@drpicox 2020

JS

Bowling Game Kata

Object Mentor, Inc.

www.objectmentor.com
blog.objectmentor.com

fitnesse.org

XProgramming.com

JU.org

www.junit.org

Origin

» Robert C. Martin created this Kata in 2005

» You can see how he solves it at:

https://cleancoders.com/video-details/clean-code-episode-6-p2

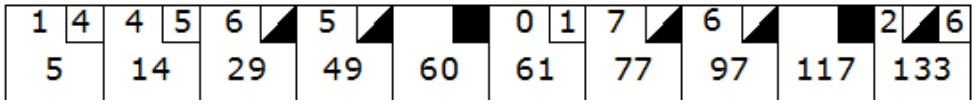
» The Objective is create «*Muscle Memory*»

Repeat frequently until be natural following the steps in all your developments

Plan

1. Analysing bowling score algotithm.
   (Robert C Martin original slides)
2. What is TDD?
3. The Kata.

Scoring Bowling.



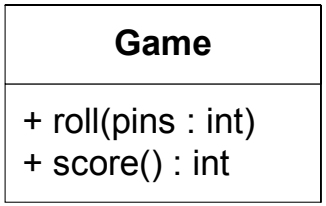
The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

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.

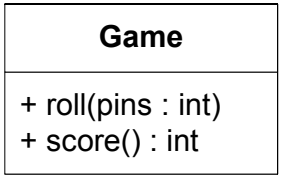
In the tenth frame a player who rolls a spare or strike is allowed to roll the extra balls to complete the frame. However no more than three balls can be rolled in tenth frame.

The Requirements.



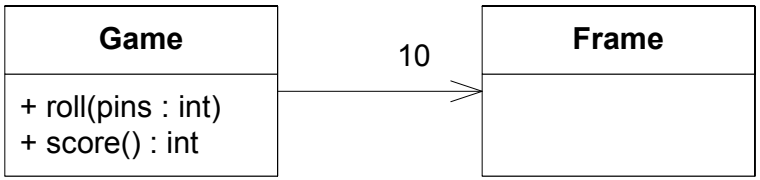
- Write a class named “Game” that has two methods
  - roll(pins : int) is called each time the player rolls a ball. The argument is the number of pins knocked down.
  - score() : int is called only at the very end of the game. It returns the total score for that game.

A quick design session



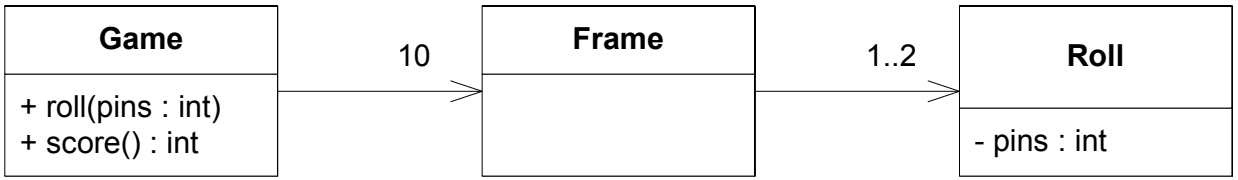
Clearly we need the Game class.

A quick design session



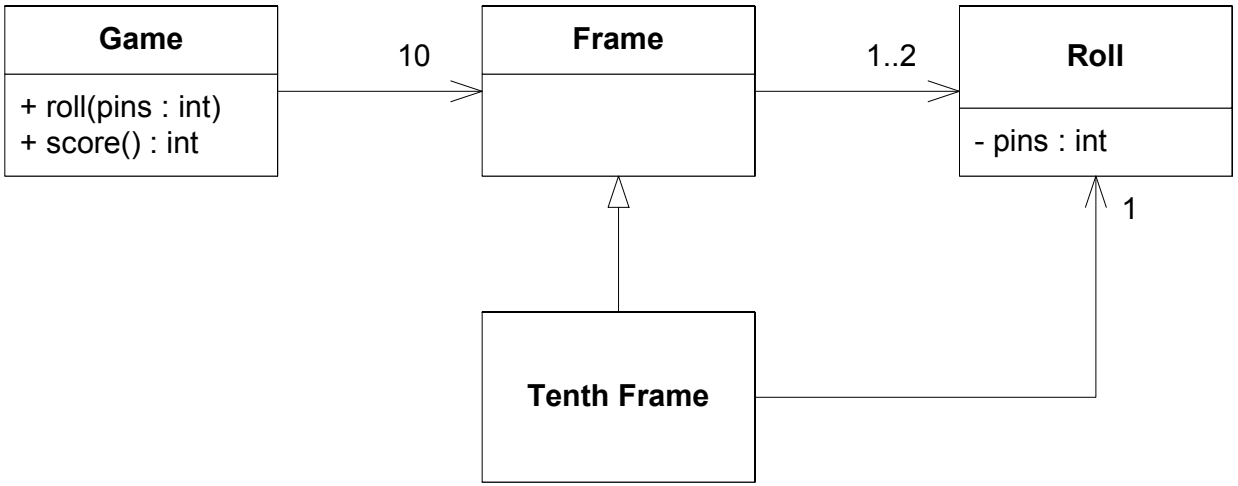
A game has 10 frames.

