# Bowling Game QA Test Plan

**Author:** Christiaan Boshoff

**Date:** 03.09.2025

**Version:** Final

**Project Scope:** Backend logic for ten-pin bowling game scoring.

I am doing a full QA cycle for the bowling game back end.

# Overview

This QA cycle focused on validation and improving the backend scoring for a bowling game implemented in Python. The goal was to achieve full compliance with the ten-pin bowling rules provided. This will be achieved through comprehensive unit testing, bug fixes, code refactoring and CI integration.This plan outlines the testing strategy, execution process and the results achieved during the QA cycle.

# Scope

## In Scope

* Bowling score calculation for 10 frames
* Rules handling: Strike, spares, open frames, gutter balls.
* 10th frame logic and bonus roll handling
* Input validation (0-10 pins per roll, validating frame totals)
* Unit test coverage for all gameplay scenarios
* CI pipeline and documentation generation

## Out of Scope

* GUI or front-end elements
* Integration with database or external API’s
* Multiplayer or network functionality

# Test Strategy

| **Area** | **Description** |
| --- | --- |
| Test Types | Automated Unit Testing (Python unittest)  Manual exploratory testing for edge cases |
| Tools | Python 3.13.7, unittest, HtmlTestRunner, pdoc, Github Actions |
| Test Execution | Local testing via VS Code and .env, CI testing via Github Actions |
| CI Pipeline | Github workflows run tests on push/PR, generate HTML reports, upload artifacts. |

# Risk and Mitigations

| **Risk** | **Mitigation** |
| --- | --- |
| Misinprestation of scoring rules | Validate against standard bowling rules, example games. |
| Incorrect 10th frame bonus points | Explicit 10th frame handling implemented and tested. |
| Negative or >10 pin inputs | Input validation added, raising a ValueError where needed. |
| Frame total > 10 in non strike game | Additional checks ensure compliance except where rules allow- bonus rolls. |

# Test coverage and Design

The test cases were designed around the business rules and edge cases. I have created a total of 12 tests.

## Unit Testing Scenarios

| **TC #** | **Test Case Name** | **Purpose** |
| --- | --- | --- |
| TC01 | Open Frame | Validate normal scoring without bonuses. |
| TC02 | Spare | Confirms spare bonus adds to the next roll. |
| TC03 | Strike | Confirms strike bonuses added to the next two rolls. |
| TC04 | Two Strikes | Validate consecutive strike bonuses. |
| TC05 | Tenth Frame Spare | Test correct bonus handling for spare in frame 10. |
| TC06 | Tenth Frame Strike | Ensures the 3-roll rule is implemented in the final frame. |
| TC07 | Perfect Game | Validates 300 score for 12 strikes |
| TC08 | Gutter Game | Validates 0 score when all rolls are 0. |
| TC09 | Negative Input | Validates rejection of negative pins. |
| TC10 | Input > 10 | Ensures rolls above 10 raise an exception. |
| TC11 | All Spares | Confirms each spare and bonus total is calculated correctly |
| TC12 | Frame > 10 pins | Validate rejection of frames with invalid pin totals. |

All the test needs to pass after debugging.

# Debugging and Fixes

* I will run my tests against the provided code
* For any failures
  + Identify the cause, business logic or calculation bug
  + Fix it
  + Document code fix and commit messages

# Refractoring

* **Code Simplification** - Remove duplicate code
* **Naming Conventions** - Improve variable/ function naming
* **PEP8 Compliance** - Break long functions into smaller, reusable ones
* **Helper Methods** - Abstract logic into private methods.
* **Readability** - Add comments and logical separation

.

# Documentation

* **PythonDoc** - I will add Google-style docstings for all public methods.
* **Generated HTML docs** - Using pdoc, it will be stored in /docs.
* **Usage Examples** - Included example\_usage.py to show valid roll sequence and expected score.

