incode

Using events to unit test smart contracts with Truffle

By Rosco Kalis

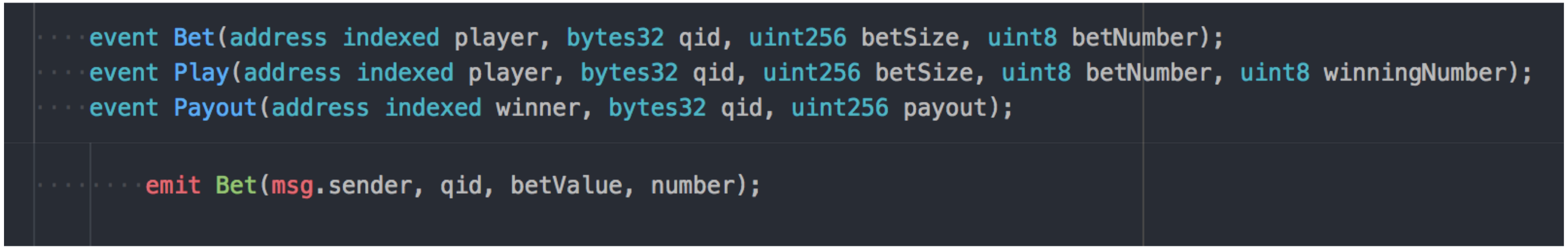
About me

- Software Engineer at Incode
- Graduate Student at University of Amsterdam
- Open Source Advocate
- Ethereum Smart Contract Developer
  - Blockchain-validated audit trail
  - ConsenSys Developer Program – Decentralised casino

Contents

What are events?

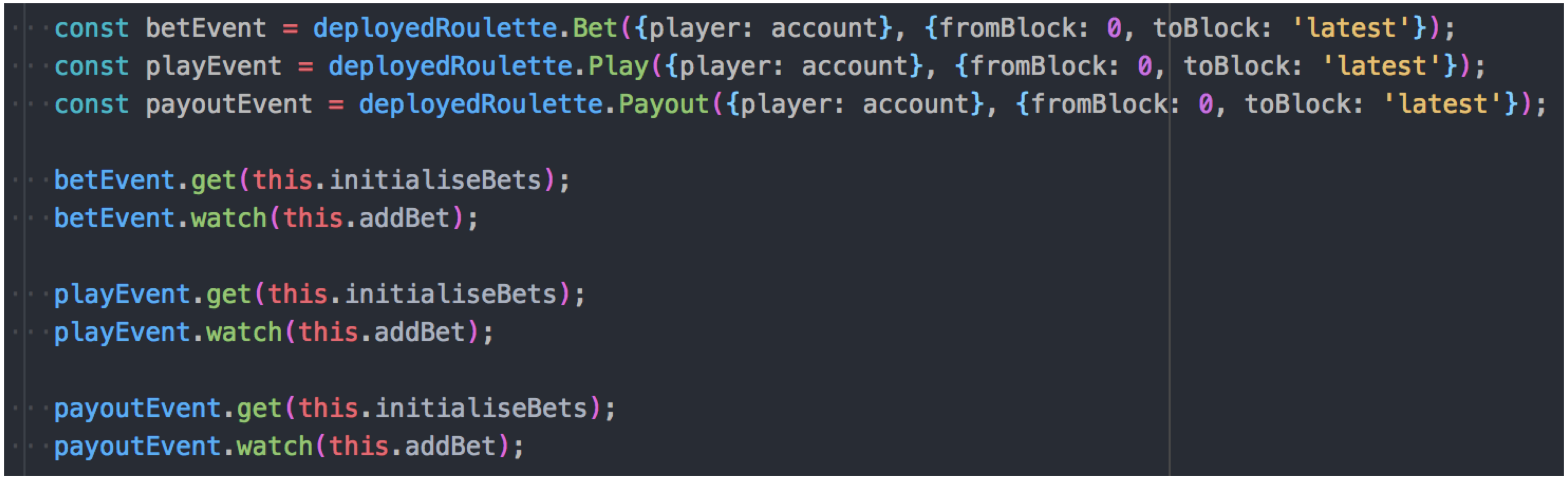
- Log of smart contract usage
- Stored in transaction log
- Can have indexed arguments for searching



