



# 區塊鏈大項目"MIT"

Blockchain assignment part-3

學生姓名：湯偉傑

學生學號：15352307

專業方向：電子政務方向

# 報告內容

- MIT (Macau Investment Token) —— 一款基於以太坊的Dapp
  - 一、項目運行環境
    - (一) 使用 solidity 語言在 Remix 上進行代碼的編譯和調試
    - (二) Chrome 瀏覽器上安裝 Metamask 插件
  - 二、MIT\_Dapp 誕生的背景
    - (一) 背景：
    - (二) 構思：
    - (三) 應用場景例子：
  - 三、編寫智能合約和測試結果
    - (一) 代幣智能合約
    - (二) 代幣發行
    - (三) 集資智能合約
    - (四) 集資智能合約的使用

# 一、項目運行環境：

(一) 使用 Remix 進行代碼的編譯和調試，網址：<http://remix.ethereum.org>

The screenshot shows the Remix IDE interface. The top part displays a Solidity code editor with the file name 'browser/erc20Interface.sol'. The code defines an ERC20Interface contract with functions like allowance, approve, transfer, and transferFrom, along with events Transfer and Approval. The right side of the interface has a 'Compiler' tab with options for selecting a compiler version (currently 0.5.0+commit.1d4f565a.Emscripten.clang), auto compile, enable optimization, and hide warnings. Below the compiler are buttons for 'Start to compile (Ctrl-S)', 'Details', 'ABI', and 'Bytecode'. A message box at the bottom says 'mock compiler: source not found'. The bottom part of the interface is a terminal window with the title '[2] only remix transactions, script'. It contains help text for the Remix environment, including commands like remix.setProviderUrl(url), remix.execute(filepath), and remix.debugHelp(). It also includes a welcome message for Remix v0.7.5 and a list of accessible libraries such as web3, ethers.js, and swarmjs.

Solidity是一種智能合約高級語言，運行在Ethereum虛擬機（EVM）之上。

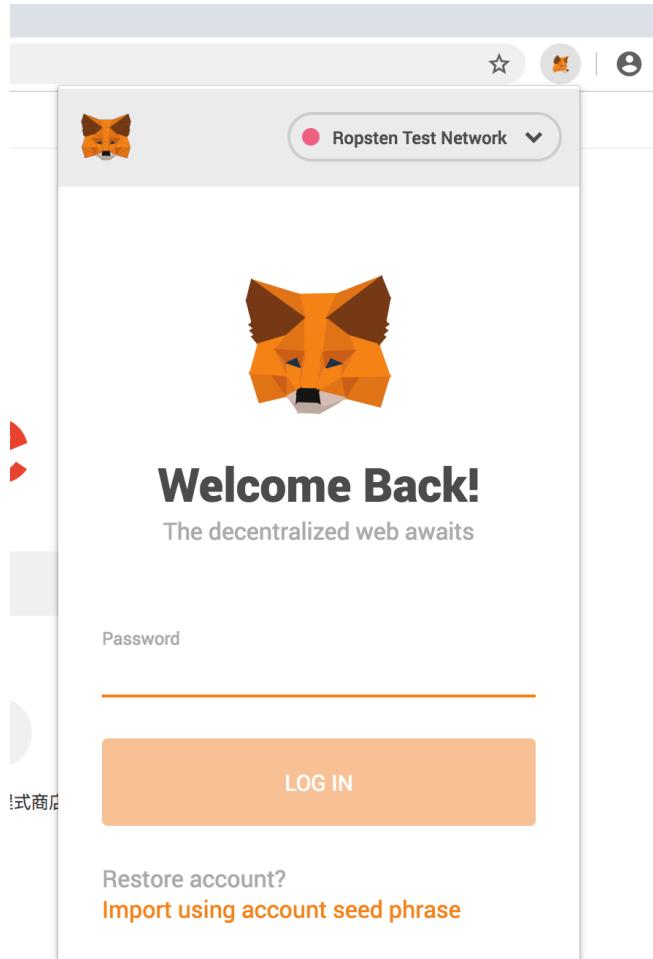
它的語法接近於Javascript，是一種面向對象的語言。但作為一種真正意義上運行在網絡上的去中心合約，它又有很多的不同，下面列舉一些：

- 以太坊底層是基於帳戶，而非UTXO的，所以有一個特殊的Address的類型。用於定位用戶，定位合約，定位合約的代碼（合約本身也是一個帳戶）。
- 由於語言內嵌框架是支持支付的，所以提供了一些關鍵字，如payable，可以在語言層面直接支持支付，而且超級簡單。
- 存儲是使用網絡上的區塊鏈，數據的每一個狀態都可以永久存儲，所以需要確定變量使用內存，還是區塊鏈。
- 運行環境是在去中心化的網絡上，會比較強調合約或函數執行的調用的方式。因為原來一個簡單的函數調用變為了一個網絡上的節點中的代碼執行，分布式的感覺。
- 最後一個非常大的不同則是它的異常機制，一旦出現異常，所有的執行都將會被回撤，這主要是為了保證合約執行的原子性，以避免中間狀態出現的數據不一致。

