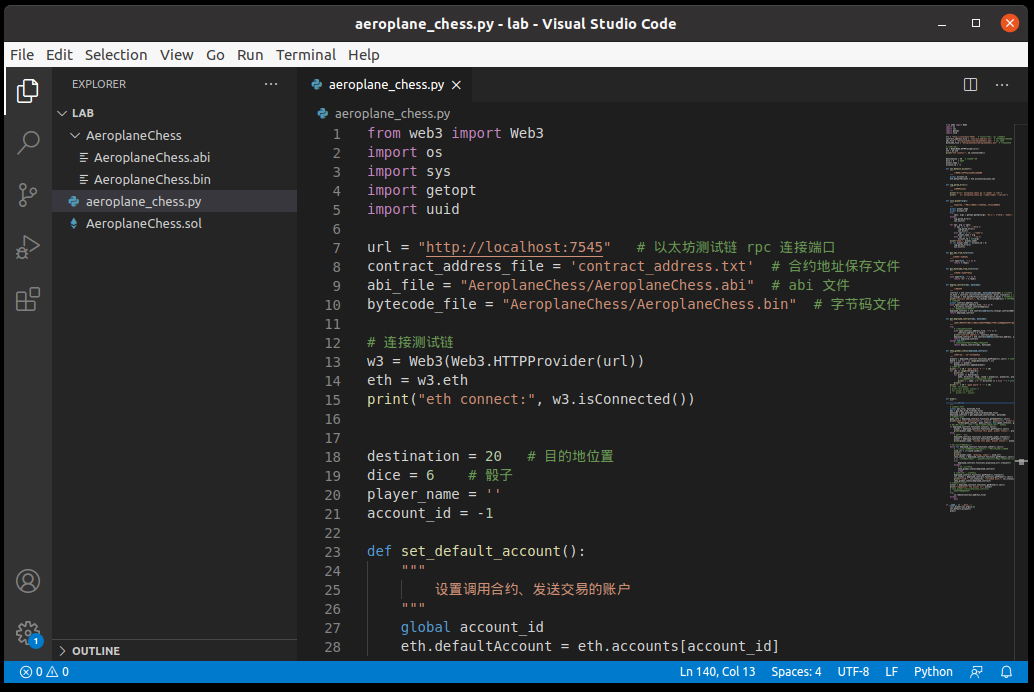
实验五 飞行棋游戏：以太坊多方参与合约

【实验介绍】

本次实验将使用智能合约实现一个有趣的多方参与游戏——飞行棋，同时使用Python远程rpc调用该合约。

本次实验代码框架为：



【实验要求】

1. 熟悉Python通过以太坊rpc接口部署并调用合约的过程
2. 掌握solidity结构体、数组、mapping等数据结构，require函数及基本代码的编写

【实验过程】

1. 安装相关依赖包
2. 安装pip3

|  |
| --- |
| **sudo** apt install python3-pip **-**y |

1. 使用pip3安装web3 python包

|  |
| --- |
| pip3 install web3 |

1. 安装ipython3

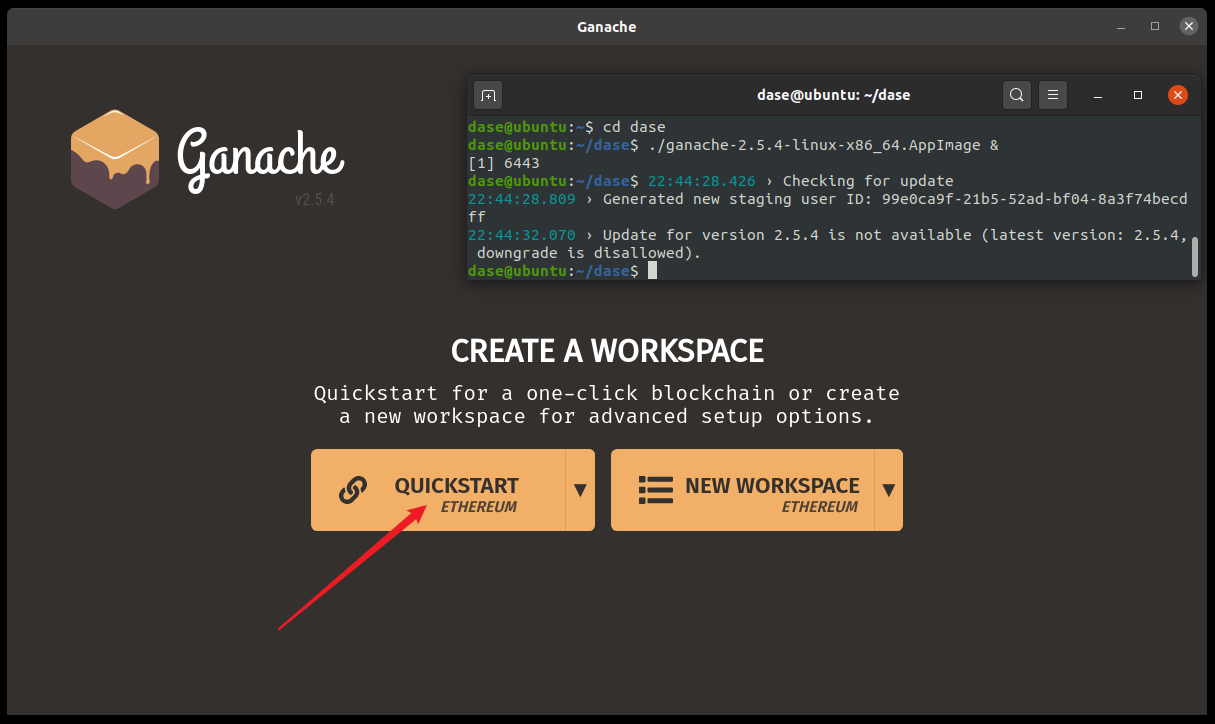
|  |
| --- |
| **sudo** apt install ipython3 -y |