A quick design session



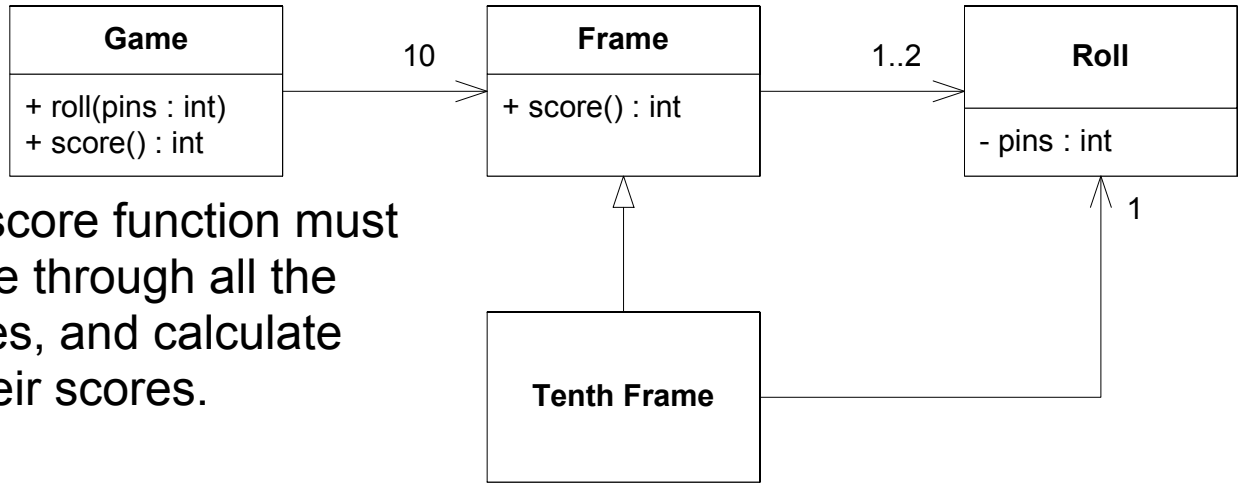
A frame has 1 or two rolls.

A quick design session



The tenth frame has two or three rolls.
It is different from all the other frames.