(二) Chrome 瀏覽器上安裝 metamask 插件

MetaMask是一款在谷歌瀏覽器Chrome上使用的插件類型的以太坊錢包，該錢包不需要下載，只需要在谷歌瀏覽器添加對應的擴展程序即可，非常輕量級，使用起來也非常方便。

整個項目的運行都會在metamask和Remix上進行，包括合約的部署，角色的變換，函數的調用和以太幣的交易。



## 二、MIT\_Dapp 誕生的背景：

### (一) 背景：

現在的澳門普遍情況就是房價高，物價高，青年人投身社會工作以後，普遍月資就月薪20000-40000塊左右，這種水平夠不著房子，每個月消耗還是挺大的，投資又沒有方向，所以很多就是一年去一兩次旅遊，或者買車，因為這是一個種力所能及的行為，但是缺乏一種長期的投資理念，也缺乏對投資的認識，以致於可能承受的風險水平很高，可抵抗環境影響的保險不足。

澳門地區的金融環境是一種特色金融，跟香港和內地的金融結構非常不同，而且在澳門的資金流動多在熟人之間互相流動，用我們那裡的話說，創業是用生命值來硬接的，因為銀行並不會給小商戶提供太大的幫助，而且澳門社會普遍的投資理念都是在保險或是房資之類的領域，和內地的大城市並不一樣，天使投資人並不多，投資人多為親戚，所以才有投資失敗是用生命值硬接這一說。

就因為這樣的環境間接使到剛出社會的年青人，因為失敗的成本高，經濟能力水平追不上已有一定基礎的階層，造成了資產階級的分層更明顯。

### (二) 構思：

所以我就想做一種基於聯盟鏈的投資應用（局限在澳門，也僅限在本澳的投資，因為可能涉及到法律相關問題），角色有三種，可信任人（銀行），投資者，投資項目發起人。而且也因為區塊鏈的可追溯性和不可篡改性，可以提供一定的安全保障。將很多小的事情進行一個量化，比如說，賣飲料機，或者是便利店，我通過銀行發行的MIT，去量化一個便利店，每個擁有代幣的人，就等同於有該投資項目的收益權，每段時間，按總項目集合的幣值，和代幣持有者所持代幣，進行成比例的利益分配，期間所有代幣可以自由買賣，只有有人接手就行了，這樣的模式可以讓資本的流通面更大，以致於小投資人，或者是中小企業能受惠。

### (三) 應用場景例子：

投資項目發起人：我有一個投資想法的項目，比如說投資一個車位，大概200萬，而我手上就只有50萬，怎麼辦好呢？那就在基於聯盟鏈的區塊鏈投資應用上發布個投資項目，看有沒有人感興趣吧。如果，以後每個月收租的話，是分配租金收益，賣的話，就只能賣自己所擁有的代幣，擁有代幣不代表能使用車位。使用權和收益權是分開的。

投資者們：我看到一個項目挺感興趣的啊，投個10萬試試。

項目不通過：把錢退回給投資者們。

項目通過了：將收集到的以太幣發回給指定的受益人。

之後的收益分配就按具體的項目安排來訂。

### 三、編寫智能合約和調試：

首先由一個可信任人（最好是銀行）創建一種代幣，在以太坊的世界中，硬通貨就是ETH-以太幣，相當於現實世界中的黃金，我們發佈的代幣，就相當於各國發行的需要與黃金掛鈎兌換比率的貨幣，國家對貨幣擁有控制權，只要合約功能進行擴展，創造代幣的人也會對代幣擁有控制權。

對於挖礦的誤解：挖礦不是挖幣，其實挖礦是挖區塊，區塊是乾嘛的，是用來打包交易的，是存儲數據的，代幣是不用挖的，當你挖到了區塊，代幣是給你的獎勵，在發行量一定的情況下，代幣會越來越少，所以挖到區塊的獎勵會變少。那獎勵少了為什麼還要挖礦呢，因為你的任何一筆交易都需要記錄，一個區塊的大小也就幾M，存儲不了那麼多交易信息，所以要持續挖區塊來記錄你的交易，同時交易的手續費，會獎勵給挖出區塊的人。

