

项目报告

数据科学与计算机学院

孙正伦

15332013

目录

1、 项目源代码地址.....	2
2、 选题背景及依据.....	2
3、 使用说明.....	2
4、 代码的简要说明.....	8
5、 测试.....	9
6、 参考文献.....	11

1、项目源代码地址

<https://github.com/show-me-code/blockchain-game-Connect-Four.git>

7 commits

1 branch

0 releases

1 contributor

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

show-me-code fixed bug

Latest commit 596ed75 18 hours ago

app

fixed bug

18 hours ago

build/contracts

final part see read me

18 hours ago

contracts

final part see read me

18 hours ago

migrations

this is the first commit but not complet, it will connctct to metamask...

3 days ago

node_modules

final part see read me

18 hours ago

test

final part see read me

18 hours ago

.babelrc

this is the first commit but not complet, it will connctct to metamask...

3 days ago

.eslinignore

this is the first commit but not complet, it will connctct to metamask...

3 days ago

.eslintrc

this is the first commit but not complet, it will connctct to metamask...

3 days ago

.vscode-janus-debug

this is the first commit but not complet, it will connctct to metamask...

3 days ago

LICENSE

final part see read me

18 hours ago

README.md

Update README.md

19 hours ago

box-img-lg.png

this is the first commit but not complet, it will connctct to metamask...

3 days ago

box-img-sm.png

this is the first commit but not complet, it will connctct to metamask...

3 days ago

package-lock.json

final part see read me

18 hours ago

package.json

final part see read me

18 hours ago

truffle-config.js

this is the first commit but not complet, it will connctct to metamask...

