# Bowling Game Kata

Object Mentor, Inc.

www.objectmentor.com
blog.objectmentor.com

fitnesse.org

www.junit.org

# …adapted for Objective-C (using Xcode 4.5)

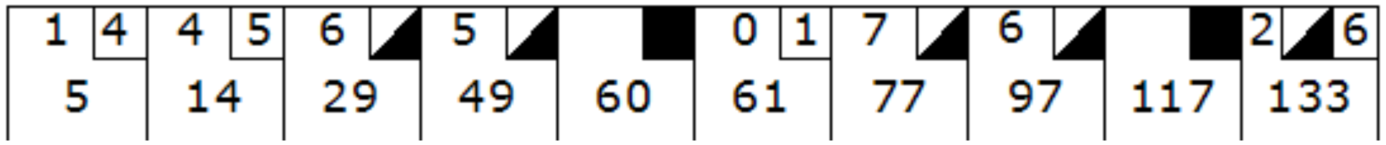
Quality Coding for iOS developers

Jon Reid
**QualityCoding.org**

**Author:**
**OCHamcrest**
**OCMockito**

Last update: 23 Dec 2012

# Scoring Bowling



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 4 | 4 5 | 6 ◢ | 5 ◢ | ■ | 0 1 | 7 ◢ | 6 ◢ | ■ | 2 ◢ 6 |
| 5 | 14 | 29 | 49 | 60 | 61 | 77 | 97 | 117 | 133 |

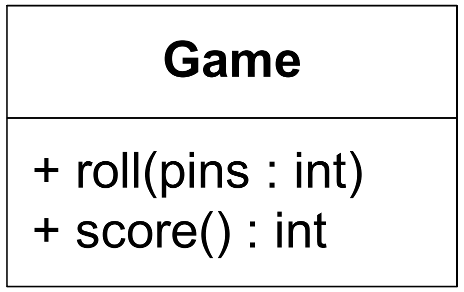
The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

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.

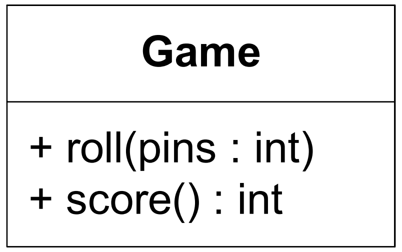
# The Requirements



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

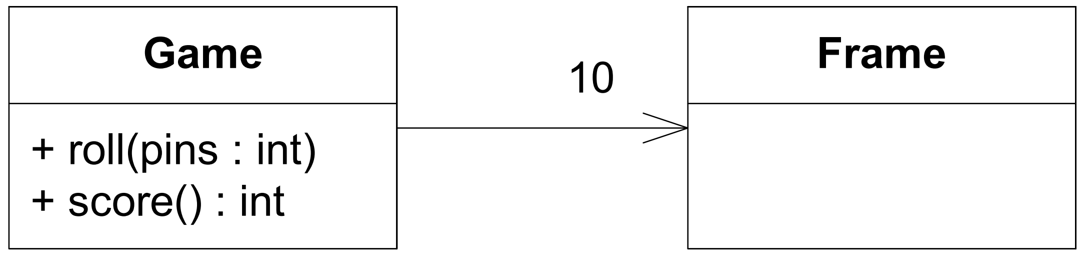
- (void)rollWithPinCount:(int)pins is called each time the player rolls a ball. The argument is the number of pins knocked down.
- (int)score is called only at the very end of the game. It returns the total score for that game.

