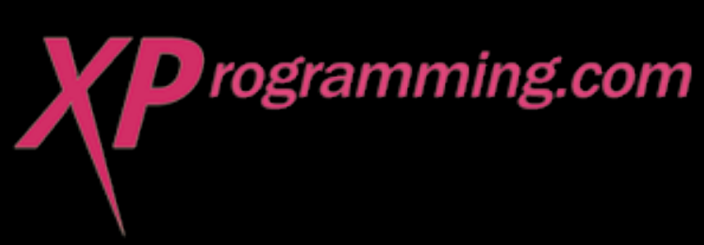
Bowling Game Kata

Object Mentor, Inc.

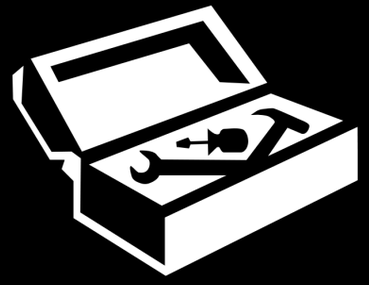
www.objectmentor.com
blog.objectmentor.com

fitnesse.org



www.junit.org

…adapted for Swift 3

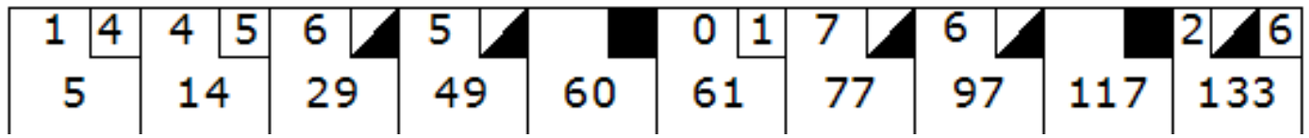


Quality Coding *with Jon Reid*

QualityCoding.org

Last update: 2 Nov 2016

Scoring Bowling



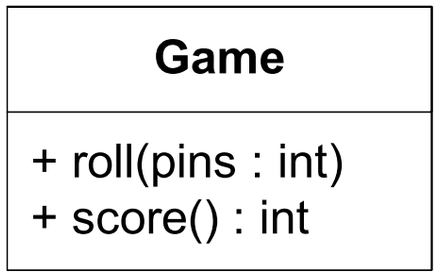
The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

A spare is when the player knocks down all 10 pins in two tries. The bonus for that frame is the number of pins knocked down by the next roll. So in frame 3 above, the score is 10 (the total number knocked down) plus a bonus of 5 (the number of pins knocked down on the next roll.)

A strike is when the player knocks down all 10 pins on his first try. The bonus for that frame is the value of the next two balls rolled.

In the tenth frame a player who rolls a spare or strike is allowed to roll the extra balls to complete the frame. However no more than three balls can be rolled in tenth frame.

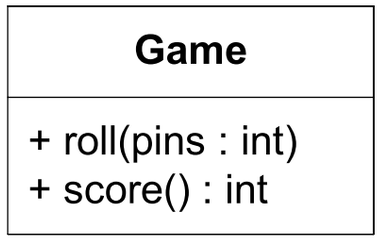
The Requirements



Write a class named “BowlingGame” that has two methods:

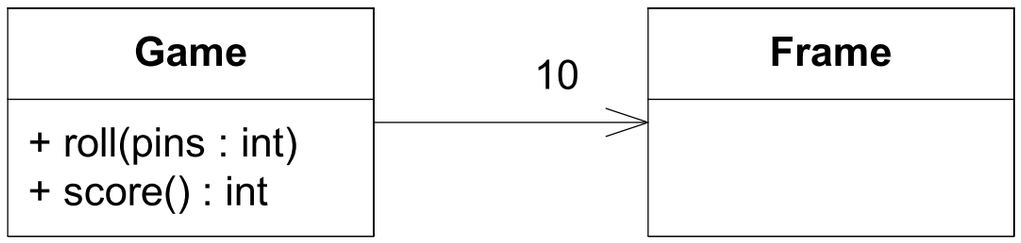
- roll(_ pins: Int) is called each time the player rolls a ball. The argument is the number of pins knocked down.
- score() -> Int is called only at the very end of the game. It returns the total score for that game.