1. 使用ganache开启以太坊测试链

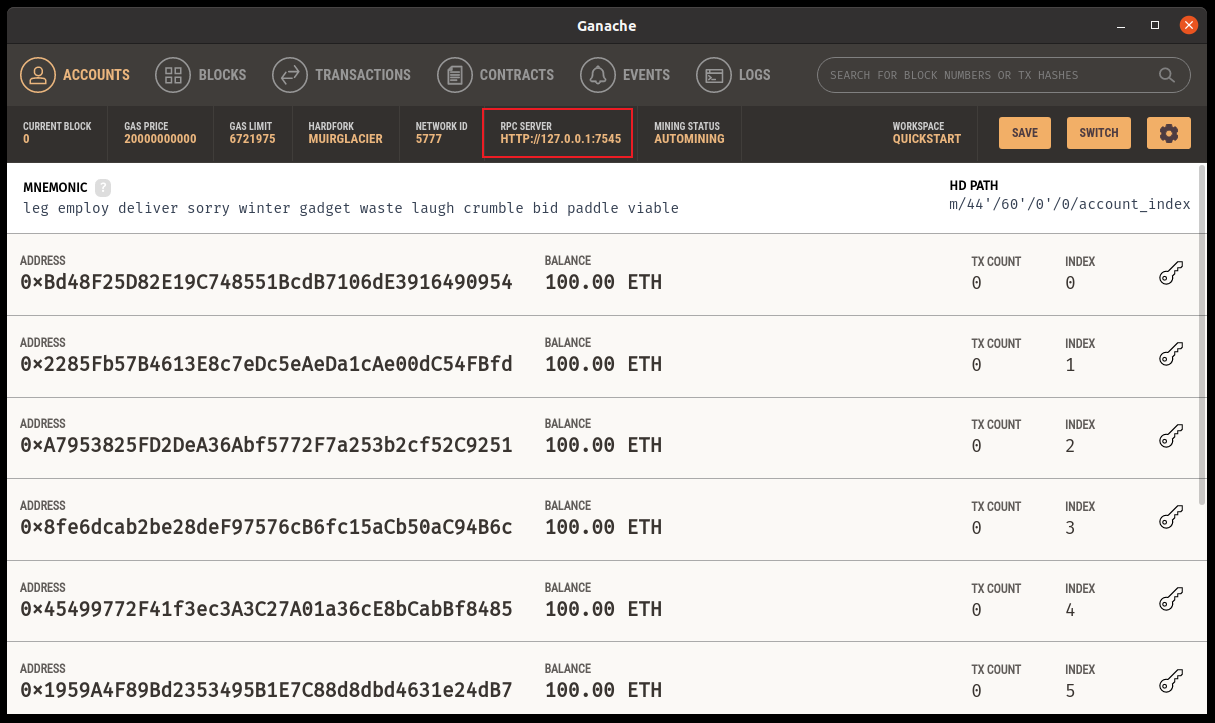
ganache是一个以太坊可视化工具，可以快速启动以太坊测试链，工具包已下载在dase目录中，在终端输入以下命令启动gannache：

