Introduction

This document will have the process of testing, the findings from the testing and the solution to the testing, the debugging, and the refactoring of the 10pin bowling project. This testing project is aiming to improve the existing back-end code into being fully functional/ well tested and more streamlined. The tests will consist of valid cases, boundary cases and invalid cases, that plan to find the bugs in functionality and fix them.

This document shows the testing, debugging, refactoring and documentation of this project.

Test plan

Test objectives

I will be testing a 10pin bowling game prototype, the source code has been developed, and I will be testing the prototype using that code. There is only code for the back end of the program, meaning that there is no GUI or system for receiving input data. This means that I will only be testing the back end of the project.

The features I will be testing from this project is:

- all 10(12) rounds our done successfully
- That all the scores are accurate
- That the strike and spare bonuses work
- That on the 10th round the extra bowls only count for the bonus

The reason I am testing these features is because I have been told to test the 10pin bowling project from the code given. The reason I am not doing more is because only the back end was given.

Test scope

- In scope:10pin bowling play and scoring features (back-end features)
- Out of scope: UI, and input features + database (front-end, input and database)

Test strategy

The testing type that will be done is Functional testing being unit testing. In that I will be doing functional testing, boundary testing, and invalid testing. I will be doing Manual testing for all the tests. It will be tested on visual studio and on my computer.

Test schedule

Duration 2 weeks

Timeline:

- Day 1-3: test planning
- Day 4-7: test case design
- Day 7-10: test execution and debugging
- Day 11-13: reporting

Resource requirements

Tools, environments, data

1 tester

1 developer

Test devices (computer)

Visual studio code

Word

Risk assessment

Potential issues and mitigations

- Risk 1: the scores are wrong
    - Impact: High (the game doesn't work if the scores are wrong)
    - Mitigation: when testing the scores show expected score to the actual score
- Risk 2: invalid inputs like words or letters or symbols
    - Impact: Medium (should normally not happen)
    - Mitigation: make sure that they know to make the inputs only accept numbers when they make the input fields

Test cases

There are no inputs and steps because there is only back-end of the code. This means that I will be saying the input, but I mean the bowls that I have put in the code for each game.

- Test case 1: normal game
    - Test case ID:TC001
    - Feature: scoring

    - Inputs: bowls round 1 (8,1)"9", round 2(6,4)"15", round 3(5,2)"7", round 4(10)"20", round 5(7,3)"10", round 6(0,7)"7", round 7(7,2)"9", round 8(10)"19", round 9(9,0)"9", round 10(7,3,7)"17"
    - Expected result:
        - Score of 122
- Test case 2: all spares game
    - Test case ID:TC002
    - Feature: scoring
    - Inputs: bowls round 1(0,10)"11", round 2(1,9)"12", round 3(2,8)"13", round 4(3,7)"14", round 5(4,6)"15", round 6(5,5)"16", round 7(6,4)"17", round 8(7,3)"18", round 9(8,2)"19", round 10(9,1,10)"20"
    - Expected result:
        - Score of 155
- Test case 3: edge case
    - Test case ID:TC003
    - Feature: scoring
    - Inputs: all bowls are a strike/ each round is a 10 on first bowl, and round 10 is 10, 10, 10
    - Expected result:
        - Score of 300
- Test case 4: edge case
    - Test case ID:TC004
    - Feature: scoring
    - Inputs: all bowls are 0
    - Expected result:
        - Score of 0
- Test case 5: edge case
    - Test case ID:TC001
    - Feature: scoring
    - Inputs: first bowl is –1 rest are 0
    - Expected result:
        - Error message: saying do between 1 and 10 and
- Test case 6: edge case
    - Test case ID:TC006
    - Feature: scoring
    - Inputs: first bowl is 11 rest are strikes
    - Expected result:

        - Error message: saying do between 1 and 10
- Test case 7: invalid game
    - Test case ID:TC007
    - Feature: scoring
    - Inputs: bowls round 1(9,4)"error", rest is 0
    - Expected result:
        - Error message: saying you can't hit more than what is left after the first bowl. (unless round 10 and you got a spare or strike)
- Test case 8: invalid round 10
    - Test case ID:TC008
    - Feature: scoring
    - Inputs: bowls rounds 1 to 9 are 0, round 10(3,3,3)"error"
    - Expected result:
        - Error message: you can't roll 3 on round 10 if you didn't get a strike or spare.

