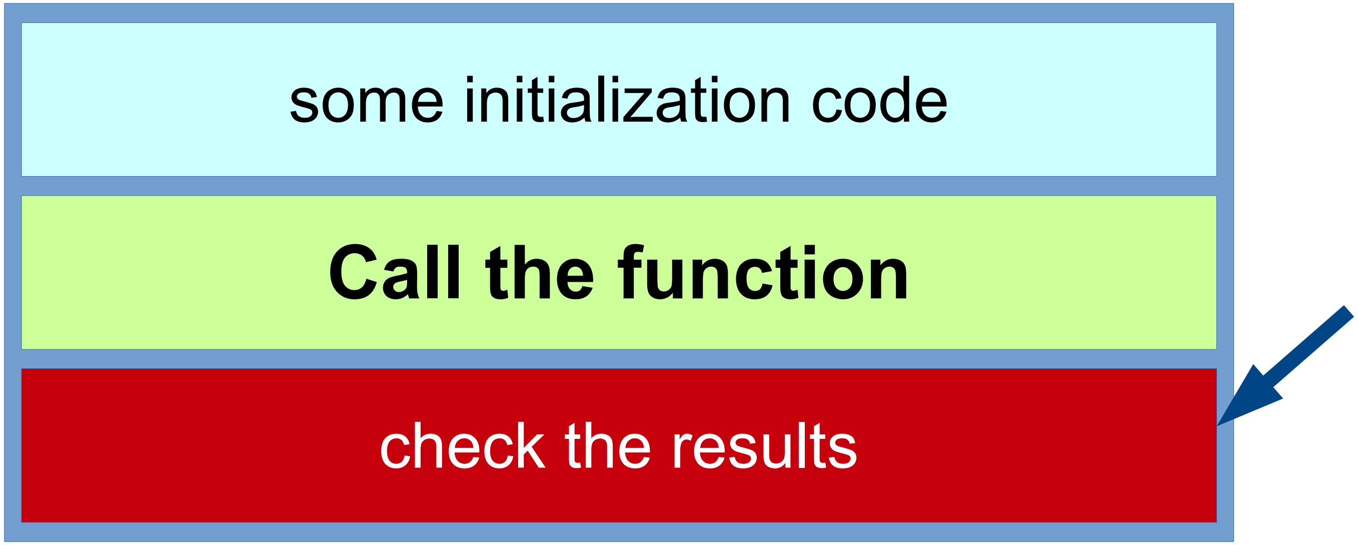
Unit testing in JavaScript with Mocha and Chai

01219245/01219246
Individual Software Process

Common structure of a test case

- How do you test a function?
  - You need to call it,
  - and check if it works correctly,
  - by looking at its return value.
- Your code would contain:



some initialization code

Call the function

check the results

The checking code is usually written as a set of **assertions**.

Our test code in Flappy Dot

```
assert( checkPlayerPillarCollision( 100, 100, 300, 200 ), false,
        'when the dot is very far left of the pillar pair' );
assert( checkPlayerPillarCollision( 300, 300, 300, 200 ), true,
        'when the dot hit the middle of the top pillar' );
```

```
var result = checkPlayerPillarCollision( 100, 100, 300, 200 );
```

```
assert( result, false,
        'when the dot is very far left of the pillar pair' );
```

The checking code is usually written as a set of **assertions**.

Testing Tools

- Test framework: Mocha
  - Calls our test methods and shows results
- Assertion library: Chai
  - Help us express our expected result
- Additional library:
  - jQuery

- Download template at:
  - http://theory.cpe.ku.ac.th/wiki/images/219245-practice.zip

The first (finished) example

```
function max3( a, b, c ) {
    if( ( a >= b ) && ( a >= c ) )
        return a;
    if( ( b >= a ) && ( b >= c ) )
        return b;
    if( ( c >= a ) && ( c >= b ) )
        return c;
}
```

```
describe( 'max3', function() {
    it( 'should return the maximum when the 1st argument is strictly maximum', function() {
        assert( max3( 10, 5, 2 ) == 10 );
    });
    it( 'should return the maximum when the 2nd argument is strictly maximum', function() {
        assert( max3( 2, 15, 5 ) == 15 );
    });
    it( 'should return the maximum when the 3rd argument is strictly maximum', function() {
        assert( max3( 5, 2, 9 ) == 9 );
    });
    it( 'should return the maximum when 1st and 2nd args are maximum', function() {
        assert( max3( 7, 7, 3 ) == 7 );
    });
    it( 'should return the maximum when 2nd and 3rd args are maximum', function() {
        assert( max3( 5, 12, 12 ) == 12 );
    });
});
```

*spaces between lines are removed so that the code fit in one page.