3 days ago

webpack.config.js

final part see read me

18 hours ago

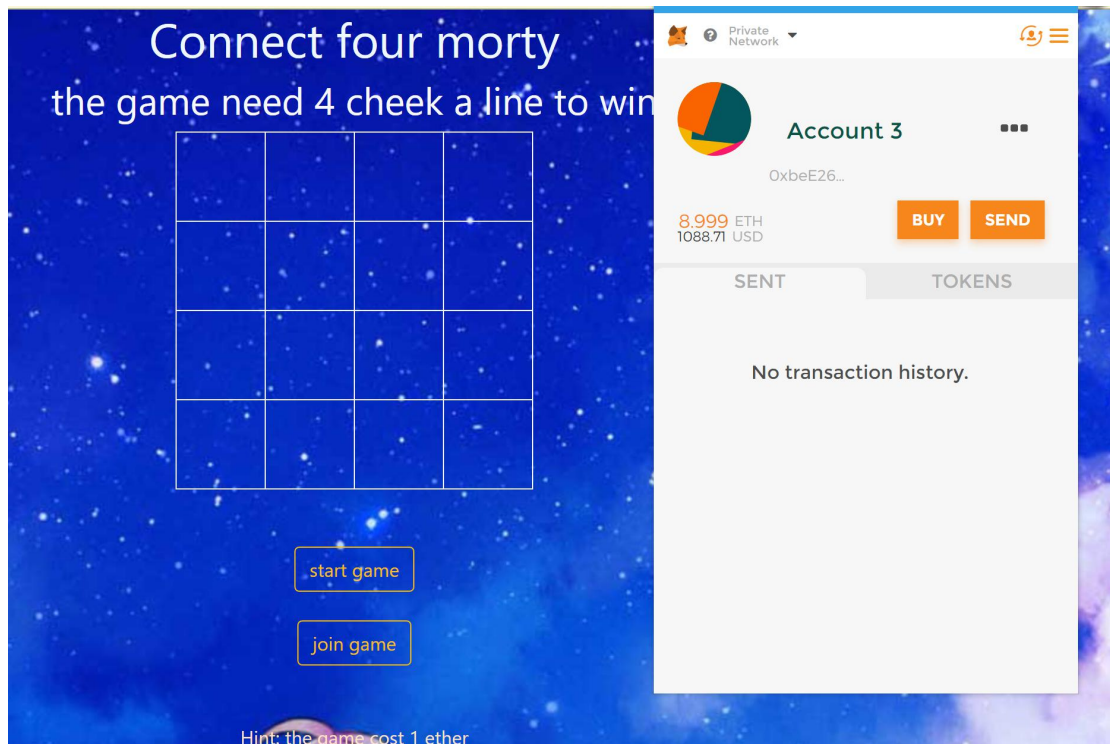
README.md

2、选题背景及依据

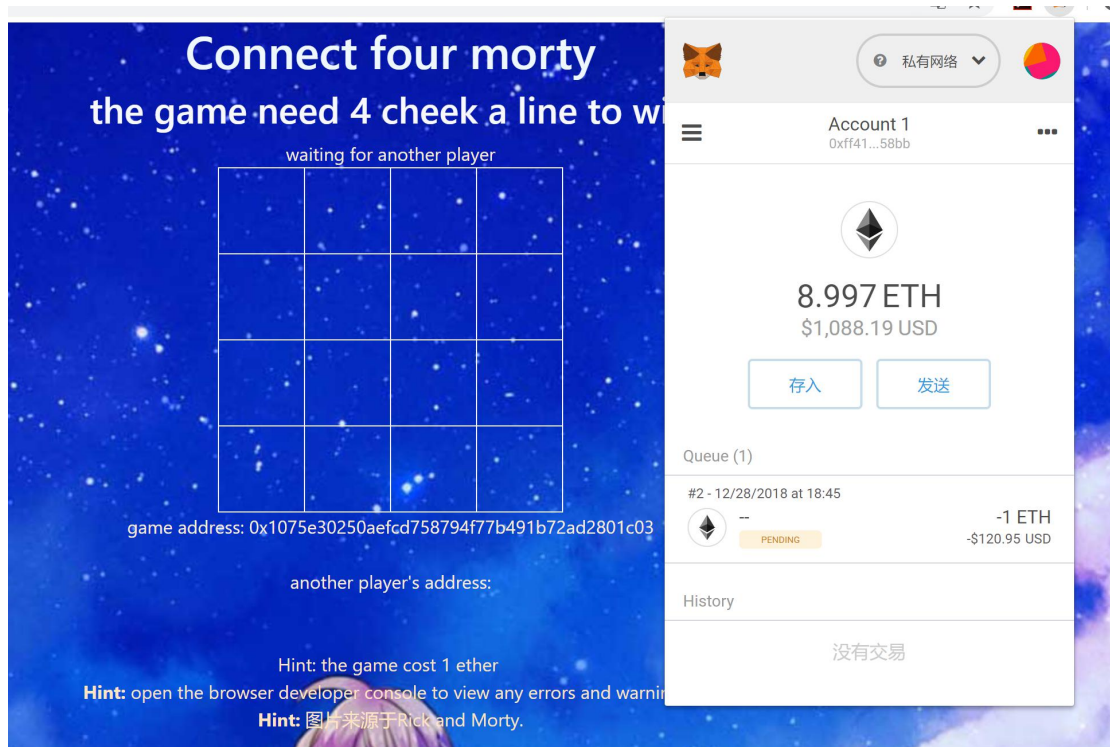
伴随着当前区块链的发展，各种 **DApp** 层出不穷，游戏更是其中热门的种类，因此我开发了一款“四子连珠”游戏，即在棋盘中将自己的棋连成四个即可获得胜利。本项目使用 **truffle** 框架进行开发，该框架开发简单，但是在调试时略有困难，关键在于环境的配置比较复杂。在开发完毕后，使用 **npm run dev** 命令监听并且配合浏览器访问本地地址 **localhost:8080** 即可进行使用或者测试。在账号方面使用了 **metamask** 在私有网络进行测试和运行。

