**Test Plan for 10-pin bowling prototype**

*Version: 1.0*

*Author: Dallas Keating*

*Date of Creation: 07.01.24*

*Project/Application Name: 10-pin bowling prototype*

1. Test Plan Identifier:

#TPB-1.0

2.0 Introduction

This document serves as a comprehensive test plan for the 10-pin bowling prototype. Its purpose is to outline the testing strategies, methodologies, and activities performed to ensure the functionality of the source code provided.

3.0 Test Items

10-pin bowling prototype back-end source code.

BowlingGame class methods and their functionality.

4.0 Features to be tested

We will be testing the following:

* **Scoring Logic** – Ensuring the game accurately calculates the score for each frame based on the number of pins knocked down.
* **Strike Handling** – Test to see if the game can identify if a strike was thrown and if the game adds the correct amount of bonus points based on the following frame(s).
* **Spare Handling** – Test to see if the game can identify if a spare is thrown and if the game adds the correct amount of bonus points based on the following throw.
* **Boundary Cases** – Test cases where players may achieve perfect games and/or consecutive strikes in the final frame.

5.0 Features not to be tested

We will not be testing the following:

* **GUI or Front-End components** – We will not be testing any GUI and Front-End components due to none being implemented at this current stage. Our focus is to validate and ensure correct functionality for the Back-End components.
* **Input from files or databases** – We will not be testing these items at this current stage as they are not required for testing the Back-End code, moreover it is not within the scope of this test plan.
* **Performance and Load Testing** – We will not be performing performance and load testing as for the current stage of the prototype, it does not require great performance and load testing as we are simply testing the functionality of the provided source code.

6.0 Test Approach

This test approach section will outline our overall strategy and methodology for conducting a comprehensive test for the 10-pin bowling game prototype.

We will be using the following testing methodologies to ensure a comprehensive and thorough assessment of the functionality for the prototype.

* **Unit Testing:** Unit testing is suitable for testing this prototype as it allows us to test individual pieces of codes such as functions and methods allowing us to ensure the absolute functionality of small pieces of code. This will result in a more comprehensive and detailed test report about how each part of the code performed when it was being tested.

We will be using the ‘unittest’ framework from python to execute this and aligning our test file to support this framework when executing our test.

* **Functional Testing:** Functional Testing is suitable for testing this prototype as it helps us ensure the correct functionality of the prototype and with this, we can validate whether the game works as expected or not from the end-user’s perspective. This helps us ensure essential functions of the prototype such as the scoring logic, strike handling and spare handling work as they should according to the game rules provided from the client’s requirements. We will be using test cases to mimic user interactions with the prototype and comparing the expected result along with the actual result given.
* **Boundary Testing:** With boundary testing, we will be able to verify how the prototype handles interactions that result in reaching the edge of the operational limits. This may be from creating scenarios such as perfect games, gutter ball games and so on. With this, we will be able test the prototype for potential bugs and unexpected behavior. We can test this when we are creating test cases and add the scenarios like gutter ball games and perfect games and compare the expected results against the actual results.

With these outlined methodologies we will be able to assess the prototype comprehensively using unit testing to check individual pieces of code, functionality test to validate the outcome of the application and boundary testing to check rare/unlikely scenarios.

With this, we will continue our test plan by outlining test cases, perform unit test, write down our expectations for these tests, record and analyze the results.

7.0 Environmental Needs:

We will require the following to perform our testing:

* An IDE (integrated development environment) capable of debugging and executing python code.
* Version Control System – We will be using git to append changes made to the code and record each version and the changes made.
* A system with an updated/current version of python installed to run test cases execute the prototypes back-end code.
* The system must also have the ‘unittest’ framework installed and available to perform our unit testing.
* Access to internet to append changes with git.

8.0 Item pass/ fail criteria

For this assessment, it is important to note down the criteria for testing to pass or fail.

Pass Criteria:

* **Scoring Logic** – Each frame’s score should be calculated accurately based on the pins knocked down.
* **Strike Handling** – The prototype should correctly identify when a strike is scored and accurately add the bonus points based on the next frame(s).
* **Spare Handling** – The prototype should correctly identify when a spare is scored and accurately add the bonus points based on the next roll.
* **Boundary Cases** – The prototype correct handles scenarios such as perfect games and gutter ball games. The prototype does not respond with unexpected behavior when dealing with edge cases.

Fail Criteria:

* **Scoring Logic** – Inaccurate calculations of frame scores or incorrect addition of bonus points for strikes or spares.
* **Strike Handling** – Failure to identify a strike and/or incorrectly adds the bonus points based on the next frame(s).
* **Spare handling** – Failure to identify a spare and/or incorrectly adds the bonus points based on the next roll.
* **Boundary Cases** – Prototype responds unexpectedly when handling edge cases resulting in scenarios such as crashes or inaccurate scores.