What do you see?

- A code with corresponding test cases.
- Enough test cases to make you feel confident about the correctness of the code.
  - Ask yourself: hide the code and look at only the test, does it make you feel comfortable to use the code?
- Enough test examples to explain what the function does.

How can we get there?

- Traditional approach
  - Write code, then write test.

- Test-driven development
  - Write test, then write code.

A few words before we start

- TDD is a well-established practice in software development in general.
- But in Game development, TDD (or even unit testing) is not a standard practice.

1st example: max3

- Let's try to work with **max3** to get to the final code as shown previously.

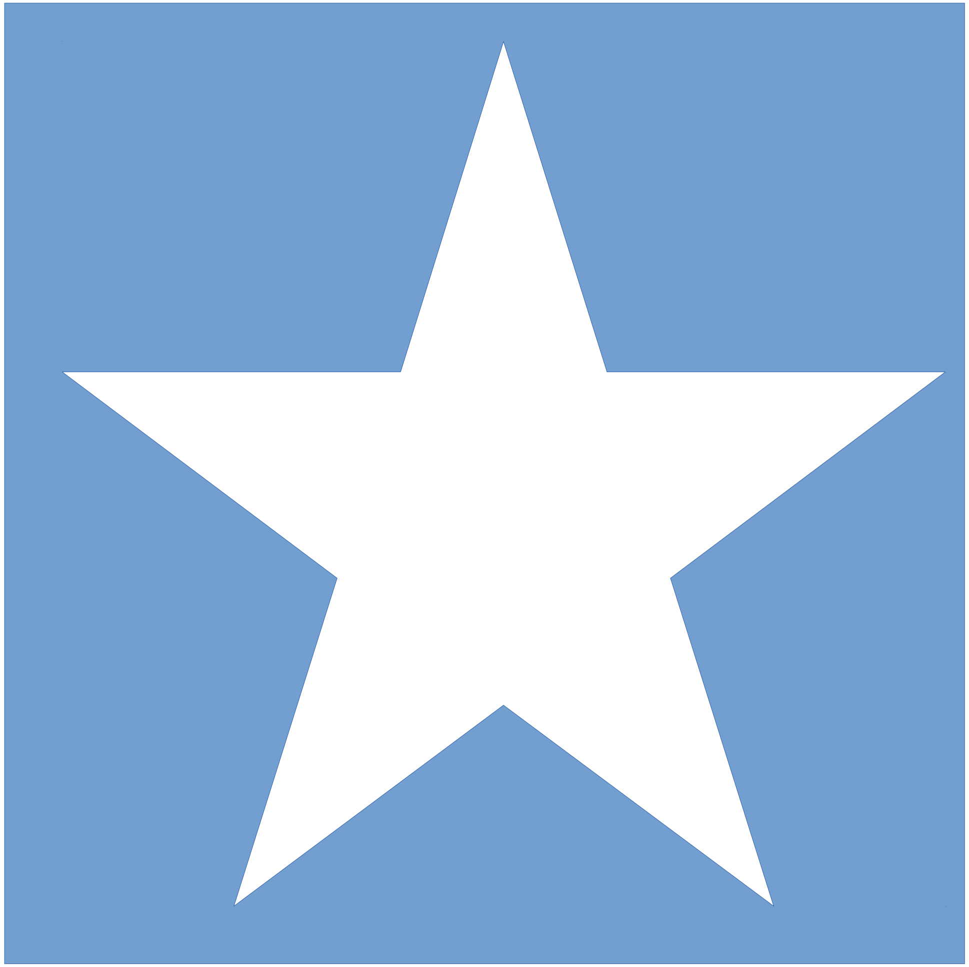
```
function max3( a, b, c ) {
}
```

- This function returns the maximum of **a**, **b**, and **c**.

