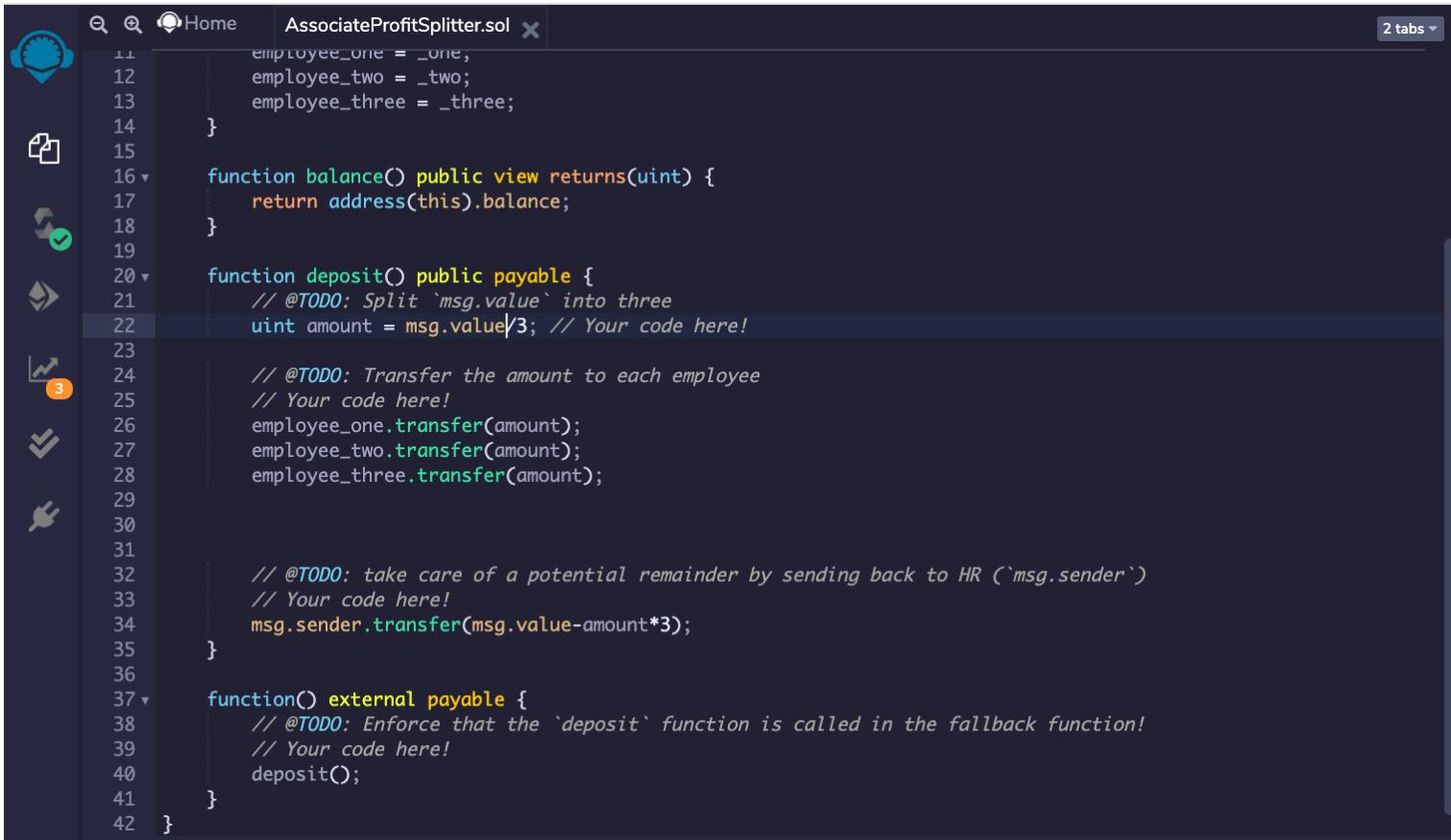


Create the
first contract



The screenshot shows a code editor interface with a dark theme. On the left, there is a vertical toolbar with several icons: a blue gear, a white square with a black outline, a green checkmark, a blue diamond, a red circle with a white '3', a green checkmark, and a blue plug. The main area is titled "AssociateProfitSplitter.sol". The code is written in Solidity and defines a single contract with the following functions:

```
11     employee_one = _one;
12     employee_two = _two;
13     employee_three = _three;
14 }
15
16 function balance() public view returns(uint) {
17     return address(this).balance;
18 }
19
20 function deposit() public payable {
21     // @TODO: Split `msg.value` into three
22     uint amount = msg.value/3; // Your code here!
23
24     // @TODO: Transfer the amount to each employee
25     // Your code here!
26     employee_one.transfer(amount);
27     employee_two.transfer(amount);
28     employee_three.transfer(amount);
29
30
31
32     // @TODO: take care of a potential remainder by sending back to HR (`msg.sender`)
33     // Your code here!
34     msg.sender.transfer(msg.value-amount*3);
35 }
36
37 function() external payable {
38     // @TODO: Enforce that the `deposit` function is called in the fallback function!
39     // Your code here!
40     deposit();
41 }
42 }
```

The code includes several TODO comments for the developer to implement the logic for splitting the deposit and handling the remainder.

Compiling first contract

The screenshot shows the Remix IDE interface. On the left, the Solidity Compiler settings are visible, including the compiler version (0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), and various configuration options like auto compile, optimization level (200), and hide warnings. A prominent blue button labeled "Compile AssociateProfitSplitter.sol" is centered below these settings. To the right, the "AssociateProfitSplitter.sol" file is open in the editor. The code defines a simple contract with three employees and a deposit function that splits the amount into thirds. The interface includes tabs for Home, AssociateProfitSplitter.sol, and another tab (partially visible). Below the editor, there's a terminal window with help messages and a welcome message for Remix 0.10.10.

```
employee_one = _one;
employee_two = _two;
employee_three = _three;

function balance() public view returns(uint) {
    return address(this).balance;
}

function deposit() public payable {
    // @TODO: Split `msg.value` into three
    uint amount = msg.value/3; // Your code here!

    // @TODO: Transfer the amount to each employee
    // Your code here!
    employee_one.transfer(amount);
    employee_two.transfer(amount);
    employee_three.transfer(amount);

    // @TODO: take care of a potential remainder by sending back to
    // Your code here!
    msg.sender.transfer(msg.value-amount*3);
}

function() external payable {
    // @TODO: Enforce that the `deposit` function is called in the
    // Your code here!
    deposit();
}
```

CONTRACT
AssociateProfitSplitter (AssociateProfitSplitter.sol)

Publish on Swarm

Publish on Ipfs

Compilation Details

API Reference

Home AssociateProfitSplitter.sol 2 tabs

Search with transaction hash or address

remix.help(): Display this help message

- Welcome to Remix 0.10.10 -

You can use this terminal to:

- Check transactions details and start debugging.