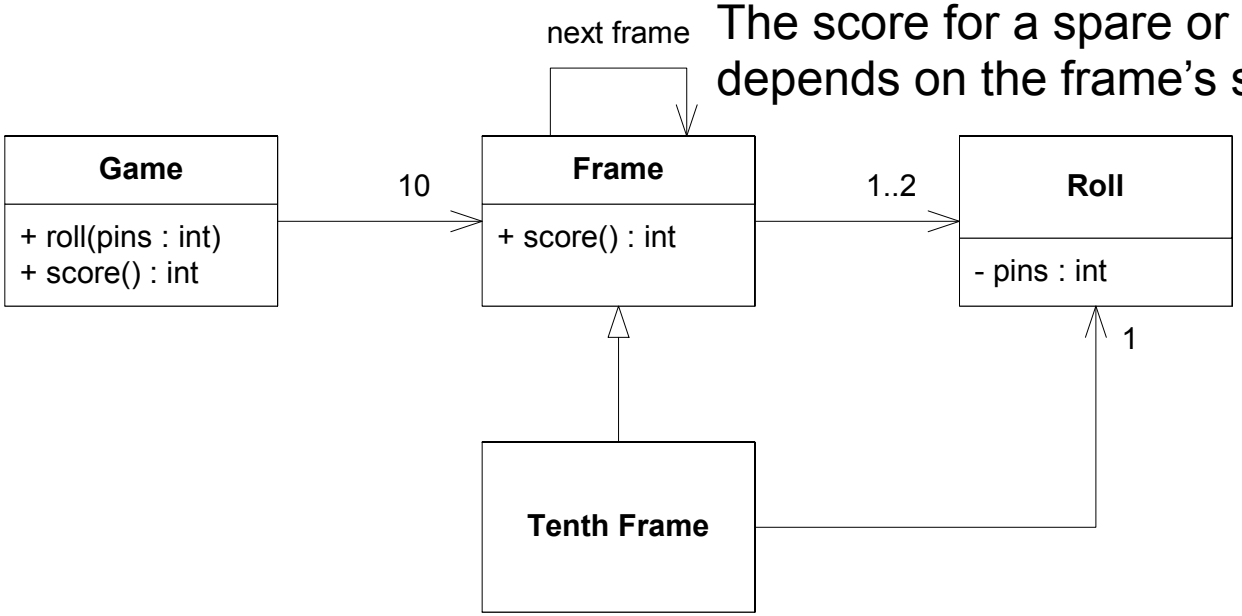
A quick design session



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



What is TDD?

» Three Rules (it is a discipline)

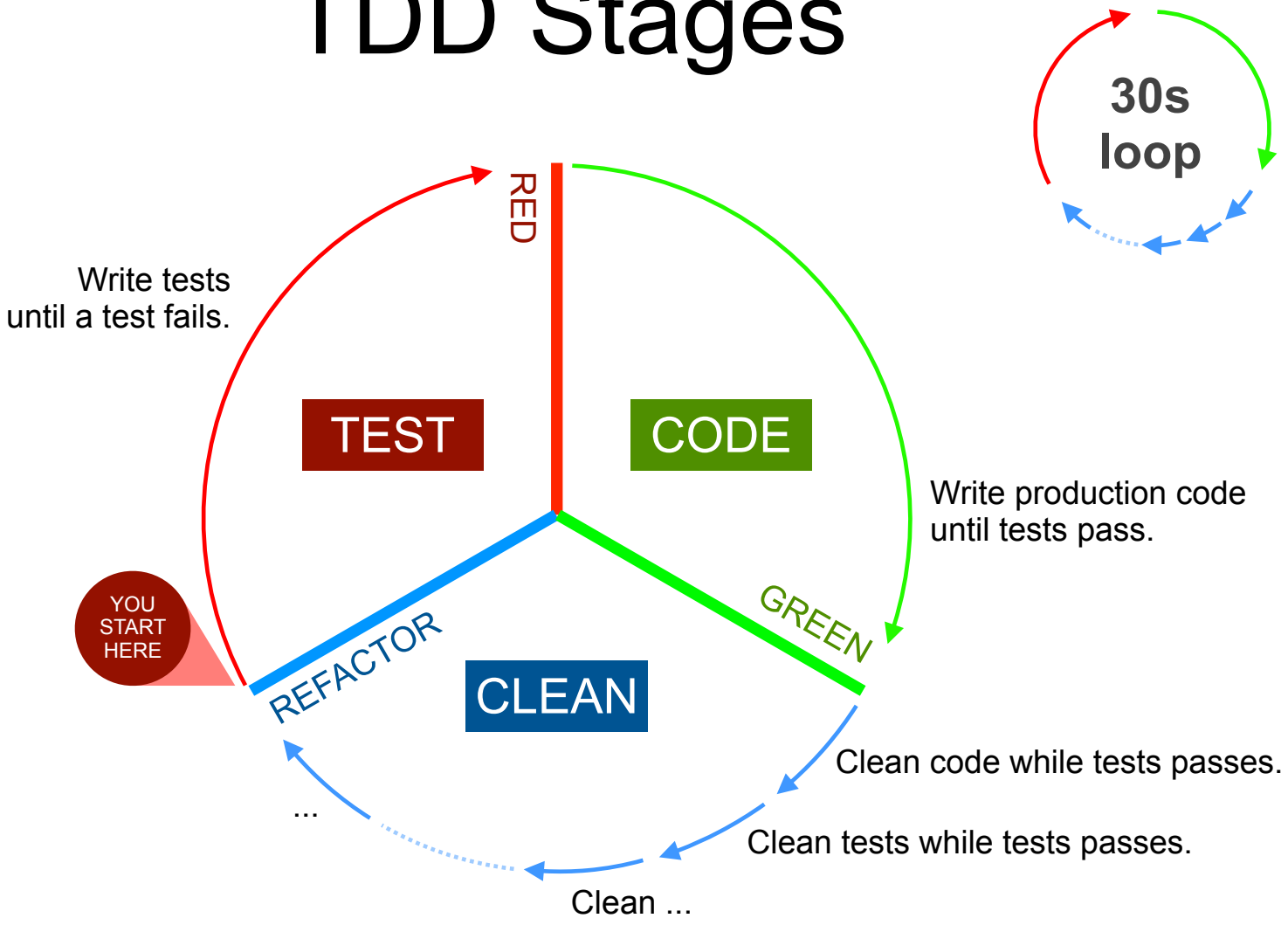
1. You are not allowed to write any production code unless it is to make a failing test pass.