# Version Control and Github Workflow

| **Task** | **Staus** |
| --- | --- |
| Create Repository | Done |
| Initial commit of base code |  |
| Test Plan committed |  |
| Unit test suite committed |  |
| Each bug fix committed separately |  |
| Refactoring commits |  |
| Github Actions Workflow |  |
| HTML test reports uploaded |  |

# Final Report

* **Intro** - Project context and the purpose of testing
* **Testing Approach** - Tools, methodology and the coverage
* **Bugs Found** - Description and the fix
* **Refactoring** - What was improved and why.
* **Git Usage** - Link to the repo
* **Recommendation**s - For future development

**Bowling Game Test Plan**

**Project:** Ten-Pin Bowling Game Backend

**Version:** 1.0

**Author:** C Boshoff

**Date:** 19.08.25

# Introduction

This test plan outlines the strategy, scope and methodology for testing the backend logic of the ten-pin bowling game prototype. The primary objective is to verify that the scoring and frame logic comply with the specified business rules for the game.

# Scope

## In Scope

* Frame by frame scoring logic
* Strike, spare and open frames rules
* Bonus calculations
* 10th frame special rules
* Validation of input pin count

## Not In Scope

* Graphical User interface (GUI)
* Input from files or databases
* Multiplayer turn taking logic, unless part of scoring

# Test Objectives

* Ensure compliance with the ten-pin bowling score rules
* Identify and fix defects in the scoring calculations
* Validate the edge cases are handled correctly
* Achieve complete unit testing coverage for all scenarios.

# Test strategy

We will use **white-sandbox** testing to directly verify back-end logic.

* **Automated Testing** - Implement via Pythons pytest framework
* **Manual Testing** - Limited exploratory testing for unexpected behaviours.

Test executed will follow these steps:

1. Create baseline tests for all scoring scenarios
2. Execute tests against the provided back-end code.
3. Debug and fix any failing cases
4. Re-run tests until all pass.
5. I will Refactor code where beneficial.

# Test Approach

## Techniques

* **UNit Testing**: To validate each scoring rule in isolation
* **Boundary Value Analysis:** Test frames with 0,1, 9, and 10 pins.
* **Equivalence Partitioning**: Group similar input scenarios.
* **Negative Testing:** Ensure invalid inputs will raise errors.

## Tools

* Python 3X
* Pytest testing framework
* Git for version control
* PythonDoc for Documentation

# Risks and Mitigation

| **Risk** | **Impact** | **Likelihood** | **Mitigation** |
| --- | --- | --- | --- |
| Misinterpretation of scoring rules | High | Medium | Corss-check test cases against official bowling scoring examples |
| Missing edge cases in test | Medium | Medium | Use checklist from business rules to ensure coverage |
| Invalid pin totals accepted | High | Low | Add validation in scoring methods |
| Code changes break other rules | High | Medium | Maintain a complete regression test suit. |

# Test case Overview

| **ID** | **Description** | **Expected result** |
| --- | --- | --- |
| TC01 | Single Open Frame - 4, 3 | Score = 7 |
| TC02 | Spare - 6, 4 + next ball 5 | Score = 15 |
| TC03 | Strike - 10 + next 2 ball 4, 3 | Score = 17 |
| TC04 | Two strikes in a row | Correct double bonus calc |
| TC05 | Tenth frame spare with bonus ball | Bonus ball added correctly |
| TC06 | Tenth frame strike with 2 bonus balls | All bonuses counted properly |
| TC07 | Perfect game (300 - 12 strikes) | Score = 300 |
| TC08 | ALl gutter balls | Score = 0 |
| TC09 | Invalid input: negative pin count | Raise ValueError |
| TC10 | Invalid input: >10 pins in a single ball | Raise ValueErro |

# Test Environment

**Hardware**: Any system capable of running Python 3.x

**OS:** Windows, macOS or Linux

**SOftware:** Python3.x, pytest, Git, IDE- VS Code

# Entry and Exit Criteria

## Entry Criteria

* Back-end code is available and compiles with out syntax errors
* Business rules are documented

## Exit Criteria

* All unit tests pass
* All defects are logged, fixed and re-tested.
* Documentation updated.

# Deliverables

* Complete the test plan
* Unit testing scripts
* Defects log
* PythonDoc-generated documentation
* Summary report