**Code Refactoring**

**1. Introduction**

During the development of the Bowling Game, several refactoring steps were applied to **improve readability, maintainability, and robustness**. Each change was carefully justified and tested to ensure no functionality was broken.

**2. Refactoring Examples**

**Example 1: Simplifying the is\_finished Method**

**Before Refactoring:**

def is\_finished(self):

roll\_index = 0

for frame in range(10):

if roll\_index >= len(self.rolls):

return False

if self.rolls[roll\_index] == 10:

roll\_index += 1

else:

roll\_index += 2

if roll\_index == 20:

tenth\_first = self.rolls[18]

tenth\_second = self.rolls[19]

# Handling bonus rolls...

**After Refactoring:**

def is\_finished(self):

roll\_index = 0

for frame in range(10):

if roll\_index >= len(self.rolls):

return False

roll\_index += 1 if self.rolls[roll\_index] == 10 else 2

# Handle 10th frame bonus safely

return len(self.rolls) >= roll\_index

**Justification:**

* Removed unnecessary variables and redundant code.
* Simplified conditional logic for clarity.
* Reduced risk of IndexError when accessing rolls in the 10th frame.

**Example 2: Using a Helper Method for Roll Validation**

**Before Refactoring:**

def roll(self, pins):

if pins < 0 or pins > 10:

raise ValueError("Pins must be between 0 and 10")

if self.is\_finished():

raise ValueError("Game already finished")

self.rolls.append(pins)

**After Refactoring:**

def roll(self, pins):

self.\_validate\_roll(pins)

self.rolls.append(pins)

def \_validate\_roll(self, pins):

if pins < 0 or pins > 10:

raise ValueError("Pins must be between 0 and 10")

if self.is\_finished():

raise ValueError("Game already finished")

**Justification:**

* Separates validation logic into a dedicated private method \_validate\_roll.
* Makes roll() cleaner and easier to read.
* Enhances maintainability if validation rules change in the future.

**Example 3: Introducing Helper Functions in Tests**

**Before Refactoring:**

def test\_all\_ones():

game = BowlingGame()

for \_ in range(20):

game.roll(1)

assert game.score() == 20

**After Refactoring:**

def roll\_many(game, rolls, pins):

for \_ in range(rolls):

game.roll(pins)

def test\_all\_ones():

game = BowlingGame()

roll\_many(game, 20, 1)

assert game.score() == 20

**Justification:**

* Eliminates repetitive code for rolling multiple pins.
* Increases readability and reduces the chance of errors in test cases.
* Makes it easier to add new tests with different roll patterns.

**3. Measurable Improvements**

| **Refactoring** | **Improvement** |
| --- | --- |
| is\_finished simplification | Reduces potential IndexError, easier to read and maintain |
| \_validate\_roll method | Centralized validation logic, clean roll() method |
| roll\_many helper in tests | Cleaner, more reusable test code, reduces repetition |