|  |
| --- |
| **cd** dase  **./**ganache-2.5.4-linux-x86\_64.AppImage **&** |

点击QUICKSTART快速启动一个以太坊测试链。

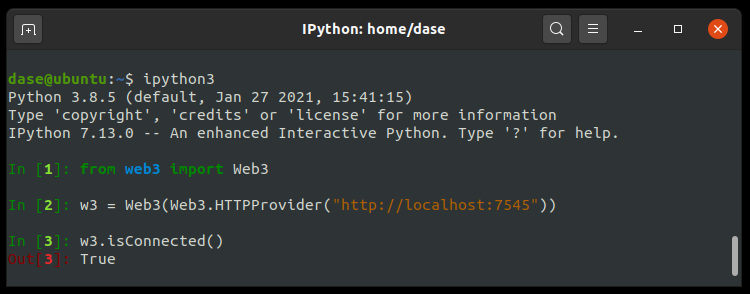


如下图所示，创建了一个以太坊测试网络，默认创建十个拥有100以太币的账户，远程rpc调用端口默认为7545，ganache工具还可以查看区块和交易信息。



1. Python连接以太坊测试链

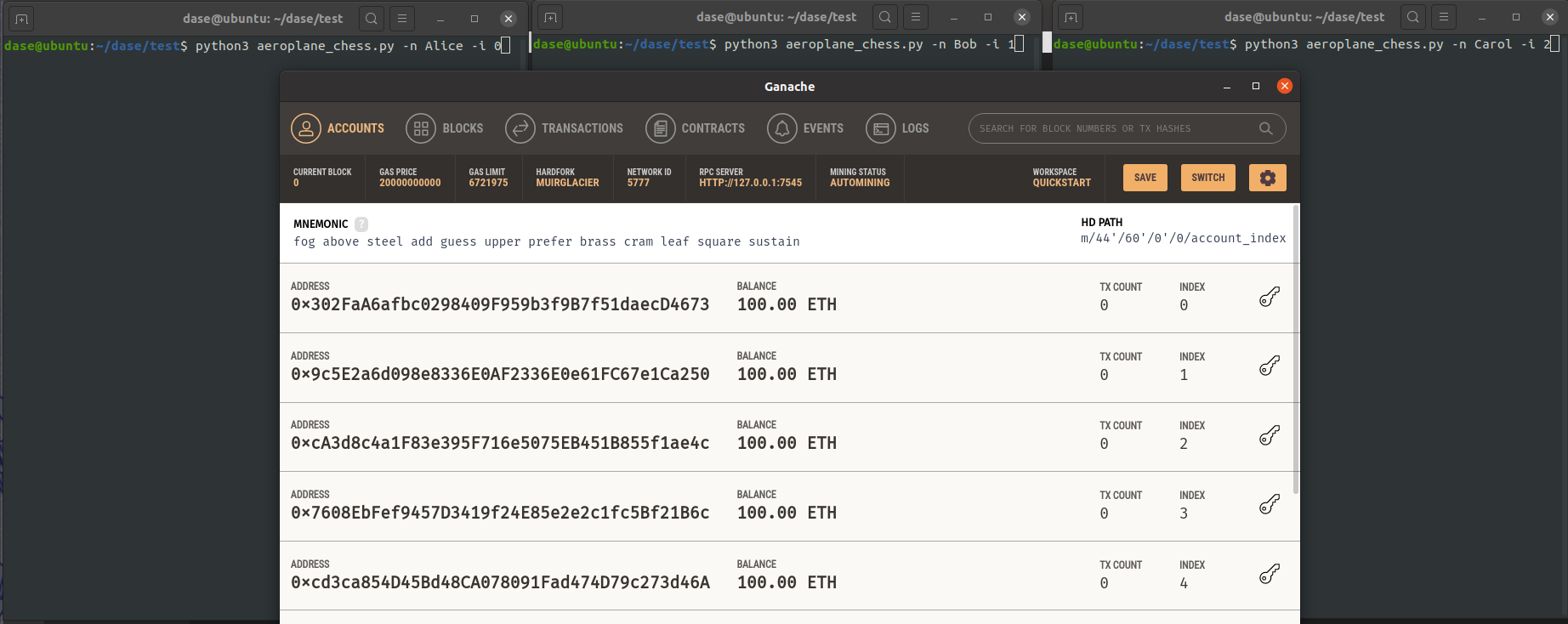
打开终端，输入ipython3进入Python交互环境，输入以下代码测试连接情况：



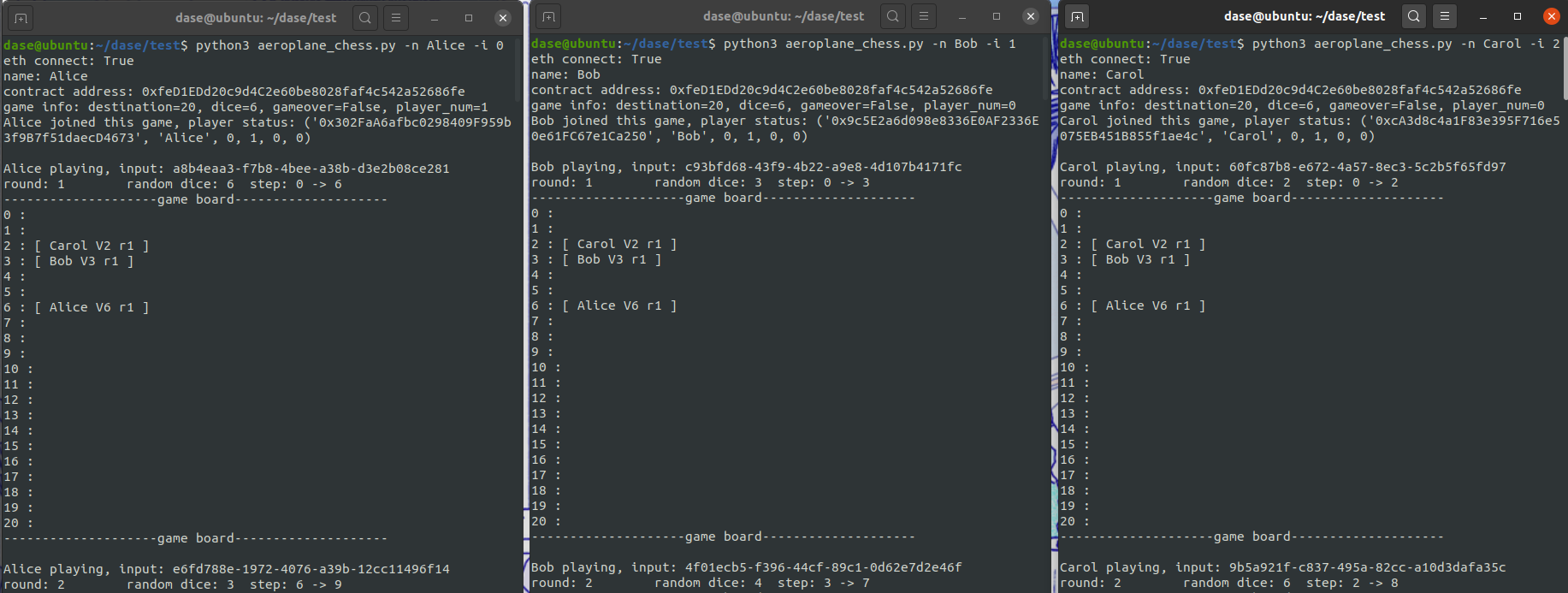
1. 飞行棋游戏Python代码设计介绍

考虑到本实验的重点在于智能合约的设计，我们实现了相应的Python代码，实验会提供相应的aeroplane\_chess.py，以太坊的多方账户参与这个简化版飞行棋游戏，最终会实现如下图的功能：