2. You are not allowed to write any more of a test than is sufficient to fail; and compilation failures are failures.

3. You are not allowed to write any more production code than is sufficient to pass the one failing test.

http://www.butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

TDD Stages



Test-Driven Development By Example — Kent Beck

Begin.

- Create the BowlingGame project
- Create a test file `bowling.spec.js`

```
// bowling.spec.js
```

Begin.

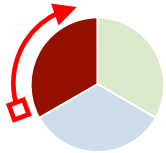
- Create the BowlingGame project
- Create a test file `bowling.spec.js`

```
// bowling.spec.js
```

Execute the test and verify that you get the following error:

```
Your test suite must contain at least one test.
```

The First test.

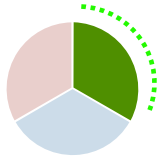


TEST

```
// bowling.spec.js
test('gutter game', () => {
  const g = new Game();
});
```

```
ReferenceError: Game is not defined
```

The First test.




```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
});
```

```
// bowling.js
export default class Game {}
```

commit 2

The First test.


```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
});
```

```
// bowling.js
export default class Game {}
```

The First test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
});
```

```
// bowling.js
export default class Game {}
```

Nothing to clean

commit 3

The First test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
});
```

```
// bowling.js
export default class Game {}
```

```
TypeError: g.roll is not a function
```

commit 4

The First test.


```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
});
```

```
// bowling.js
export default class Game {
  roll() {}
}
```

The First test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
});
```

```
// bowling.js
export default class Game {
  roll() {}
}
```

Nothing to clean

commit 5

The First test.

TEST

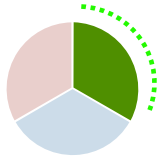
```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});
```

```
// bowling.js
export default class Game {
  roll() {}
}
```

```
TypeError: g.score is not a function
```

commit 6

The First test.



CODE

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {}
}
```

```
Expected: 0. Received: undefined.
```

commit 7

The First test.


```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {
    return 0;
  }
}
```

The First test.

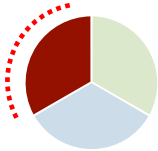
CLEAN

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {
    return 0;
  }
}
```

Nothing to clean

The Second test.



TEST

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {
    return 0;
  }
}
```

```
TypeError: g.score is not a function
```

- Roll loop is duplicated

The Second test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {
    return 0;
  }
}
```

- Roll loop is duplicated
- Game creation duplicated

The Second test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {
    return 0;
  }
}
```

commit 8

- Roll loop is duplicated
- Game creation duplicated

The Second test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  roll() {}

  score() {
    return 0;
  }
}
```

Expected: 20. Received: 0.

commit 9

- Roll loop is duplicated
- Game creation duplicated

The Second test.

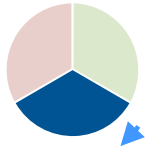

```
// bowling.spec.js
import Game from "./bowling";

test("gutter game", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 10

- Roll loop is duplicated
- Game creation duplicated

The Second test.



CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

test("gutter game", () => {
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  const g = new Game();
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

X

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

- Roll loop is duplicated
- Game creation duplicated

The Second test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

test("gutter game", () => {
  for (let i = 0; i < 20; i++)
    g.roll(0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  for (let i = 0; i < 20; i++)
    g.roll(1);
  expect(g.score()).toBe(20);
});
```

X

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 12

- Roll loop is duplicated
- ~~Game creation duplicated~~

The Second test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

test("gutter game", () => {
  const pins = 0;
  const rolls = 20;
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);

  expect(g.score()).toBe(0);
});

test("all ones", () => {
  for (let i = 0; i < 20; i += 1)
    g.roll(1);

  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 13

- Roll loop is duplicated
- ~~Game creation duplicated~~

The Second test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

test("gutter game", () => {
  const pins = 0;
  const rolls = 20;
  rollMany(rolls, pins);

  expect(g.score()).toBe(0);
});

test("all ones", () => {
  for (let i = 0; i < 20; i += 1)
    g.roll(1);

  expect(g.score()).toBe(20);
});

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 14

- Roll loop is duplicated
- ~~Game creation duplicated~~

The Second test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  for (let i = 0; i < 20; i += 1)
    g.roll(1);

  expect(g.score()).toBe(20);
});

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 15

- Roll loop is duplicated
- ~~Game creation duplicated~~

The Second test.

CLEAN

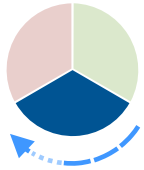
```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 16

- Roll loop is duplicated
- ~~Game creation duplicated~~

The Second test.

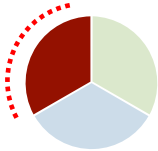


CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

The Third test.




```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

- ugly comment in test.

The Third test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 17

- ugly comment in test.

The Third test.

TEST

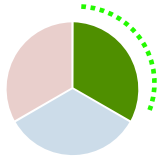
```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

```
Expected: 16. Received: 13.
```

- ugly comment in test.

The Third test.




```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

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

- ugly comment in test.

The Third test.


```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

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

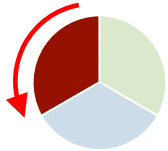
score() does not calculate score, but name implies that it does.

Design is wrong. Responsibilities are misplaced.

commit 18

- ugly comment in test.

The Third test.



TEST

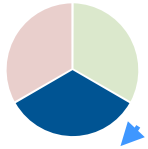
```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

```
// bowling.js
export default class Game {
  _score = 0;

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

- ugly comment in test.

commit 19

The Third test.



CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

```
// bowling.js
export default class Game {
  _score = 0;
  _rolls = [];

  roll(pins) {
    this._score += pins;
  }

  score() {
    return this._score;
  }
}
```

commit 20

- ugly comment in test.

The Third test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

```
// bowling.js
export default class Game {
  _score = 0;
  _rolls = [];

  roll(pins) {
    this._score += pins;
    this._rolls.push(pins);
  }

  score() {
    return this._score;
  }
}
```

- ugly comment in test.

commit 21

The Third test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

```
// bowling.js
export default class Game {
  _score = 0;
  _rolls = [];

  roll(pins) {
    this._score += pins;
    this._rolls.push(pins);
  }

  score() {
    let score = 0;
    for (let i = 0; i < this._rolls.length; i++) {
      score += this._rolls[i];
    }
    return score;
  }
}
```

- ugly comment in test.

commit 22

The Third test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

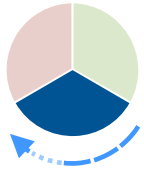
X

```
// bowling.js
export default class Game {
  _score = 0;
  _rolls = [];

  roll(pins) {
   this._rolls.push(pins);
  }

  score() {
    let score = 0;
    for (let i = 0; i < this._rolls.length; i++) {
      score += this._rolls[i];
    }
    return score;
  }
}
```

commit 23

- ugly comment in test.

The Third test.



CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

X

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
   this._rolls.push(pins);
  }

  score() {
    let score = 0;
    for (let i = 0; i < this._rolls.length; i++) {
      score += this._rolls[i];
    }
    return score;
  }
}
```

- ugly comment in test.

commit 24

The Third test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
   this._rolls.push(pins);
  }

  score() {
    let score = 0;
    for (let i = 0; i < this._rolls.length; i++) {
      score += this._rolls[i];
    }
    return score;
  }
}
```

```
Expected: 16. Received: 13.
```

- ugly comment in test.

