Tests of SportEth contract.

The specific numbers calculated in more detail in the corresponding Excel workbook. The following helps explain what those numbers mean.

# Test 1: margins and odds

Test of basic bet payoffs and margin accounting. The spreadsheet shows how various bets should affect the margin before and after results are processed in the settlement. Several separate bets are made on four different matches, for a total of 12 bets. Each match includes bets for both home and away teams, and so involves netting exposure for the bookies.

Document that the bets correctly apportion bookie margin to accommodate the size and odds of the bets, including netting offsetting bets.

The bettors should receive their initial bet, plus the payout implied by the odds on their bet. By looking at the resulting balance of the players after redeeming their winning bets we can see that they are attributed their payouts correctly. Specifically, the contract that passes the test will have to get the following correct:

1. Odds for match
2. result for match
3. fee to LP
4. fee to Oracle

Test of system solvency

The amount of eth going in should be attributed to the correct parties

Initial End

LP1 30000 29154

Bettor 2 10000 6950

Bettor 3 10000 13554

Oracle 0 34.22

Sum: 50000 50000

Here we assert the individual values, noting they match the above numbers. Note that the test program measures the Oracle balance in finney (1e15) while in the contract the smallest unit of denomination is tenths of a finney. Thus, we must anticipate rounding the digit when comparing to the Solidity contracts.

# Test 2: margin and bet test with different outcomes

This is just a repeat of Test 1 with different outcomes for the same matches and odds.

# Test 3: Odds Updates

Here we test updating odds by looking at the ultimate bettor redemptions. Initial ‘unadjusted’ odds for the favorite was originally 1.999, which is then updated to 1.800 (input as 999 and 800).

Acct1 balance should be 10k + 0.05\*(1.999\*1000) = 949

Acct2 balance should be 10k + 0.05\*(1.800\*1000) = 760

Accounts 1 and 2 started at 1000, and after redeeming their bets ended at 10949 and 10760 respectively, highlighting this account received the winnings accurately.

Also note that the oracleBalance is 8.995 finney, which is 0.05\*(999+800).

# Test 4: Redemptions and Withdrawals

This tests bettor redemptions and withdrawals. Users bet, redeem, and withdraw. Withdrawal checks to make sure gas fee accounts for the difference between the ether going out and the EOA’s change in ether.

Fails:

1. An attempt to redeem a winning bet from the wrong account
   * account3 trying to get account 2’s winning
2. An attempt to redeem a bet that was a loss
   * Account 3 trying to redeem his losing bet on match 0, team 0
3. An attempt by a bettor to redeem a bet he already redeemed
   * Account 2’s attempt to redeem a bet he had already redeemed

# Test 5: Big Bets

A “Big” bet can have different odds than a regular bet. We want to test and make sure this works.

Three regular bets are taken, generating a net liability of 1030. Given the ‘big bets’ are fully collateralized, this is the extent of the LP liability.

Bookie Locked = net on match 0 + net on match 2 = (1111-1000) + 607= 718

The regular odds on match 2 was 955 on team 0, 963 on team 1

A big bet on match 2 had account 2 bet 2000 with odds of 2.111, so when account 3 took the bet, that meant account 3 put down 1.111x2000 or 2222 on the other side (there’s no 4.5% LP vig here). Account 2 bet 2000 with odds of 2.111, so when account 3 took the bet, that meant account 3 put down 1.111x2000 or 2222 on the other side (there’s no 4.5% LP vig here).

Another big bet was on match 4, where account 2 put down 2002 on team 0 with odd of 1.955, which wins. Account 3 took this bet, so it should be processed.

A big bet was offered on match 3, but not taken, so it should not affect the final payoffs.