# A quick design session



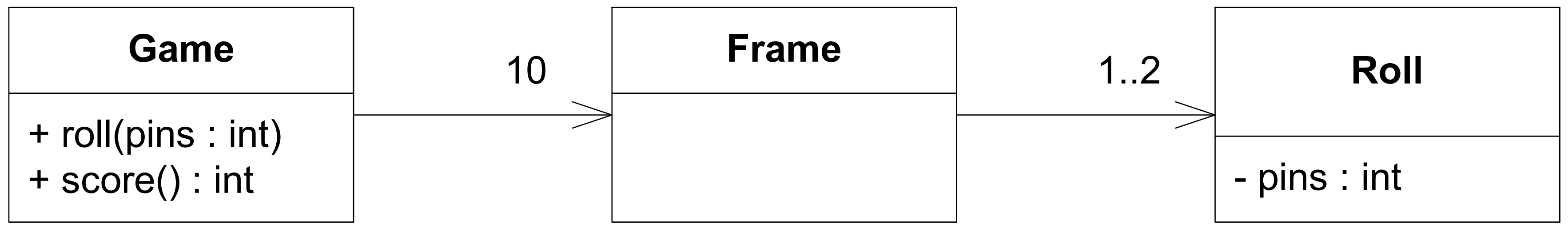
Clearly we need the Game class.

# A quick design session



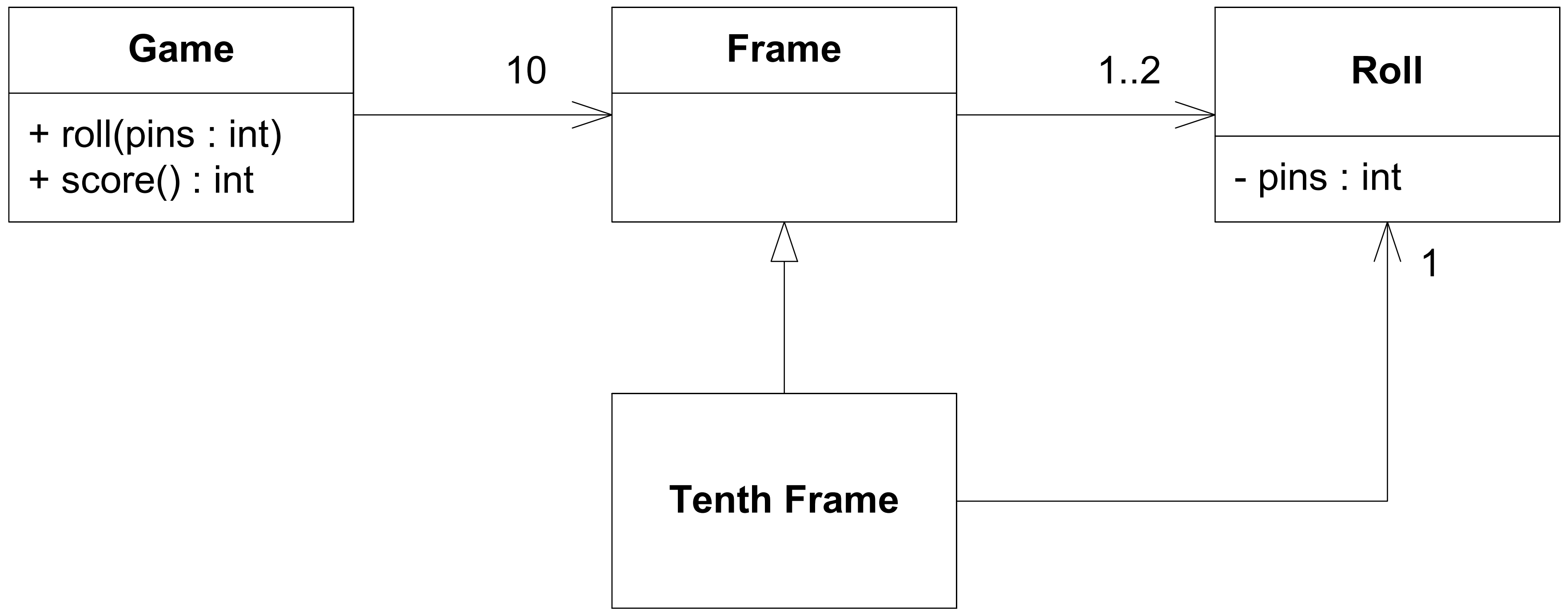
A game has 10 frames.

