

## FT5004 Lab 3

### Truffle Suite

#### Introduction

In this lab, we'll cover the following:

1. Setting up a Truffle Environment
2. Writing Contracts in Truffle
3. Deploying contracts/ Migrating contracts
4. Testing Contracts

#### Setting Up Truffle

To download truffle, you first need Node : <https://nodejs.org/en/download/>

Please download and install Node JS

Once Node JS is installed, please open your terminal and run: ***npm install truffle -g***

This will install the Truffle environment

Apart from Truffle, we also need **Ganache**

Ganache is a development blockchain running the Ethereum Virtual Machine which can simulate the Ethereum blockchain. You can think of it as the “backend” when you deployed your smart contracts on Remix IDE.

To download Ganache, click here: <https://trufflesuite.com/ganache/>

Follow the instructions to download and install ganache

The last item we need is a local IDE (recall that Remix is an online IDE). The recommendation is to use **Visual Studio Code** which can be downloaded here:

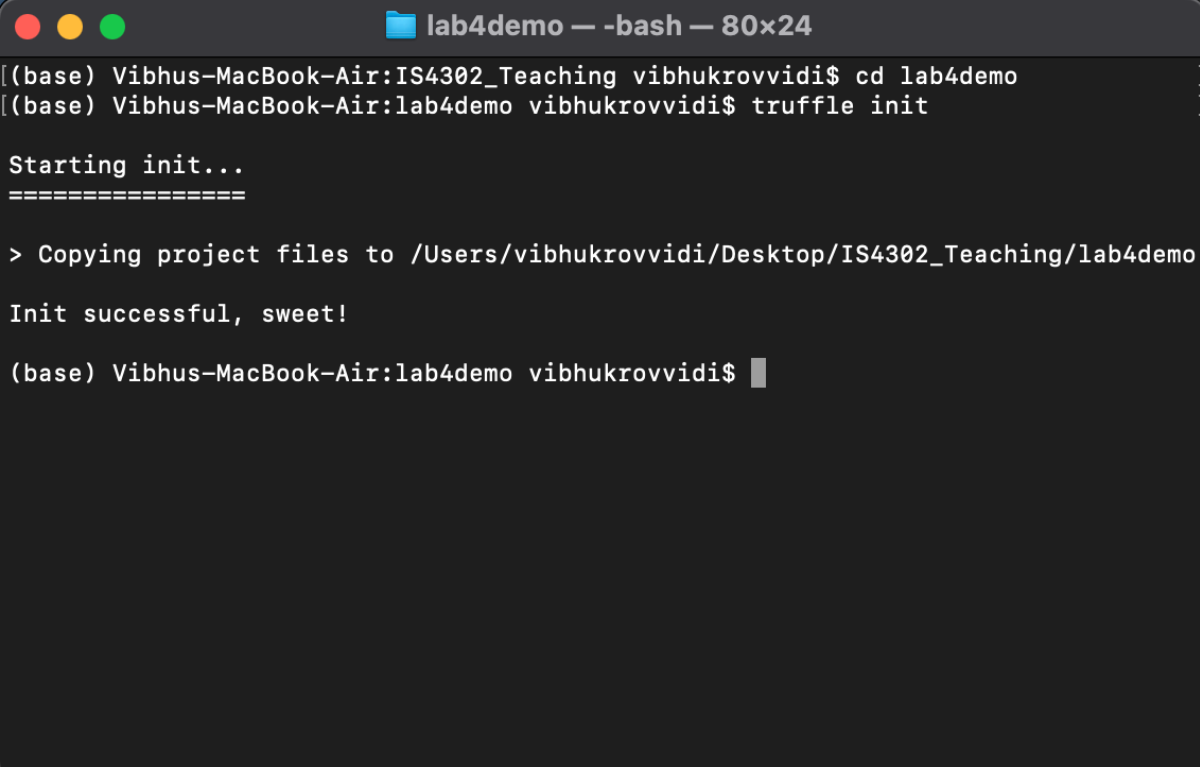
<https://code.visualstudio.com/download>

Go ahead and install all three

## Writing Contracts in Truffle

Once all the tools are downloaded appropriately, you should be all set to create your environment. Open your terminal or command prompt and create a folder to store your project.