How to get started

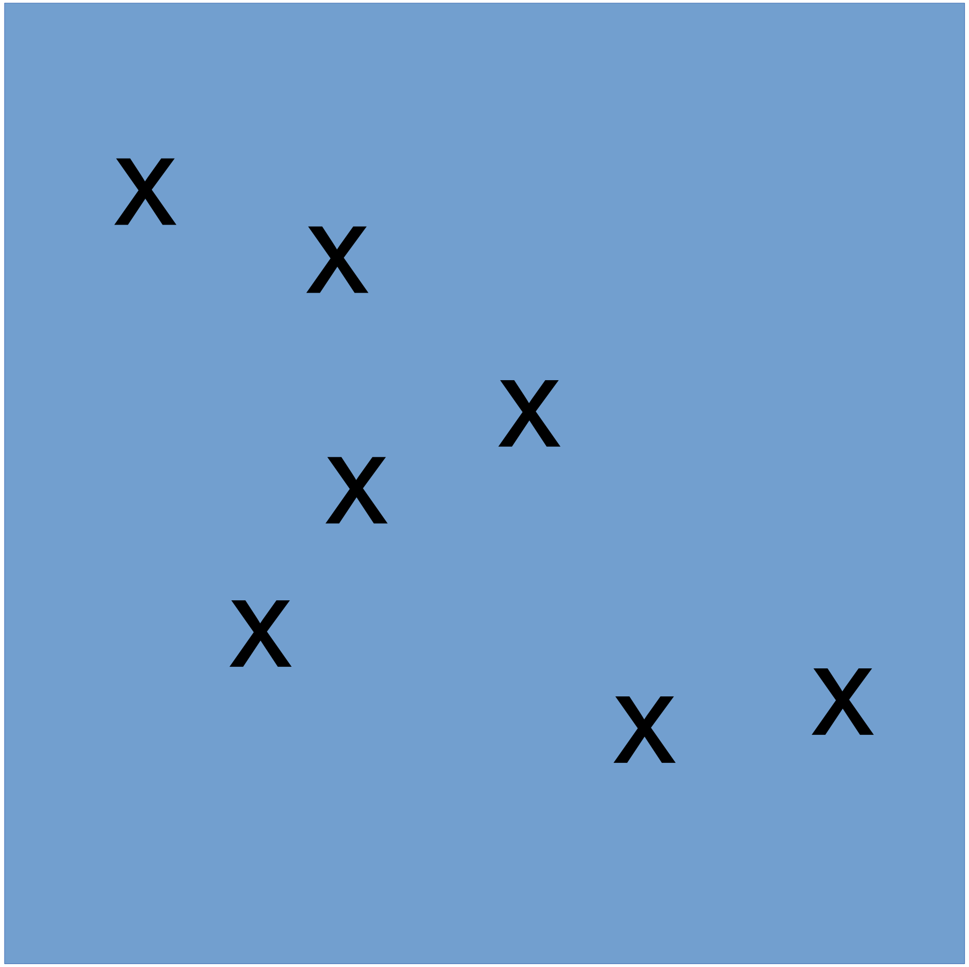
- If you are fluent with the techniques, you can just start writing test cases right away.
- But sometimes it might be easier to start by thinking about what you would like to test.
- In other words, let ask:
  - how do we know that max3 works correctly?

What's in this box?



Is it a star-shaped object?