# A quick design session



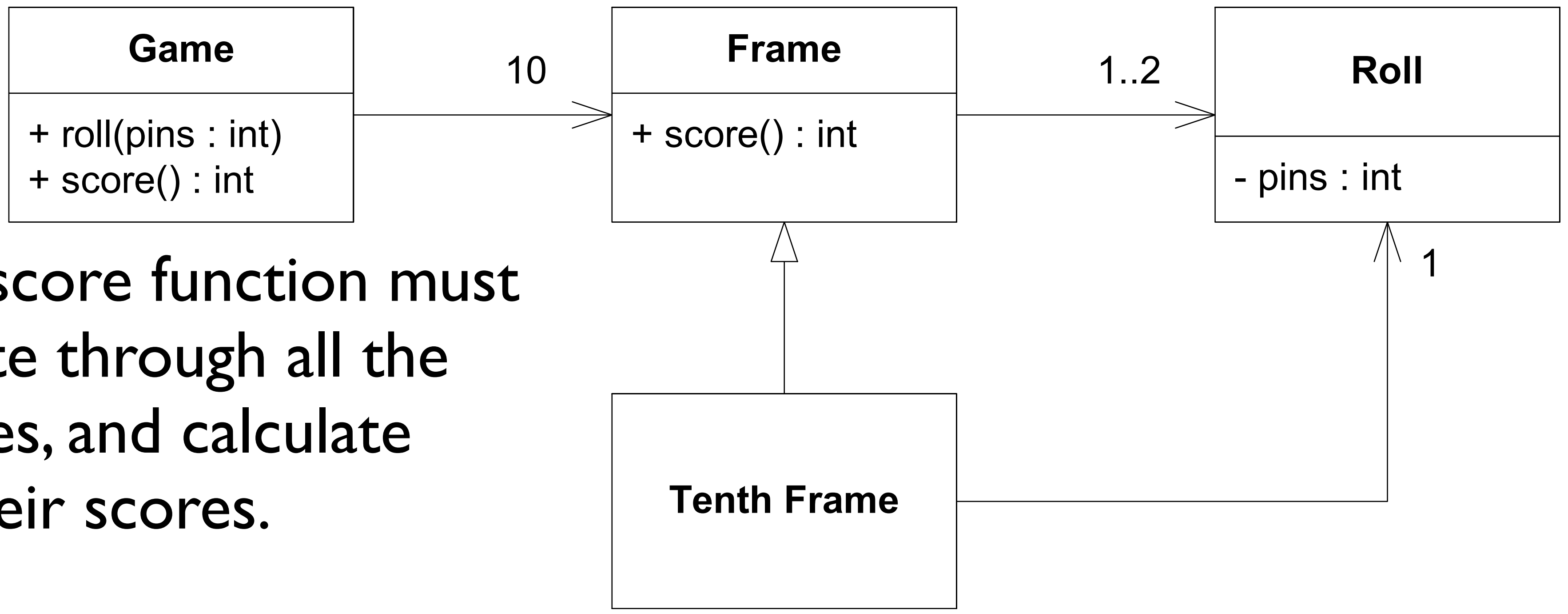
A frame has 1 or 2 rolls.