The final results correctly generate the account balances for accounts 2 and 3 given their various odds, showing that the big bets were processed correctly: win on match 2/team 0, match 4/team 1, and no take on match 3/team 0. was a win for team 1 on match 2, generating a total payoff to accounts 2 and 3 consistent with these old and new bet amounts

# Test 6: data submission

Tests that data sent to the oracle are processed correctly in their sequencing requirements, and also whether the votes were majorities for passage or not. There are three token owners

Account[0]: 4.0MM tokens

Account[1]: 2.5MM tokens

Account[2]: 1.5MM tokens

Total of 8.0MM tokens

* The initial process in line 283 succeeds because there were no other votes, and the initial proposer has all of his tokens count as a yes vote; the majority yes vote is unanimous.
* Attempt to send updated odds is successful, as the total vote is 5.5MM yes, 2.5MM no. Odds for match 1 change from 999 to 2000
* Attempt to send updated odds fails, as the vote total is 4.0MM yes and 4.0MM no (lines 211-212). A majority yes is needed, so odds stay at 2000.

Fails

1. Try to send data to the betting contract prior to the end of the 5-hour cure period
2. Try to send odds to the oracle while the odds currently being voted upon have not been processed.
3. Try to process a settlement though the data under review are an odds update.

# Test 7: LP payout

This tests that bookies are allocated eth correctly given their eth investment. LPs are credited with shares based on the current share price, which is the ratio of LP eth to LP shares. A withdrawal involves redeeming those shares, which then sends the requisite amount of eth to the LP.

From the start of the first contest to settlement, no investment or withdrawal is allowed by the bookies. Outside of this time window the ratio of eth allocated to the bookies (margin[0]) and the shares owned by the bookies, is the share price. Investors are given shares at this price, and shares are redeemed at this price.

Initially, we have two bookies who invest a total of 100,000 units. Their initial share allocation is also 100000 units, as the ratio of

shares eth

account 0 60000 60000 finney

account 1 40000 40000 finney

Initial shares and eth invested are consistent

In period 2 account[0] withdraws 100 shares, and account[1] withdraws 50 shares, the contract now has 850 shares outstanding. It also has 850 finney eth in the bookie margin.

A bet is made for 5000 finney, which loses. This finney is then sent to the bookie margin at settlement, adding to the LP’s eth.

document that there is now 9000 units in bookie margin, and 8500 shares shares outstanding, so the price of shares is 1.059.

A withdrawal of 10,000 and 5,000 shares by accounts 0 and 1 result in an outflow of 15882 finney eth. This is because 15000\*1.059=15882. Acct 0 and 1 receive 10588 and 5294 units respectively.

Account[0] adds 10000 units to his bookie account. The withdrawal did not change the eth/share value, since no new revenue came in. Unlike in the first period where LP’s received equal shares per eth invested, now the 10000 unit investment generates 9444 shares, because the share price rose from 1.0 to 1.0588.

# Test 8: Oracle token holder payout

This tests the oracle’s ability to pay out its revenue correctly. Oracle token holders can withdraw their share of the oracle’s eth every 13 epochs. Three accounts deposit tokens in the oracle contract at various times. In each epoch the oracle receives 10 units of revenue that needs to be allocated to the oracle owners. Oracle revenue is only claimed by having tokens deposited in the oracle contract.

There are 5 payments generated, and the oracles enter and exit and different times. In epoch 3 account 0 increases her token investment. The spreadsheet shows how the payments are calculated in the orange cells, and this is equivalent to the amounts calculated in the green cells. The contract uses the algorithm in the green cells. The difference is due to rounding in the contract. I used bet amounts of ~20 finneys, because this was convenient for test nets where I had limited amounts of ether, so the rounding truncation is economically inconsequential, and does not create an insolvency risk because it shortchanges withdrawers, not the contract.

# Test 9: Oracle token holder payout

This is a rerun of test 8, only account 0 adds to her token deposit as opposed to withdrawing. I thought this would be silly but it identified a bug, so I’m very glad I did this test. Using the same logic as in test 8, it highlights how oracle revenue is allocated to depositors as a function of their proportional deposited amount each week.

# Test 10: input check

Betting odds and start times are packed into a single uint256, along with other data, for each epoch’s match. To demonstrate this method is correct, we input the data, make bets that adjust these other data, and then pull and decode the information from this uint256.

The tests show that the odds input can be successful decoded, and that the update in the odds was accurately recorded and did not affect the other variables.