Deploy the first contract

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area is titled "DEPLOY & RUN TRANSACTIONS". Under "ENVIRONMENT", it says "Injected Web3" and "Custom (5777) network". Under "ACCOUNT", it shows "0xb9D..2C553 (96.825442)". Under "GAS LIMIT", it's set to "3000000". Under "VALUE", it shows "0 wei". In the "CONTRACT" section, the code for "AssociateProfitSplitter" is visible. A large orange "Deploy" button is prominent. Below it, there are options for "Publish to IPFS" and "At Address". The status bar at the bottom indicates "Transactions recorded 0" and "Deployed Contracts". A message box says "Currently you have no contract instances to interact with."

On the right side, the code editor window is open with the file "AssociateProfitSplitter.sol". The code is as follows:

```
employee_one = _one;
employee_two = _two;
employee_three = _three;

function balance() public view returns(uint) {
    return address(this).balance;
}

function deposit() public payable {
    // @TODO: Split `msg.value` into three
    uint amount = msg.value/3; // Your code here!

    // @TODO: Transfer the amount to each employee
    // Your code here!
    employee_one.transfer(amount);
    employee_two.transfer(amount);
    employee_three.transfer(amount);

    // @TODO: take care of a potential remainder by sending back to
    // Your code here!
    msg.sender.transfer(msg.value-amount*3);
}

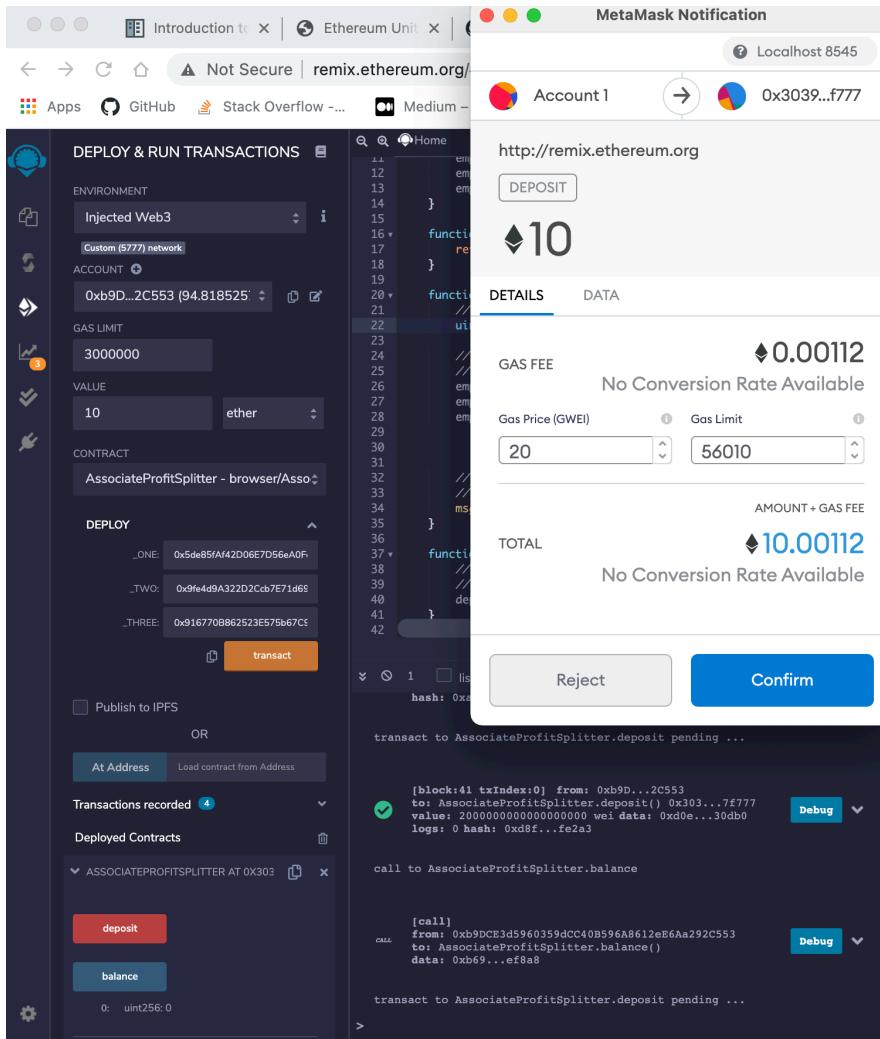
function() external payable {
    // @TODO: Enforce that the `deposit` function is called in the
    // Your code here!
    deposit();
}
```

The status bar at the bottom of the code editor shows "remix.help(): Display this help message" and "- Welcome to Remix 0.10.10 -". It also has a terminal section with instructions: "You can use this terminal to:" followed by a list: "• Check transactions details and start debugging.", "• Execute JavaScript scripts.", and "• Run your own command-line tools via Node.js".

Addresses used for the Employees

CURRENT BLOCK 38	GAS PRICE 2000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:8545	MINING STATUS AUTOMINING	WORKSPACE FINTECH	SWITCH	
MNEMONIC fantasy trip night legend neck naive cave share harbor album hen hood					HD PATH m/44'/60'/0'/0/account_index				
ADDRESS 0xb9DCE3d5960359dCC40B596A8612eE6Aa292C553	BALANCE 96.83 ETH						TX COUNT 38	INDEX 0	
ADDRESS 0x4f40f3D0169a02D37035548B6cE82284A4C083De	BALANCE 100.00 ETH						TX COUNT 0	INDEX 1	
ADDRESS 0x19632B1e8811863aCaBfdC331F1F638368DE2318	BALANCE 100.00 ETH						TX COUNT 0	INDEX 2	
ADDRESS 0xF0b19a0ef4894d4B32cDEc0e8Bd7a42bD95225c3	BALANCE 100.00 ETH						TX COUNT 0	INDEX 3	
ADDRESS 0x5de85fAf42D06E7D56eA0FeF469dd8BeF94Ed7D8	BALANCE 100.00 ETH						TX COUNT 0	INDEX 4	
ADDRESS 0x9fe4d9A322D2Ccb7E71d690dB01452f3116C7183	BALANCE 100.00 ETH						TX COUNT 0	INDEX 5	
ADDRESS 0x916770B862523E575b67C9aFBBe8420b8B3c14e5	BALANCE 100.00 ETH						TX COUNT 0	INDEX 6	