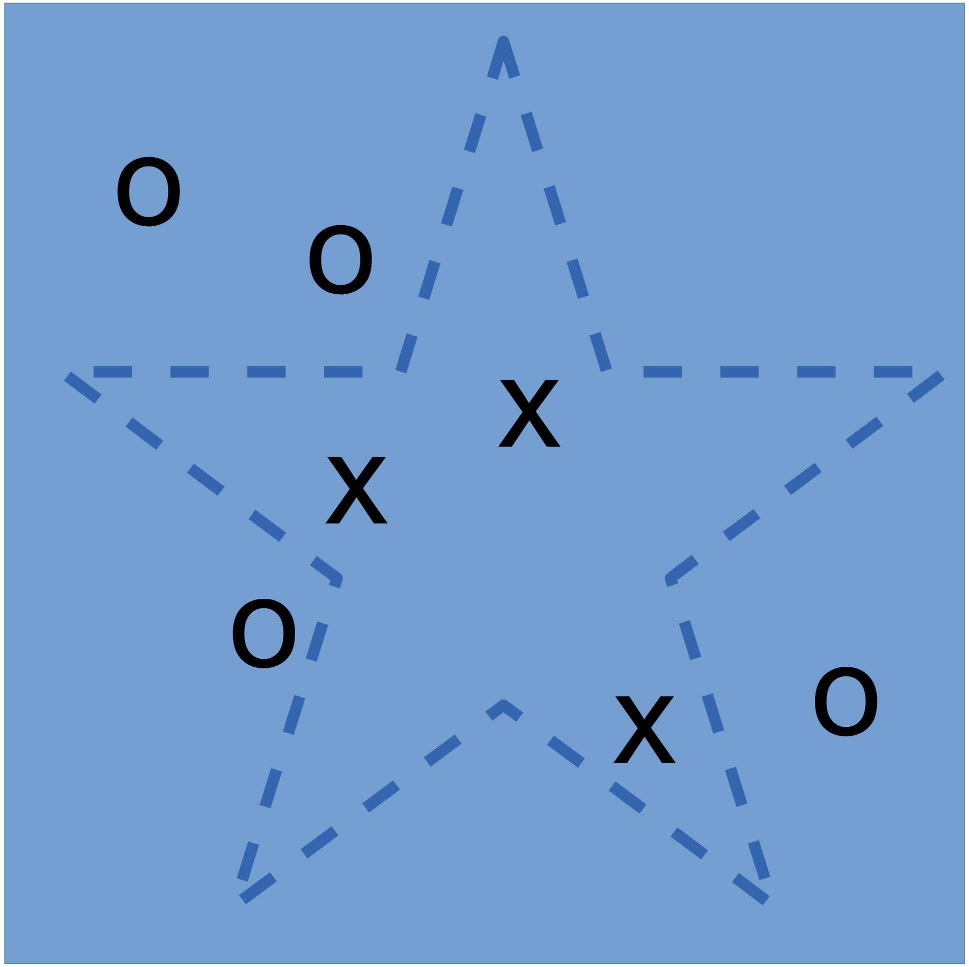
Let's try to “peak” into the box with a pin



These are the positions that we plan to use a pin to check if there is anything at that position

Is it a star-shaped object?

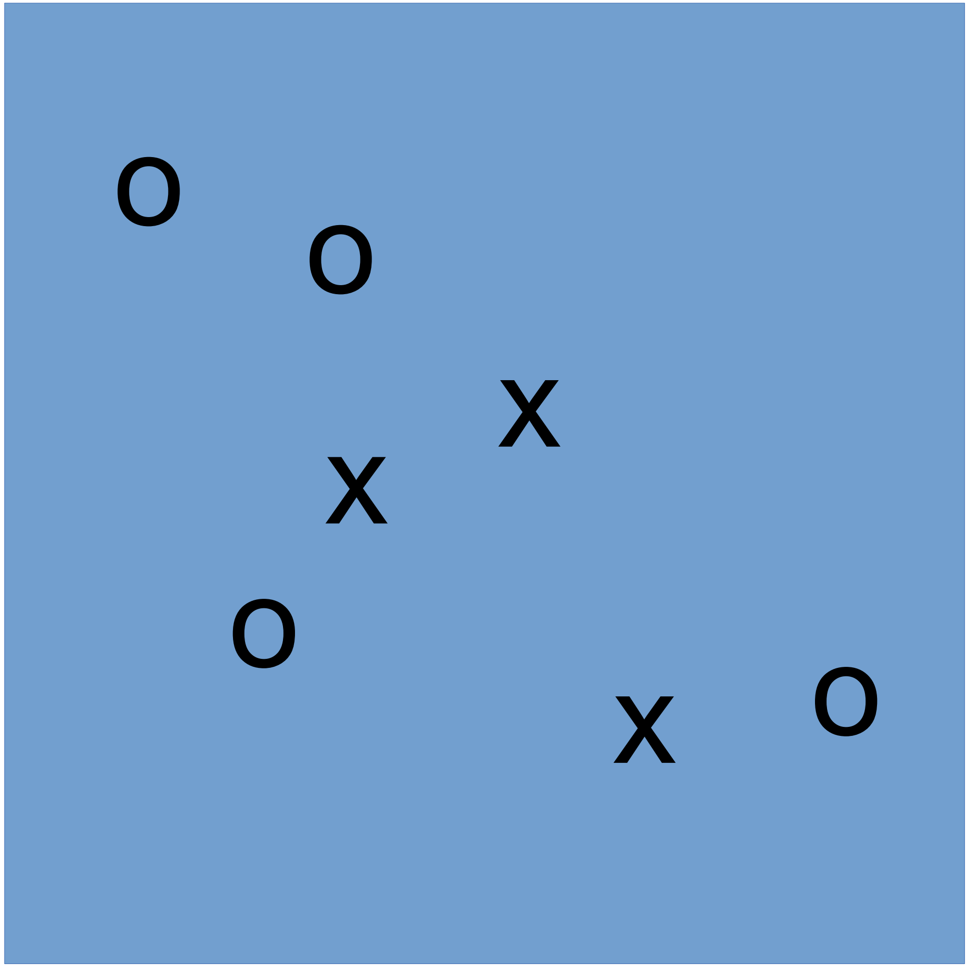
Expectations:
if there is a star in the box