A quick design session



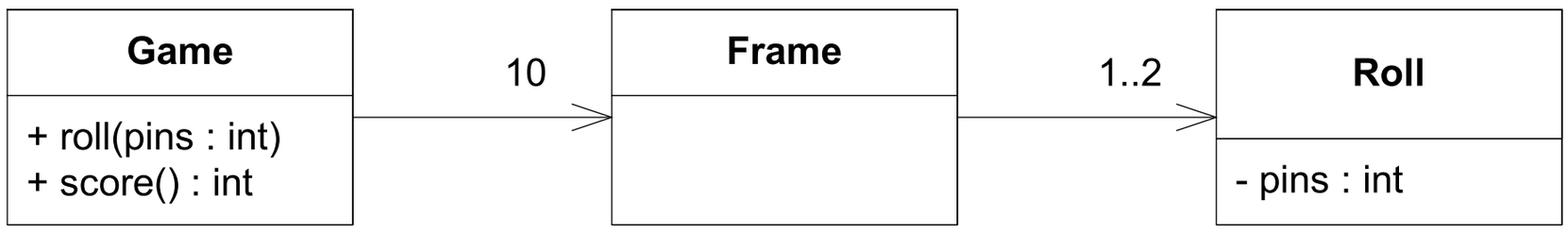
Clearly we need the Game class.

A quick design session



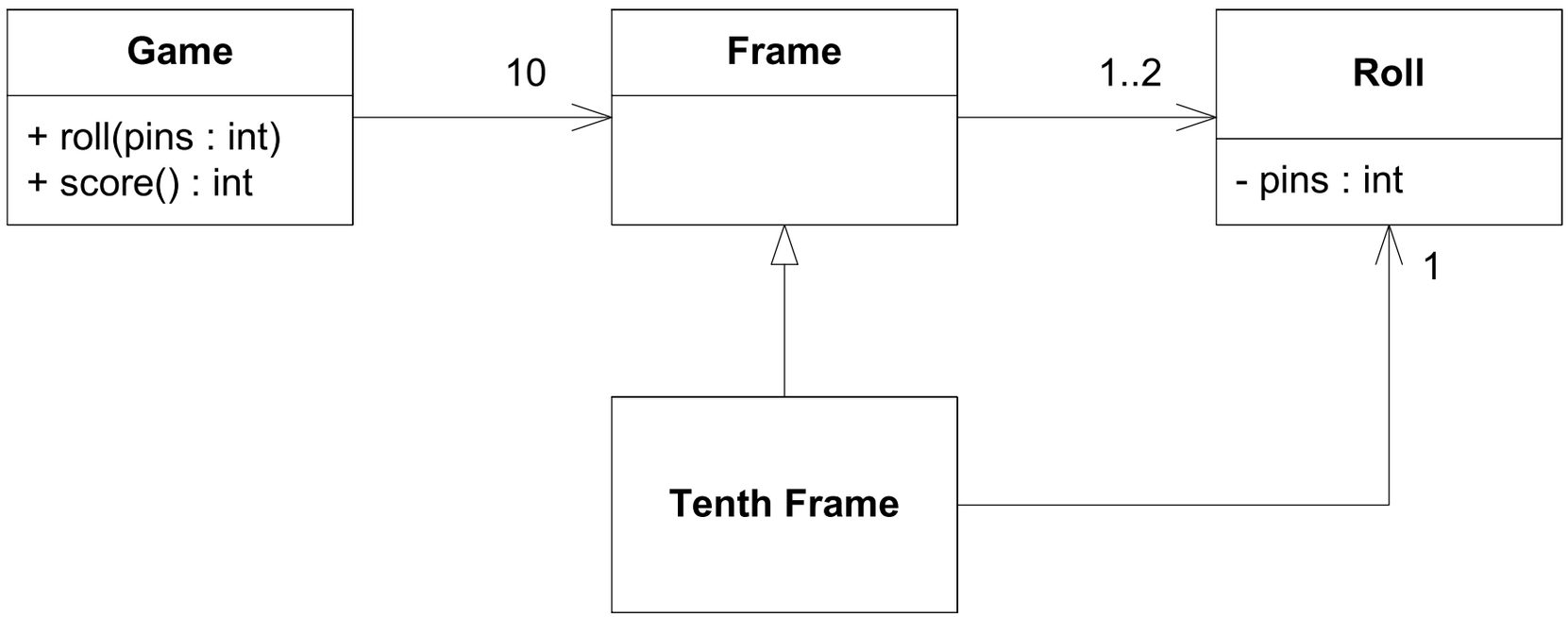
A game has 10 frames.