Use the command **truffle init** to create a folder structure with the basic files needed in your project.

A terminal window titled 'lab4demo — -bash — 80x24' shows the following commands and output:

```
[(base) Vibhus-MacBook-Air:IS4302_Teaching vibhukrovvidi$ cd lab4demo
[(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$ truffle init

Starting init...
=====

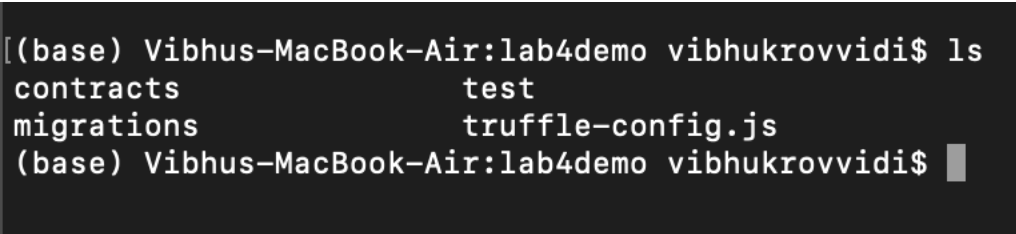
> Copying project files to /Users/vibhukrovvidi/Desktop/IS4302_Teaching/lab4demo

Init successful, sweet!

(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$
```

If initialisation is successful, you should see the message above.

Now if you examine the contents of this folder, you should see some files have been created for you. For **Windows** type **dir**. For **Mac** type **ls**. Then hit enter

A terminal window shows the output of the 'ls' command:

```
[(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$ ls
contracts          test
migrations         truffle-config.js
(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$
```

You will see 3 folders:

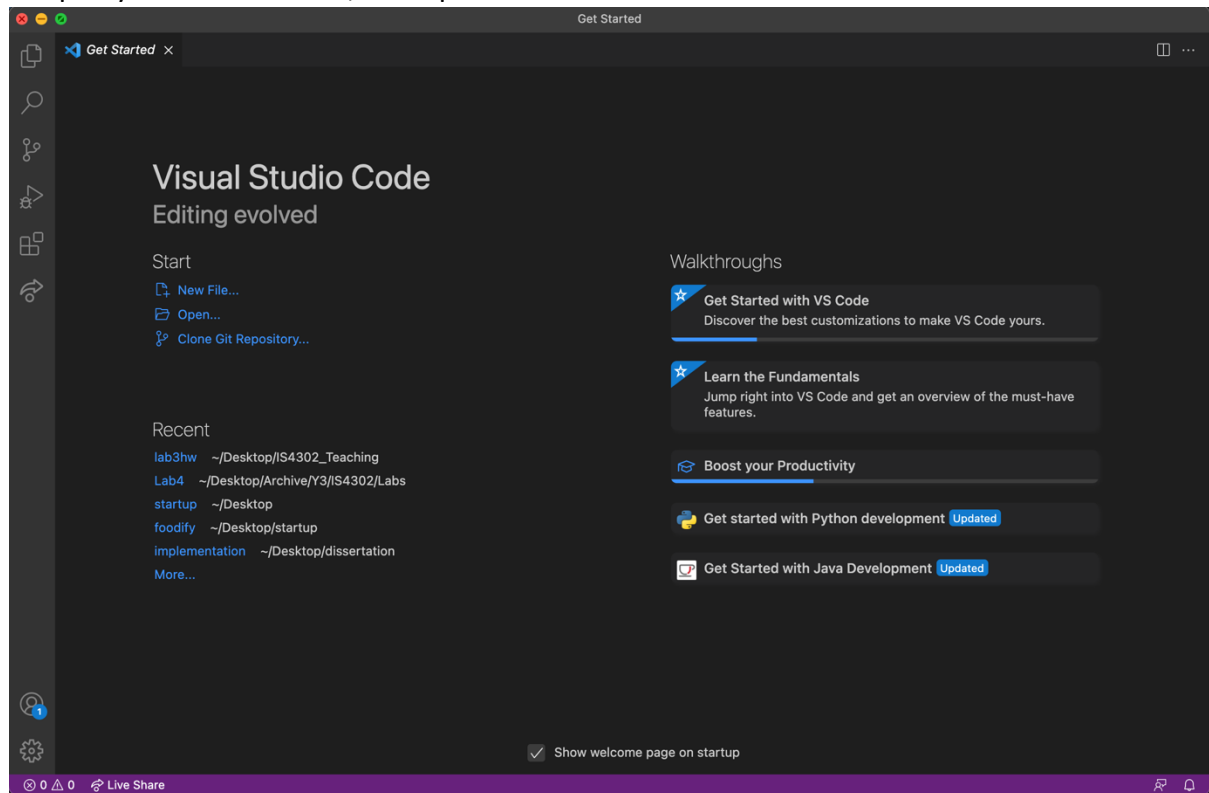
1. Contracts
2. Migrations
3. Test

You will also see truffle-config.js, a file that configures the truffle environment.

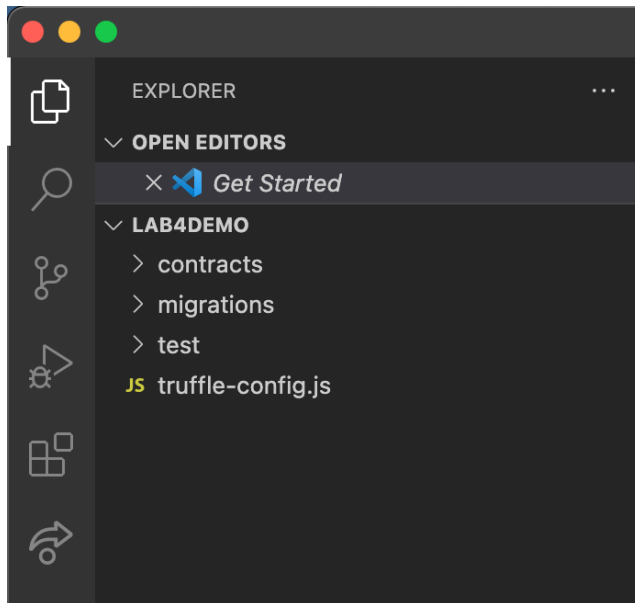
Folder	What it does
contracts	Stores your .sol files
migrations	Stores instructions on how your contracts should be compiled and then committed on the blockchain
test	Stores your unit test files

Let's start real slow. We are going to use the Dice.sol and DiceMarket.sol contracts for this lab (not the ones using DiceToken).

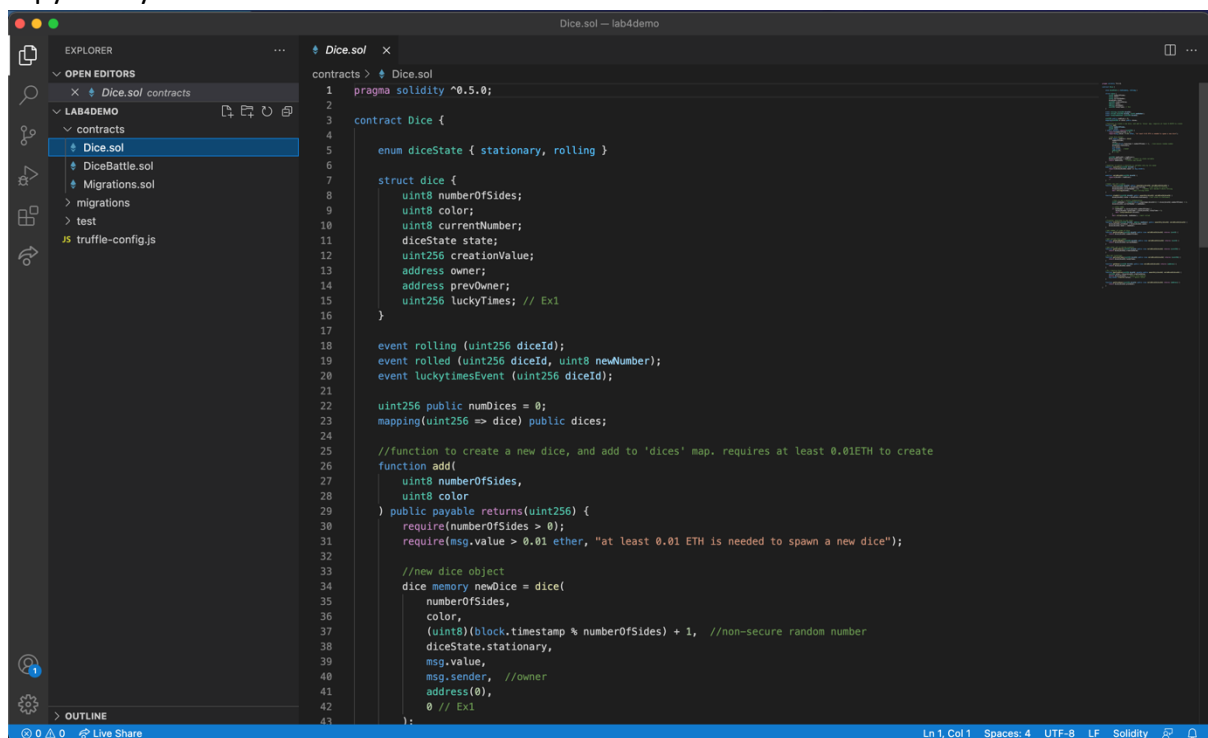
To open your environment, first open Visual Studio Code