# A quick design session



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

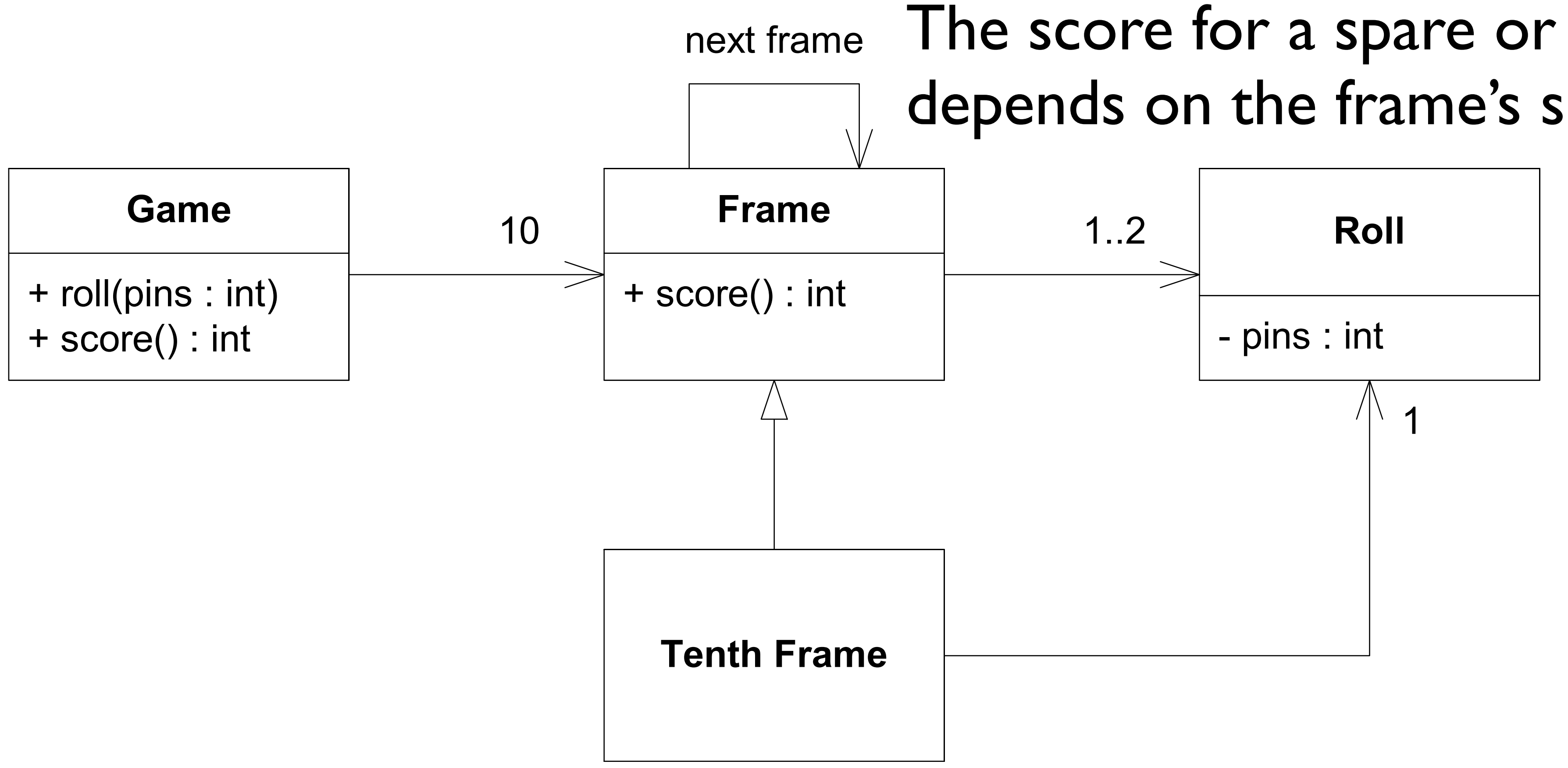
# A quick design session



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



# Xcode one-time set-up

- Go to qualitycoding.org/resources
- Download “Unit Test Template” and install using Terminal
- Relaunch Xcode 4
- In Preferences → Behaviors, enable “Show bezel alert” for Testing succeeds

# Begin.

- Create a new iOS project of type “Cocoa Touch Static Library”
- Name it BowlingGame and make sure “Include Unit Tests” and “Use Automatic Reference Counting” are both checked
- Delete BowlingGameTests.h and .m
- Add new file of type “OCUnit test case (iOS)” named GameTest. Make sure it goes in the tests target, not the main target.