#### (一) 代幣智能合約

erc20interface.iso

```
1 pragma solidity ^0.4.20;
2
3 contract ERC20Interface {
4
5     string public constant name = "MCoin";
6     string public constant symbol = "MIToken";
7     uint8 public constant decimals = 18; // 18 is the most common number of decimal places
8     // 0.0000000000000001 个代币
9
10    function totalSupply() public constant returns (uint);
11
12    function balanceOf(address tokenOwner) public constant returns (uint balance);
13
14    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
15    function approve(address spender, uint tokens) public returns (bool success);
16
17    function transfer(address to, uint tokens) public returns (bool success);
18    function transferFrom(address from, address to, uint tokens) public returns (bool success);
19
20
21    event Transfer(address indexed from, address indexed to, uint tokens);
22    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
23 }
```

name : 代幣名稱

symbol : 代幣符號

decimals : 代幣小數點位數，代幣的最小單位，18表示我們可以擁有 .000000000000000001單位個代幣。

totalSupply() : 發行代幣總量。

balanceOf(): 查看對應賬號的代幣餘額。

transfer(): 實現代幣交易，用於給用戶發送代幣（從我們的賬戶里）。

transferFrom(): 實現代幣用戶之間的交易。

allowance(): 控制代幣的交易，如可交易賬號及資產。

approve(): 允許用戶可花費的代幣數。

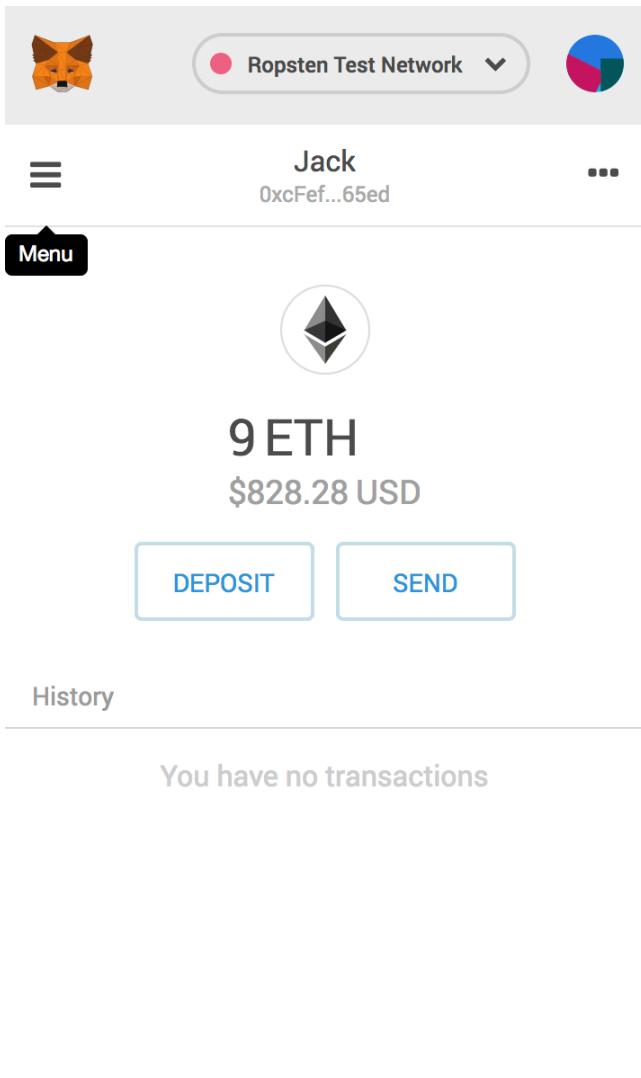
其實還可以再增加幾項功能，但是按目前所需要用到的，上面幾項功能就足夠了，代幣代碼來自於技術博客，我作出了一點修改，使之更符合所需要的條件。

## erc20.iso

```
1 pragma solidity ^0.4.20;
2 import './erc20interface.sol';
3 contract ERC20 is ERC20Interface {
4     string public name;
5     string public symbol;
6     uint8 public decimals = 18; // 18 是建议的默认值
7     uint256 public totalSupply;
8
9     // 对自动生成对应的balanceOf方法
10    mapping(address => uint256) public balanceOf;
11
12    // allowed保存每个地址 (第一个address) 授权给其他地址(第二个address)的额度 (uint256)
13    mapping(address => mapping(address => uint256)) allowed;
14
15
16    constructor(string _name) public {
17        name = _name; // "MIToken";
18        symbol = "MIT";
19        decimals = 0;
20        totalSupply = 1000000;
21        balanceOf[msg.sender] = totalSupply;
22    }
23
24    // 转移
25    function transfer(address _to, uint256 _value) returns (bool success) {
26        require(_to != address(0));
27        require(balanceOf[msg.sender] >= _value);
28        require(balanceOf[_to] + _value >= balanceOf[_to]); // 防止溢出
29
30        balanceOf[msg.sender] -= _value;
31        balanceOf[_to] += _value;
32
33        // 发送事件
34        emit Transfer(msg.sender, _to, _value);
35
36        return true;
37    }
38
39    function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
40        require(_to != address(0));
41        require(allowed[_from][msg.sender] >= _value);
42        require(balanceOf[_from] >= _value);
43        require(balanceOf[_to] + _value >= balanceOf[_to]);
44
45        balanceOf[_from] -= _value;
46        balanceOf[_to] += _value;
47
48        allowed[_from][msg.sender] -= _value;
49
50        emit Transfer(msg.sender, _to, _value);
51        return true;
52    }
53
54    function approve(address _spender, uint256 _value) returns (bool success) {
55        allowed[msg.sender][_spender] = _value;
56
57        emit Approval(msg.sender, _spender, _value);
58        return true;
59    }
60
61    function allowance(address _owner, address _spender) view returns (uint256 remaining) {
62        return allowed[_owner][_spender];
63    }
64}
```