Go to File -> Open and select **the folder containing the 3 folders + 1 script**. Open that folder.



This is what the left hand side menu should show. Feel free to open the contracts folder and copy over your .sol files.



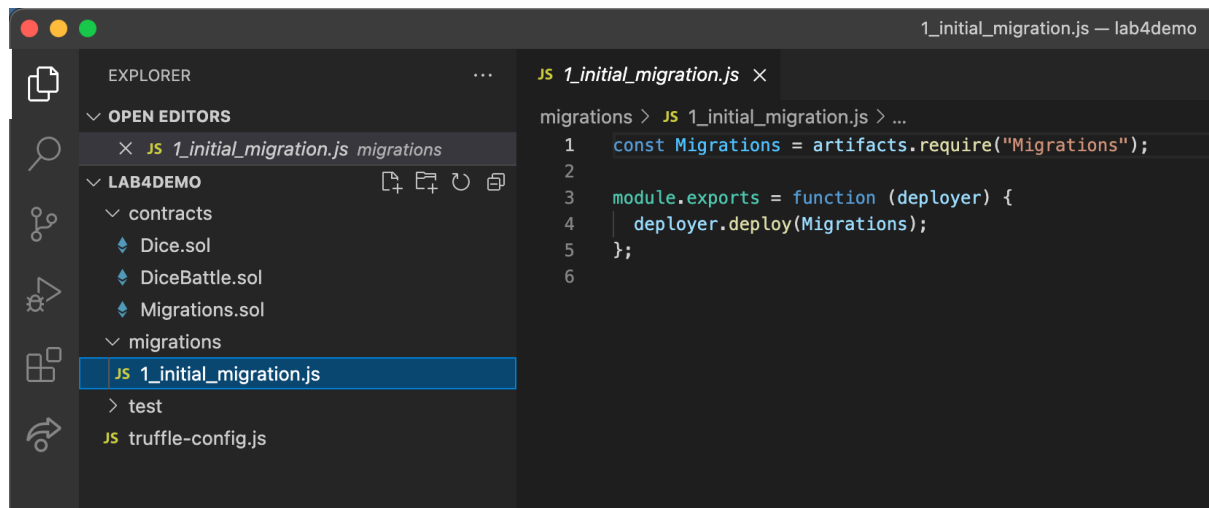
If prompted to install an extension that allows you to code on Solidity, please do so. It will give you hints about errors to help make your code bug free.

You can always create new files, write your contracts and save them using VS Code.