The Third test.


```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    for (let i = 0; i < rolls.length; i++) {
      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.

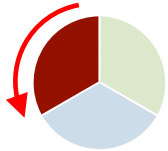
Design is still wrong.

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

- ugly comment in test.

commit 25

The Third test.



TEST

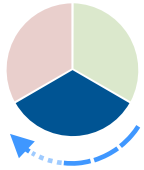
```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    let score = 0;
    for (let i = 0; i < this._rolls.length; i++) {
      score += this._rolls[i];
    }
    return score;
  }
}
```

- ugly comment in test.

commit 26

The Third test.



CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

// test("one spare", () => {
//   g.roll(5);
//   g.roll(5); // spare
//   g.roll(3);
//   rollMany(17, 0);
//   expect(g.score()).toBe(16);
// });
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let i = 0;
    for (let frame = 0; frame < 10; frame++) {
      score += rolls[i] + rolls[i + 1];
      i += 2;
    }
    return score;
  }
}
```

- ugly comment in test.

commit 27

The Third test.

TEST

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let i = 0;
    for (let frame = 0; frame < 10; frame++) {
      score += rolls[i] + rolls[i + 1];
      i += 2;
    }
    return score;
  }
}
```

```
Expected: 16. Received: 13.
```

- ugly comment in test.

commit 28

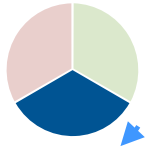
The Third test.


```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let i = 0;
    for (let 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;
  }
}
```

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

The Third test.



CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let i = 0;
    for (let 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;
  }
}
```

commit 29

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

The Third test.

CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (rolls[frameIndex] + rolls[frameIndex + 1] == 10) {
        // spare
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex] + rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}
```

commit 30

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

The Third test.

CLEAN

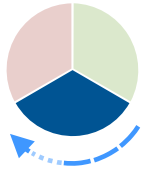
```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1)
    g.roll(pins);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  g.roll(5);
  g.roll(5); // spare
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isSpare(rolls, frameIndex)) {
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex] + rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}

function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

commit 31

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

The Third test.



CLEAN

```
// bowling.spec.js
import Game from "./bowling";

let g;
beforeEach(() => (g = new Game()));

function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1) g.roll(pins);
}

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isSpare(rolls, frameIndex)) {
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex] + rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}

function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

- ugly comment in test.

commit 32

The Fourth test.

TEST

```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];

  roll(pins) {
    this._rolls.push(pins);
  }

  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isSpare(rolls, frameIndex)) {
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex] + rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}

function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

```
Expected: 17. Received: NaN.
```

- ugly comment in test

commit 33

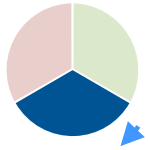
The Fourth test.


```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];
  roll(pins) {
    this._rolls.push(pins);
  }
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (rolls[frameIndex] == 10) {
        // strike
        score += 10 +
          rolls[frameIndex + 1] +
          rolls[frameIndex + 2];
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex] + rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}

function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

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

The Fourth test.



CLEAN

```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
// bowling.js
export default class Game {
  _rolls = [];
  roll(pins) {
    this._rolls.push(pins);
  }
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (rolls[frameIndex] == 10) {
        // strike
        score += 10 +
          rolls[frameIndex + 1] +
          rolls[frameIndex + 2];
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex] + rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}

function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

commit 34

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

The Fourth test.

CLEAN

```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
// bowling.js
export default class Game {
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (rolls[frameIndex] == 10) {
        // strike
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + rolls[frameIndex + 2];
        frameIndex += 2;
      } else {
        score += rolls[frameIndex]+rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

commit 35

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

The Fourth test.

CLEAN

```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
// bowling.js
export default class Game {
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (rolls[frameIndex] == 10) {
        // strike
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += rolls[frameIndex]+rolls[frameIndex + 1];
        frameIndex += 2;
      }
    }
    return score;
  }
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

commit 36

- ugly comment in test.
- ugly comment in conditional.
- ~~ugly expressions.~~

The Fourth test.