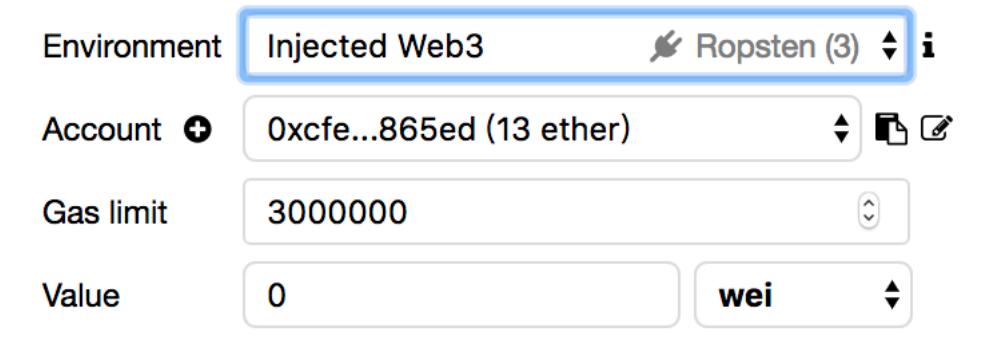
## (二) 代幣發行

創建自己的metamask錢包，並申請測試用的以太幣。（一瞬間覺得自己找到了一夜暴富的方法）



(好啦～該醒醒了)

接著在injectweb3上部署並測試一下智能合約的代碼功能。



部署好代幣發行的智能合約，初期發行1000000個MIT幣（Macau Investment Token）

ERC20

Deploy "MICChain"

or

At Address Load contract from Address

**Transactions recorded:** 5

All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment. e.g Transactions created in Javascript VM can be replayed in the Injected Web3.

Deploy Contracts

**ERC20 at 0xd7f...f37df (blockchain)**

approve address\_spender, uint256\_value

transfer 0xd7f513d795cf325d0e13dc0451e3098!

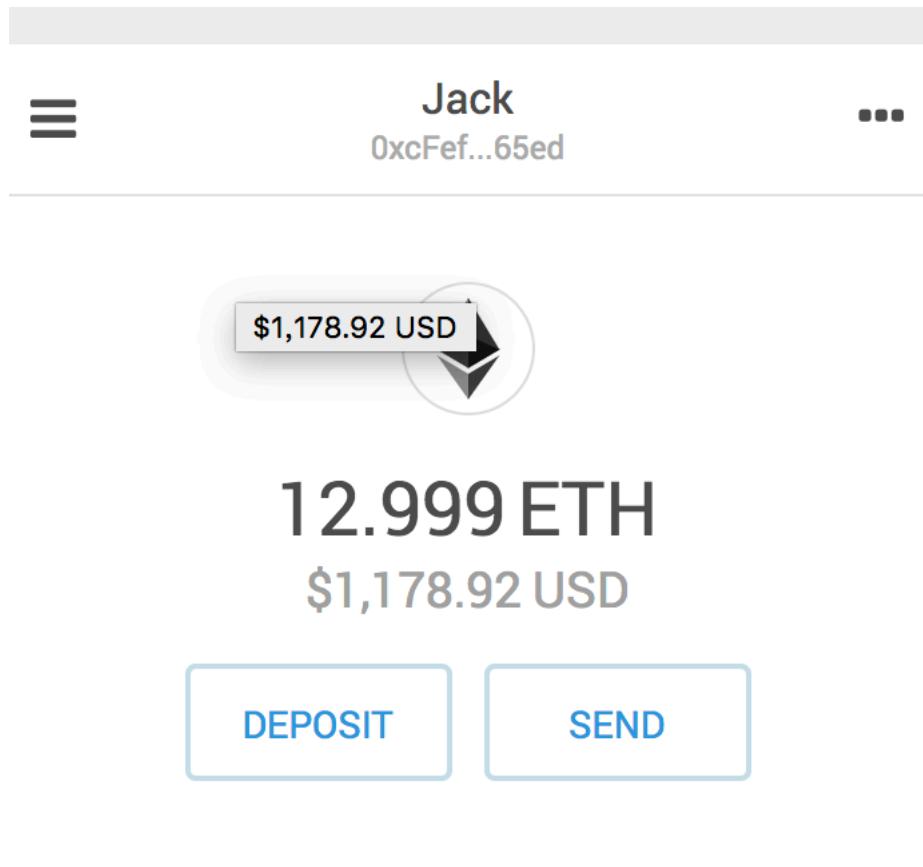
transferFrom address\_from, address\_to, uint256\_valu