Debugging

Bug identifications

ID	Bug description	Impact/ Severity	Steps to reproduce	expected	actual	Consistent evidence
BUG 1	Round 10 doesn't happen	High: makes all scores wrong but all 0s	All strikes function, All Spares function	For all Stikes score = 300 For all spares score = 155	For all Stikes score = 270 For all spares score = 135	These both show that the gap in the scores is the same as round 10s points
BUG 2	open frames don't take the second bowl	High: makes all scores with an open frame wrong	Normal game function	Score =122	Score =110	This shows that there are rolls being missed
BUG 3	If rolls are less, then 0 it impacts the score	Mid: makes it that if people put negative rolls, it	Negative game function	Score = 0 and a message saying you are	Score = -1	This shows that it doesn't check if the

		impacts the game score		cheating appears		rolls are above 0
BUG 4	If rolls are more then 10 it crashes	high: it crashes the game	Bowls 11 game function	Score = 270 and a message saying you are cheating appears	Crashes the game	This shows that is doesn't check if the rolls are below 10 (and crashes)
BUG 5	If Open frames are more then 10 it impacts the score	Mid: makes it if people put more then 10 in the two bowls for an open frame it impacts the game score	More than 10 game function	Score = 9 and message saying you are cheating appears	Score = 13	This shows that if the bowls for a round are for example 8 and 5 it will damage the score

Bug solutions

Bug 1 solution:

Bug 1 was that only rounds 1 to 9 happened, this meant that in all the tests there was an error because the final round never happened. This meant that there was a big impact on the project because none of the scores would ever be correct.

The solation to fixing this:

```
def score(self):
    """Calculate the score for the current game."""
    score = 0
    frame_index = 0

    for frame in range(10):
        #changing from 9 to 10 makes it that round 10 happens
        if self._is_strike(frame_index):
```

That is changing the range from 9 to 10 making it that the 10 round happens

Bug 2 solution:

Bug 2 was that if the round was an open frame, it wouldn't count the second bowl in the score. This meant that all tests with an open frame wouldn't pass. This meant that there was a big impact because all normal games scores would be incorrect.

The solation to fixing this:

```
    else:
        # Open frame
        score += self._open_frame(frame_index)
        #changed from taking the roll to taking both rolls in the open frame
        frame_index += 2
return score
```

```
    def _open_frame(self, frame_index):
        """
        Calculate the open frame

        Args:
            frame_index: Index of the first roll in the open frame

        Returns:
            The value of the open frame
        """
        return self.rolls[frame_index] + self.rolls[frame_index + 1]
```

Being that I made a function to plus the current and second roll for the open frame thus making it count both in the score.

Bug 3 solution:

Bug 3 was that if a negative bowl was given it would be accepted and ruin the score. There isn't a big impact because you aren't meant to send negatives, but it still has an impact because the score is altered.

The solution to fixing this:

```
def roll(self, pins):
    """
    Records a roll in the game.

    Args:
        pins: Number of pins knocked down in this roll
    """
    if pins < 0 :
        #this happens if a roll is below 0 and makes it 0 by defult and says you cheated
        print( "\ncheating detected " , pins , "is lower then 0 and thus cheating so this roll will be made 0")
        self.rolls.append(0)
        self.current_roll += 1
```

As can be seen I made it so that if it is below 0 it prints that cheating was detected and made the roll = 0.