CLEAN

```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
// bowling.js
export default class Game {
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (rolls[frameIndex] == 10) {
        // strike
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += sumOfBallsInFrame(rolls, frameIndex);
        frameIndex += 2;
      }
    }
    return score;
  }
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function sumOfBallsInFrame(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

commit 37

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

The Fourth test.

CLEAN

```
...

function rollSpare() {
  g.roll(5);
  g.roll(5);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  g.roll(10); // strike
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isStrike(rolls, frameIndex)) {
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += sumOfBallsInFrame(rolls, frameIndex);
        frameIndex += 2;
      }
    }
    return score;
  }
}
function isStrike(rolls, frameIndex) {
  return rolls[frameIndex] === 10;
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function sumOfBallsInFrame(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

commit 38

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

The Fourth test.

CLEAN

```
...
function rollMany(rolls, pins) {
  for (let i = 0; i < rolls; i += 1) g.roll(pins);
}
function rollSpare() {
  g.roll(5);
  g.roll(5);
}
function rollStrike() {
  g.roll(10);
}

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  rollStrike();
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});
```

```
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isStrike(rolls, frameIndex)) {
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += sumOfBallsInFrame(rolls, frameIndex);
        frameIndex += 2;
      }
    }
    return score;
  }
}
function isStrike(rolls, frameIndex) {
  return rolls[frameIndex] === 10;
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function sumOfBallsInFrame(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

The Fifth test.

CLEAN

```
...

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  rollStrike();
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});

test("perfect game", () => {
  rollMany(12, 10);
  expect(g.score()).toBe(300);
});
```

```
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isStrike(rolls, frameIndex)) {
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += sumOfBallsInFrame(rolls, frameIndex);
        frameIndex += 2;
      }
    }
    return score;
  }
}
function isStrike(rolls, frameIndex) {
  return rolls[frameIndex] === 10;
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function sumOfBallsInFrame(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

The Fifth test.

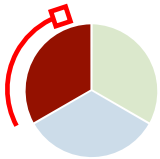
TEST

```
...

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  rollStrike();
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});

test("perfect game", () => {
  rollMany(12, 10);
  expect(g.score()).toBe("fail");
});
```

```
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isStrike(rolls, frameIndex)) {
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += sumOfBallsInFrame(rolls, frameIndex);
        frameIndex += 2;
      }
    }
    return score;
  }
}
function isStrike(rolls, frameIndex) {
  return rolls[frameIndex] === 10;
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function sumOfBallsInFrame(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```