allowance address\_owner, address\_spender

balanceOf 0xcfefd3089ab3dbb54b5e374570a0a07

0: uint256: 1000000

部署合約成功，可能看到ETH只剩 12.999個，因為部署合約需要消耗少量gas。



## History

#0 - 12/09/2018 at 19:32



Contract Deployment

CONFIRMED

-0 ETH

-\$0.00 USD

接著測試一下代幣所實現的功能。從部署了合約的地址，向隨便一個地址發放100個MIT幣。

## Deployed Contracts



\_value    **ERC20 at 0xd7f...f37df (blockchain)**

approve address \_spender, uint256 \_value

**transfer**

\_to: "0xd7f513d795cf325d0e13dc0451e3098"

\_value: "100"

**transact**

這是錢包中顯示的發送確認，檢查沒錯後，點確認。

Jack → 0xd7f5...37df

TRANSFER

100 MIT

No Conversion Rate Available

DETAILS DATA

**EDIT**

GAS FEE ♦ 0.000052  
\$0.00

---

AMOUNT + GAS FEE

TOTAL 100 MIT + ♦ 0.000052  
\$0.00

---

**REJECT** **CONFIRM**

最後在Token的帳本中可以看到代幣的交易情況。

The screenshot shows the Etherscan interface for the Ropsten (Revival) TESTNET. The top navigation bar includes links for HOME, BLOCKCHAIN, TOKEN, and MISC. A search bar at the top right allows searching by Address / Txhash / Block / Token / Ens, with a GO button and a Language selection dropdown.

The main content area is titled "Token" and "MITChain". Below this, there's a summary table for the MIT token:

Total Supply:	1,000,000 MIT
Holders:	2 addresses
Transfers:	1

On the right side of the summary table, there are details about the contract: Contract: 0xd7f513d795cf325d0e13dc0451e30985b22f37df, Decimals: 0, and Links: Not Available, Update ?.

Below the summary, there's a filter section labeled "Filtered By:" with a text input field "Enter Address/TxHash" and an "Apply" button.

The main table displays the single transfer found:

TxHash	Age	From	To	Quantity
0xecd76bb5be9175...	1 min ago	0xcfef3089ab3dbb...	0xd7f513d795cf325...	100

At the bottom right of the table, there are buttons for First, Prev, Page 1 of 1, Next, and Last.

[ Download CSV Export ]

代幣合約部署完畢。

### (三) 集資智能合約

要求：

設定集資時間，目標，兌換格價，受益人

實現以太和代幣的兌換

提款和退款功能

```
1 pragma solidity ^0.4.16;
2
3 interface token {
4     function transfer(address receiver, uint amount) external ;
5 }
6
7 contract Ico {
8     address public beneficiary;
9     uint public fundingGoal;
10    uint public amountRaised;
11
12    uint public deadline;
13    uint public price;
14    token public tokenReward;
15
16    mapping(address => uint256) public balanceOf;
17    bool crowdsaleClosed = false;
18
19    event GoalReached(address recipient, uint totalAmountRaised);
20    event FundTransfer(address backer, uint amount, bool isContribution);
21
22
23    constructor (
24        uint fundingGoalInEthers,
25        uint durationInMinutes,
26        uint etherCostOfEachToken,
27        address addressOfTokenUsedAsReward
28    ) public {
29        beneficiary = msg.sender;
30        fundingGoal = fundingGoalInEthers * 1 ether;
31        deadline = now + durationInMinutes * 1 minutes;
32        price = etherCostOfEachToken * 1 ether;
33        tokenReward = token(addressOfTokenUsedAsReward);
34    }
35
36
37    function () public payable {
38        require(!crowdsaleClosed);
39
40        uint amount = msg.value; // wei
41        balanceOf[msg.sender] += amount;
42
43        amountRaised += amount;
44
45        tokenReward.transfer(msg.sender, amount / price);
46
47        emit FundTransfer(msg.sender, amount, true);
48    }
49
50    modifier afterDeadline() {
51        if (now >= deadline) {
52            ;
53        }
54    }
55
56    function checkGoalReached() public afterDeadline {
57        if (amountRaised >= fundingGoal) {
58            emit GoalReached(beneficiary, amountRaised);
59        }
60        crowdsaleClosed = true;
61    }
62
63
64    function safeWithdrawal() public afterDeadline {
65
66        if (amountRaised < fundingGoal) {
67            uint amount = balanceOf[msg.sender];
68            balanceOf[msg.sender] = 0;
69            if (amount > 0) {
70                msg.sender.transfer(amount);
71                emit FundTransfer(msg.sender, amount, false);
72            }
73        }
74
75        if (fundingGoal <= amountRaised && beneficiary == msg.sender) {
76            beneficiary.transfer(amountRaised);
77            emit FundTransfer(beneficiary, amountRaised, false);
78        }
79    }
80 }
81 }
```