Bug 4 solution:

Bug 4 was that if a roll greater than 10 was given it would be accepted which would crash the game. This has a big impact because it crashes the whole game, making the game not useable.

The solution to fixing this:

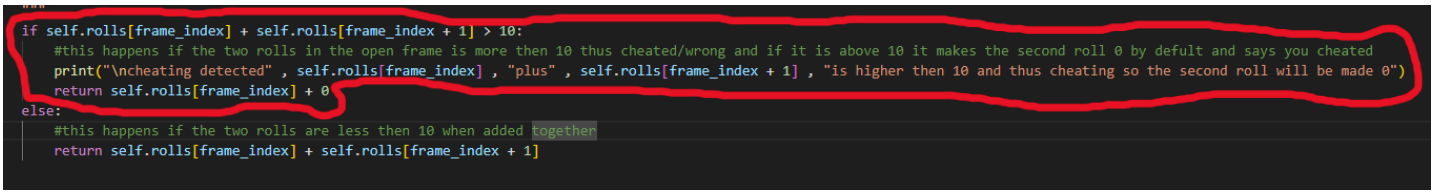
```
if pins < 0 :
    #this happens if a roll is below 0 and makes it 0 by defult and says you cheated
    print( "\ncheating detected " , pins , "is lower then 0 and thus cheating so this roll will be made 0")
    self.rolls.append(0)
    self.current_roll += 1
elif pins > 10 :
    #this happens if a roll is above 10 and makes it 0 by defult and says you cheated
    print("\ncheating detected ", pins , "is higher then 10 and thus cheating so this roll will be made 0")
    self.rolls.extend([0, 0])
    self.current_roll += 2
else:
    #this happens if the roll is between 0 and 10 / a normal roll
    self.rolls.append(pins)
    self.current_roll += 1
```

As can be seen I made it so that if it is higher than 10 it prints that cheating was detected and made the roll = 0 (it does two 0s because something 10 or higher would be seen as a strike meaning 2 rolls are needed to replace the one)

Bug 5 solution:

Bug 5 was that if you rolled for example an 8 and then something higher than a 2 like 7 it would accept it which would ruin the score. There isn't much of an impact because you aren't meant to do a second bowl above the first in a round, but it still has an impact because the score is altered.

The solution to fixing this:



```
if self.rolls[frame_index] + self.rolls[frame_index + 1] > 10:
    #this happens if the two rolls in the open frame is more then 10 thus cheated/wrong and if it is above 10 it makes the second roll 0 by defult and says you cheated
    print("\ncheating detected" , self.rolls[frame_index] , "plus" , self.rolls[frame_index + 1] , "is higher then 10 and thus cheating so the second roll will be made 0")
    return self.rolls[frame_index] + 0
else:
    #this happens if the two rolls are less then 10 when added together
    return self.rolls[frame_index] + self.rolls[frame_index + 1]
```

As can be seen, I made it so that if the two rolls together are higher than 10 it prints that cheating was detected and makes the second roll =0.

Refactoring

Optimization of the code

The code I optimized is the test cases that had different rolls they looked like this:

```
def test_normal_game(self):
    """Test a normal game."""
    self.game.roll(8)
    self.game.roll(1)

    self.game.roll(6)
    self.game.roll(4)

    self.game.roll(5)
    self.game.roll(2)

    self.game.roll(10)

    self.game.roll(7)
    self.game.roll(3)

    self.game.roll(0)
    self.game.roll(7)

    self.game.roll(7)
    self.game.roll(2)

    self.game.roll(10)

    self.game.roll(9)
    self.game.roll(0)

    self.game.roll(7)
    self.game.roll(3)
    self.game.roll(7)
    # Expected score: 122
    self.assertEqual(122, self.game.score())
```

They were in the test bowling and in-depth usage test cases. I decided to make it that they would take up less space and not repeat the same code over and over, so I changed it to this:

```
def test_normal_game(self):
    """Test a normal game."""
    #this has all the rolls in it the space is to seperate the rounds
    rolls = [8,1,   6, 4,   5, 2,   10,   7, 3,   0, 7,   7, 2,    10,   9, 0,   7, 3, 7]

    #this puts all the rolls in the game
    for pins in rolls:
        self.game.roll(pins)

    # Expected score: 122
    self.assertEqual(122, self.game.score())
```

This made it still do everything it could do but, in a lot less lines and repetition. I made there be bigger gaps between the rounds so that you can still see the rounds so that what was in the more expanded version wasn't lost.

