**Bowling Game Backend Test Plan**

**1. Test Plan Identifier**

**BowlingGame\_TestPlan\_v1.0**

**2. Introduction**

This test plan describes the comprehensive testing strategy for the backend logic of an educational ten-pin bowling game prototype. The system is designed to demonstrate accurate bowling mechanics and scoring suitable for teaching purposes. The objective is to ensure that all scoring rules, game frames, and bonus calculations are implemented correctly, providing a reliable basis for further development including GUI and database enhancements.

**Project Background**

The backend provides core game functionality including managing game frames, recording rolls, and calculating scores according to official ten-pin bowling rules. Its correctness is critical to delivering valid educational content and accurate game feedback.

The testing approach follows Test-Driven Development (TDD) principles (Red → Green → Refactor), ensuring each feature is validated against defined criteria before integration.

**3. Test Items**

The testing scope includes the following components within the backend:

* The main BowlingGame class responsible for game state
* The roll() method to record each pinfall
* The score() method to compute the player's total score
* Helper functions supporting score calculations: \_is\_strike(), \_is\_spare(), \_strike\_bonus(), \_spare\_bonus()

**4. Features to be Tested**

**Core Functionality**

* Correct initialization of a new game and player data structures
* Accurate recording of pin counts for each roll across all 10 frames
* Proper calculation of open frames (where no strike or spare occurs)
* Detection and scoring of strikes, including the application of bonuses from subsequent rolls
* Detection and scoring of spares with appropriate one-roll bonuses
* Handling of special rules in the 10th frame, including bonus roll allowances
* Calculation of perfect games (i.e., 12 consecutive strikes for a 300 score)
* Handling of special rules in the 10th frame, including bonus roll allowances and strike–spare combinations.

**Boundary Conditions**

* Games with all gutter balls (zero pins) resulting in a 0 score
* Games with all strikes representing a perfect 300 score
* Spares where first roll counts differ, testing edge cases around scoring logic
* Boundary pin counts such as minimum (0) and maximum (10) pins per roll

**Error Conditions**

* Handling invalid pin counts such as negative numbers or values exceeding 10
* Managing excessive rolls beyond usual frame limits
* Preventing frame overflow with too many recorded rolls or improper input

Unit test coverage for helper methods will be verified individually (\_is\_strike, \_is\_spare, \_strike\_bonus, spare\_bonus) to ensure correctness before integration into the main scoring logic.

**5. Features Not to be Tested**

* The graphical user interface (GUI) portion, as it is not developed yet
* File input/output or database operations
* Multithreading or concurrent user interactions in this prototype

**6. Approach**

**Testing Strategy**

* **Unit testing**: Individual method behaviors are validated with focused test cases using the Python unittest framework
* **Integration testing**: Comprehensive game scenarios spanning all frames and scoring outcomes
* **Regression testing**: Regression testing: The automated test suite will be re-executed systematically after every bug fix and refactoring cycle to confirm no new defects are introduced.
* **Boundary testing**: Focused on edge cases and input limits

**Test Design Techniques**

* Use of equivalence partitioning to reduce duplicate test inputs
* Boundary value analysis ensuring robustness at input extremes
* Decision table testing covering combinations of strikes, spares, and open frames
* Error guessing will be applied to anticipate common mistakes such as miscounting rolls in consecutive strike sequences.

**7. Item Pass/Fail Criteria**

**Pass Criteria**

* All test cases execute without runtime exceptions or failures
* Scores computed precisely match predefined expected values for all scenarios
* Strict adherence to business rules regarding scoring and frame progression
* Complete handling of all edge and error cases
* Test coverage meets or exceeds 90% of the codebase
* Minimum of 95% test pass rate is required for acceptance.

**Fail Criteria**

* Any test case results in incorrect scoring or unexpected program behavior
* Presence of unhandled exceptions or crashes
* Violations of documented bowling rules or project requirements
* Noticeable performance degradation under typical usage

**8. Suspension and Resumption Requirements**

**Suspension Criteria**

* Identification of critical defects that prevent core game features from functioning
* Failure of test environment setup inhibiting further testing
* Incomplete or unavailable source code needed for testing

**Resumption Criteria**

* All critical defects resolved
* Test environment restored
* Code delivery completed with proper documentation

**9. Test Deliverables**

* Test case specifications and results
* Defect reports and resolution tracking
* Test coverage reports
* Performance test results
* Final test summary report
* Screenshots evidence of test execution and Git workflow is provided in the project report

**10. Testing Tasks**

1. Design and implement unit test cases
2. Execute functional testing scenarios
3. Perform boundary and edge case testing
4. Conduct regression testing after fixes
5. Generate coverage and performance reports
6. Document findings and recommendations

**11. Environmental Needs**

**Software**

* Python 3.8 or higher
* unittest or pytest framework
* Git for version control
* IDE (PyCharm, VSCode, or similar)

**12. Responsibilities**

* **Test Lead**: Piyush Tatwani
* **Developer**: Piyush Tatwani
* **Reviewer**: Piyush tatwani

**13. Staffing and Training Needs**

* Individual project - no additional staffing required
* Familiarity with Python unittest framework
* Understanding of bowling scoring rules

**14. Schedule**

* week 1: Test planning and design
* week 2: Test implementation and execution
* week 3: Debugging and refactoring
* week 4: Documentation and reporting

**15. Risks and Contingencies**

**Risks**

* Complex 10th frame bonus logic may pose implementation challenges (High)
* Accurate bonus point calculation requires thorough testing (Medium)
* Performance with multiple consecutive games may degrade (Low)
* Version control misconfiguration

**Contingencies**

* Allocate additional time for complex feature testing
* Peer reviews and code walkthroughs to identify issues early
* Consult reference implementations if available

**16. Approvals**

|  |  |
| --- | --- |
| Role | Name |
| Test Lead | Piyush Tatwani |
| Course Tutor | Sana Alyaseri |