A quick design session



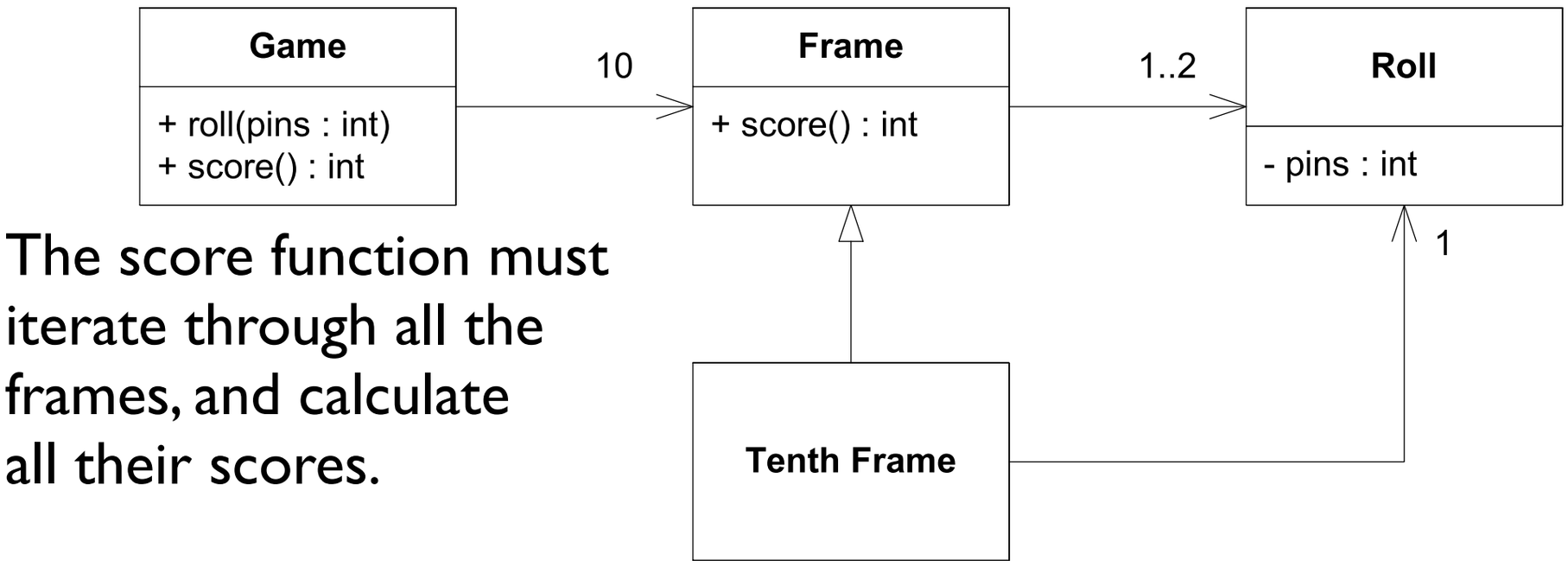
A frame has 1 or 2 rolls.

A quick design session



The tenth frame has 2 or 3 rolls.
It is different from all the other frames.