o = nothing
x = something

Is it a star-shaped object?

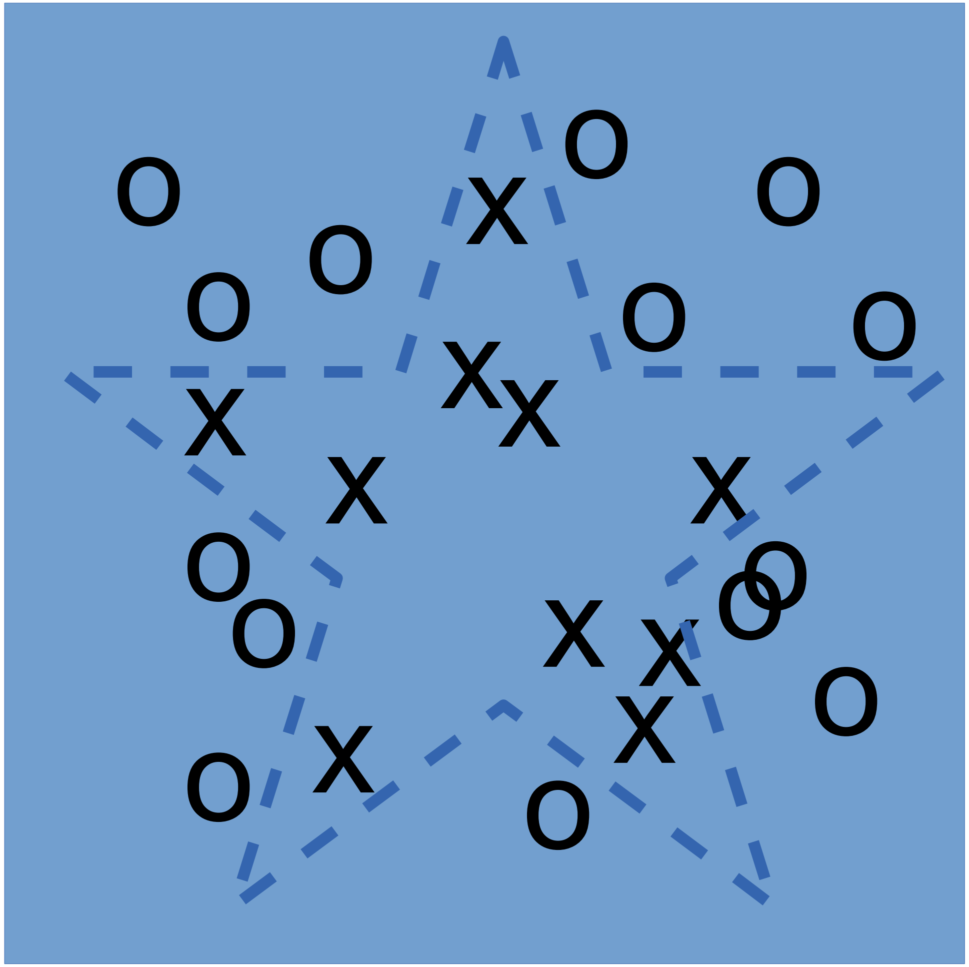
Actual results



o = nothing
x = something

Do you believe that it is a star-shaped object?