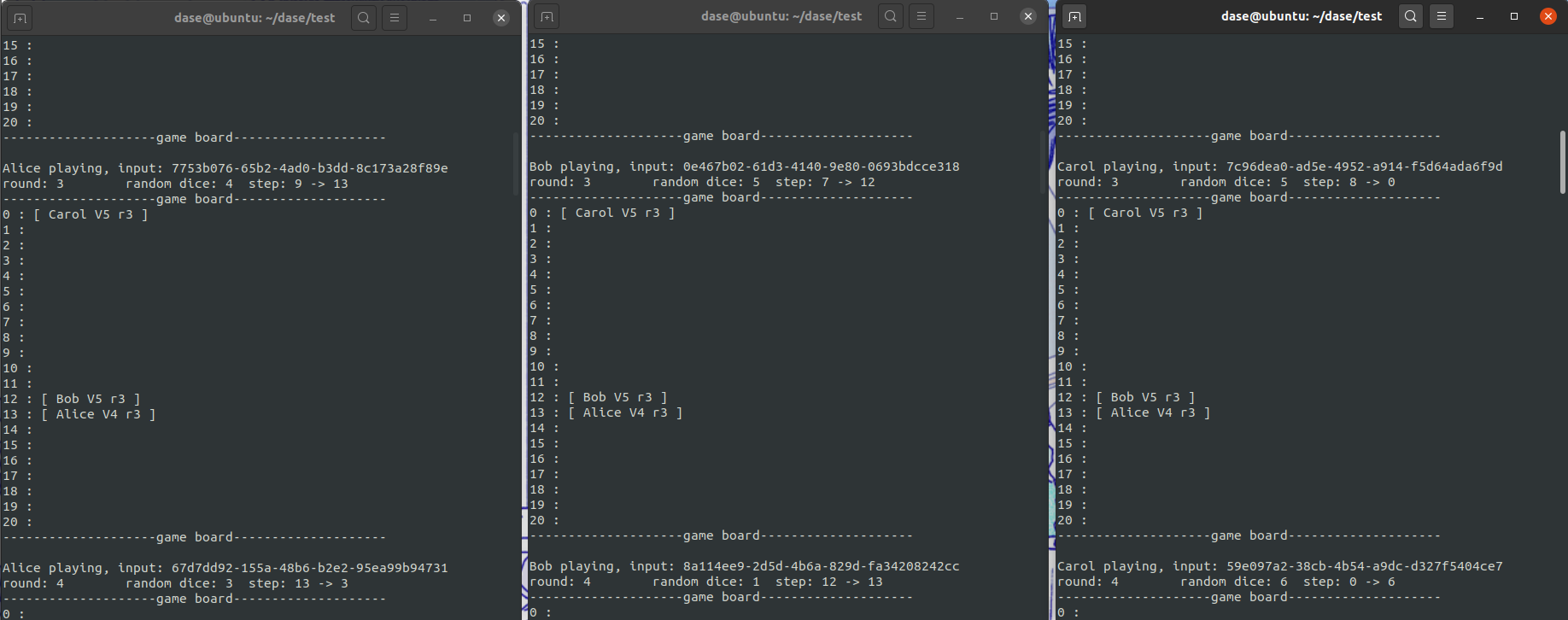
首先，我们查看以下测试链的信息，我们将使用账户0、1、2作为参与者，现在他们的以太币都是100eth：



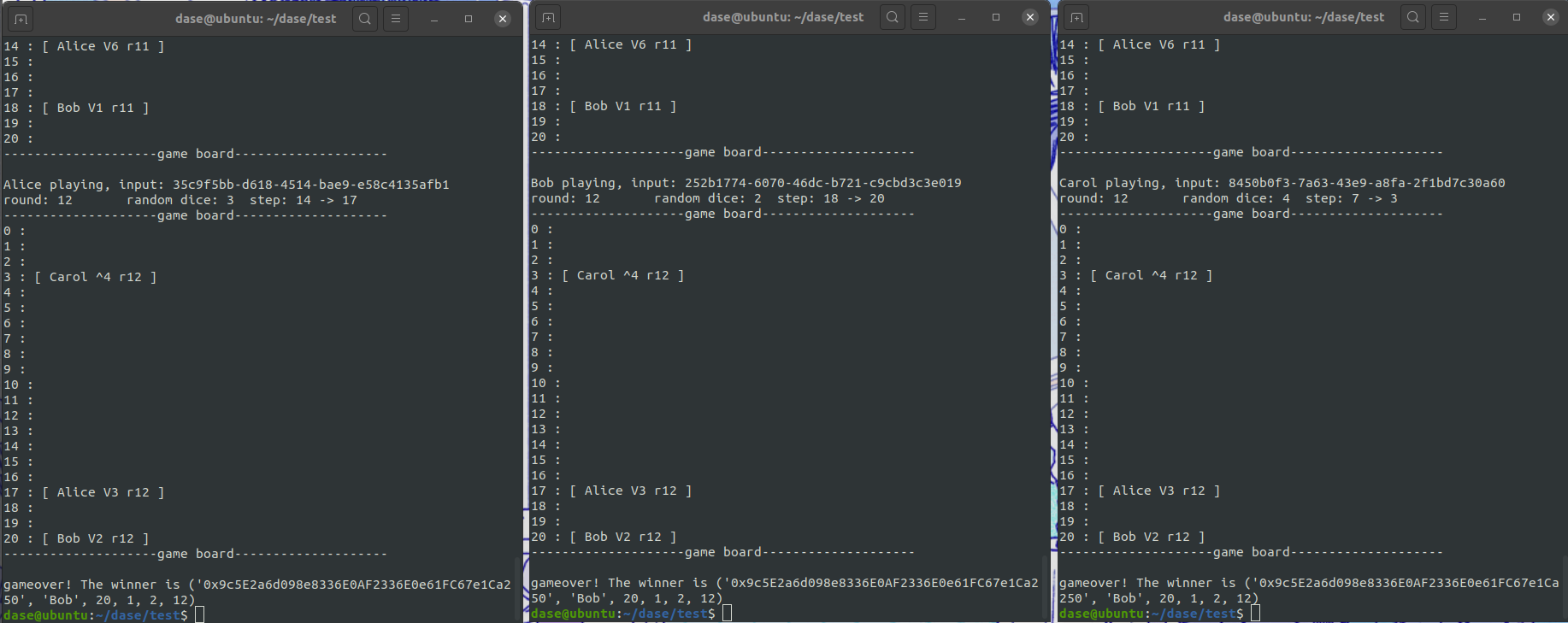
开始游戏如下图所示，Alice、Bob、Carol三位玩家参与游戏，他们的飞机开始处于0位置，然后他们输入不同且随机的字符串，在合约中会被处理为随机骰子的值，开始向终点进发，下图中名称右方的箭头为其方向（v或^）,数字为当前骰子的步数（已走完），r+数字为当前用户游玩的轮次（因为参与时间不一致或进程运行速度轮次会不一样）。