## Deploying Contracts

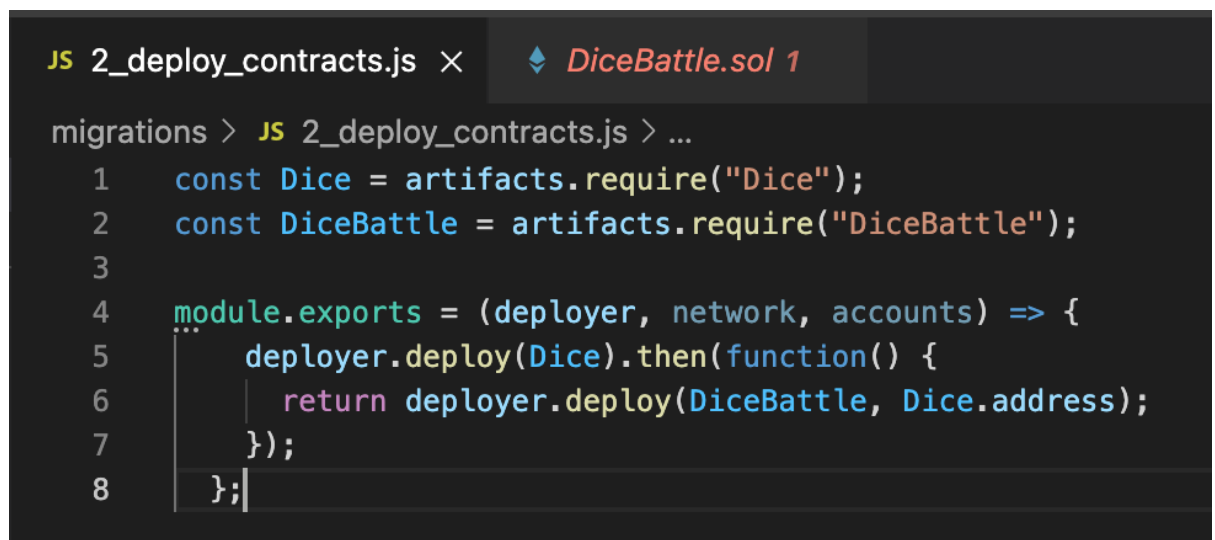
Contracts written like this are not useful unless we are able to deploy them to our development blockchain and test them.

In order to deploy blockchains, you need to use the **Migrations** folder.



You will see that the migrations folder already has a javascript file. We need to add another file and describe how we want the blockchain to work.

1. Create a file called **"2\_deploy\_contracts.js"**
2. Inside this file, follow the structure shown below



First, by declaring Dice and DiceBattle, we are requiring these contracts to exist in our environment

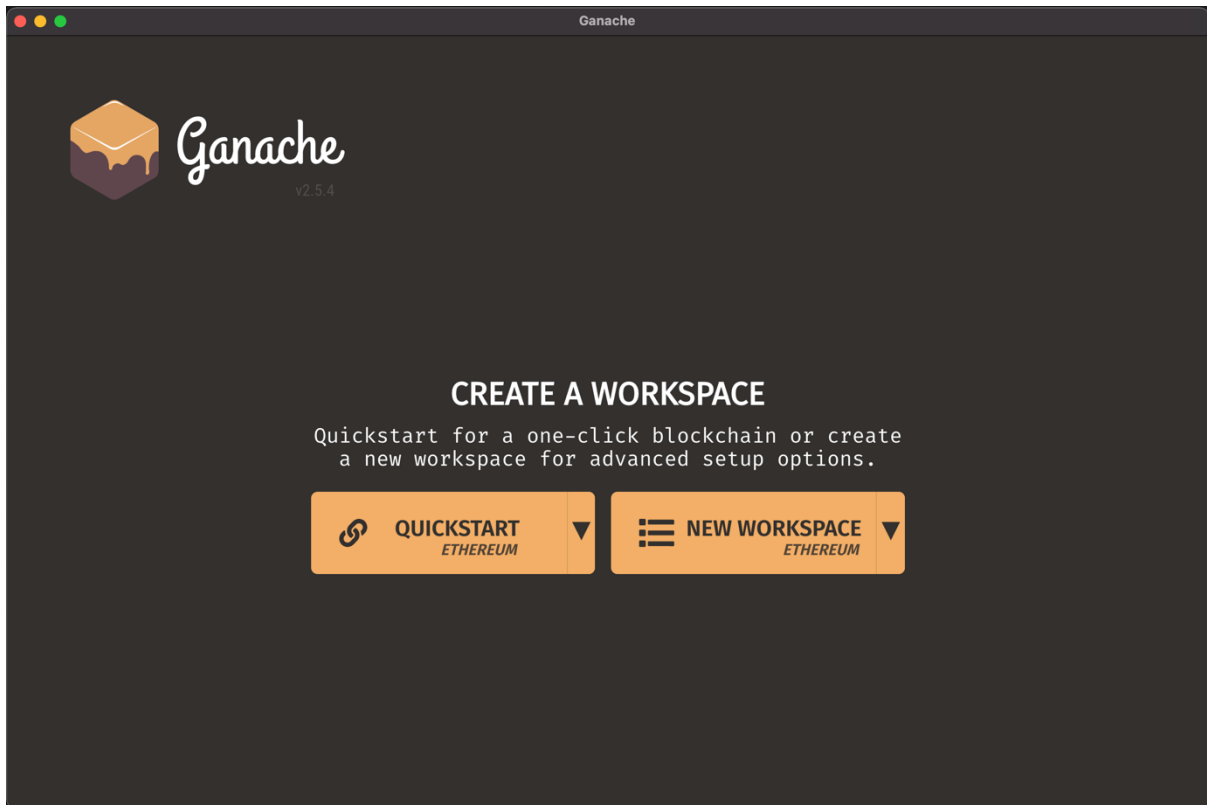
Then we are using deployer to deploy Dice first. Once Dice has been deployed, we ask it to deploy DiceBattle and pass Dice's address to DiceBattle's constructor.