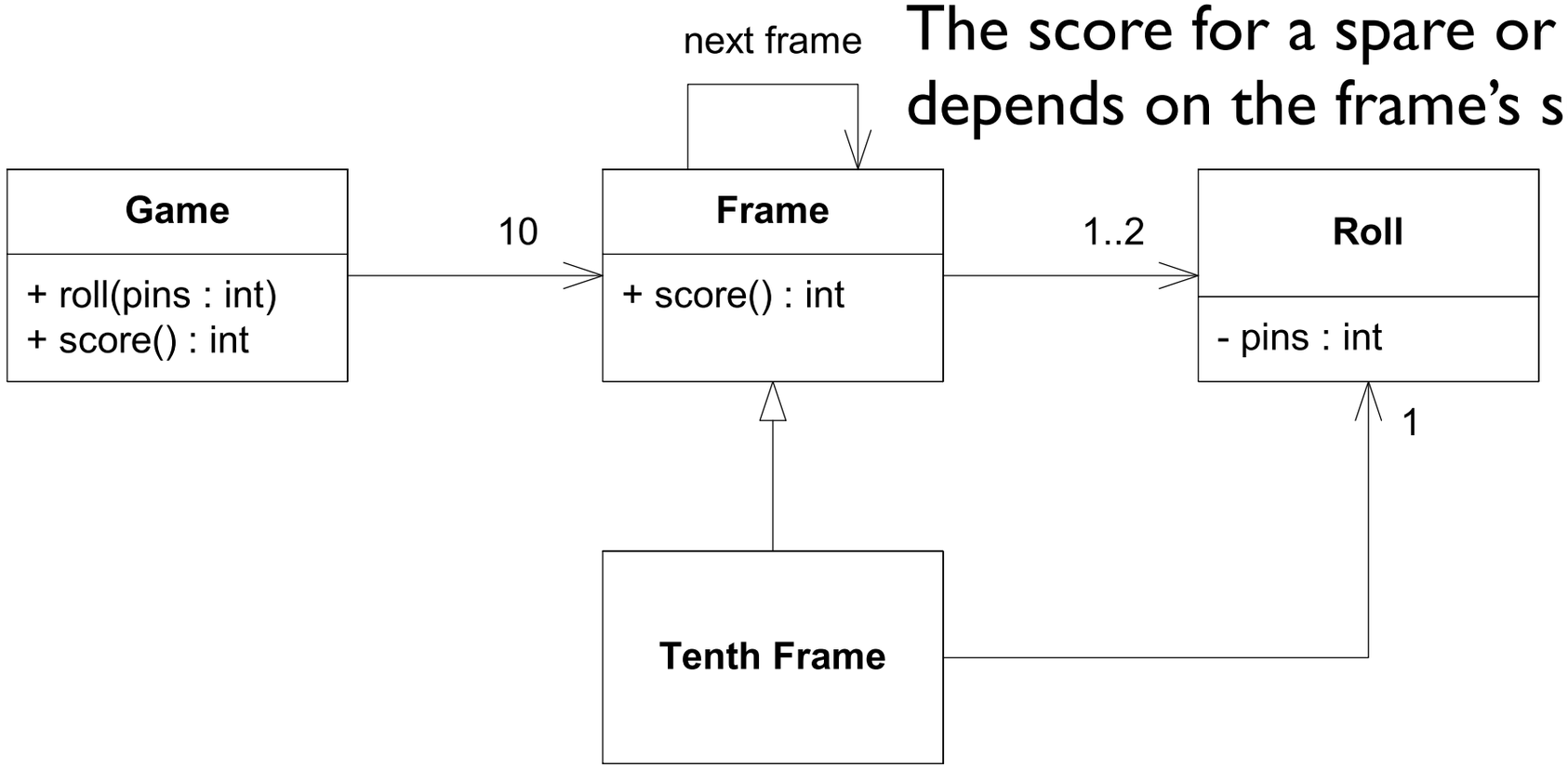
A quick design session



The score function must iterate through all the frames, and calculate all their scores.

A quick design session

The score for a spare or a strike depends on the frame's successor



Begin.

- Create an iOS application of type Cocoa Touch Framework. Name it *BowlingGame*. Select Swift as the language. Select "Include Unit Tests"
- Select a Simulator
- ⌘U to run unit tests
- Verify that testExample successfully ran
- Delete setUp, tearDown, testExample and testPerformanceExample, and run tests again

The first test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
    }

}
```

```
class Game {
}
```

The first test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
    }

}
```

```
class Game {
}
```

The first test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
    }

}
```

```
class Game {

    func roll(_ pins: Int) {
    }

}
```

The first test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

}
```

```
class Game {

    func roll(_ pins: Int) {
    }

    func score() -> Int {
        return -1
    }

}
```

("–1") is not equal to ("0")