9.0 Suspension Criteria and Resumption Requirements

Testing will be suspended if any defects are encountered that render the prototype inoperable and/or cause the prototype to behave unexpectedly. This will assist with debugging and fixing the provided source code.

Once the defects have been corrected and the prototype is operable, we can resume testing.

10.0 Test Deliverables

* Test Cases
* Test Summary Report
* Python Documentation
* Test Execution logs

11.0 Test Cases

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Description | Pre-Conditions | Test Steps | Test Data | Expected Result | Actual Result | Pass / Fail |
| TPB01-Score | Validate frame score calculation | Null | * Roll 2 balls in 1 frame | Roll 1: 5  Roll 2: 4 | 9 / True | 9 / True | Pass |
| TPB02-Score | Validate frame score calculation | Null | * Roll 2 balls in 1 frame | Roll 1: 1  Roll 2: 0 | 1 / True | 1 / True | Pass |
| TPB01-Strike | Validate identification of a strike | Null | * Roll 1 ball in 1 frame * Roll 2 balls in frame 2 | Frame 1:  Roll 1: 10  Frame 2:  Roll 1: 2  Roll 2: 5 | Score: 24  Strike identified | Score: 24  Striked | Pass |
| TPB02-Strike | Validate no strike identified | Null | * Roll 2 balls in frame 1 | Roll 1: 0  Roll 2: 3 | Score: 3  No Strike Scored | Score: 3  No Strike | Pass |
| TPB01-Spare | Validate identification of a spare | Null | * Roll 2 balls in frame 1 * Roll 2 balls in frame 2 | Frame 1:  Roll 1: 5  Roll 2: 5  Frame 2:  Roll 1: 2  Roll 2: 5 | Score: 19  Spare identified | Score: 19  Spared | Pass |
| TPB02-Spare | Validate no spare identified | Null | * Roll 2 balls in frame 1 | Roll 1: 5  Roll 2: 3 | Score: 8  No spare scored | Score: 8  No spare | Pass |
| TPB01-Boundary | Validate a perfect game | Null | * Roll 12 strikes | All rolls are strikes | Score: 300  Perfect Game | Score: 300 | Pass |
| TPB02-Boundary | Validate a gutter game | Null | * All rolls miss | All rolls are gutter balls | Score: 0  Gutter Game | Score: 0 | Pass |

11.1 Unit Testing

We will be testing the following methods of the BowlingGame class

* Roll() method - validated
* Score() method - validated
* isStrike() method - validated
* isSpare() method - validated
* StrikeScore() method - validated
* SpareScore() method - validated
* frameScore() method - validated

These are all the main methods of the BowlingGame class that will require testing.

**Test Summary Report**

*Version: 1.0*

*Author: Dallas Keating*

*Date of Creation: 08.01.23*

*Project/Application: Ten Pin Bowling Prototype*

1.0 Test Report Identifier:

#TPB-1.0

2.0 Summary

The object of the testing process was verify the functionality of the provided source code for the ten pin bowling game prototype and ensure that the application worked as expected. We were able to perform unit testing and use test cases to verify this. Using the unit tests, we could check that the individual methods worked.

3.0 Variances

We had encountered issues due to spelling mistakes, functions not working as intended and difficulty reading the code. With these test, we were able to outline multiple issues in the code and fix them. Through this process, we were able to ensure that code performed as expected once debugging concluded.

4.0 Comprehensive Assessment

We could conclude that initially, the application was unable to perform reliably and accurately track the score of the bowling games. It was also unable to identify strikes and spares. We were able to fix these and get the prototype back in working order.

4.1 Refactoring Summary

We refactored multiple items in the bowling game class as well as fixing the test class provided in the source code. Some of the items we refactored and why are as follows:

* Method Name clarification – Renaming items such as strikeScore, spareScore and frameScore for better clarity on what the method actually did. We renamed theses items by putting calculate at the front like such: calculateStrikeScore, calculateSpareScore, calculateFrameScore. This allowed for a more readable and clear code allowing future developers to easily maintain the code.
* Conditional Logic – I focused on some of the conditional statements within the score method as we were encountering inaccurate results such as bonus points not being applied correctly. By changing, we were able to ensure that the code was more accurate and more clear.

5.0 Summary Results

Total Test Executed: 7

Test Passed: 3

Test Failed: 4

Total Defects Found: 3

High Severity: 0

Medium Severity: 2

Low Severity: 0

6.0 Evaluation

From the executed unit test and use of the test cases, I can say the current state of the application is stable and performs as expected. The code is refactored now so it is more clear and easier to understand. This will help our developers who will be working on the front-end to have an easier time of creating the GUI.

Dallas Keating