First transaction

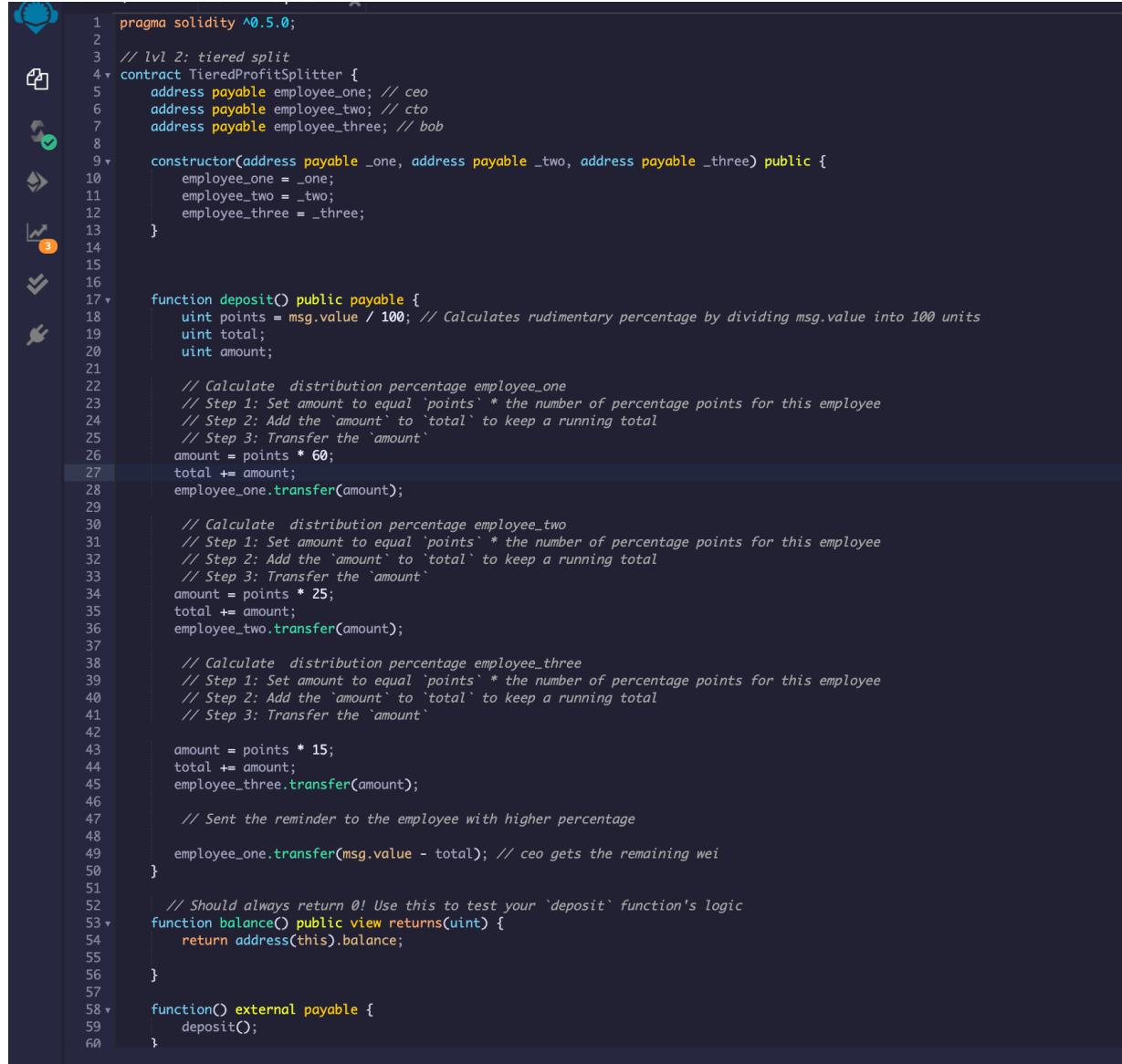


Deploy the first contract in live Tesnet

The screenshot shows the Remix IDE interface with the following details:

- Environment:** Injected Web3, Kovan (42) network.
- Account:** 0xb9D...2C553 (1.0345595).
- Gas Limit:** 3000000.
- Value:** 0 wei.
- Contract:** DeferredEquityPlan - browser/DeferredEquityPlan.sol.
- Deployment Status:** Deploy button is orange, indicating a successful deployment to address 0xF0b19a0ef4894d4B32cD.
- Transactions Recorded:** 2 transactions recorded.
- Deployed Contracts:** Two contracts listed: DEFERREDEQUITYPLAN AT 0XACB..06 and DEFERREDEQUITYPLAN AT 0XFCE..67.
- Code View:** Solidity code for AssociateProfitSplitter.sol and DeferredEquityPlan.sol.
- Right Panel:** Shows Account 1 (0xb9D...2C553) with 1.0346 ETH, and two Contract Deployment entries from Feb 22, both costing 0 ETH.