## Verify that tests run.

- Add new file of type “Objective-C class” named Game.
- In GameTest, edit #import replacing Class Header with Game.h
- Select iPhone Simulator in Scheme picker
- ⌘U to run unit tests
- Verify that you get error in testExample
- Delete testExample and run tests again

## The first test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
@end
```

```
#import "Game.h"

@implementation Game
@end
```

## The first test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
@end
```

```
#import "Game.h"

@implementation Game
@end
```

# The first test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
@end
```

```
#import "Game.h"

@implementation Game

- (void)rollWithPinCount:(int)pins {
}

@end
```

# The first test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game

- (void)rollWithPinCount:(int)pins {
}

- (int)score {
    return -1;
}

@end
```

```
'-1' should be equal to '0'
```

# The first test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game

- (void)rollWithPinCount:(int)pins {
}

- (int)score {
    return 0;
}

@end
```

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game

- (void)rollWithPinCount:(int)pins {
}

- (int)score {
    return 0;
}

@end
```

- Game creation is duplicated
- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game

- (void)rollWithPinCount:(int)pins {
}

- (int)score {
    return 0;
}

@end
```

- Game creation is duplicated
- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game

- (void)rollWithPinCount:(int)pins {
}

- (int)score {
    return 0;
}

@end
```

```
'0' should be equal to '20'
```

- Game creation is duplicated
- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- Game creation is duplicated
- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest

- (void)testGutterGame {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    Game *game = [[Game alloc] init];
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

- (void)setUp {
    [super setUp];
    game = [[Game alloc] init];
}

- (void)tearDown {
    game = nil;
    [super tearDown];
}

- (void)testGutterGame {
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:0];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

- (void)setUp {
    [super setUp];
    game = [[Game alloc] init];
}

- (void)tearDown {
    game = nil;
    [super tearDown];
}

- (void)testGutterGame {
    int n = 20;
    int pins = 0;
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

*Refactor menu / Extract*

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

- (void)setUp {
    [super setUp];
    game = [[Game alloc] init];
}

- (void)tearDown {
    game = nil;
    [super tearDown];
}

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];

}
- (void)testGutterGame {
    int n = 20;
    int pins = 0;
    [self rollPins:pins times:n];

    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- roll loop is duplicated

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

- (void)setUp {
    [super setUp];
    game = [[Game alloc] init];
}

- (void)tearDown {
    game = nil;
    [super tearDown];
}

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    for (int i = 0; i < 20; ++i)
        [game rollWithPinCount:1];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

# The second test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

- (void)setUp {
    [super setUp];
    game = [[Game alloc] init];
}

- (void)tearDown {
    game = nil;
    [super tearDown];
}

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5]; // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

```
'13' should be equal to '16'
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

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

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

*-rollWithPinCount: 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.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

//- (void)testOneSpare {
//    [game rollWithPinCount:5];
//    [game rollWithPinCount:5];  // spare
//    [game rollWithPinCount:3];
//    [self rollPins:0 times:17];
//    STAssertEquals([game score], 16, nil);
//}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
}

- (int)score {
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

//- (void)testOneSpare {
//    [game rollWithPinCount:5];
//    [game rollWithPinCount:5];  // spare
//    [game rollWithPinCount:3];
//    [self rollPins:0 times:17];
//    STAssertEquals([game score], 16, nil);
//}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    for (int i = 0; i < 21; ++i)
        score += rolls[i];
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

//- (void)testOneSpare {
//    [game rollWithPinCount:5];
//    [game rollWithPinCount:5];  // spare
//    [game rollWithPinCount:3];
//    [self rollPins:0 times:17];
//    STAssertEquals([game score], 16, nil);
//}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int score;
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    score += pins;
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    for (int i = 0; i < 21; ++i)
        score += rolls[i];
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

//- (void)testOneSpare {
//    [game rollWithPinCount:5];
//    [game rollWithPinCount:5];  // spare
//    [game rollWithPinCount:3];
//    [self rollPins:0 times:17];
//    STAssertEquals([game score], 16, nil);
//}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    for (int i = 0; i < 21; ++i)
        score += rolls[i];
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    for (int i = 0; i < 21; ++i)
        score += rolls[i];
    return score;
}

@end
```

```
'13' should be equal to '16'
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    for (int i = 0; i < 21; ++i) {
        if (rolls[i] + rolls[i+1] == 10)  // spare
            score += ...
        score += rolls[i];
    }
    return score;
}

@end
```