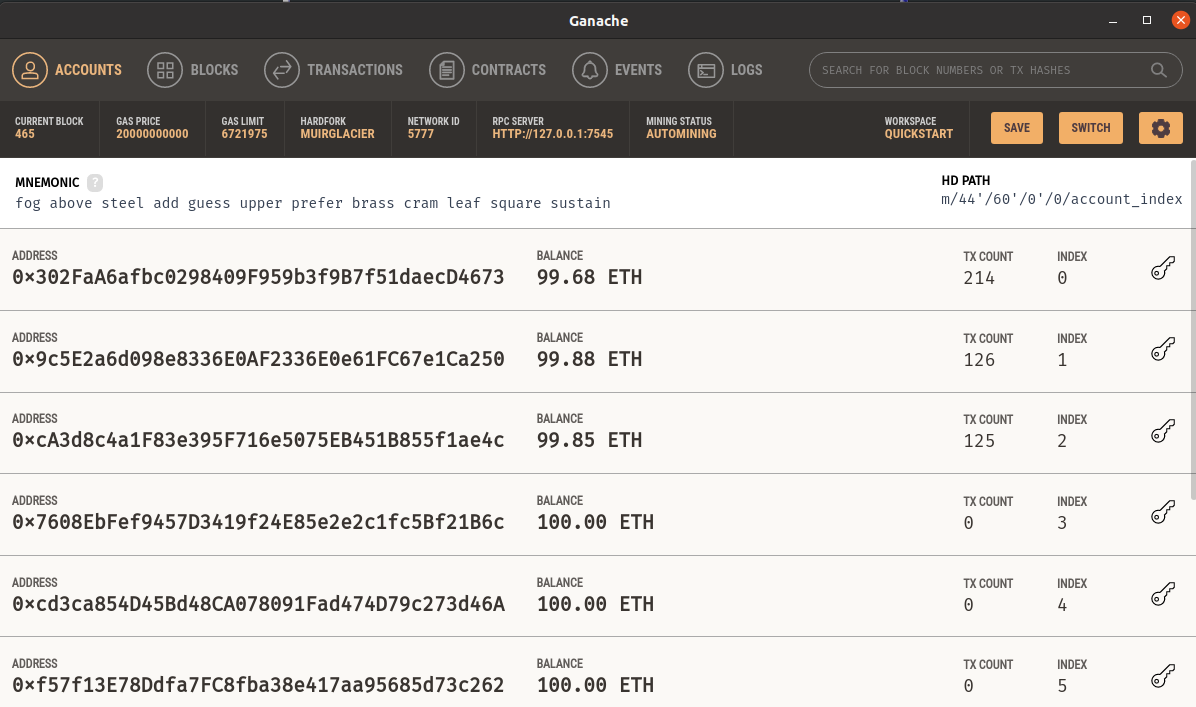
下图右方玩家Carol回到0是因为被Alice的飞机撞回到起点，因为编码原因暂时无法提供更细节的反馈。