Create second contract



```
pragma solidity ^0.5.0;

// lvl 2: tiered split
contract TieredProfitSplitter {
    address payable employee_one; // ceo
    address payable employee_two; // cto
    address payable employee_three; // bob

    constructor(address payable _one, address payable _two, address payable _three) public {
        employee_one = _one;
        employee_two = _two;
        employee_three = _three;
    }

    function deposit() public payable {
        uint points = msg.value / 100; // Calculates rudimentary percentage by dividing msg.value into 100 units
        uint total;
        uint amount;

        // Calculate distribution percentage employee_one
        // Step 1: Set amount to equal 'points' * the number of percentage points for this employee
        // Step 2: Add the 'amount' to 'total' to keep a running total
        // Step 3: Transfer the 'amount'
        amount = points * 60;
        total += amount;
        employee_one.transfer(amount);

        // Calculate distribution percentage employee_two
        // Step 1: Set amount to equal 'points' * the number of percentage points for this employee
        // Step 2: Add the 'amount' to 'total' to keep a running total
        // Step 3: Transfer the 'amount'
        amount = points * 25;
        total += amount;
        employee_two.transfer(amount);

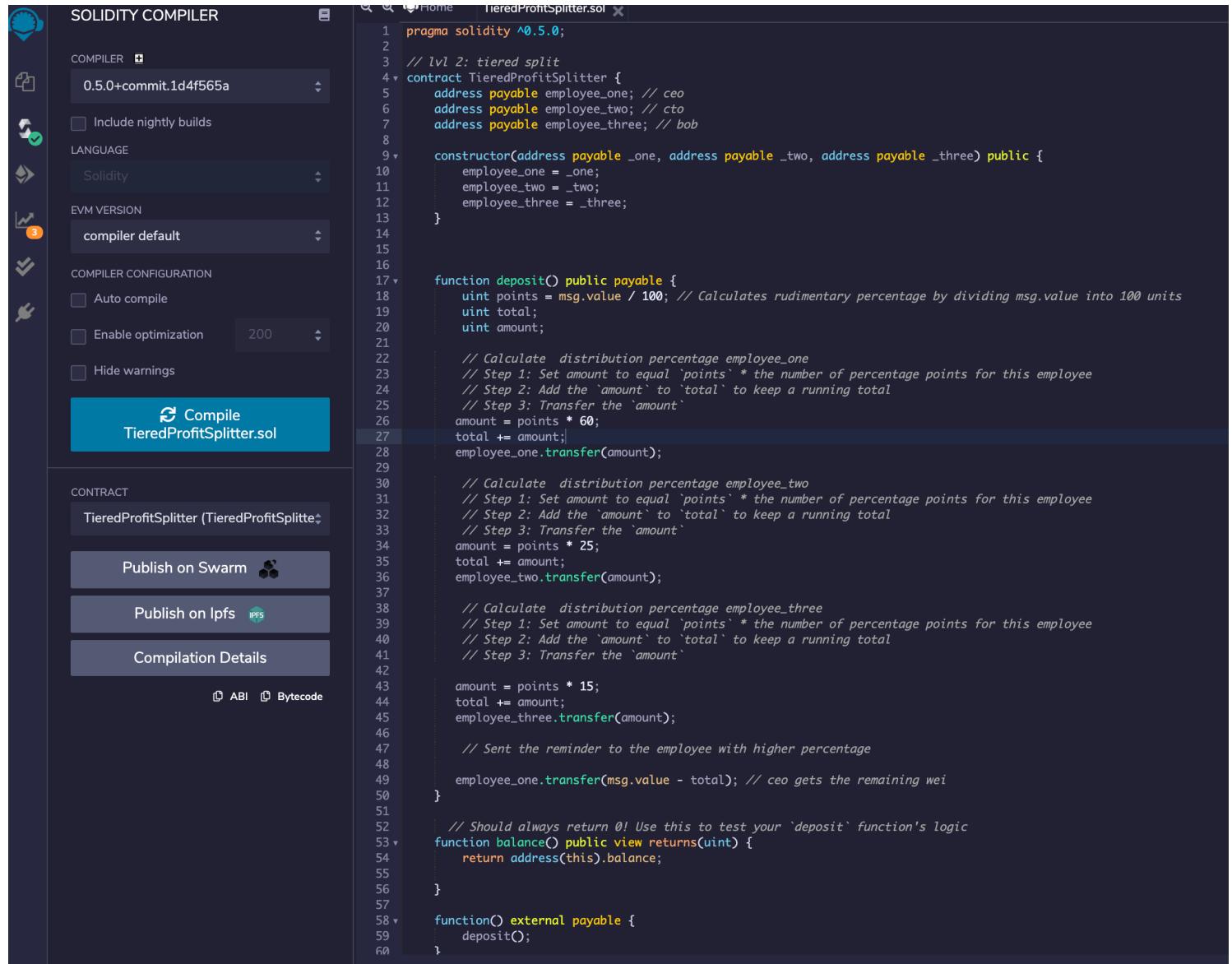
        // Calculate distribution percentage employee_three
        // Step 1: Set amount to equal 'points' * the number of percentage points for this employee
        // Step 2: Add the 'amount' to 'total' to keep a running total
        // Step 3: Transfer the 'amount'
        amount = points * 15;
        total += amount;
        employee_three.transfer(amount);

        // Sent the remainder to the employee with higher percentage
        employee_one.transfer(msg.value - total); // ceo gets the remaining wei
    }

    // Should always return 0! Use this to test your 'deposit' function's logic
    function balance() public view returns(uint) {
        return address(this).balance;
    }

    function() external payable {
        deposit();
    }
}
```

Compile second contract



The image shows the Solidity Compiler interface with the following details:

- SOLIDITY COMPILER** tab is selected.
- COMPILER**: 0.5.0+commit.1d4f565a
- Include nightly builds
- LANGUAGE**: Solidity
- EVM VERSION**: compiler default
- COMPILER CONFIGURATION**:
 - Auto compile
 - Enable optimization (set to 200)
 - Hide warnings
- Compile TieredProfitSplitter.sol** button
- CONTRACT**: TieredProfitSplitter (TieredProfitSplitter.sol)
- Publish on Swarm** button
- Publish on Ipfs** button
- Compilation Details** button
- ABI** and **Bytecode** links

The code editor on the right contains the `TieredProfitSplitter.sol` file:

```
pragma solidity ^0.5.0;

// lvl 2: tiered split
contract TieredProfitSplitter {
    address payable employee_one; // ceo
    address payable employee_two; // cto
    address payable employee_three; // bob

    constructor(address payable _one, address payable _two, address payable _three) public {
        employee_one = _one;
        employee_two = _two;
        employee_three = _three;
    }

    function deposit() public payable {
        uint points = msg.value / 100; // Calculates rudimentary percentage by dividing msg.value into 100 units
        uint total;
        uint amount;

        // Calculate distribution percentage employee_one
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 60;
        total += amount;
        employee_one.transfer(amount);

        // Calculate distribution percentage employee_two
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 25;
        total += amount;
        employee_two.transfer(amount);

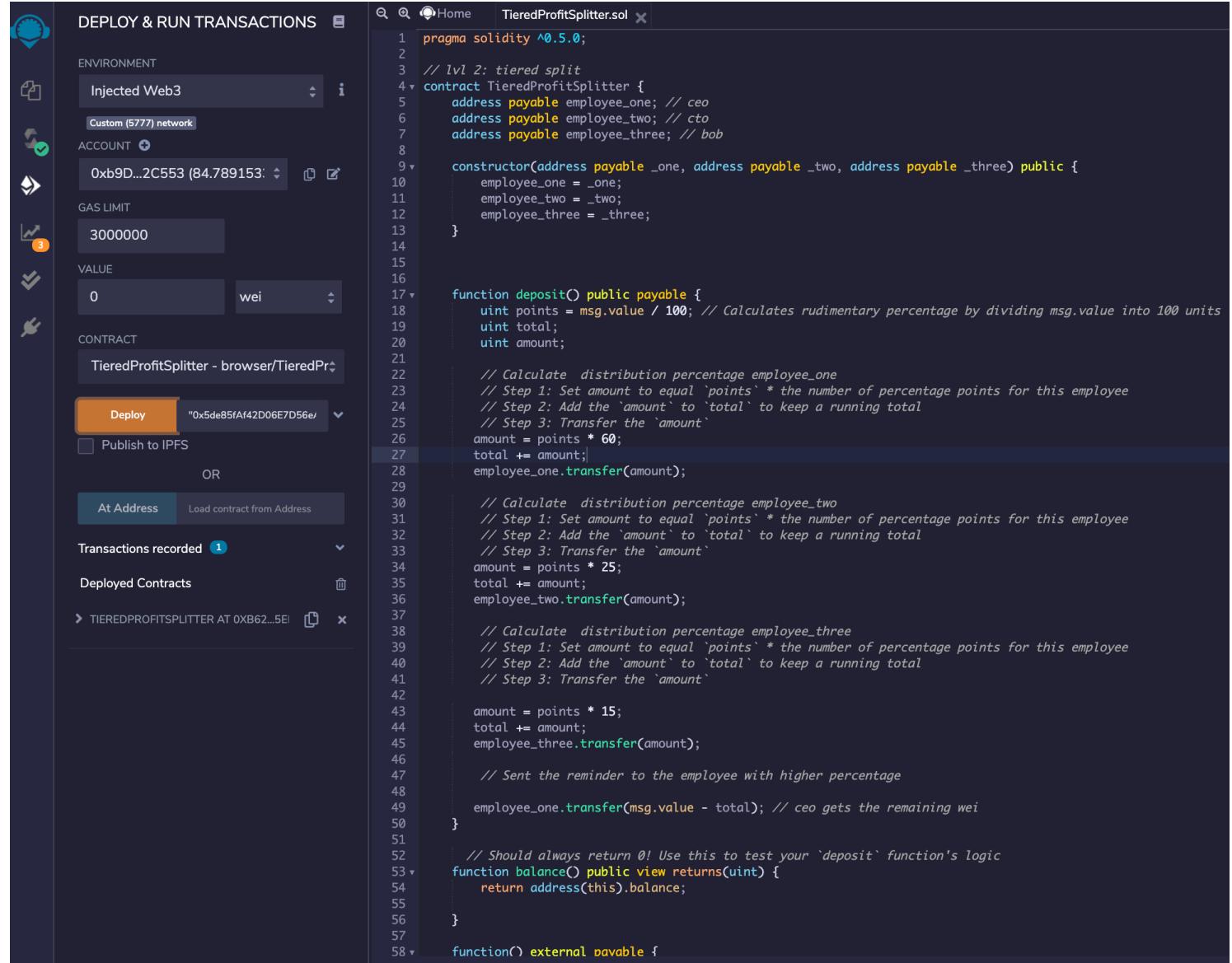
        // Calculate distribution percentage employee_three
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 15;
        total += amount;
        employee_three.transfer(amount);

        // Send the remainder to the employee with higher percentage
        employee_one.transfer(msg.value - total); // ceo gets the remaining wei
    }

    // Should always return 0! Use this to test your `deposit` function's logic
    function balance() public view returns(uint) {
        return address(this).balance;
    }

    function() external payable {
        deposit();
    }
}
```