The cases where I made these changes are for test bowling "test normal game, test all spares game" for the in-depth usage "example game, normal game, all spares game"

Justification

The reason why this refactoring is ok is because I made it so that you can still see the rounds because of the gaps otherwise I think that the older version would have been better because it would be easier to understand what is happening. This means that there is less space being used and makes it that the same code with just a different roll isn't constantly repeating. This makes it better, smaller and easier to read, which is why I think it is justified.

evidence

The performance of these tests is already low being 0.002 so it didn't change but when you look at the python programs themselves there is clear improvement being that for example in test bowling the number of lines in the program when from 133 lines to 91 lines.

challenges

The biggest challenge was finding code that wasn't already optimized during the debugging process. This was very apparent when I couldn't find anything to optimize in the bowling game code.

Also, when I found the tests to optimize, I feared it would ruin the readability which luckily didn't happen because I found a way to still be able to identify the separate round. This was the only other big challenge I had to deal with.

Version control

GitHub Link

https://github.com/lordsepticplier/10-Pin_bowling

Key examples

All the commits have the same format as this is an example:

```
fix: open frames bug

fixed the issue where open frames wouldn't count the second bowl. did this by making a function to add the first and second roll together.
```

Or

```
refactor: tests with changing rolls

I made it so that the tests with many different rolls wouldn't take so many lines of code and duplication of used code wouldn't happen.
```

These show that I am following a consistent format and describing the changes that happened in the commit.

Important notice for my GitHub repo

I didn't commit the sphinx files because this is a public repo that can be seen in my CV, and I don't want people to think I know how to use sphinx because I can't, and this was very hard.

Documentation

There are automated documents in the docs folder of the project. I used sphinx to do this, and it was the hardest thing about this project. I have made the documentation for all of the python files in this project. I have also made sure that through all my python files there are python docstrings.

Lastly there is a slight difference between the automated docs and the code being that in the code:

```
from bowling_game import BowlingGame
```

Is used for the test files

Whereas in the automated docs it is:

```
from python_code.bowling_game import BowlingGame
```

Because if I use the codes when making the docs it breaks and if I use the docs when testing it breaks so they will just be separate.

Summary

I discovered that there were a lot of critical errors that made it that the 10pin bowling was never going to give an accurate score. Through the course of testing 7 of the 8 tests failed this made it clear that there were bugs that needed to be fixed. After debugging I fixed all the bugs that were found. Now the site works for normal games, edge cases, and numerical invalid cases. Now all the test cases have passed and there are no errors with the scoring.

The main approach to discover these issues was to do normal test cases as well as edgy/boundary cases and invalid test cases, this made it easier to identify the bugs that were in the code and figure out how to fix them. Lastly, making two lots of test files, one for pass and fail and the other with the same cases but more detailed made it easier to find when there was a consistent problem that could be seen across multiple cases.

I recommend firstly when building the inputs make it only accept integers because that will make it less likely for problems to arise in the code. Also, when building the interface, I recommend making it call them out for cheating if they do an impossible role. My last recommendation is that you should make the scores visible throughout the game to make it easier to identify bugs and make the project more enjoyable for the "player".

Conclusion

In conclusion this document has shown the testing documentation for the back end of the 10pin bowling project. In it bugs were found and fixed as well as refactoring that was done to make the code better. I hope you consider the recommendations that were mentioned in the summary that is all for this test project.