Actual results with more tests



o = nothing
x = something

Do you believe that it is a star-shaped object?

Usage examples

- Think about the test cases as usage examples for the function.

a	b	c	expected results

Try to be lazy

- Many usage examples look at the same situation.
- We don't need to include all of them.

a	b	c	expected results
10	20	5	20
~~50~~	~~700~~	~~12~~	~~700~~
~~13~~	~~15~~	~~12~~	~~15~~
1	2	3	3
~~9~~	~~30~~	~~40~~	~~40~~
10	10	5	10

Pick one to start

- We need to get started.
- Pick one example, and let's code.
  - Which one? Let's try the one that is easiest to code.

a	b	c	expected results
10	20	5	20
1	2	3	3
10	10	5	10

See the demo

Test structure

Assertions

Let's try

- Let's start with a simple function:

```
function addWithCap( a, b, cap ) {
}
```

- This function adds **a** and **b**, but ensure that the return value is not greater than **cap**. (Think about the HP in game after you drink a magic recovery portion.)

Examples

- Before you start writing the test and code, think about the examples that you would need to show that `addWithCap` works correctly.
- Think about a table like the one below.
- After you have listed a few test cases, think about which one to start testing first.

a	b	cap	expected results

Practice time

Function pronounce

- Write function `pronounce` that takes an integer `x` from 1 to 999 and return how `x` is pronounced in English.

```
function pronounce( x ) {
}
```

- For example:
  - pronounce(1) should return 'one'
  - pronounce(57) should return 'fifty-seven'

Function `getTopK`

- Write function getTopK that takes an array of integers and returns the k-th largest integer.

```
function getTopK( arr, k ) {
}
```

- For example:
  - `getTopK( [ 1, 2, 3, 4 ], 3 )` should return 2
  - `getTopK( [ 10, 9, 8, 100 ], 2 )` should return 10

Testing object behavior

- We want to have a **`Player`**:
  - a `Player` has property `healthPoint`
  - valid value of `healthPoint` is from 0 to 100
- Player has the following methods
  - `setHealthPoint( point)`
  - `takeHit( attackPoint )`
    - decrease the healthPoint by attackPoint but healthPoint should never be less than 0
  - `recoverHealth( recoveryPoint )`
    - increase the healthPoint by recoveryPoint but healthPoint should never be more than 100
  - `isDead()` and `isAlive()` which return true/false

See demo

Current code

```
function Player() {
    this.healthPoint = Player.MAX_HEALTHPOINT;
}
Player.MAX_HEALTHPOINT = 100;
Player.MIN_HEALTHPOINT = 0;

Player.prototype.setHealthPoint = function( point ) {
    this.healthPoint = point;
};
```

```
describe( 'Player', function() {

    it( 'should have healthPoint', function() {
        var p = new Player();
        assert( p.healthPoint != undefined );
    });

    it( 'should be able to set health point', function() {
        var p = new Player();
        p.setHealthPoint( 67 );
        assert( p.healthPoint == 67 );
    });
});
```

var and beforeEach

```
describe( 'Player', function() {

    beforeEach( function() {
        this.player = new Player();
    });

    it( 'should have healthPoint', function() {
        assert( this.player.healthPoint != undefined );
    });

    it( 'should be able to set health point', function() {
        this.player.setHealthPoint( 67 );
        assert( this.player.healthPoint == 67 );
    });
});
```

Note that we change the variable name from **p** to **player** because now the scope of this variable gets larger so that we need a more meaningful name.

OXBoard

- An `OXBoard` represent a 3x3 O-X board.
- It has the following methods
  - `placeO( row, column )`
  - `placeX( row, column )`
  - `show()`
    - returns,e.g., an array of string `['XX.','OXO', 'OO.']`.
  - `hasEnded()`
  - `getWinner()`
    - returns 'X' or 'O' or `null` if the game has not ended or the game ends in draw.
  - `isDraw()`
  - `hasOWon()`
  - `hasXWon()`