1、Ico : 集資合約

addressOfTokenUsedAsReward : 募資成功後的收款方

fundingGoalInEthers : 募資額度

durationInMinutes : 募資時間

etherCostOfEachToken : 代幣和以太幣匯率

2、function () payable payable: 回調函數

沒有函數名的payable函數為回調函數，意思是往智能合約地址打ETH的時候，則會自動調用該函數執行。

3. checkGoalReached : 檢查眾籌目標是否達到

該調用函數修改器modifier的函數afterDeadline，只是表示截止時間前執行這個代碼，實際不會

checkGoalReached執行函數體的代碼，只會執行afterDeadline的代碼後就返回。

該函數的功能是設置fundingGoalReached為true表示眾籌目標達到。

4. safeWithdrawal : 眾籌結束執行代碼

該調用函數修改器modifier的函數afterDeadline，只是表示截止時間前執行這個代碼，實際不會

checkGoalReached執行函數體的代碼，只會執行afterDeadline的代碼後就返回。

如果眾籌目標沒有達到，則在當前執行賬號為眾籌賬號的情況下，把募集的ETH打回給眾籌發送的賬號。

如果眾籌目標賬號達到，則把募集的ETH打給智能合約創建的賬號。

創建角色：

tokencreator 代幣發行人

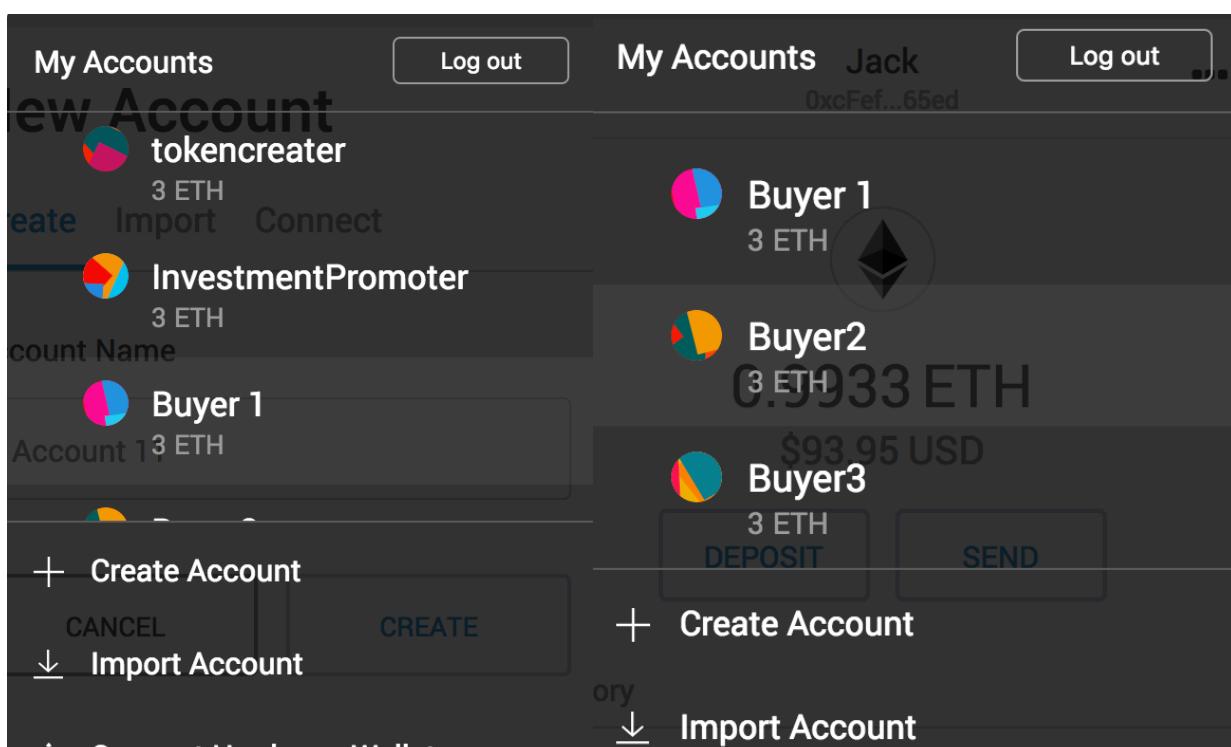
InvestmentPromoter 投資項目發起人

Buyer1 買家1

Buyer2 買家2

Buyer3 買家3

每人都有三個以太幣，接著會在tokencreator部署代幣合約。在InvestmentPromoter部署投資項目合約。



合約部署的參數如下，20分鐘內，總共需要集個5個以太幣，每一個以太幣等於1個代幣。最下面的地址是剛剛代幣合約的地址（說明是同該合約裡的代幣進行交易）。

Ico

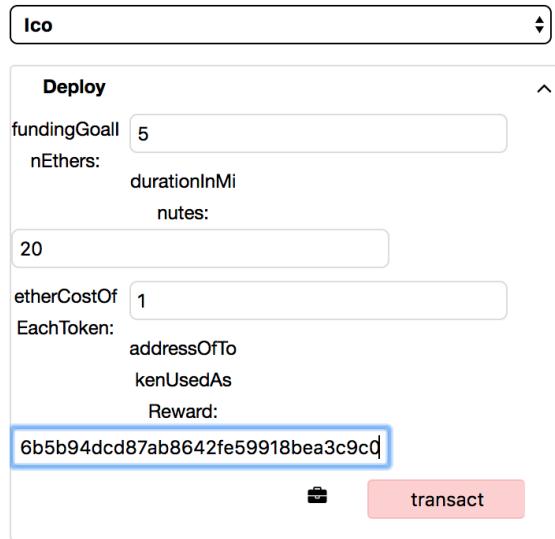
Deploy

fundingGoal: 5  
nEthers:  
durationInMi  
notes:  
20  
etherCostOf: 1  
EachToken:  
addressOfTo  
kenUsedAs  
Reward:  
6b5b94dcd87ab8642fe59918bea3c9cq

or

At Address Load contract from Address

transact



接著切換回tokencreator發送5個代幣給投資項目合約的智能合約地址。要不然投資者就沒法投錢進去。

Deployed Contracts

ERC20 at 0x19e...3c9c0 (blockchain)

approve address\_spender, uint256\_value

transfer

