#3. Client said Sometimes the armour bonus doesn’t seem to work

Index.js

class Person

{

    constructor(name, strength, dexterity, constitution, equipment)

    {

        this.name = name;

        this.strength = strength;

        this.dexterity = dexterity;

        this.constitution = constitution;

        this.hitPoints = 10 + constitution;

        this.equipment = equipment;

        this.calculateEquipment();

        // this.armorBonus = 0;

    }

    static rollDice(howMany, type)

    {

        var total = 0;

        for(var i=0;i<howMany;i++)

        {

            total += Math.round(Math.random() \* type);

        }

        return total;

    }

    attack(target)

    {

        var roll = Person.rollDice(1, 20);

        roll += this.strength;

        roll = \_.clamp(roll, 1, 20);

        var toHit = 10 + target.armorBonus + target.dexterity;

        return roll >= toHit;

    }

    addEquipment(item)

    {

        this.equipment.push(item);

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    removeEquipment(item)

    {

        for(var i=0; i<this.equipment.length; i++)

        {

            var e = this.equipment[i];

            if(e === item)

            {

                this.equipment.splice(i, 1);

            }

        }

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    calculateEquipment()

    {

        this.armorBonus = 0;

        for(var i=0; i<this.equipment.length; i++)

        {

            var item = this.equipment[i];

            if(item instanceof Armor)

            {

                this.armorBonus += item.bonus;

            }

        }

    }

}

class Armor

{

    constructor(name, bonus)

    {

        this.name = name;

        this.bonus = bonus;

    }

}

class Weapon

{

    constructor(name, bonsu, damageDieAmount, damageDieType)

    {

        this.name = name;

        this.bonsu = bonsu;

        this.damageDieAmount = damageDieAmount;

        this.damageDieType = damageDieType;

    }

}

var person;

function setupPerson()

{

    var leatherArmor = new Armor("Leather", 2);

    var shortSword = new Weapon("Short Sword", 0, 1, 6)

    person = new Person('McFly Bojo', 2, 4, 1, [leatherArmor, shortSword]);

}

setupPerson();

getPerson = () => person;

module.exports = {getPerson}

index.test.js

const log = console.log;

const \_ = require('lodash');

const should = require('chai').should();

const{

    getPerson

} = require('./index')

describe('#index initial conditions', ()=>{

    it('initial person is an object', () =>{

        const person = getPerson();

        \_.isObject(person).should.be.true;

    });

    it.only('armourBonus by default is 0 wearing leatherArmour', ()=>{

        const person = getPerson();

        person.armorBonus.should.equal(0);

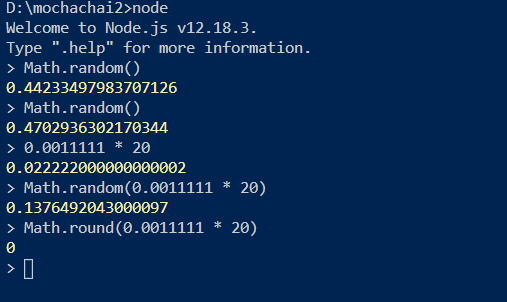
        //FIXME: should be 2 by default using leatherArmour,

        //fix is to not reset armorBonus to 0

    })

});

#4. Testing Static class methods (rollDice). Math.random has a problem



And to top it all Math.random is nested inside a for loop

Index.js

class Person

{

    constructor(name, strength, dexterity, constitution, equipment)

    {

        this.name = name;

        this.strength = strength;

        this.dexterity = dexterity;

        this.constitution = constitution;

        this.hitPoints = 10 + constitution;

        this.equipment = equipment;

        this.calculateEquipment();

        // this.armorBonus = 0;

    }

    static rollDice(howMany, type)

    {

        var total = 0;

        for(var i=0;i<howMany;i++)

        {

            total += Math.round(Math.random() \* type);

        }

        return total;

    }

    attack(target)

    {

        var roll = Person.rollDice(1, 20);

        roll += this.strength;

        roll = \_.clamp(roll, 1, 20);

        var toHit = 10 + target.armorBonus + target.dexterity;

        return roll >= toHit;

    }

    addEquipment(item)

    {

        this.equipment.push(item);

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    removeEquipment(item)

    {

        for(var i=0; i<this.equipment.length; i++)

        {

            var e = this.equipment[i];

            if(e === item)

            {

                this.equipment.splice(i, 1);

            }

        }

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    calculateEquipment()

    {

        this.armorBonus = 0;

        for(var i=0; i<this.equipment.length; i++)

        {

            var item = this.equipment[i];

            if(item instanceof Armor)

            {

                this.armorBonus += item.bonus;

            }

        }

    }

}

class Armor

{

    constructor(name, bonus)