Deploy Second contract



The screenshot shows the Truffle UI interface for deploying a Solidity smart contract named `TieredProfitSplitter.sol`. The left panel, titled "DEPLOY & RUN TRANSACTIONS", contains configuration fields for the environment (set to "Injected Web3" and "Custom (5777) network"), account (0xb9D...2C553), gas limit (3000000), and value (0 wei). The "CONTRACT" section shows the selected contract as "TieredProfitSplitter - browser/TieredPr". Below it, there is a "Deploy" button and a dropdown menu for publishing to IPFS. The right panel displays the Solidity code for the `TieredProfitSplitter` contract, which handles profit distribution across three employees based on their percentage points.

```
pragma solidity ^0.5.0;

// lvl 2: tiered split
contract TieredProfitSplitter {
    address payable employee_one; // ceo
    address payable employee_two; // cto
    address payable employee_three; // bob

    constructor(address payable _one, address payable _two, address payable _three) public {
        employee_one = _one;
        employee_two = _two;
        employee_three = _three;
    }

    function deposit() public payable {
        uint points = msg.value / 100; // Calculates rudimentary percentage by dividing msg.value into 100 units
        uint total;
        uint amount;

        // Calculate distribution percentage employee_one
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 60;
        total += amount;
        employee_one.transfer(amount);

        // Calculate distribution percentage employee_two
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 25;
        total += amount;
        employee_two.transfer(amount);

        // Calculate distribution percentage employee_three
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 15;
        total += amount;
        employee_three.transfer(amount);

        // Sent the remainder to the employee with higher percentage
        employee_one.transfer(msg.value - total); // ceo gets the remaining wei
    }

    // Should always return 0! Use this to test your `deposit` function's logic
    function balance() public view returns(uint) {
        return address(this).balance;
    }

    function() external payable {
```

Test second contract

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' tab is active, displaying the deployment parameters: Environment (Injected Web3), Account (0xb9b...2C553), Gas Limit (3000000), Value (11110 wei), and Contract (TieredProfitSplitter). The 'Deploy' button is highlighted. Below it, the 'Transactions recorded' section shows a single transaction for 'deposit' to the deployed contract address (0x5de85fA42D06E7D56e). The 'Deployed Contracts' section lists 'TIEREDPROFITSPLITTER AT 0XB62...5E' with two functions: 'deposit' and 'balance'. The 'Low level interactions' section has a 'CALLDATA' button and a 'Transact' button. On the right, the 'CODE' tab is open, showing the Solidity source code for the 'TieredProfitSplitter' contract. The code implements a tiered profit splitting logic where the amount is divided into 100 units and distributed to three employees based on their percentage points. The 'DETAILS' tab is also visible, showing the transaction details: GAS FEE (0.001139 GWEI), TOTAL (0.001139 GWEI), and buttons for 'Reject' and 'Confirm'. A 'MetaMask Notification' is visible at the top right, indicating a balance of 0.

```
pragma solidity ^0.5.0;

// lvl 2: tiered split
contract TieredProfitSplitter {
    address payable employee_one; // ceo
    address payable employee_two; // cto
    address payable employee_three; // bob

    constructor(address payable _one, address payable _two, address payable _three) public {
        employee_one = _one;
        employee_two = _two;
        employee_three = _three;
    }

    function deposit() public payable {
        uint points = msg.value / 100; // Calculates rudimentary percentage by dividing msg.value into 100 units
        uint total;
        uint amount;

        // Calculate distribution percentage employee_one
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the amount
        amount = points * 60;
        total += amount;
        employee_one.transfer(amount);

        // Calculate distribution percentage employee_two
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the amount
        amount = points * 25;
        total += amount;
        employee_two.transfer(amount);

        // Calculate distribution percentage employee_three
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the amount
        amount = points * 15;
        total += amount;
        employee_three.transfer(amount);

        // Sent the reminder to the employee with higher percentage
        employee_one.transfer(msg.value - total); // ceo gets the remaining wei
    }

    // Should always return 0! Use this to test your `deposit` function's logic
    function balance() public view returns(uint) {
        return address(this).balance;
    }
}

function() external payable {
```