The first test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

}
```

```
class Game {

    func roll(_ pins: Int) {
    }

    func score() -> Int {
        return 0
    }

}
```

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        let game = Game()
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {

    func roll(_ pins: Int) {
    }

    func score() -> Int {
        return 0
    }

}
```

- Game creation is duplicated
- roll loop is duplicated

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        let game = Game()
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {

    func roll(_ pins: Int) {
    }

    func score() -> Int {
        return 0
    }

}
```

- Game creation is duplicated
- roll loop is duplicated

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        let game = Game()
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {

    func roll(_ pins: Int) {
    }

    func score() -> Int {
        return 0
    }

}
```

("0") is not equal to ("20")

- Game creation is duplicated
- roll loop is duplicated

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        let game = Game()
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- Game creation is duplicated
- roll loop is duplicated

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {

    func testGutterGame() {
        let game = Game()
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        let game = Game()
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- roll loop is duplicated

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    func testGutterGame() {
        for _ in 1...20 {
            game.roll(0)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- roll loop is duplicated

The second test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    func testGutterGame() {
        let n = 20
        let pins = 0
        for _ in 1...n {
            game.roll(pins)
        }
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

Extract Method

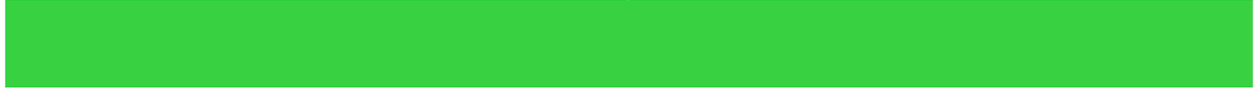
```swift
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- roll loop is duplicated

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        let n = 20
        let pins = 0
        rollMany(pins: pins, times: n)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```



- roll loop is duplicated

The second test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        for _ in 1...20 {
            game.roll(1)
        }
        XCTAssertEqual(game.score(), 20)
    }

}
```

```swift
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

The second test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

("13") is not equal to ("16")

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

Tempted to use flag to remember previous roll. So design must be wrong.

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

roll() calculates score, but name does not imply that.

score() does not calculate score, but name implies that it does.

Design is wrong. Responsibilities are misplaced.

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

//    func testOneSpare() {
//        game.roll(5)
//        game.roll(5) // spare
//        game.roll(3)
//        rollMany(pins: 0, times: 17)
//        XCTAssertEqual(game.score(), 16)
//    }

}
```

```swift
class Game {
    private var theScore = 0

    func roll(_ pins: Int) {
        theScore += pins
    }

    func score() -> Int {
        return theScore
    }

}
```

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

//    func testOneSpare() {
//        game.roll(5)
//        game.roll(5) // spare
//        game.roll(3)
//        rollMany(pins: 0, times: 17)
//        XCTAssertEqual(game.score(), 16)
//    }

}
```

```
class Game {
    private var theScore = 0
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        theScore += pins
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        for i in 0...20 {
            score += rolls[i]
        }
        return score
    }

}
```

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

//    func testOneSpare() {
//        game.roll(5)
//        game.roll(5) // spare
//        game.roll(3)
//        rollMany(pins: 0, times: 17)
//        XCTAssertEqual(game.score(), 16)
//    }

}
```

```
class Game {
    private var theScore = 0
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        theScore += pins
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        for i in 0...20 {
            score += rolls[i]
        }
        return score
    }

}
```

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

//    func testOneSpare() {
//        game.roll(5)
//        game.roll(5) // spare
//        game.roll(3)
//        rollMany(pins: 0, times: 17)
//        XCTAssertEqual(game.score(), 16)
//    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        for i in 0...20 {
            score += rolls[i]
        }
        return score
    }

}
```

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        for i in 0...20 {
            score += rolls[i]
        }
        return score
    }

}
```

("13") is not equal to ("16")

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        for i in 0...20 {
            if (rolls[i] + rolls[i+1] == 10) { // spare
                score += ...
            }
            score += rolls[i]
        }
        return score
    }

