Question:

Kickstarter is the world's largest funding platform for creative projects. Project team sets minimum and maximum funding goals. They should trust Kickstarter that if the amount of money will be reached, Kickstarter will pay the team.

On the other hand, backers need to trust Kickstarter that they will return the money invested in if the goal is not reached or send money to the project team if the goal is reached. Such solution is very **centralized**.

Rewards in crowd-funding are usually handled by a central unchangeable database that keeps track of all donors: anyone who missed the deadline for the campaign cannot get in anymore and any donor who changed their mind can't get out.

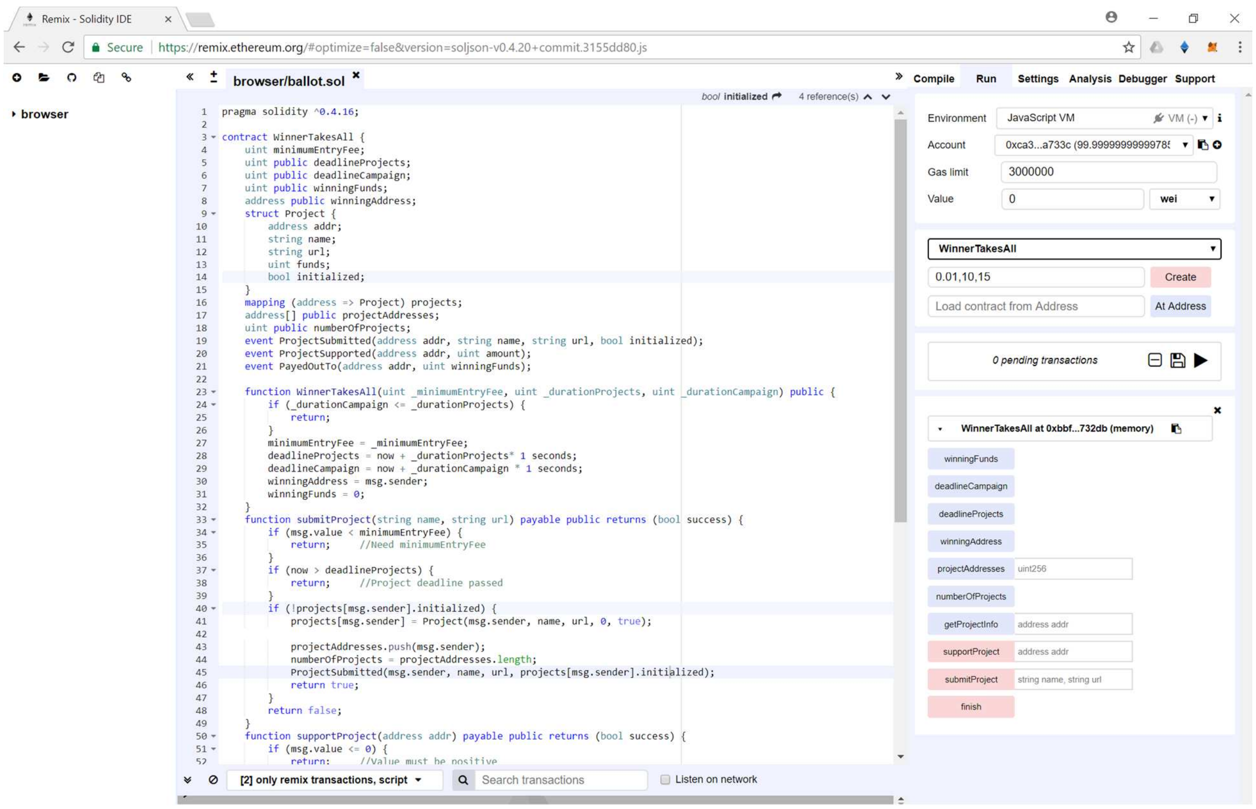
Here is what we can do to solve the problem of centralization and developing a trustless system.

We are going to do this the decentralized way and just create a token to keep track of rewards, anyone who contributes gets a token that they can trade, sell or keep for later. When the time comes to give the physical reward the producer only needs to exchange the tokens for real products. Donors get to keep their tokens, even if the project doesn't achieve its goals, as a souvenir.