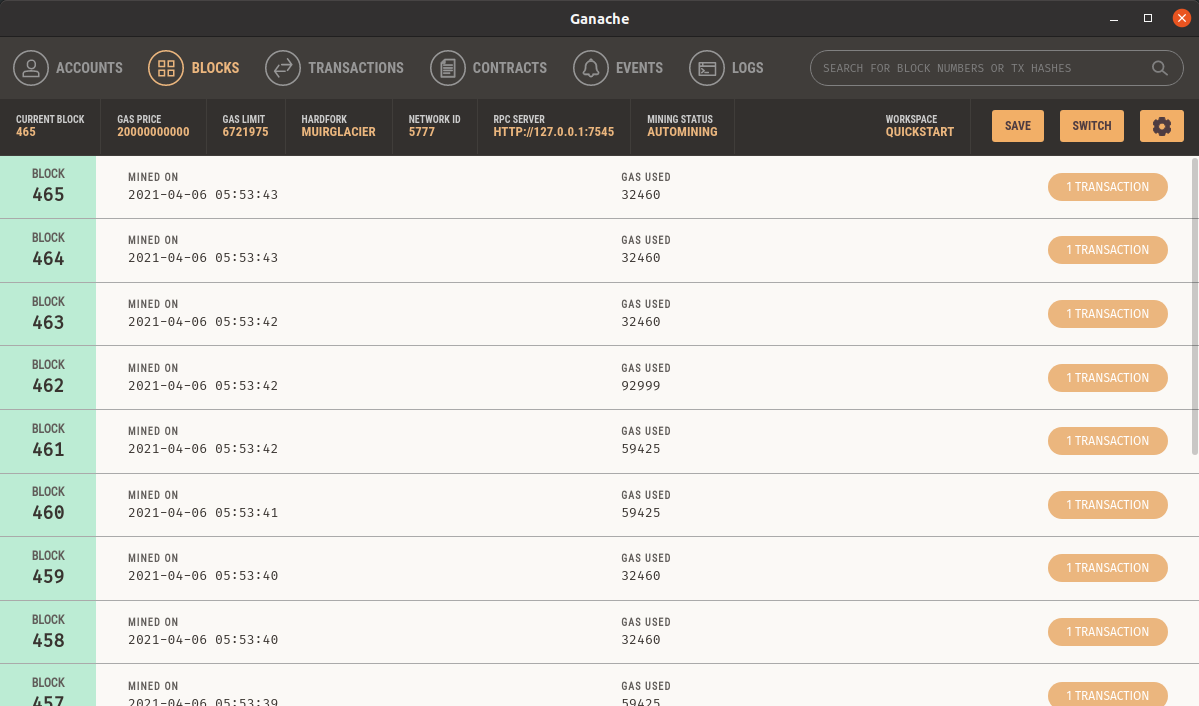
最终，Bob的飞机飞到了终点，赢得了本次游戏。



此时我们回看ganache可视化工具，发现三个账户都消耗了一定量的以太币发送交易执行合约代码。



以太坊也产生了一定量的区块去打包这些交易：



下面介绍下相应的功能函数。

1. 导入包及相应配置

目的地和骰子可以自行设定，但必须都是正数。注意abi文件和bytecode文件的位置。



1. 通过命令行获取游戏玩家及相应id

id对应于区块链账户数组的索引，即eth.accounts[id]。

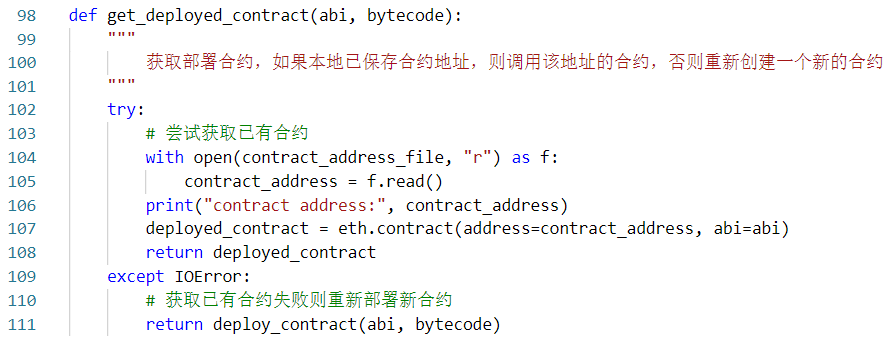


1. 从文件中读取abi和bytecode并部署合约

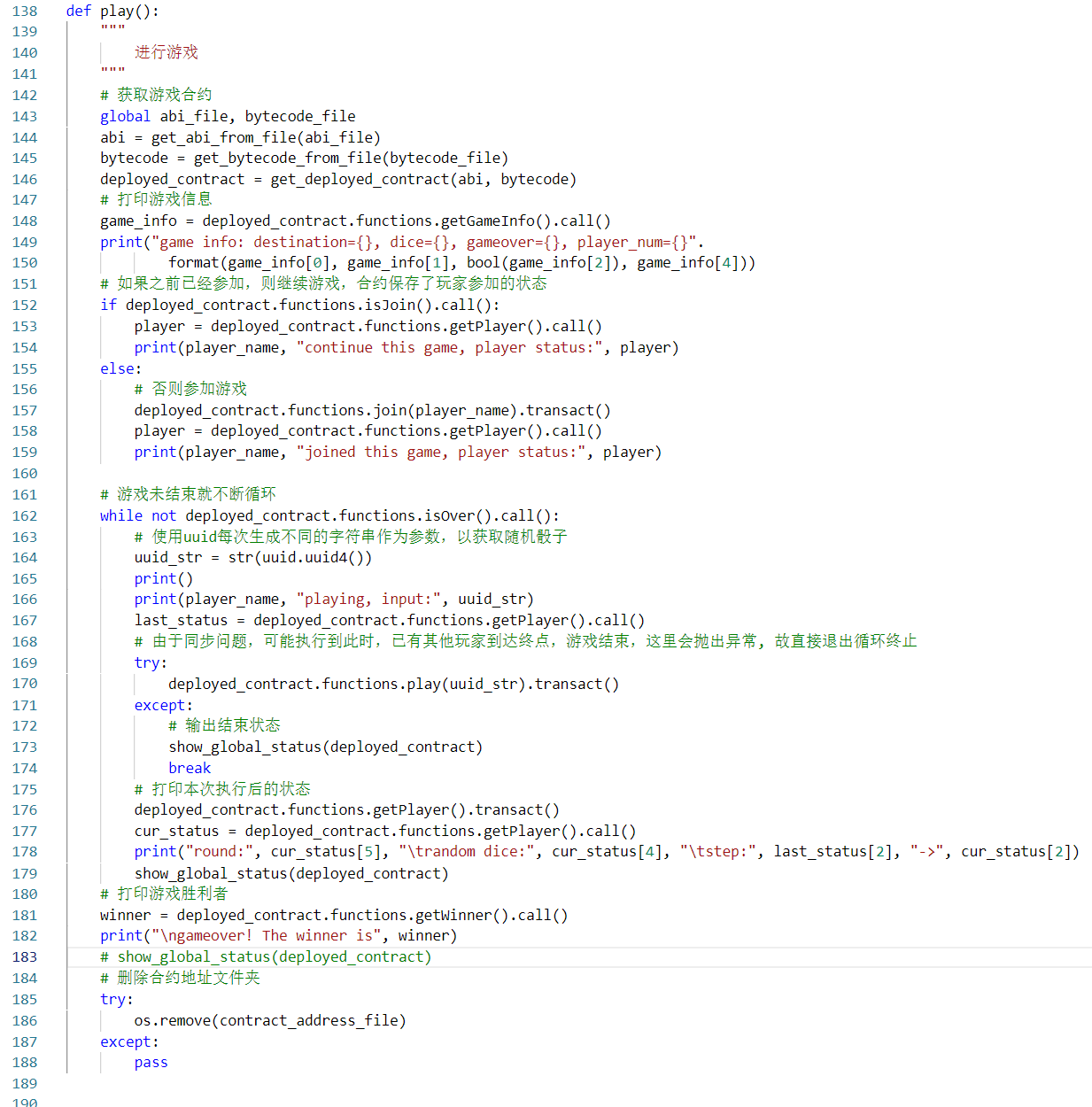
本次实验编写完合约，使用solc编译后，配置相关文件路径，即可通过Python自动读取文件内容并部署，减少复制粘贴的步骤。