Deploy the first contract in live Testnet

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is open, showing the environment as 'Injected Web3' (Kovan (42) network), account address '0xb9D...2C553 (1.0425959)', gas limit set to 3000000, and value set to 0 wei. The 'CONTRACT' section displays the Solidity code for 'TieredProfitSplitter'. The code defines a constructor that takes three payable addresses for employees one, two, and three. It includes placeholder logic for splitting the deposit and sending it to each employee. The 'DEPLOY' section shows the deployed contract at address '0x5de85fa42d06e7d56ea0f...' with three associated employee contracts at addresses '0x9fe4d9A322D2Cb7E71d69...', '0x916770B862523E57b67CE...', and '0x5de85fa42d06e7d56ea0f...'. A large orange 'transact' button is visible. On the right, the 'Account 1' panel shows a balance of 1.0426 ETH with buttons for Buy, Send, and Swap. Below it, the 'Activity' section shows a pending 'Contract Deployment' from 'remix.ethereum.org' with a status of 'Pending'.

```
1 pragma solidity ^0.5.0;
2
3 // lvl 1: equal split
4 v contract AssociateProfitSplitter {
5     // @TODO: Create three payable addresses representing `employee_one`, `employee_two` and `employee_three`.
6     address payable employee_one;
7     address payable employee_two;
8     address payable employee_three;
9
10    constructor(address payable _one, address payable _two, address payable _three) public {
11        employee_one = _one;
12        employee_two = _two;
13        employee_three = _three;
14    }
15
16    function balance() public view returns(uint) {
17        return address(this).balance;
18    }
19
20    function deposit() public payable {
21        // @TODO: Split `msg.value` into three
22        uint amount = msg.value/3; // Your code here!
23
24        // @TODO: Transfer the amount to each employee
25        // Your code here!
26        employee_one.transfer(amount);
27        employee_two.transfer(amount);
28        employee_three.transfer(amount);
29
30
31        // @TODO: take care of a potential remainder by sending back to HR (`msg.sender`)
32        // Your code here!
33        msg.sender.transfer(msg.value-amount*3);
34    }
35
36    function() external payable {
37        // @TODO: Enforce that the `deposit` function is called in the fallback function!
38        // Your code here!
39        deposit();
40    }
41
42 }
```

Deploy second contract in the live Testnet

The screenshot shows the Truffle UI interface for deploying a Solidity smart contract named `TieredProfitSplitter`. The left sidebar contains navigation links like Home, Contracts, Migrations, and Settings. The main area is titled "DEPLOY & RUN TRANSACTIONS".

ENVIRONMENT: Set to "Injected Web3" and "Kovan (42) network".

ACCOUNT: Address "0xb9D...2C553 (1.0389462)".

GAS LIMIT: Set to 3000000.

VALUE: 0 wei.

CONTRACT: Selected "TieredProfitSplitter - browser/TieredPr..."

DEPLOY: Fields for "_ONE", "_TWO", and "_THREE" with their respective addresses filled in. A "transact" button is present.

Publish to IPFS: Unchecked checkbox.

OR

At Address: Input field for loading a contract from an address.

Transactions recorded: 6 transactions.

Deployed Contracts: A dropdown menu showing "TIEREDPROFITSPLITTER AT 0XB62...5E".

Low level interactions: Buttons for "deposit" and "balance".

CALLDATA: A section for entering call data.

Code View: Displays the Solidity code for `TieredProfitSplitter.sol`.

```
pragma solidity ^0.5.0;

// lvl 2: tiered split
contract TieredProfitSplitter {
    address payable employee_one; // ceo
    address payable employee_two; // cto
    address payable employee_three; // bob

    constructor(address payable _one, address payable _two, address payable _three) public {
        employee_one = _one;
        employee_two = _two;
        employee_three = _three;
    }

    function deposit() public payable {
        uint points = msg.value / 100; // Calculates rudimentary percentage by dividing msg.value into 100 units
        uint total;
        uint amount;

        // Calculate distribution percentage employee_one
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 60;
        total += amount;
        employee_one.transfer(amount);

        // Calculate distribution percentage employee_two
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: Transfer the `amount`
        amount = points * 25;
        total += amount;
        employee_two.transfer(amount);

        // Calculate distribution percentage employee_three
        // Step 1: Set amount to equal `points` * the number of percentage points for this employee
        // Step 2: Add the `amount` to `total` to keep a running total
        // Step 3: transfer the `amount`
        amount = points * 15;
        total += amount;
        employee_three.transfer(amount);

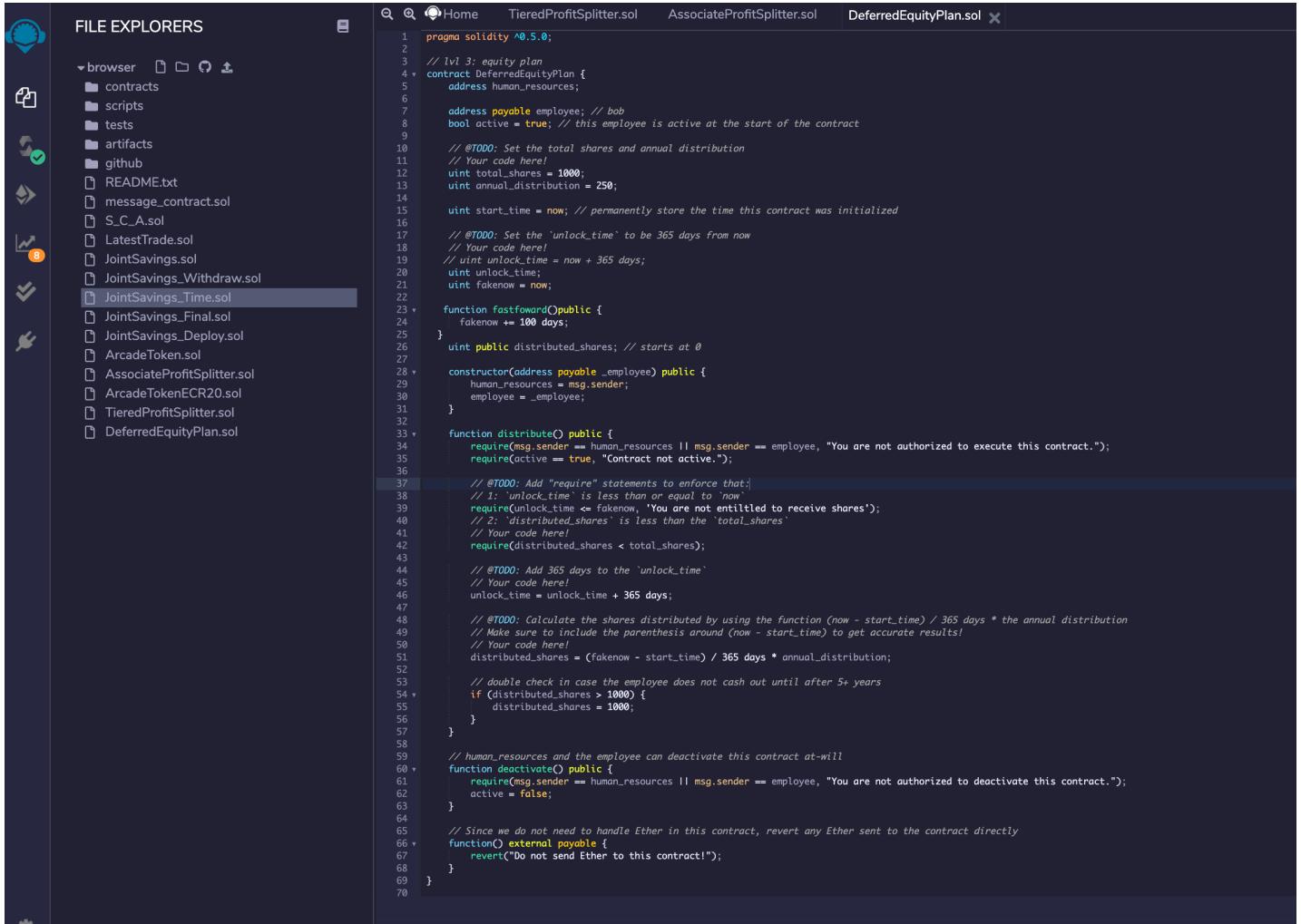
        // Send the remainder to the employee with higher percentage
        employee_one.transfer(msg.value - total); // ceo gets the remaining wei
    }

    // Should always return 0! Use this to test your `deposit` function's logic
    function balance() public view returns(uint) {
        return address(this).balance;
    }

    function() external payable {
```