3、使用说明

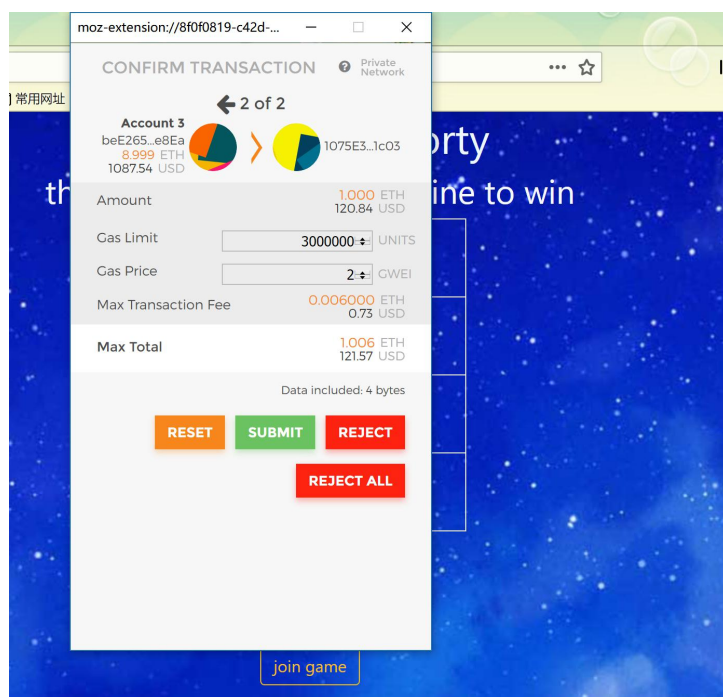
本次使用火狐和谷歌浏览器来作为游戏的两个运行端，这是游戏的主要界面，点击 **start game**，两位玩家均需要投入 **1** 以太币以进行游戏。此时开始计时。



游戏的发起方支付 1 以太币，得到游戏地址，将这一游戏地址复制给对面的玩家。采取复制的原因是当多个游戏同时存在的时候，玩家会不清楚自己加入的房间，因此不采用自动显示。



另外一名玩家点击 join game 按钮并且在输入框中输入游戏地址，支付 1 以太币。



此时游戏开始。开始游戏的一方会在表格上放显示‘it’s your turn’，其他一方则会显示等待，这些文字的显示带有动画。同时表格下方会显示游戏地址和玩家地址，由于交易的打包需要时间，点击表格后可能不会立刻显示棋子，同时，下棋也要支付相应的交易费用。在不是自己的回合下棋交易会失败，因此只有在自己的回合才能下棋。




此时，当前玩家下棋了，他的界面和对手的界面显示如下：

Connect four morty

the game need 4 cheek a line to win

waiting for another player

game address: [0x1075e30250aefcd758794f77b491b72ad2801c03](#)

another player's address: [0xff417244ea481d294ee45ba6fda0c839fa4058bb](#)

Hint: the game cost 1 ether


Hint: open the browser developer console to view any errors and warnings.

Hint: 图片来源于Rick and Morty.

Connect four morty

the game need 4 cheek a line to win

It's your turn

game address: [0x1075e30250aefcd758794f77b491b72ad2801c03](#)

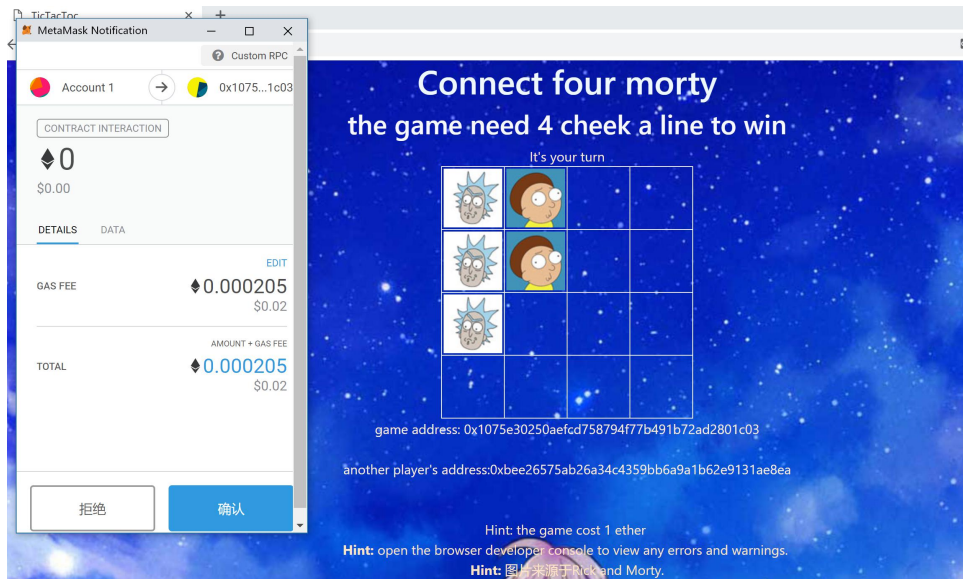
another player's address: [0xbec26575ab26a34c4359bb6a9a1b62e9131ae8ea](#)

Hint: the game cost 1 ether

Hint: open the browser developer console to view any errors and warnings.