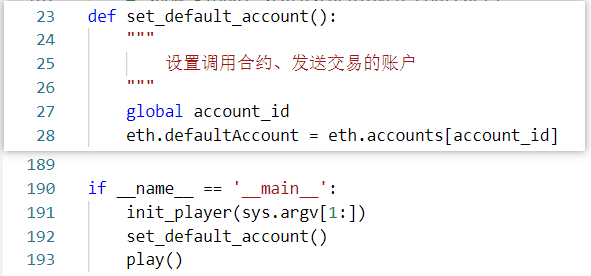
1. 保存合约地址，如果文件及地址有效，就读取旧有合约，继续仍在进行的游戏



1. play进行游戏函数，该游戏可中断后继续进行，详细信息请看注释：

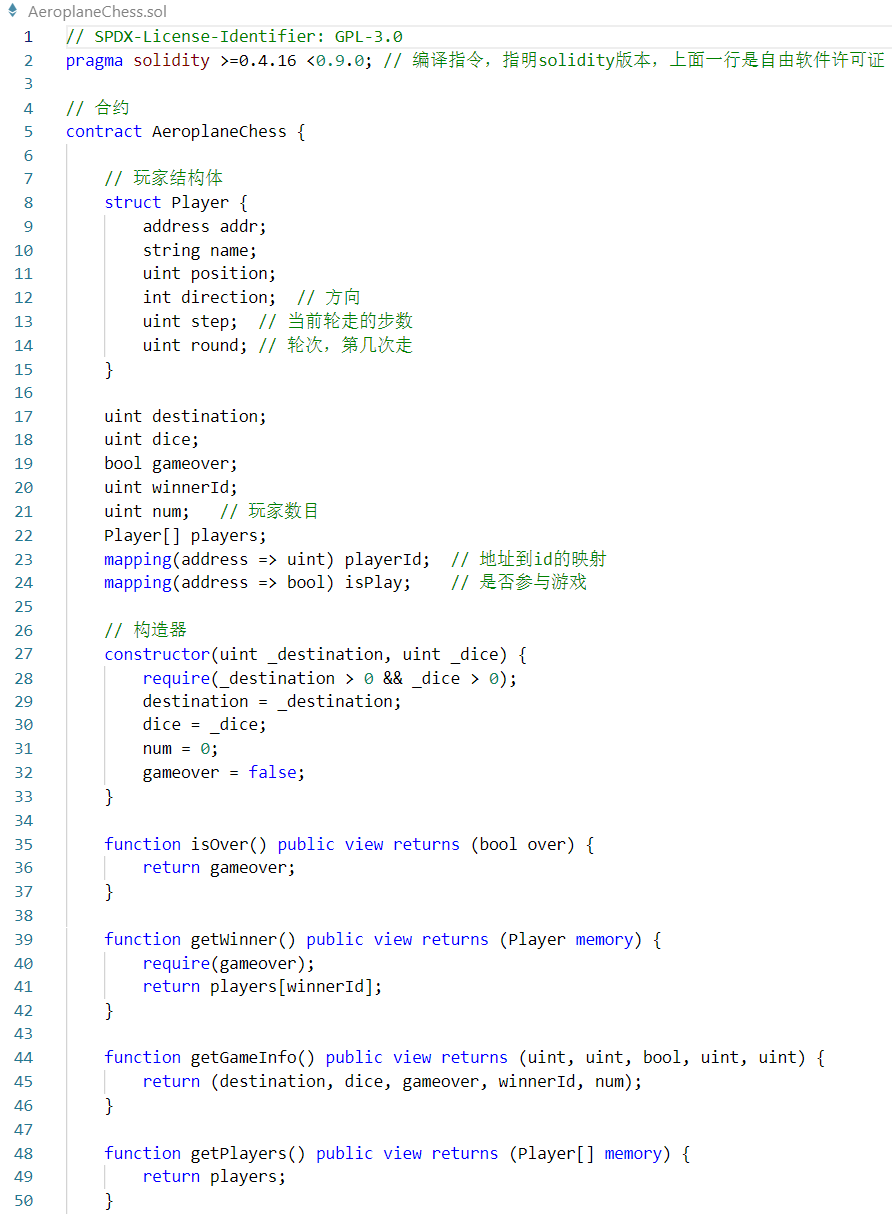


1. 程序入口，先初始化玩家名称和账户id，然后设置默认账户，最后参与游戏：



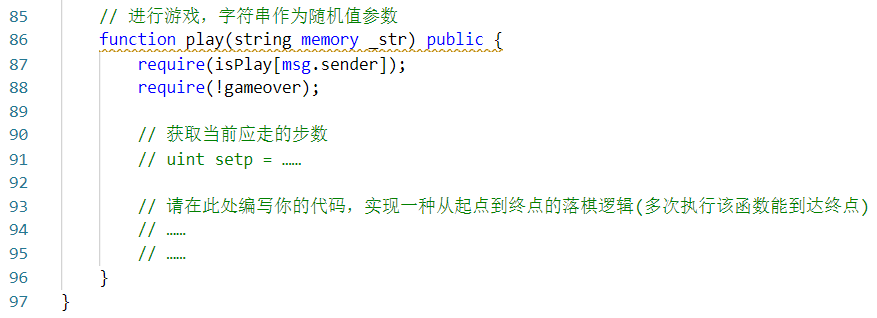
1. 飞行棋游戏智能合约设计

根据Python代码调用合约的函数接口模块，我们可以得出合约的几个接口设计，初步的智能合约设计如下图所示：



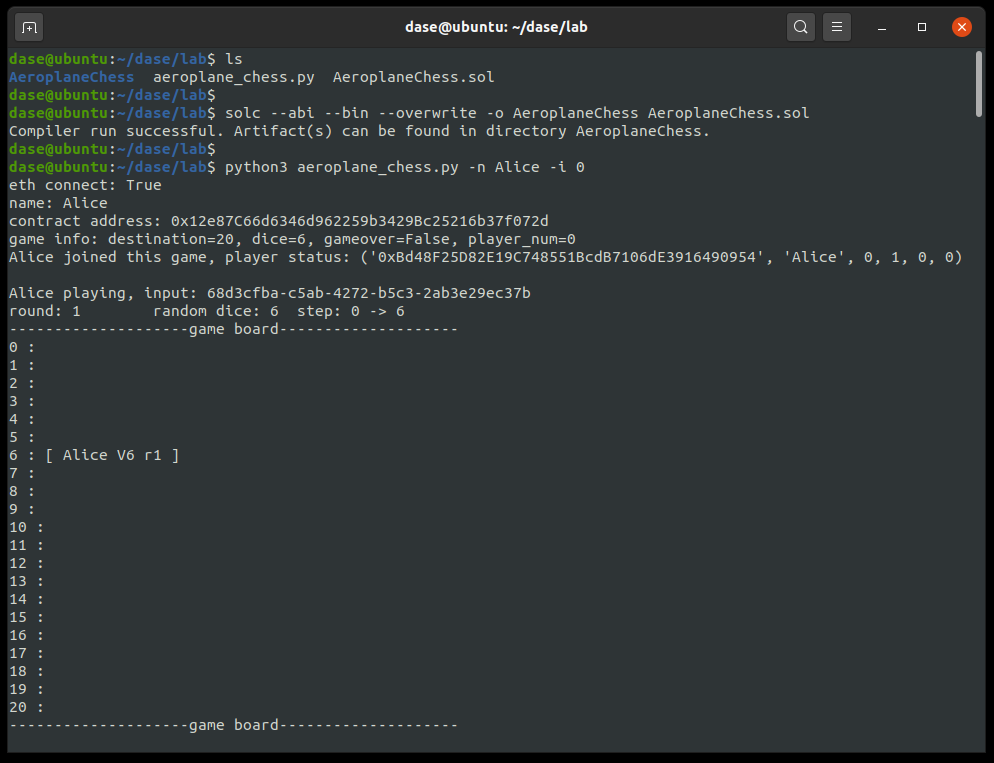


我们预留了一个play函数交由你去设计并实现，前面游戏的逻辑是：从起点出发，随机掷骰子走步数，过了终点就反向向原点进发，过了原点才能转回正向，如果多人参与游戏，还可将落点的其他玩家打回起点（如果他是反向可能还要谢谢你:)）。