Bottom Bar: Includes a "Transact" button, a search bar for transaction hash or address, and a "listen on network" checkbox.

Create the third contract



The screenshot shows a code editor interface with two main panes. The left pane is titled 'FILE EXPLORERS' and lists several Solidity files: browser, contracts, scripts, tests, artifacts, github, README.txt, message_contract.sol, S.C_A.sol, LatestTrade.sol, JointSavings.sol, JointSavings_Withdraw.sol, JointSavings_Time.sol (which is selected), JointSavings_Final.sol, JointSavings_Deploy.sol, ArcadeToken.sol, AssociateProfitSplitter.sol, ArcadeTokenERC20.sol, TieredProfitSplitter.sol, and DeferredEquityPlan.sol. The right pane displays the content of the 'DeferredEquityPlan.sol' file, which is a Solidity smart contract. The code includes pragmas, imports, and various functions for managing employee equity plans, including calculations for unlock times and share distributions.

```
pragma solidity ^0.5.0;
// lvl 3: equity plan
contract DeferredEquityPlan {
    address human_resources;
    address payable employee; // bob
    bool active = true; // this employee is active at the start of the contract
    // @TODO: Set the total shares and annual distribution
    // Your code here!
    uint total_shares = 1000;
    uint annual_distribution = 250;
    uint start_time = now; // permanently store the time this contract was initialized
    // @TODO: Set the 'unlock_time' to be 365 days from now
    // Your code here!
    uint unlock_time = now + 365 days;
    uint unlock_time;
    uint fakenow = now;
    function fastforward() public {
        fakenow += 100 days;
    }
    uint public distributed_shares; // starts at 0
    constructor(address payable _employee) public {
        human_resources = msg.sender;
        employee = _employee;
    }
    function distribute() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to execute this contract.");
        require(active == true, "Contract not active.");
        // @TODO: Add "require" statements to enforce that:
        // 1: 'unlock_time' is less than or equal to 'now'
        require(unlock_time <= fakenow, 'You are not entitled to receive shares');
        // 2: 'distributed_shares' is less than the 'total_shares'
        require(distributed_shares < total_shares);
        // @TODO: Add 365 days to the 'unlock_time'
        // Your code here!
        unlock_time = unlock_time + 365 days;
        // @TODO: Calculate the shares distributed by using the function (now - start_time) / 365 days * the annual distribution
        // Make sure to include the parenthesis around (now - start_time) to get accurate results!
        // Your code here!
        distributed_shares = (fakenow - start_time) / 365 days * annual_distribution;
        // double check in case the employee does not cash out until after 5+ years
        if (distributed_shares > 1000) {
            distributed_shares = 1000;
        }
    }
    // human_resources and the employee can deactivate this contract at-will
    function deactivate() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to deactivate this contract.");
        active = false;
    }
    // Since we do not need to handle Ether in this contract, revert any Ether sent to the contract directly
    function() external payable {
        revert("Do not send Ether to this contract!");
    }
}
```

Compile the third contract

The screenshot shows the Remix Ethereum IDE interface. On the left, there's a sidebar with a large orange arrow pointing right. The main area has three tabs at the top: Home, TieredProfitSplitter.sol, AssociateProfitSplitter.sol, and DeferredEquityPlan.sol (which is currently active). The DeferredEquityPlan.sol tab displays the Solidity code for the contract.

```
pragma solidity >=0.5;
```

```
// lvl 3: equity plan
contract DeferredEquityPlan {
    address human_resources;
    bool active = true; // this employee is active at the start of the contract
    uint total_shares = 1000;
    uint annual_distribution = 250;
    uint start_time = now; // permanently store the time this contract was initialized
    uint unlock_time = now + 365 days; // Your code here!
    uint unlock_time_c = now + 365 days; // Your code here!
    uint tokensnow = now;
    function fastForward() public {
        tokensnow += 100 days;
    }
    uint public distributed_shares; // starts at 0
    constructor(address payable _employee) public {
        human_resources = msg.sender;
        employee = _employee;
    }
    function distribute() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to execute this contract.");
        require(active == true, "Contract not active.");
        // #7000: Add "require" statements to enforce that:
        // 1: unlock_time is less than now
        require(unlock_time < now, "You are not entitled to receive shares");
        // 2: "distributed_shares" is less than the "total_shares"
        require(distributed_shares < total_shares);
        // #7000: Add 365 days to the "unlock_time"
        unlock_time = unlock_time + 365 days;
        // #7000: calculate the shares distributed by using the function (now - start_time) / 365 days * the annual distribution
        // Make sure to include the parenthesis around (now - start_time) to get accurate results!
        distributed_shares = (tokensnow - start_time) / 365 days * annual_distribution;
        // double check in case the employee does not cash out until after 5+ years
        if (distributed_shares > 1000) {
            distributed_shares = 1000;
        }
    }
    // human_resources and the employee can deactivate this contract at-will
    function deactivate() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to deactivate this contract.");
        active = false;
    }
    // Since we do not need to handle Ether in this contract, revert any Ether sent to the contract directly
    function() external payable {
        revert("Do not send Ether to this contract!");
    }
}
```

The central part of the interface shows the "DEPLOY & RUN TRANSACTIONS" section. It includes fields for ENVIRONMENT (Injected Web3), ACCOUNT (0xb9D...2C553), GAS LIMIT (300000), and VALUE (0 wei). Below these are sections for CONTRACT (DeferredEquityPlan - browser/Deferec) and DEPLOYED CONTRACTS (DEFERREDEQUITYPLAN at 0x088...B9). A "Deploy" button is present, along with options to "Publish to IPFS" or "At Address".