*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.

```objc
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

//- (void)testOneSpare {
//    [game rollWithPinCount:5];
//    [game rollWithPinCount:5];  // spare
//    [game rollWithPinCount:3];
//    [self rollPins:0 times:17];
//    STAssertEquals([game score], 16, nil);
//}

@end
```

```objc
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```objc
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int i = 0;
    for (int frame = 0; frame < 10; ++frame) {
        score += rolls[i] + rolls[i + 1];
        i += 2;
    }
    return score;
}

@end
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int i = 0;
    for (int frame = 0; frame < 10; ++frame) {
        score += rolls[i] + rolls[i + 1];
        i += 2;
    }
    return score;
}

@end
```

```
'13' should be equal to '16'
```

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int i = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if (rolls[i] + rolls[i + 1] == 10)  // spare
        {
            score += 10 + rolls[i + 2];
            i += 2;
        } else {
            score += rolls[i] + rolls[i + 1];
            i += 2;
        }
    }
    return score;
}

@end
```

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

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int i = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if (rolls[i] + rolls[i + 1] == 10) // spare
        {
            score += 10 + rolls[i + 2];
            i += 2;
        } else {
            score += rolls[i] + rolls[i + 1];
            i += 2;
        }
    }
    return score;
}

@end
```

- ugly comment in test
- ugly comment in conditional

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import <Foundation/Foundation.h>

@interface Game : NSObject
- (void)rollWithPinCount:(int)pins;
- (int)score;
@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if (rolls[ballIndex] +
            rolls[ballIndex + 1] == 10)  // spare
        {
            score += 10 + rolls[ballIndex + 2];
            ballIndex += 2;
        } else {
            score += rolls[ballIndex] +
                     rolls[ballIndex + 1];
            ballIndex += 2;
        }
    }
    return score;
}

@end
```

*Renamed using "Edit All in Scope"*

- ugly comment in test

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)testOneSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];  // spare
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if ([self isSpare:ballIndex]) {
            score += 10 + rolls[ballIndex + 2];
            ballIndex += 2;
        } else {
            score += rolls[ballIndex] +
                     rolls[ballIndex + 1];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

@end
```

# The third test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollPins:(int)pins times:(int)n {
    for (int i = 0; i < n; ++i)
        [game rollWithPinCount:pins];
}

- (void)testGutterGame {
    [self rollPins:0 times:20];
    STAssertEquals([game score], 0, nil);
}

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if ([self isSpare:ballIndex]) {
            score += 10 + rolls[ballIndex + 2];
            ballIndex += 2;
        } else {
            score += rolls[ballIndex] +
                     rolls[ballIndex + 1];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

@end
```

- ugly comment in test

# The fourth test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [game rollWithPinCount:10]; // strike
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if ([self isSpare:ballIndex]) {
            score += 10 + rolls[ballIndex + 2];
            ballIndex += 2;
        } else {
            score += rolls[ballIndex] +
                     rolls[ballIndex + 1];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

@end
```

```
'17' should be equal to '24'
```

- ugly comment in test

# The fourth test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [game rollWithPinCount:10];  // strike
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if (rolls[ballIndex] == 10)  // strike
        {
            score += 10 +
                     rolls[ballIndex + 1] +
                     rolls[ballIndex + 2];
            ++ballIndex;
        } else if ([self isSpare:ballIndex]) {
            score += 10 + rolls[ballIndex + 2];
            ballIndex += 2;
        } else {
            score += rolls[ballIndex] +
                     rolls[ballIndex + 1];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

@end
```

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

# The fourth test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [game rollWithPinCount:10];  // strike
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if (rolls[ballIndex] == 10)  // strike
        {
            score += 10 +
                     rolls[ballIndex + 1] +
                     rolls[ballIndex + 2];
            ++ballIndex;
        } else if ([self isSpare:ballIndex]) {
            score += 10 + rolls[ballIndex + 2];
            ballIndex += 2;
        } else {
            score += rolls[ballIndex] +
                     rolls[ballIndex + 1];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

@end
```