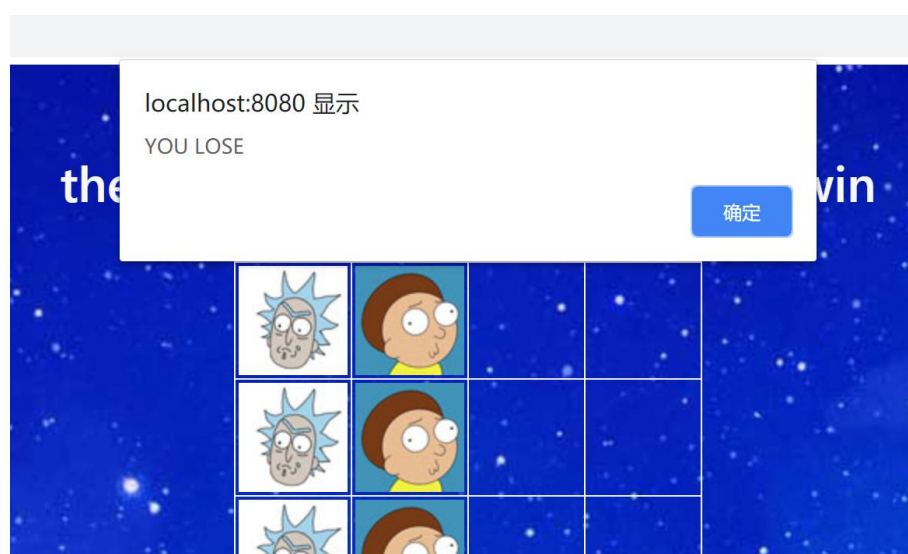
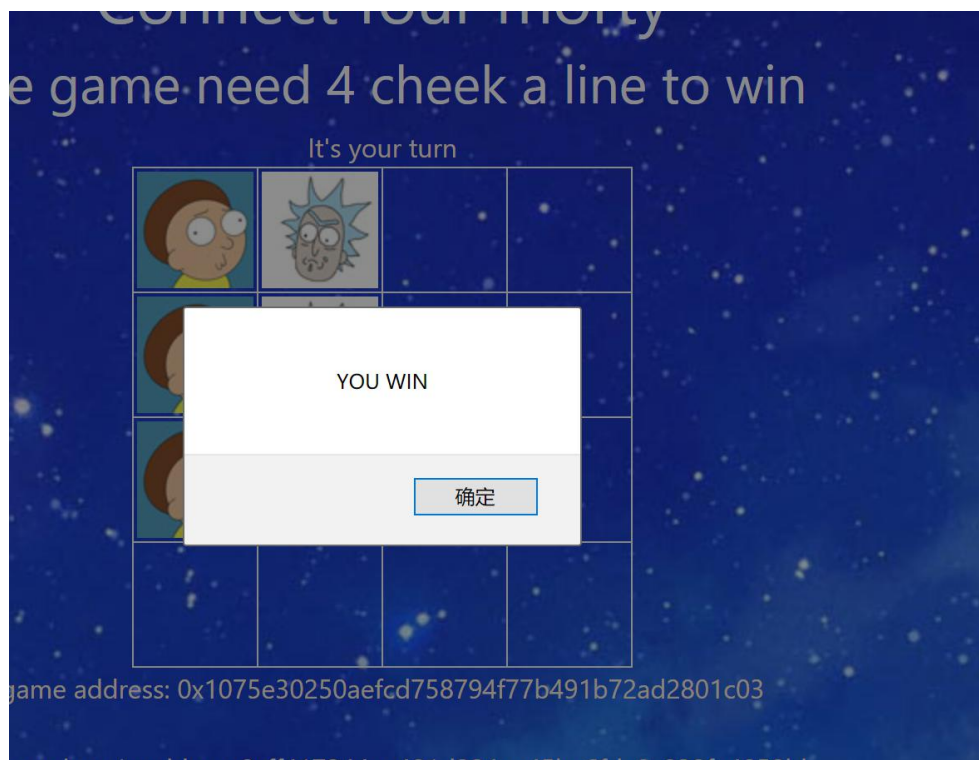
Hint: 图片来源于Rick and Morty.