```
Expected: 17. Received: "fail".
```

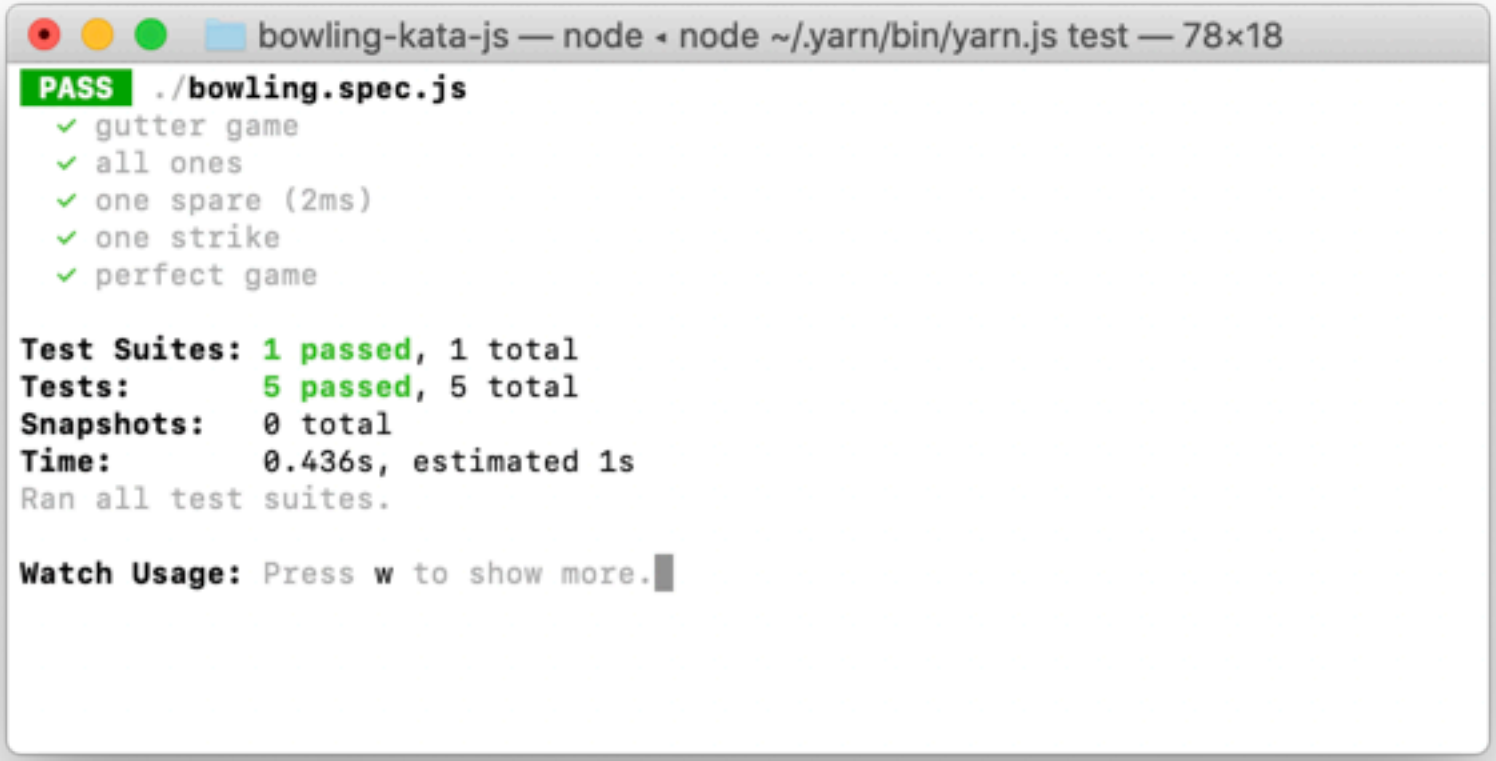
The Fifth test.



TEST

```
...

test("gutter game", () => {
  rollMany(20, 0);
  expect(g.score()).toBe(0);
});

test("all ones", () => {
  rollMany(20, 1);
  expect(g.score()).toBe(20);
});

test("one spare", () => {
  rollSpare();
  g.roll(3);
  rollMany(17, 0);
  expect(g.score()).toBe(16);
});

test("one strike", () => {
  rollStrike();
  g.roll(3);
  g.roll(4);
  rollMany(16, 0);
  expect(g.score()).toBe(24);
});

test("perfect game", () => {
  rollMany(12, 10);
  expect(g.score()).toBe(300);
});
```

```
  ...
  score() {
    const rolls = this._rolls;
    let score = 0;
    let frameIndex = 0;
    for (let frame = 0; frame < 10; frame++) {
      if (isStrike(rolls, frameIndex)) {
        score += 10 + strikeBonus(rolls, frameIndex);
        frameIndex += 1;
      } else if (isSpare(rolls, frameIndex)) {
        score += 10 + spareBonus(rolls, frameIndex);
        frameIndex += 2;
      } else {
        score += sumOfBallsInFrame(rolls, frameIndex);
        frameIndex += 2;
      }
    }
    return score;
  }
}
function isStrike(rolls, frameIndex) {
  return rolls[frameIndex] === 10;
}
function strikeBonus(rolls, frameIndex) {
  return rolls[frameIndex + 1] + rolls[frameIndex + 2];
}
function spareBonus(rolls, frameIndex) {
  return rolls[frameIndex + 2];
}
function sumOfBallsInFrame(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1];
}
function isSpare(rolls, frameIndex) {
  return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
}
```


bowling-kata-js — node ◂ node ~/.yarn/bin/yarn.js test — 78×18
PASS ./bowling.spec.js
✓ gutter game
✓ all ones
✓ one spare (2ms)
✓ one strike
✓ perfect game
Test Suites: 1 passed, 1 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 0.436s, estimated 1s
Ran all test suites.
Watch Usage: Press w to show more.

End.