- ugly comment in test
- ugly comment in conditional

# The fourth test.

```
...

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [game rollWithPinCount:10];  // strike
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}
```

```
- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if (rolls[ballIndex] == 10)  // strike
        {
            score += 10 + [self strikeBonus:ballIndex];
            ++ballIndex;
        } else if ([self isSpare:ballIndex]) {
            score += 10 + [self spareBonus:ballIndex];
            ballIndex += 2;
        } else {
            score += [self sumOfBallsInFrame:ballIndex];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

- (int)strikeBonus:(int)ballIndex {
    return rolls[ballIndex + 1] + rolls[ballIndex + 2];
}

- (int)spareBonus:(int)ballIndex {
    return rolls[ballIndex + 2];
}

- (int)sumOfBallsInFrame:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1];
}

@end
```

- ugly comment in test

# The fourth test.

```
...

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [game rollWithPinCount:10];  // strike
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}
```

```
- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if ([self isStrike:ballIndex]) {
            score += 10 + [self strikeBonus:ballIndex];
            ++ballIndex;
        } else if ([self isSpare:ballIndex]) {
            score += 10 + [self spareBonus:ballIndex];
            ballIndex += 2;
        } else {
            score += [self sumOfBallsInFrame:ballIndex];
            ballIndex += 2;
        }
    }
    return score;
}

- (BOOL)isStrike:(int)ballIndex {
    return rolls[ballIndex] == 10;
}

- (BOOL)isSpare:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1] == 10;
}

- (int)strikeBonus:(int)ballIndex {
    return rolls[ballIndex + 1] + rolls[ballIndex + 2];
}

- (int)spareBonus:(int)ballIndex {
    return rolls[ballIndex + 2];
}

- (int)sumOfBallsInFrame:(int)ballIndex {
    return rolls[ballIndex] + rolls[ballIndex + 1];
}

@end
```

# The fourth test.

```
#import "Game.h"
#import <SenTestingKit/SenTestingKit.h>

@interface GameTest : SenTestCase
@end

@implementation GameTest {
    Game *game;
}

...

- (void)rollStrike {
    [game rollWithPinCount:10];
}

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [self rollStrike];
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if ([self isStrike:ballIndex]) {
            score += 10 + [self strikeBonus:ballIndex];
            ++ballIndex;
        } else if ([self isSpare:ballIndex]) {
            score += 10 + [self spareBonus:ballIndex];
            ballIndex += 2;
        } else {
            score += [self sumOfBallsInFrame:ballIndex];
            ballIndex += 2;
        }
    }
    return score;
}

...
```

# The fifth test.

```
...

- (void)testAllOnes {
    [self rollPins:1 times:20];
    STAssertEquals([game score], 20, nil);
}

- (void)rollStrike {
    [game rollWithPinCount:10];
}

- (void)rollSpare {
    [game rollWithPinCount:5];
    [game rollWithPinCount:5];
}

- (void)testOneSpare {
    [self rollSpare];
    [game rollWithPinCount:3];
    [self rollPins:0 times:17];
    STAssertEquals([game score], 16, nil);
}

- (void)testOneStrike {
    [self rollStrike];
    [game rollWithPinCount:3];
    [game rollWithPinCount:4];
    [self rollPins:0 times:16];
    STAssertEquals([game score], 24, nil);
}

- (void)testPerfectGame {
    [self rollPins:10 times:12];
    STAssertEquals([game score], 300, nil);
}

@end
```

```
#import "Game.h"

@implementation Game {
    int rolls[21];
    int currentRoll;
}

- (void)rollWithPinCount:(int)pins {
    rolls[currentRoll++] = pins;
}

- (int)score {
    int score = 0;
    int ballIndex = 0;
    for (int frame = 0; frame < 10; ++frame) {
        if ([self isStrike:ballIndex]) {
            score += 10 + [self strikeBonus:ballIndex];
            ++ballIndex;
        } else if ([self isSpare:ballIndex]) {
            score += 10 + [self spareBonus:ballIndex];
            ballIndex += 2;
        } else {
            score += [self sumOfBallsInFrame:ballIndex];
            ballIndex += 2;
        }
    }
    return score;
}

...
```

# End

**QualityCoding.org**