\_to: "0xba1b71dd065a4e634d7dfbae5aed62a"  
\_value: "5"

transact

transferFrom address\_from, address\_to, uint256\_value

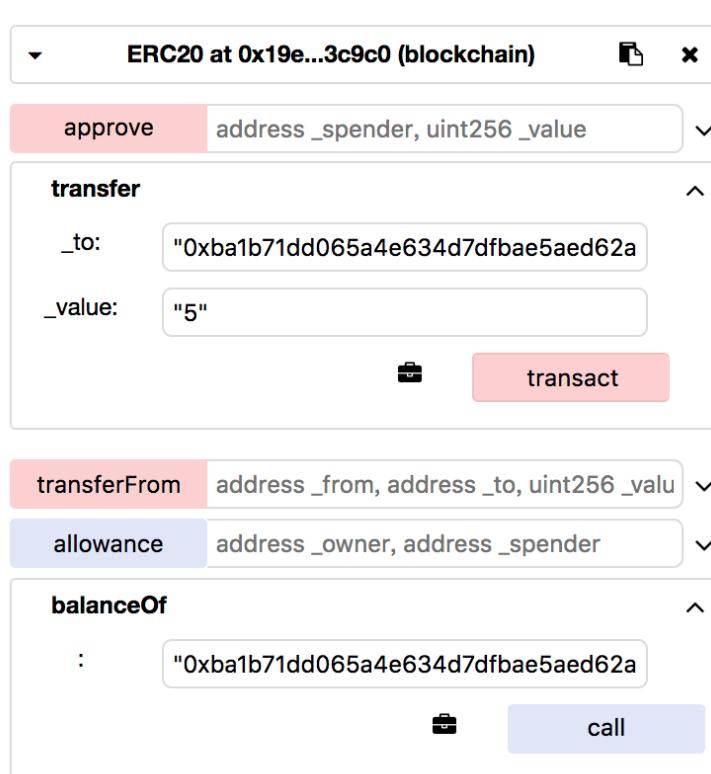
allowance address\_owner, address\_spender

balanceOf

: "0xba1b71dd065a4e634d7dfbae5aed62a"

call

0: uint256: 5



The image shows three separate MetaMask notifications for the Ropsten Test Network. Each notification details a transfer of tokens from a buyer's account to a contract address. The transfers are:

- Buyer 1:** UNKNOWN FUNCTION, Amount: ♦1 \$94.62, Gas Fee: ♦0.000094 \$0.01, Total: ♦1.000094 \$94.63.
- Buyer2:** UNKNOWN FUNCTION, Amount: ♦2 \$189.24, Gas Fee: ♦0.000079 \$0.01, Total: ♦2.000079 \$189.25.
- Buyer3:** UNKNOWN FUNCTION, Amount: ♦2 \$189.24, Gas Fee: ♦0.000079 \$0.01, Total: ♦2.000079 \$189.25.

Below each notification are two buttons: "REJECT" and "CONFIRM".

合約部署好，而且合約裡的代幣足夠了，接著就是Buy1,Buy2,Buy3分別向合約轉賬以太幣，成功以後收到1:1的MIT幣。

The screenshot shows the Token MIChain interface for the ERC-20 token MIT. The summary section provides the following information:

Total Supply:	1,000,000 MIT
Holders:	4 addresses
Transfers:	4

The contract details are:

Contract:	0x19ea4f326b5b94dcd87ab8642fe59918bea3c9c0
Decimals:	0
Links:	Not Available, <a href="#">Update</a> ?

There is a search bar for "Filtered By: Enter Address/TxHash" and an "Apply" button.