Creating a Winner Takes All Crowdfunding Contract:

- Create the contract with
- Minimum Entry Fee for project proposals (to avoid spam)
- Project proposal deadline
- Campaign deadline
- Before the Project proposal deadline, people can enter their projects providing a name, a url and the entry fee
- When the Project proposal deadline is over, people can vote with ether (money) for the projects they want to support
- When the campaign deadline is over, ALL the pledged money (incl. entry fees) goes to the project with the highest votes (most ether received)
- And the contract is closed thereafter

Solution:



Environment: JavaScript VM
Account: 0xca3...a733c (99.99999999999978
Gas limit: 3000000
Value: 0 wei
WinnerTakesAll
0.01,10,15
Load contract from Address

```
pragma solidity ^0.4.16;

contract WinnerTakesAll {
    uint minimumEntryFee;
    uint public deadlineProjects;
    uint public deadlineCampaign;
    uint public winningFunds;
    address public winningAddress;
    struct Project {
        address addr;
        string name;
        string url;
        uint funds;
        bool initialized;
    }
    mapping (address => Project) projects;
    address[] public projectAddresses;
    uint public numberOfProjects;
    event ProjectSubmitted(address addr, string name, string url, bool initialized);
    event ProjectSupported(address addr, uint amount);
    event PayedOutTo(address addr, uint winningFunds);

    function WinnerTakesAll(uint _minimumEntryFee, uint _durationProjects, uint _durationCampaign) public {
        if (_durationCampaign <= _durationProjects) {
            return;
        }
        minimumEntryFee = _minimumEntryFee;
        deadlineProjects = now + _durationProjects* 1 seconds;
        deadlineCampaign = now + _durationCampaign * 1 seconds;
        winningAddress = msg.sender;
        winningFunds = 0;
    }
    function submitProject(string name, string url) payable public returns (bool success) {
        if (msg.value < minimumEntryFee) {
            return;     //Need minimumEntryFee
        }
        if (now > deadlineProjects) {
            return;     //Project deadline passed
        }
        if (!projects[msg.sender].initialized) {
            projects[msg.sender] = Project(msg.sender, name, url, 0, true);

            projectAddresses.push(msg.sender);
            numberOfProjects = projectAddresses.length;
            ProjectSubmitted(msg.sender, name, url, projects[msg.sender].initialized);
            return true;
        }
        return false;
    }
    function supportProject(address addr) payable public returns (bool success) {
        if (msg.value <= 0) {
            return;     //Value must be positive
```

Solidity Code

```
pragma solidity ^0.4.16;

contract WinnerTakesAll {

    uint minimumEntryFee;

    uint public deadlineProjects;

    uint public deadlineCampaign;

    uint public winningFunds;

    address public winningAddress;

    struct Project {

        address addr;

        string name;

        string url;

        uint funds;
        bool initialized;

    }

    mapping (address => Project) projects;

    address[] public projectAddresses;

    uint public numberOfProjects;

    event ProjectSubmitted(address addr, string name, string url, bool
initialized);

    event ProjectSupported(address addr, uint amount);

    event PayedOutTo(address addr, uint winningFunds);


    function WinnerTakesAll(uint _minimumEntryFee, uint
_durationProjects, uint _durationCampaign) public {

        if (_durationCampaign <= _durationProjects) {

            return;

        }

        minimumEntryFee = _minimumEntryFee;

        deadlineProjects = now + _durationProjects* 1 seconds;

        deadlineCampaign = now + _durationCampaign * 1 seconds;

        winningAddress = msg.sender;

        winningFunds = 0;

    }

    function submitProject(string name, string url) payable public
returns (bool success) {

        if (msg.value < minimumEntryFee) {

            return;     //Need minimumEntryFee

        }

        if (now > deadlineProjects) {

            return;     //Project deadline passed

        }

        if (!projects[msg.sender].initialized) {

            projects[msg.sender] = Project(msg.sender, name, url, 0,
true);
```

```
            projectAddresses.push(msg.sender);

            numberOfProjects = projectAddresses.length;

            ProjectSubmitted(msg.sender, name, url,
projects[msg.sender].initialized);

            return true;

        }

        return false;

    }

    function supportProject(address addr) payable public returns (bool
success) {

        if (msg.value <= 0) {

            return;     //Value must be positive

        }

        if (now > deadlineCampaign || now <= deadlineProjects) {

            return;     //Passed project deadline or passed campaign
deadline

        }

        if (!projects[addr].initialized) {

            return;         //Return if not initialized

        }

        projects[addr].funds += msg.value;

        if (projects[addr].funds > winningFunds) {

            winningAddress = addr;

            winningFunds = projects[addr].funds;

        }

        ProjectSupported(addr, msg.value);

        return true;

    }

    function getProjectInfo(address addr) public constant returns
(string name, string url, uint funds) {

        var project = projects[addr];
        if (!project.initialized) {
           return;      //Return if project has not initialized
        }
        return (project.name, project.url, project.funds);
    }
    function finish() public {
        if (now >= deadlineCampaign) {
            PayedOutTo(winningAddress, winningFunds);
            selfdestruct(winningAddress);
        }
    }
}
```