing**  
   Intentional invalid inputs will be supplied to confirm the system properly raises exceptions. Examples include negative rolls, rolls over 10 pins, or rolling after the game is finished.
4. **Integration Testing**  
   Multiple features and rolls will be combined in sequences to ensure the game calculates **total scores accurately**across full matches.

**5. Test Approach**

**Test Environment**

* Python 3.12
* pytest for unit testing
* pdoc for generating documentation

**Test Design**

* Use helper functions like roll\_many() to simulate multiple rolls efficiently.
* Create **dedicated test cases** for spares, strikes, and 10th-frame scenarios.
* Ensure each test checks both **expected output** (score) and **game state behavior** (e.g., preventing further rolls once the game ends).

**Test Execution**

1. Place the tutor-provided or implemented BowlingGame class in src/.
2. Run tests using:
3. PYTHONPATH=. pytest -q
4. Verify that all tests pass and match expected outcomes.
5. Generate documentation for reference:
6. pdoc --output-dir docs src/bowling

**Example Test Cases**

| **Test Case** | **Input** | **Expected Outcome** | **Notes** |
| --- | --- | --- | --- |
| **All gutters** | 20 rolls of 0 | Score = 0 | Tests basic scoring with minimum pins |
| **All ones** | 20 rolls of 1 | Score = 20 | Tests accumulation without spares/strikes |
| **Single spare** | 5 + 5 + 3 | Score = 16 | Confirms spare bonus applied correctly |
| **Single strike** | 10 + 3 + 4 | Score = 24 | Confirms strike bonus applied correctly |
| **Perfect game** | 12 strikes | Score = 300 | Maximum score scenario |
| **10th-frame spare** | First 18 rolls zero, last two rolls 7 + 3, bonus 5 | Score = 15 | Tests correct 10th-frame bonus for spares |
| **10th-frame strike** | First 18 rolls zero, last three rolls 10 + 7 + 2 | Score = 19 | Tests correct 10th-frame bonus for strikes |
| **Invalid rolls** | -1, 11, or roll after game finished | Raises ValueError | Ensures error handling for improper input |

**6. Risk Assessment**

| **Risk** | **Likelihood** | **Impact** | **Mitigation** |
| --- | --- | --- | --- |
| Incorrect scoring logic | Medium | High | Implement comprehensive unit tests for all scenarios |
| 10th-frame bonus miscalculation | Medium | High | Separate test cases specifically for 10th frame |
| Invalid input not caught | Low | Medium | Negative testing and exception handling |
| Test environment mismatch | Low | Medium | Use standardized Python version and dependencies |
| Misinterpretation of rules | Low | High | Follow official 10-pin bowling rules and verify with multiple test cases |

**7. Test Deliverables**

* pytest results showing passed/failed tests.
* Generated documentation using pdoc.
* Summary report highlighting test coverage, risks, and observations.

**8. Entry & Exit Criteria**

**Entry Criteria**

* BowlingGame class implemented and accessible.
* Test environment set up with pytest and pdoc.

**Exit Criteria**

* All planned test cases executed.
* Critical tests pass without errors.
* Documentation generated and verified.