3. Open truffle-config.js and scroll down to networks. **Uncomment out the section shown below and change the port to 7545**

```
module.exports = {  
  /**  
   * Networks define how you connect to your ethereum client and let you  
   * defaults web3 uses to send transactions. If you don't specify one  
   * will spin up a development blockchain for you on port 9545 when you  
   * run `develop` or `test`. You can ask a truffle command to use a specific  
   * network from the command line, e.g  
   *  
   * $ truffle test --network <network-name>  
   */  
  
  networks: {  
    // Useful for testing. The `development` name is special - truffle  
    // if it's defined here and no other network is specified at the command  
    // You should run a client (like ganache-cli, geth or parity) in a separate  
    // tab if you use this network and you must also set the `host`, `port`  
    // options below to some value.  
    //  
    development: {  
      host: "127.0.0.1",      // Localhost (default: none)  
      port: 7545,            // Standard Ethereum port (default: none)  
      network_id: "*",       // Any network (default: none)  
    },  
  
    // Another network with more advanced options...  
    // advanced: {  
    //   port: 8777,           // Custom port  
    //   network_id: 1342,    // Custom network  
    //   gas: 8500000,        // Gas sent with each transaction (default: 6700000)  
    //   gasPrice: 20000000000, // 20 gwei (in wei) (default: 100 gwei)  
    //   from: <address>,     // Account to send txs from (default: account 0)  
    //   websocket: true      // Enable EventEmitter interface for web3  
    // },  
  
    // Useful for deploying to a public network.  
    // NB: This is not a recommended function, it's just a way to avoid  
    // creating multiple networks that don't have hard-coded values.  
  },  
}
```

Awesome, now we've made all the changes needed to deploy our contracts. But, we still don't have a blockchain to deploy these to.

Open the Ganache app. It should look something like this:



Click quickstart Ethereum. That should end up with something like this:

ADDRESS	BALANCE	TX COUNT	INDEX
0xD8DeD8818039bB4248909A4928E1929D30BF130	100.00 ETH	0	0
0xaFC5d94e895Ba7BA92FC16C12bA7553c5d8Ed29E	100.00 ETH	0	1
0x93eBBd64785d439575757DaD8F7523f317bd875a	100.00 ETH	0	2
0x987BD4BA2aE6Dd2df3ac18B2509685D503359d60	100.00 ETH	0	3
0xFdFC034a89a549305c661aEbdD39634d4caC6bB5	100.00 ETH	0	4
0x3dC39271057f9C64Edf9752BB9f937787a65CFeF	100.00 ETH	0	5
0x2599bF68F38041954AeFAd4e5C589067251220cd	100.00 ETH	0	6

Nice!