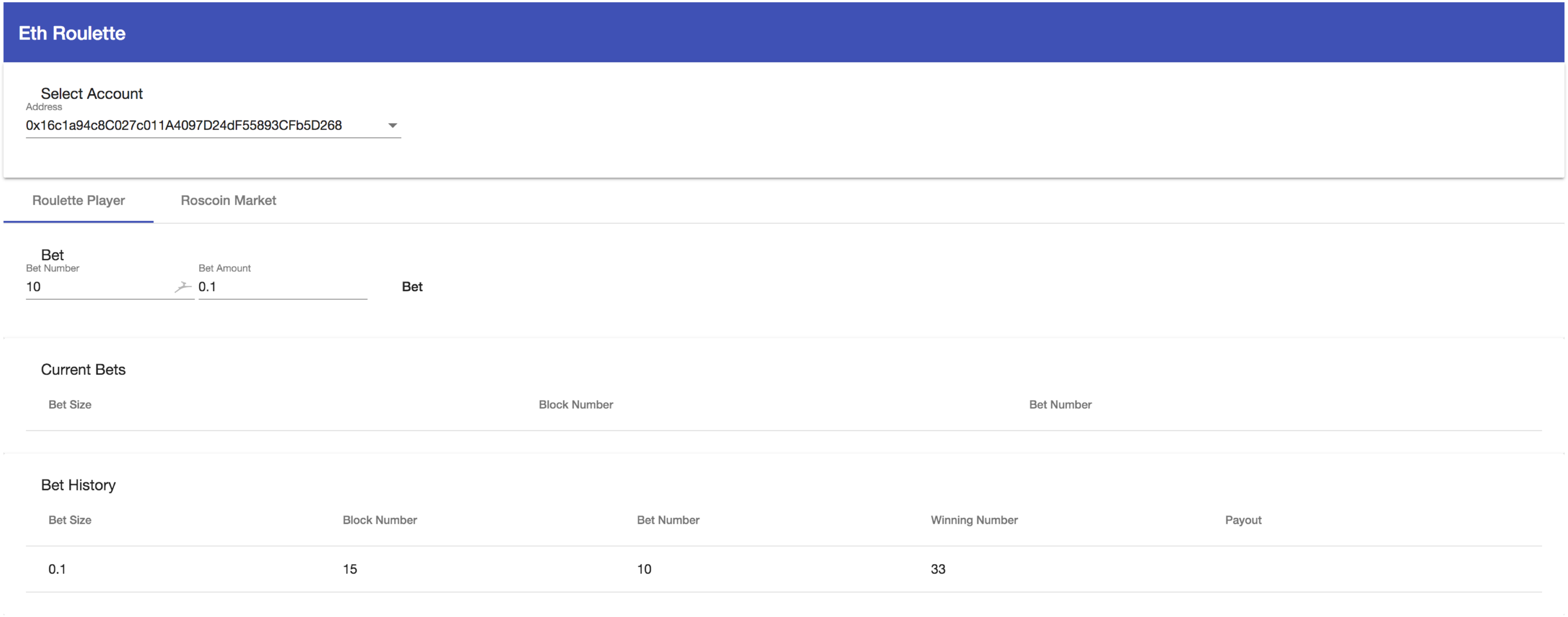
```
    event Bet(address indexed player, bytes32 qid, uint256 betSize, uint8 betNumber);
    event Play(address indexed player, bytes32 qid, uint256 betSize, uint8 betNumber, uint8 winningNumber);
    event Payout(address indexed winner, bytes32 qid, uint256 payout);

        emit Bet(msg.sender, qid, betValue, number);
```

What are events used for in dapps?

- Logging smart contract functionality & updating user interface
- Event.watch() & Event.get()
- Filtering on indexed parameters



```
const betEvent = deployedRoulette.Bet({player: account}, {fromBlock: 0, toBlock: 'latest'});
const playEvent = deployedRoulette.Play({player: account}, {fromBlock: 0, toBlock: 'latest'});
const payoutEvent = deployedRoulette.Payout({player: account}, {fromBlock: 0, toBlock: 'latest'});

betEvent.get(this.initialiseBets);
betEvent.watch(this.addBet);

playEvent.get(this.initialiseBets);
playEvent.watch(this.addBet);

payoutEvent.get(this.initialiseBets);
payoutEvent.watch(this.addBet);
```



Eth Roulette

Select Account

Address

0x16c1a94c8C027c011A4097D24dF55893CFb5D268

Bet

Bet Number: 10

Bet Amount: 0.1

Current Bets

Bet Size	Block Number	Bet Number

Bet History

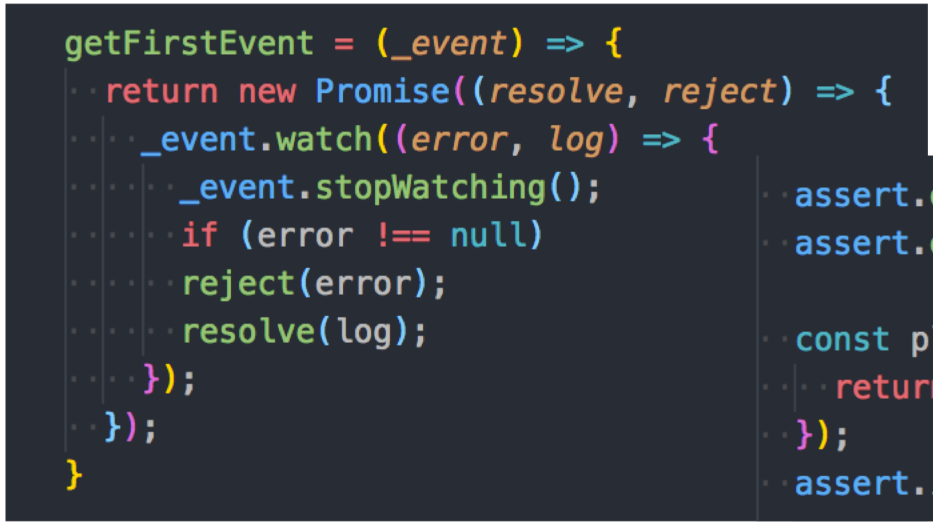
Bet Size	Block Number	Bet Number	Winning Number	Payout
0.1	15	10	33	