The navigation tabs at the top are: Transfers, Holders (selected), ReadContract, Write Contract, Beta.

The Token Holders Chart shows the following data:

Rank	Address	Quantity	Percentage
1	0x456e9095bdae2ddaa1c07376455b8787790972ab2	999995	99.9995%
2	0x4970760c3bc165a5fa0022e7cbcbf38830d6bd19	2	0.0002%
3	0x130246b217355a34faeffd47716799aae4301991	2	0.0002%
4	0x527bd4e8d97ebc3b8ab294d2ce812b98bb88b9e7	1	0.0001%

在MIT代幣合約的holder中可以清楚的看到，現在發行出去的5個代幣已經被Buy1,Buy2,Buy3所擁有。

20分鐘到了項目發起人需要驗收項目是否成功啟動。手動點擊checkGoalReached，發現項目成功以後，就點擊safeWithdrawal，將合約收到的收益發送到受益人那裡。

帳本如下：

Contract Overview

Balance: 5 Ether

Transactions: 6 txns

Misc:

TxHash	Block	Age	From	To	Value	[TxFee]	
0x772573f11fe0826...	4597786	1 min ago	0x527bd4e8d97ebc...	IN	0xba1b71dd065a4e...	0 Ether	0.000022947
0xf75cdf0a66716fb...	4597782	2 mins ago	0x527bd4e8d97ebc...	IN	0xba1b71dd065a4e...	0 Ether	0.000044123
0x87796b1534a90d...	4597695	18 mins ago	0x130246b217355a...	IN	0xba1b71dd065a4e...	2 Ether	0.000064498
0x8d717615c28ccb...	4597694	19 mins ago	0x4970760c3bc165...	IN	0xba1b71dd065a4e...	2 Ether	0.000079498
0x5a899db47ddce1...	4597692	19 mins ago	0x527bd4e8d97ebc...	IN	0xba1b71dd065a4e...	1 Ether	0.000094498
0x34830292310586...	4597660	29 mins ago	0x9868b975cda371...	IN	Contract Creation	0 Ether	0.000760095

[ Download CSV Export ]

接著在受益人的帳本中可以看到，項目的智能合約地址向受益人轉賬了5個代幣。

Address Overview

Balance: 7.999206933 Ether

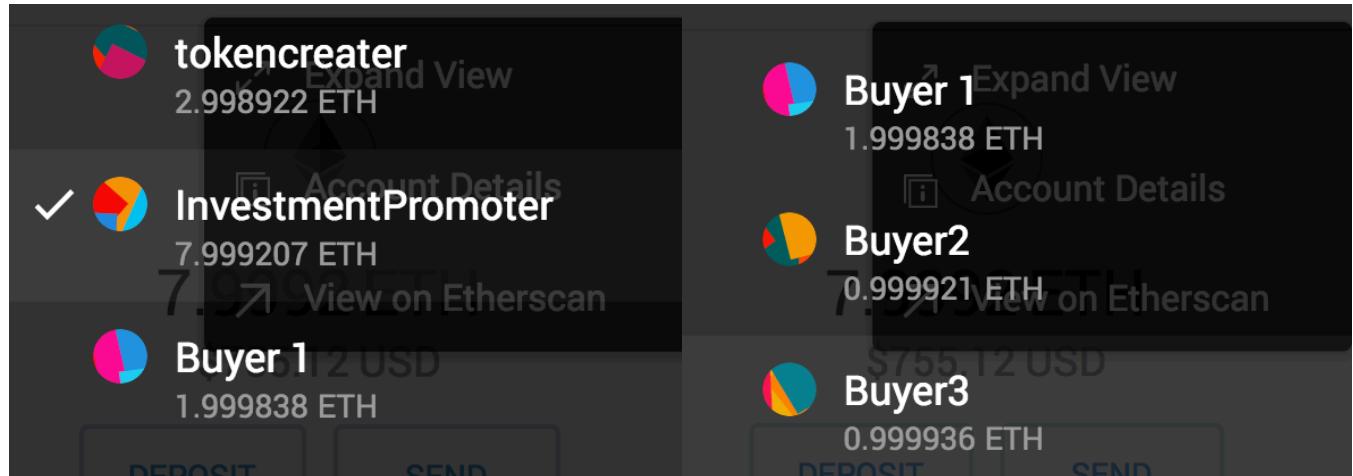
Transactions: 3 txns

Internal Txns

Internal Transactions as a result of Contract Execution

ParentTxHash	Block	Age	From	To	Value
0xfcfae59a1302cbef...	4597800	1 min ago	0xba1b71dd065a4e...	→ 0x9868b975cda371...	5 Ether

最後再來看一下交易後各帳號的資金情況。



一個關於投資的一個區塊鏈智能合約，在這裡說已經完成了。接下來實際利益分配等的情況，就只能在項目開始前就通過一系列的協商協議來制定。

調試結束，項目完成。