Test that everything is working now. Go to your terminal and type:

*truffle compile*

then

*truffle migrate*

```
Compiling your contracts...
=====
> Compiling ./contracts/Dice.sol
> Compiling ./contracts/DiceBattle.sol
> Compiling ./contracts/Migrations.sol
> Artifacts written to /Users/vibhukrovvidi/Desktop/IS4302_Teaching/lab4demo/build/contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang

(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$
```

### Truffle Compile

```
> account:      0xD8DeD8818039bB4248909A4928E1929D30BfE130
> balance:      99.99616114
> gas used:     191943 (0x2edc7)
> gas price:    20 gwei
> value sent:   0 ETH
> total cost:   0.00383886 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:   0.00383886 ETH

2_deploy_contracts.js
=====

Deploying 'Dice'
-----
> transaction hash: 0xf8e8549f92f43a7bb2ff9b58d06f1f3f8ea7cce671672eb199aee3c5be3d21ba
> Blocks: 0        Seconds: 0
> contract address: 0x67d05494697a25CEBE09Fe473733A0359bBE91bB
> block number:    3
> block timestamp: 1644043730
> account:         0xD8DeD8818039bB4248909A4928E1929D30BfE130
> balance:         99.97520282
> gas used:        1005578 (0xf580a)
> gas price:       20 gwei
> value sent:      0 ETH
> total cost:      0.02011156 ETH

Deploying 'DiceBattle'
-----
> transaction hash: 0x5688453eb3364db86969cb8e6ae5eea52364c9a74477a453788a9b7529ca0aba
> Blocks: 0        Seconds: 0
> contract address: 0x14e2DE0c66c49436b6defc7e034E717d065C2250
> block number:    4
> block timestamp: 1644043731
> account:         0xD8DeD8818039bB4248909A4928E1929D30BfE130
> balance:         99.95376206
> gas used:        1072038 (0x105ba6)
> gas price:       20 gwei
> value sent:      0 ETH
> total cost:      0.02144076 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:      0.04155232 ETH

Summary
=====
> Total deployments: 3
> Final cost:       0.04539118 ETH
```

### Truffle Migrate

You will see that your ganache shows some accounts have less than 100 ETH. This is the contract deployment process.



## Testing Contracts

Go to the test folder and create a file called 'test\_dice.js'.

You can opt to download Truffle Assertions to test your events here:

<https://www.npmjs.com/package/truffle-assertions>

---

To start, let's first require the contracts to be deployed and our assertion frameworks to be correctly initialised:

```
const _deploy_contracts = require("../migrations/2_deploy_contracts");
const truffleAssert = require('truffle-assertions');
var assert = require('assert');
```

Now, let us create variables to represent these contracts:

```
const _deploy_contracts = require("../migrations/2_deploy_contracts");
const truffleAssert = require('truffle-assertions');
var assert = require('assert');

var Dice = artifacts.require("../contracts/Dice.sol");
var DiceBattle = artifacts.require("../contracts/DiceBattle.sol");
```

Now, let us establish our testing :

```
contract('DiceBattle', function(accounts) {

  before(async () => {
    diceInstance = await Dice.deployed();
    diceBattleInstance = await DiceBattle.deployed();
  });
  console.log("Testing Trade Contract");
```

This essentially waits for the 2 contracts to be deployed before any testing can occur.

### Test 1: Test the ability to get dice

```
it('Get Dice', async () => {  
  let makeD1 = await diceInstance.add(1, 1, {from: accounts[1], value: 10000000000000000});  
  let makeD2 = await diceInstance.add(30, 1, {from: accounts[2], value: 10000000000000000});  
  
  assert.notStrictEqual(  
    makeD1,  
    undefined,  
    "Failed to make dice"  
  );  
  
  assert.notStrictEqual(  
    makeD2,  
    undefined,  
    "Failed to make dice"  
  );  
})
```

### Test 2: Test transfer ownership

```
it('transfer ownership of dice', async () => {  
  
  let t1 = await diceInstance.transfer(0, diceBattleInstance.address, {from: accounts[1]});  
  let t2 = await diceInstance.transfer(1, diceBattleInstance.address, {from: accounts[2]});  
  
  let enemy_adj1 = await diceBattleInstance.setBattlePair(accounts[2], {from: accounts[1]});  
  let enemy_adj2 = await diceBattleInstance.setBattlePair(accounts[1], {from: accounts[2]});  
  
  truffleAssert.eventEmitted(enemy_adj1, 'add_enemy');  
  truffleAssert.eventEmitted(enemy_adj2, 'add_enemy');  
})
```