双方玩家点击表格下棋，直到一方取得胜利或者平局。如图，下棋需要支付一定的费用，并且表格上方的提示也会改变。

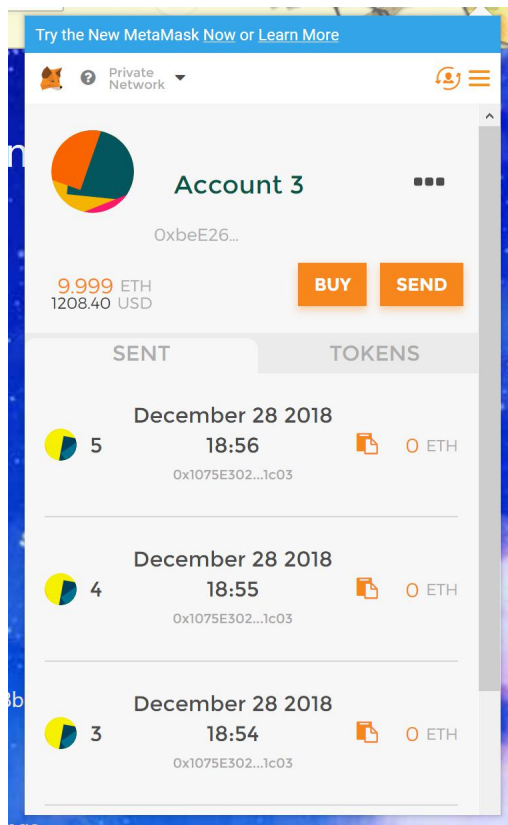


在点击最左下方格子后，一名玩家胜利，弹出提示“you win”，另外一名玩家弹出“you lose”

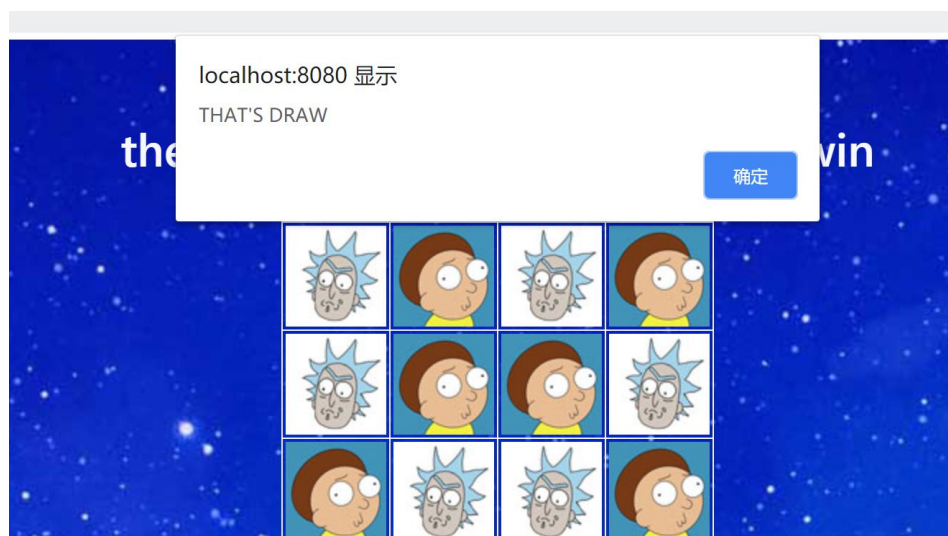




此时观察赢家的账号，发现账号得到了 2 以太币（即自己投入的 1 以太币和对手投入的 1 以太币）。



在平局情况下则会显示 “that’s draw” ,此时以太币会被退回。



4、代码的简要说明

智能合约中的函数及其作用：

构造函数：加入第一名玩家。

`joinGame()`：首先检查地址并且防止重复加入，第二名玩家加入并且令其进行支付，先下棋的玩家通过 `block.number` 的奇偶来判断。此时引入时间来提示谁是下一名玩家。

`setChess()`：通过检查棋盘，当前玩家和时间限制后，将玩家选择的 `board[4][4]` 中某一个

位置设为玩家的地址。同时，在内部会对行，列，对角线，反向对角线进行检查，检查是否有连成一条线的，有则判定胜利，判定的方法是通过遍历。如果有平局，则调用 `setDraw()` 函数。