To the right, the "Account 1" panel is visible, showing a balance of 84.7689 ETH with "Buy", "Send", and "Swap" buttons. Below this, the "Activity" tab is selected, displaying two entries for "Contract Deployment" on Feb 22 from remix.ethereum.org, both with a balance of -0 ETH.

Deploy the third contract

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
Injected Web3
Custom (577) network

ACCOUNT
0xb9D...2C53 (84.768864)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
DeferredEquityPlan - browser/DeferredEquityPlan.sol

Deploy address _employee

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

DEFERREDEQUITYPLAN AT 0X088...B9

```
pragma solidity ^0.5.0;

// lvl 3: equity plan
contract DeferredEquityPlan {
    address human_resources;
    address payable employee; // bob
    bool active = true; // this employee is active at the start of the contract

    // @TODO: Set the total shares and annual distribution
    // Your code here!
    uint total_shares = 1000;
    uint annual_distribution = 250;

    uint start_time = now; // permanently store the time this contract was initialized
    // @TODO: Set the `unlock_time` to be 365 days from now
    // Your code here!
    uint unlock_time = now + 365 days;

    uint public distributed_shares; // starts at 0

    constructor(address payable _employee) public {
        human_resources = msg.sender;
        employee = _employee;
    }

    function distribute() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to execute this contract.");
        require(active == true, "Contract not active.");

        // @TODO: Add "require" statements to enforce that:
        // 1: `unlock_time` is less than or equal to `now`
        require(unlock_time <= now, "You are not entitled to receive shares");
        // 2: `distributed_shares` is less than the `total_shares`
        // Your code here!
        require(distributed_shares < total_shares);

        // @TODO: Add 365 days to the `unlock_time`
        // Your code here!
        unlock_time = unlock_time + 365 days;

        // @TODO: Calculate the shares distributed by using the function (now - start_time) / 365 days * the annual distribution
        // Make sure to include the parenthesis around (now - start_time) to get accurate results!
        // Your code here!
        distributed_shares = (now - start_time) / 365 days * annual_distribution;

        // double check in case the employee does not cash out until after 5+ years
        if (distributed_shares > 1000) {
            distributed_shares = 1000;
        }
    }

    // human_resources and the employee can deactivate this contract at-will
    function deactivate() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to deactivate this contract.");
        active = false;
    }

    // Since we do not need to handle Ether in this contract, revert any Ether sent to the contract directly
    function() external payable {
        revert("Do not send Ether to this contract!");
    }
}
```

Test the third contract

The screenshot shows the Remix IDE interface with three main panels:

- Left Panel (Deploy & Run Transactions):** Shows the environment set to "Injected Web3" and the account set to "0xb9D...2C553 (84.759116)". It has fields for GAS LIMIT (3000000) and VALUE (0 wei). The CONTRACT dropdown is set to "DeferredEquityPlan - browser/DeferredEquityPlan.sol". A "Deploy" button is visible, along with options to "Publish to IPFS" or "At Address". Below these are sections for "Transactions recorded" and "Deployed Contracts".
- Middle Panel (Code Editor):** Displays the Solidity code for the `DeferredEquityPlan.sol` contract. The code includes comments for initializing variables, setting unlock times, distributing shares, and handling deactivate requests.
- Right Panel (Account Overview):** Shows a balance of 84.7591ETH. It has tabs for "Assets" (selected) and "Activity". Under "Assets", there are two entries for "Contract Deployment" on Feb 22 from remix.ethereum.org, both showing a balance of -0 ETH.

```
pragma solidity ^0.5.0;
// lvl 3: equity plan
contract DeferredEquityPlan {
    address payable employee; // bob
    bool active = true; // this employee is active at the start of the contract
    // #TODO: Set the total shares and annual distribution
    // Your code here!
    uint total_shares = 1000;
    uint annual_distribution = 250;
    uint start_time = now; // permanently store the time this contract was initialized
    // #TODO: Set the unlock_time to be 365 days from now
    uint unlock_time = now + 365 days;
    uint public distributed_shares; // starts at 0
    constructor(address payable _employee) public {
        human_resources = msg.sender;
        employee = _employee;
    }
    function distribute() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to execute this contract.");
        require(active == true, "Contract not active.");
        // #TODO: Add "require" statements to enforce that:
        // 1: unlock_time is less than or equal to now
        // 2: unlock_time is now. "You are not entitled to receive shares";
        // Your code here!
        require(distributed_shares < total_shares);
        // Your code here!
        unlock_time = unlock_time + 365 days;
        // #TODO: Calculate the shares distributed by using the function (now - start_time) / 365 days * the annual distribution
        // Make sure to include the parenthesis around (now - start_time) to get accurate results!
        distributed_shares = (now - start_time) / 365 days * annual_distribution;
        // double check in case the employee does not cash out until after 5+ years
        if (distributed_shares > 1000) {
            distributed_shares = 1000;
        }
    }
    // human_resources and the employee can deactivate this contract at-will
    function deactivate() public {
        require(msg.sender == human_resources || msg.sender == employee, "You are not authorized to deactivate this contract.");
        active = false;
    }
    // Since we do not need to handle Ether in this contract, revert any Ether sent to the contract directly
    function() external payable {
        revert("Do not send Ether to this contract!");
    }
}
```

Deploy the Third Contract in live Tesnet

The screenshot shows the Remix IDE interface with three main panels:

- Left Panel (Deploy & Run Transactions):** Shows the environment set to "Injected Web3" and "Kovan (42) network". The account is "0xb9D...C553 (1.0367528)". Gas limit is set to 3000000. Value is 0 wei. The contract selected is "DeferredEquityPlan - browser/Defere...". A "Deploy" button is highlighted in orange, showing the address "0xF0b19a0ef4894d4B32c". Below it are options to "Publish to IPFS" or "At Address". The status shows "Transactions recorded 1" and "Deployed Contracts" with one entry: "DEFERREDEQUITYPLAN AT 0xACB...06".
- Middle Panel (Code Editor):** Displays the Solidity code for the `DeferredEquityPlan` contract. The code defines a contract with variables for employee, total_shares, annual_distribution, start_time, unlock_time, and distributed_shares. It includes functions for distribution, deactivate, and a fallback function for external payable calls.
- Right Panel (Account Overview):** Shows Account 1 with address 0xb9D...C553 and balance 1.0368 ETH. It has buttons for Buy, Send, and Swap. Below that, the "Activity" tab is selected, showing two entries for "Contract Deployment" on Feb 22 from remix.ethereum.org, both with 0 ETH.

Copy the new addresses

0xB62DcFc3d9B5889cCbAf01d15a2EB81befF5EeE3

0xB62DcFc3d9B5889cCbAf01d15a2EB81befF5EeE3

0xACbbc4b7faC243eB95f1a12dFAd11E736E10613f