    {

        this.name = name;

        this.bonus = bonus;

    }

}

class Weapon

{

    constructor(name, bonsu, damageDieAmount, damageDieType)

    {

        this.name = name;

        this.bonsu = bonsu;

        this.damageDieAmount = damageDieAmount;

        this.damageDieType = damageDieType;

    }

}

var person;

function setupPerson()

{

    var leatherArmor = new Armor("Leather", 2);

    var shortSword = new Weapon("Short Sword", 0, 1, 6)

    person = new Person('McFly Bojo', 2, 4, 1, [leatherArmor, shortSword]);

}

setupPerson();

getPerson = () => person;

module.exports = {getPerson, Person}

index.test.js

const log = console.log;

const \_ = require('lodash');

const should = require('chai').should();

const{

    getPerson,

    Person

} = require('./index')

// describe('#index initial conditions', ()=>{

//     it('initial person is an object', () =>{

//         const person = getPerson();

//         \_.isObject(person).should.be.true;

//     });

//     it.only('armourBonus by default is 0 wearing leatherArmour', ()=>{

//         const person = getPerson();

//         person.armorBonus.should.equal(0);

//         //FIXME: should be 2 by default using leatherArmour,

//         //fix is to not reset armorBonus to 0

//     })

// });

describe("#Person", ()=>{

    describe("#rollDice", ()=>{

        it.only("should return a finite number (not NaN nor Infinity)",() =>{

            // log("Person:", Person.rollDice);

            const number = Person.rollDice(1,20);

            \_.isFinite(number).should.be.true

            // log("number:", number)

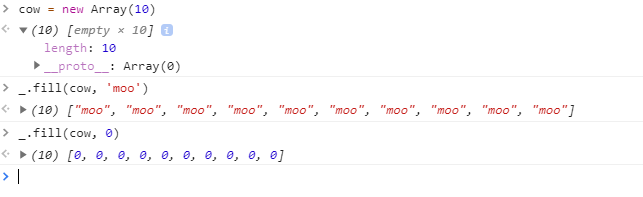
        })

    })

})

#5. Unit testing Sampling Data

Lets create a sample size.



Index.js

class Person

{

    constructor(name, strength, dexterity, constitution, equipment)

    {

        this.name = name;

        this.strength = strength;

        this.dexterity = dexterity;

        this.constitution = constitution;

        this.hitPoints = 10 + constitution;

        this.equipment = equipment;

        this.calculateEquipment();

        // this.armorBonus = 0;

    }

    static rollDice(howMany, type)

    {

        var total = 0;

        for(var i=0;i<howMany;i++)

        {

            total += Math.round(Math.random() \* type);

        }

        return total;

    }

    attack(target)

    {

        var roll = Person.rollDice(1, 20);

        roll += this.strength;

        roll = \_.clamp(roll, 1, 20);

        var toHit = 10 + target.armorBonus + target.dexterity;

        return roll >= toHit;

    }

    addEquipment(item)

    {

        this.equipment.push(item);

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    removeEquipment(item)

    {

        for(var i=0; i<this.equipment.length; i++)

        {

            var e = this.equipment[i];

            if(e === item)

            {

                this.equipment.splice(i, 1);

            }

        }

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    calculateEquipment()

    {

        this.armorBonus = 0;

        for(var i=0; i<this.equipment.length; i++)

        {

            var item = this.equipment[i];

            if(item instanceof Armor)

            {

                this.armorBonus += item.bonus;

            }

        }

    }

}

class Armor

{

    constructor(name, bonus)

    {

        this.name = name;

        this.bonus = bonus;

    }

}

class Weapon

{

    constructor(name, bonsu, damageDieAmount, damageDieType)

    {

        this.name = name;

        this.bonsu = bonsu;

        this.damageDieAmount = damageDieAmount;

        this.damageDieType = damageDieType;

    }

}

var person;

function setupPerson()

{

    var leatherArmor = new Armor("Leather", 2);

    var shortSword = new Weapon("Short Sword", 0, 1, 6)

    person = new Person('McFly Bojo', 2, 4, 1, [leatherArmor, shortSword]);

}

setupPerson();

getPerson = () => person;

module.exports = {getPerson, Person}

index.test.js

const log = console.log;

const \_ = require('lodash');

const should = require('chai').should();

const{

    getPerson,

    Person

} = require('./index')

// describe('#index initial conditions', ()=>{

//     it('initial person is an object', () =>{

//         const person = getPerson();

//         \_.isObject(person).should.be.true;

//     });

//     it.only('armourBonus by default is 0 wearing leatherArmour', ()=>{

//         const person = getPerson();

//         person.armorBonus.should.equal(0);

//         //FIXME: should be 2 by default using leatherArmour,

//         //fix is to not reset armorBonus to 0

//     })