}
```

This isn't going to work because "i" might not refer to the first ball of the frame.

Design is still wrong.

Need to walk through array two balls (one frame) at a time.

- ugly comment in test

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

//    func testOneSpare() {
//        game.roll(5)
//        game.roll(5) // spare
//        game.roll(3)
//        rollMany(pins: 0, times: 17)
//        XCTAssertEqual(game.score(), 16)
//    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var i = 0
        for _ in 1...10 {
            score += rolls[i] + rolls[i + 1]
            i += 2
        }
        return score
    }

}
```

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var i = 0
        for _ in 1...10 {
            score += rolls[i] + rolls[i + 1]
            i += 2
        }
        return score
    }

}
```

("13") is not equal to ("16")

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var i = 0
        for _ in 1...10 {
            // spare
            if rolls[i] + rolls[i + 1] == 10 {
                score += 10 + rolls[i + 2]
                i += 2
            } else {
                score += rolls[i] + rolls[i + 1]
                i += 2
            }
        }
        return score
    }

}
```

- ugly comment in test
- ugly comment in conditional
- i is a bad name for this variable

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var i = 0
        for _ in 1...10 {
            // spare
            if rolls[i] + rolls[i + 1] == 10 {
                score += 10 + rolls[i + 2]
                i += 2
            } else {
                score += rolls[i] + rolls[i + 1]
                i += 2
            }
        }
        return score
    }

}
```

- ugly comment in test
- ugly comment in conditional

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            // spare
            if rolls[roll] + rolls[roll + 1] == 10 {
                score += 10 + rolls[roll + 2]
                roll += 2
            } else {
                score += rolls[roll] +
                         rolls[roll + 1]
                roll += 2
            }
        }
        return score
    }

}
```

Renamed using "Edit All in Scope"

- ugly comment in test

The third test.

```swift
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

    override func setUp() {
        super.setUp()
        game = Game()
    }

    override func tearDown() {
        game = nil
        super.tearDown()
    }

    private func rollMany(pins: Int, times: Int) {
        for _ in 1...times {
            game.roll(pins)
        }
    }

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    func testOneSpare() {
        game.roll(5)
        game.roll(5) // spare
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```swift
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if isSpare(roll) {
                score += 10 + rolls[roll + 2]
                roll += 2
            } else {
                score += rolls[roll] +
                         rolls[roll + 1]
                roll += 2
            }
        }
        return score
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

}
```

The third test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

...

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if isSpare(roll) {
                score += 10 + rolls[roll + 2]
                roll += 2
            } else {
                score += rolls[roll] +
                         rolls[roll + 1]
                roll += 2
            }
        }
        return score
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

}
```

- ugly comment in test

The fourth test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

...

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        game.roll(10) // strike
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if isSpare(roll) {
                score += 10 + rolls[roll + 2]
                roll += 2
            } else {
                score += rolls[roll] +
                         rolls[roll + 1]
                roll += 2
            }
        }
        return score
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

}
```

("17") is not equal to ("24")

- ugly comment in test

The fourth test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

...

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        game.roll(10) // strike
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if rolls[roll] == 10 { //strike
                score += 10 +
                         rolls[roll + 1] +
                         rolls[roll + 2]
                roll += 1
            } else if isSpare(roll) {
                score += 10 + rolls[roll + 2]
                roll += 2
            } else {
                score += rolls[roll] +
                         rolls[roll + 1]
                roll += 2
            }
        }
        return score
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

}
```

- ugly comment in test
- ugly comment in conditional
- ugly expressions

The fourth test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

...

    func testGutterGame() {
        rollMany(pins: 0, times: 20)
        XCTAssertEqual(game.score(), 0)
    }

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        game.roll(10) // strike
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if rolls[roll] == 10 { //strike
                score += 10 +
                         rolls[roll + 1] +
                         rolls[roll + 2]
                roll += 1
            } else if isSpare(roll) {
                score += 10 + rolls[roll + 2]
                roll += 2
            } else {
                score += rolls[roll] +
                         rolls[roll + 1]
                roll += 2
            }
        }
        return score
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

}
```