因为一种棋的走法是多样的，你可以选择一种简单的落子规则，如取终点长度的模运算落子，过了终点就回到起点重新进发，也可以取消将其他玩家打回起点的规则。需要注意的是，你的play函数最终要改变当前玩家的postion变量，同时也要更新step、round（如果你要变换方向的话，也别忘了direction），否则他将在原地不动，同时也须保证他能到达终点，这样游戏合约方可终止。

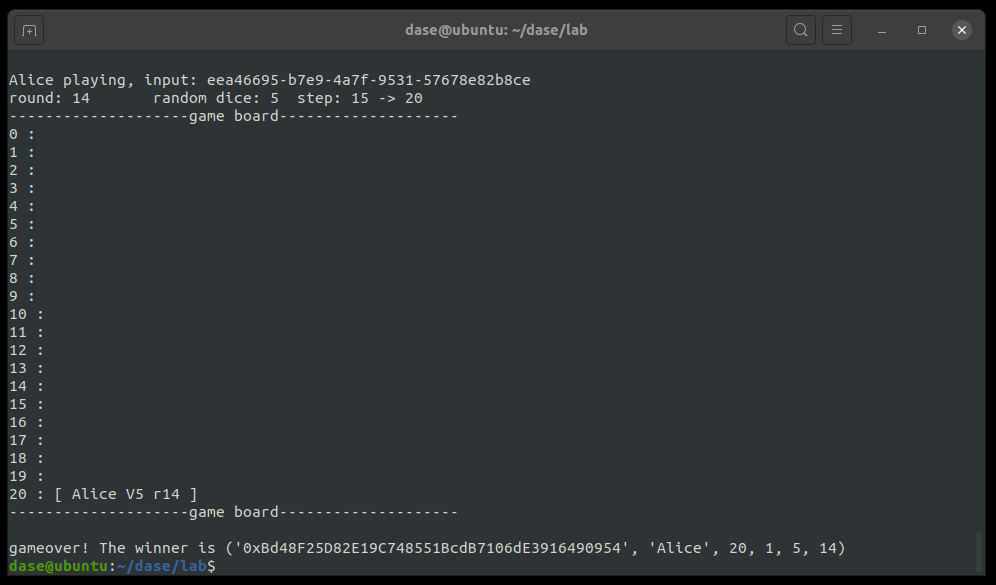


1. 游戏合约编译、部署及运行

如下图所示，写好合约后，使用solc进行编译，其中—overwrite是覆盖旧有的编译结果，便于修改后重新编译。然后就可以运行python代码，注意命令是python3，-n后面是玩家姓名（可自定），-i后面是以太坊账户eth.accounts数组的下标，所以不要超过gananche中账户的数目减一，同时相同的下标会被视作统一账户继续参赛，故开启多个终端时最好设定不同的姓名和下标值。（最后一行python3命令可执行在多个终端中让多个玩家参与游戏合约。）



14轮后Alice到达终点，游戏终止。



多用户参加的示例在前文已给出，这里就不再演示。