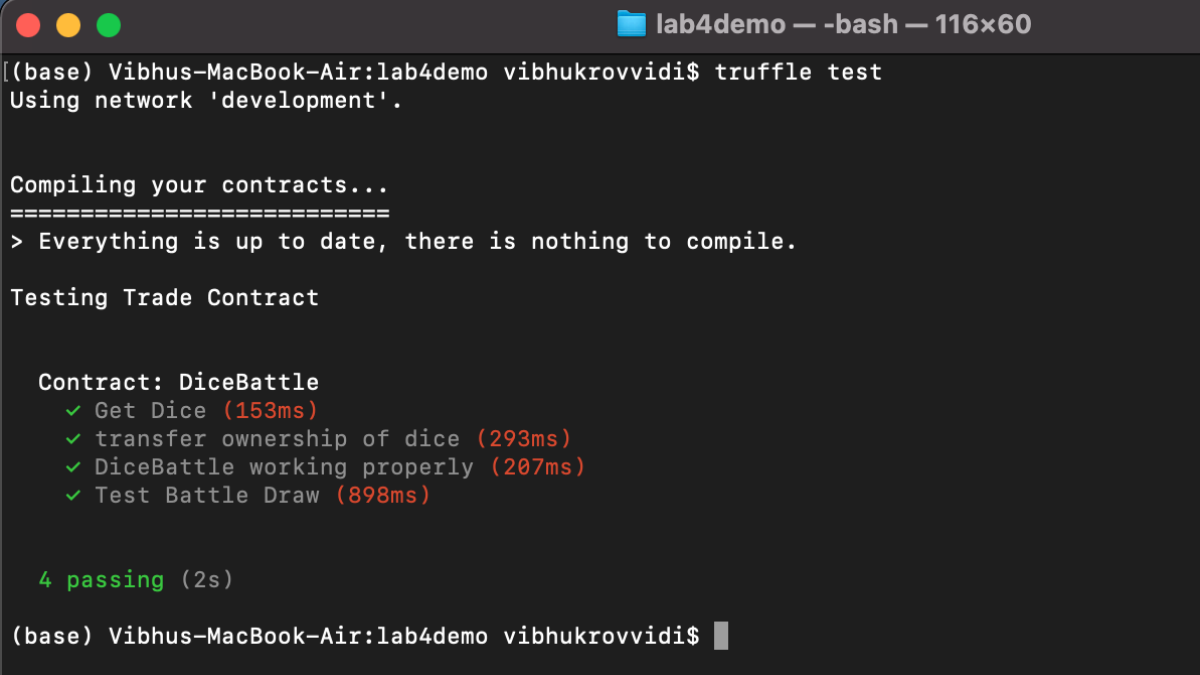
### Test 3: Dice Battle working

```
it('DiceBattle working properly', async () => {  
  
  let doBattle = await diceBattleInstance.battle(0, 1, {from: accounts[1]});  
  console.log(doBattle);  
  truffleAssert.eventEmitted(doBattle, 'battlewin');  
});
```

Nice! We have written unit tests!

## Actual Testing

Once your test file is done, go to your terminal and type 'truffle test'. Hit enter and see how your test cases perform!



```
lab4demo — -bash — 116x60
[(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$ truffle test
Using network 'development'.

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Testing Trade Contract

Contract: DiceBattle
  ✓ Get Dice (153ms)
  ✓ transfer ownership of dice (293ms)
  ✓ DiceBattle working properly (207ms)
  ✓ Test Battle Draw (898ms)

4 passing (2s)

[(base) Vibhus-MacBook-Air:lab4demo vibhukrovvidi$
```

**NOTE:** Unit tests must test small aspects of code to isolate bugs or errors. They should not test large chunks because if they fail, you are not sure where the bug is.

---

## Homework Exercises:

### Ex 1:

Create a truffle environment and use Truffle and ganache to create unit tests for the DiceMarket contract from our prior lab.

Test cases must include:

1. Test the creation of the dice
2. Test that if ether is not supplied to the Dice contract's add function, an error is returned
3. Test that the Dice can be transferred to the DiceMarket contract
4. Test that a Die cannot be listed if the price is less than value + commission
5. Test that a Dice can be listed
6. Test that the owner can unlist a die
7. Test that another party can buy the die

Feel free to expand the test cases beyond this.