- ugly comment in test
- ugly comment in conditional

The fourth test.

```
...

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        game.roll(10) // strike
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if rolls[roll] == 10 { //strike
                score += 10 + strikeBonus(roll)
                roll += 1
            } else if isSpare(roll) {
                score += 10 + spareBonus(roll)
                roll += 2
            } else {
                score += sumOfBallsInFrame(roll)
                roll += 2
            }
        }
        return score
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

    private func strikeBonus(_ roll: Int) -> Int {
        return rolls[roll + 1] + rolls[roll + 2]
    }

    private func spareBonus(_ roll: Int) -> Int {
        return rolls[roll + 2]
    }

    private func sumOfBallsInFrame(_ roll: Int) -> Int {
        return rolls[roll] + rolls[roll + 1]
    }

}
```

- ugly comment in test

The fourth test.

```
...

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        game.roll(10) // strike
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if isStrike(roll) {
                score += 10 + strikeBonus(roll)
                roll += 1
            } else if isSpare(roll) {
                score += 10 + spareBonus(roll)
                roll += 2
            } else {
                score += sumOfBallsInFrame(roll)
                roll += 2
            }
        }
        return score
    }

    private func isStrike(_ roll: Int) -> Bool {
        return rolls[roll] == 10
    }

    private func isSpare(_ roll: Int) -> Bool {
        return rolls[roll] + rolls[roll + 1] == 10
    }

    private func strikeBonus(_ roll: Int) -> Int {
        return rolls[roll + 1] + rolls[roll + 2]
    }

    private func spareBonus(_ roll: Int) -> Int {
        return rolls[roll + 2]
    }

    private func sumOfBallsInFrame(_ roll: Int) -> Int {
        return rolls[roll] + rolls[roll + 1]
    }

}
```

The fourth test.

```
import XCTest
@testable import BowlingGame

class BowlingGameTests: XCTestCase {
    var game: Game!

...

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    private func rollStrike() {
        game.roll(10)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        rollStrike()
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

}
```

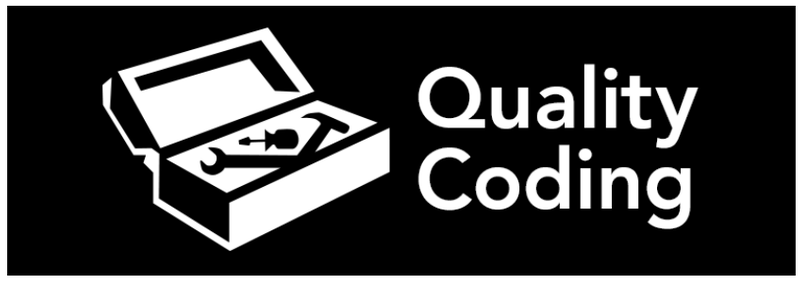
```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if (isStrike(roll)) {
                score += 10 + strikeBonus(roll)
                roll += 1
            } else if isSpare(roll) {
                score += 10 + spareBonus(roll)
                roll += 2
            } else {
                score += sumOfBallsInFrame(roll)
                roll += 2
            }
        }
        return score
    }

...
```

The fifth test.

```
...

    func testAllOnes() {
        rollMany(pins: 1, times: 20)
        XCTAssertEqual(game.score(), 20)
    }

    private func rollStrike() {
        game.roll(10)
    }

    private func rollSpare() {
        game.roll(5)
        game.roll(5)
    }

    func testOneSpare() {
        rollSpare()
        game.roll(3)
        rollMany(pins: 0, times: 17)
        XCTAssertEqual(game.score(), 16)
    }

    func testOneStrike() {
        rollStrike()
        game.roll(3)
        game.roll(4)
        rollMany(pins: 0, times: 16)
        XCTAssertEqual(game.score(), 24)
    }

    func testPerfectGame() {
        rollMany(pins: 10, times: 12)
        XCTAssertEqual(game.score(), 300)
    }

}
```

```
class Game {
    private var rolls = [Int](repeating: 0, count: 21)
    private var currentRoll = 0

    func roll(_ pins: Int) {
        rolls[currentRoll] = pins
        currentRoll += 1
    }

    func score() -> Int {
        var score = 0
        var roll = 0
        for _ in 1...10 {
            if (isStrike(roll)) {
                score += 10 + strikeBonus(roll)
                roll += 1
            } else if isSpare(roll) {
                score += 10 + spareBonus(roll)
                roll += 2
            } else {
                score += sumOfBallsInFrame(roll)
                roll += 2
            }
        }
        return score
    }

...
```

End



QualityCoding.org