`getBoard()`: 返回棋盘。

`setDraw()`: 平局，将智能合约中的以太坊平分还给原来的两位玩家。平局的情况有：超过时限，棋盘下满了还未分出胜负。

`setWinner()`: 设置赢家，此时将游戏设置为未激活状态，并且将智能合约中的余额送入赢家的账户中。

index.js 中的函数及其作用：

`newGame()`: 创建新游戏，此时显示游戏地址，并将开始游戏的按钮和加入游戏的按钮隐藏防止多次加入，在这一函数中会监听四个事件：加入、提示下一名玩家、赢家、平局。由于不是某一方胜利就是平局，因此平局和胜利使用同一个函数进行 `watch`。

`joinGame()`: 加入游戏，在加入之前需要输入所需要的游戏地址，输入后会监听三个事件：提示下一名玩家、赢家、平局。此时会显示游戏的地址，其他玩家的地址同时隐藏按钮。

`setChess()`: 点击事件传入，调用实例中的 `setChess` 函数，在将点击的位置设为点击者的地址，同时还要在 `setChess` 中指明账号。

`print()`: 调用实例的 `getBoard()`，得到当前的棋盘，并且遍历棋盘，如果棋盘中的地址是当前的地址，那么就将格子中设置为 `morty` 图片，对手则会设置为 `rick` 图片。

`GameOverWinner()`: 赢家和平局事件，即为 `GameOverWinnerEvent`, `GameOverDrawEvent` 所监听的函数，如果事件是 `GameOverWinner`，且赢家的地址是当前玩家的地址，那么则显示 “you win”，否则则显示 “you lose”，若事件名不是 `GameOverWinner`，则此时为平局的情况，显示 “that’s draw”。同时由于游戏胜利或平局意味着游戏的结束，此时会停止其他所有时间的监听。

`playerRemind()`: 提示当前玩家，并且在这一步中会检测棋盘，如果当前的玩家应该下棋且要下的格子为空，那么久可以在其中设置棋子，调用先前所说的 `setChess` 函数。同时，在当前可以下棋的玩家处显示 “it’s your turn”，另外一位玩家会显示 “waiting for another player”。

5、测试

本次的测试采用 `truffle` 内部的测试，通过编写 `test` 文件来进行测试，本次测试的内容

是双方下棋直到一方胜利。首先设置两方玩家加入，之后下棋直到一方胜利。

```
import { AssertionError } from "assert";

var TicTacToc = artifacts.require("TicTacToc");
//测试游戏胜利
//var moneyUsed = 1000000000000000000;
var moneyUsed = web3.utils.toWei('1', 'ether');
contract("TicTacToc", function(accounts){
    it("Join Game and test winner", function() {
        var Instance;
        var player1 = accounts[0];
        var player2 = accounts[1];
        return TicTacToc.new({from : accounts[0], value : moneyUsed}).then(instance => {
            Instance = instance;
            //加入第二个玩家
            return instance.joinGame({from : player2, value : moneyUsed})
        }).then(board => {
            return Instance.setChess(0,0,{from : board.logs[1].args.player})
        }).then(board =>{
            return Instance.setChess(1,1,{from : board.logs[0].args.player})
        }).then(board =>{
            return Instance.setChess(0,1,{from : board.logs[0].args.player})
        }).then(board =>{
            return Instance.setChess(1,0,{from : board.logs[0].args.player})
        }).then(board =>{
            return Instance.setChess(0,2,{from : board.logs[0].args.player})
        }).then(board =>{
            return Instance.setChess(1,2,{from : board.logs[0].args.player})
        }).then(board =>{
            return Instance.setChess(0,3,{from : board.logs[0].args.player})
        }).then(board =>{
            console.log(board)
        }).catch(err => {
            console.log(err)
        })
    })
})
```

测试结果如下：

首先使用 truffle develop 命令。

```
PS C:\Users\UNSC> cd C:\Users\UNSC\truffle
PS C:\Users\UNSC\truffle> truffle develop
```

在进入控制台之后使用命令 test。

```
truffle(develop)> test
Using network 'develop'.
```

在测试之后输入日志并且发现此时已经成功。