// });

describe("#Person", ()=>{

    describe("#rollDice", ()=>{

        // it.only("should return a finite number (not NaN nor Infinity)",() =>{

        //     // log("Person:", Person.rollDice);

        //     const number = Person.rollDice(1,20);

        //     \_.isFinite(number).should.be.true

        //     // log("number:", number)

        // });

        it("should not have a 0 in a 1000 sample size", ()=>{

            const sample = new Array(1000);

            \_.fill(sample, 0);

            // log(sample);

            const rollDiceSamples = \_.map(sample, item => Person.rollDice(1,20));

            log("rollDiceSamples",rollDiceSamples);

            const anyZeros = \_.filter(rollDiceSamples, item => item === 0);

            anyZeros.length.should.equal(0);

        })

    })

})

With higher sample size there will be more zeros

#7 Code Coverage

Npm install Istanbul

Add following to package.json

"covarage":"istanbul cover \_mocha index.test.js -x \*.test.js"

For windows,

"covarage":"istanbul cover node\_modules/mocha/bin/\_mocha index.test.js -x \*.test.js"

"showcoverage":"open coverage/lcov-report/index.html"

For windows

"showcoverage":"start coverage/lcov-report/index.html"

Index.js

class Person

{

    constructor(name, strength, dexterity, constitution, equipment)

    {

        this.name = name;

        this.strength = strength;

        this.dexterity = dexterity;

        this.constitution = constitution;

        this.hitPoints = 10 + constitution;

        this.equipment = equipment;

        this.calculateEquipment();

        // this.armorBonus = 0;

    }

    static rollDice(howMany, type)

    {

        var total = 0;

        for(var i=0;i<howMany;i++)

        {

            total += Math.round(Math.random() \* type);

        }

        return total;

    }

    attack(target)

    {

        var roll = Person.rollDice(1, 20);

        roll += this.strength;

        roll = \_.clamp(roll, 1, 20);

        var toHit = 10 + target.armorBonus + target.dexterity;

        return roll >= toHit;

    }

    addEquipment(item)

    {

        this.equipment.push(item);

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    removeEquipment(item)

    {

        for(var i=0; i<this.equipment.length; i++)

        {

            var e = this.equipment[i];

            if(e === item)

            {

                this.equipment.splice(i, 1);

            }

        }

        if(item instanceof Armor)

        {

            this.calculateEquipment();

        }

    }

    calculateEquipment()

    {

        this.armorBonus = 0;

        for(var i=0; i<this.equipment.length; i++)

        {

            var item = this.equipment[i];

            if(item instanceof Armor)

            {

                this.armorBonus += item.bonus;

            }

        }

    }

}

class Armor

{

    constructor(name, bonus)

    {

        this.name = name;

        this.bonus = bonus;

    }

}

class Weapon

{

    constructor(name, bonsu, damageDieAmount, damageDieType)

    {

        this.name = name;

        this.bonsu = bonsu;

        this.damageDieAmount = damageDieAmount;

        this.damageDieType = damageDieType;

    }

}

var person;

function setupPerson()

{

    var leatherArmor = new Armor("Leather", 2);

    var shortSword = new Weapon("Short Sword", 0, 1, 6)

    person = new Person('McFly Bojo', 2, 4, 1, [leatherArmor, shortSword]);

}

setupPerson();

getPerson = () => person;

module.exports = {getPerson, Person}

index.test.js

const log = console.log;

const \_ = require('lodash');

const should = require('chai').should();

const{

    getPerson,

    Person

} = require('./index')

// describe('#index initial conditions', ()=>{

//     it('initial person is an object', () =>{

//         const person = getPerson();

//         \_.isObject(person).should.be.true;

//     });

//     it.only('armourBonus by default is 0 wearing leatherArmour', ()=>{

//         const person = getPerson();

//         person.armorBonus.should.equal(0);

//         //FIXME: should be 2 by default using leatherArmour,

//         //fix is to not reset armorBonus to 0

//     })

// });

describe("#Person", ()=>{

    describe("#rollDice", ()=>{

        it("should return a finite number (not NaN nor Infinity)",() =>{

            // log("Person:", Person.rollDice);

            const number = Person.rollDice(1,20);

            \_.isFinite(number).should.be.true

            // log("number:", number)

        });

        it("should not have a 0 in a 1000 sample size", ()=>{

            const sample = new Array(1000);

            \_.fill(sample, 0);

            // log(sample);

            const rollDiceSamples = \_.map(sample, item => Person.rollDice(1,20));

            log("rollDiceSamples",rollDiceSamples);

            const anyZeros = \_.filter(rollDiceSamples, item => item === 0);

            // anyZeros.length.should.equal(0);

        })

    })

})