Testing for a purpose

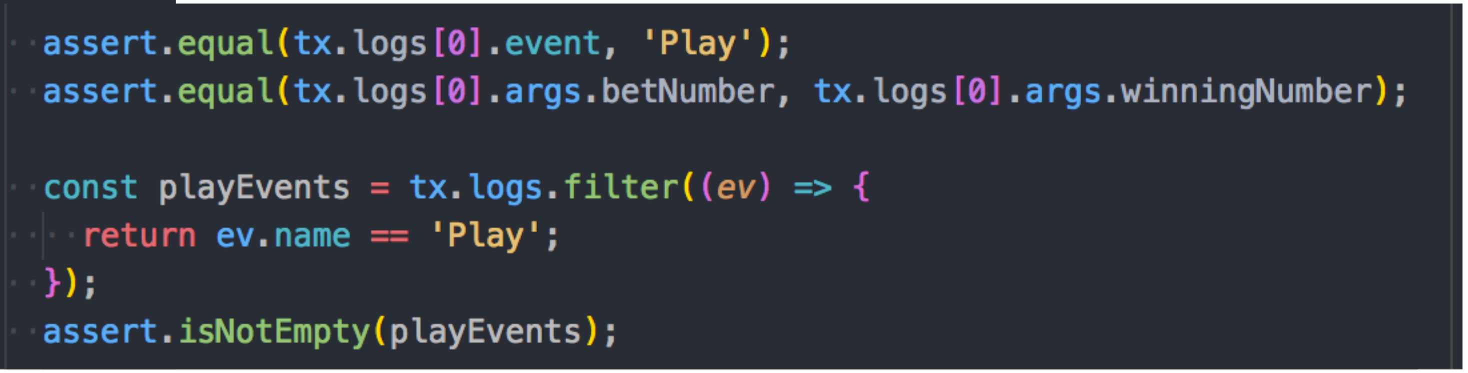
- Tests should be based on actual application usage
- Solidity testing vs JavaScript testing

Why are events not widely used in tests?

- Cumbersome to use Event.watch() in unit tests
- Logs are difficult to parse



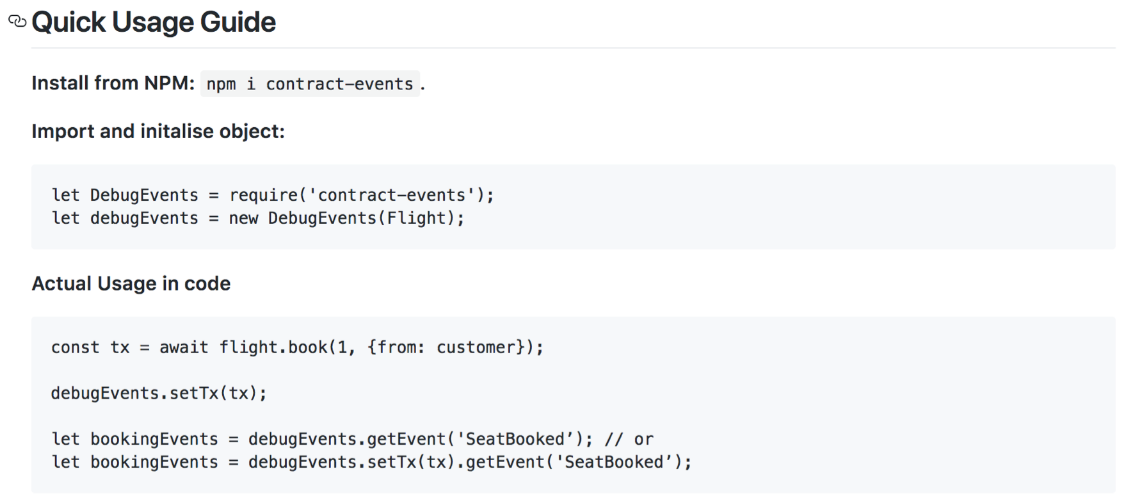
```
getFirstEvent = (_event) => {
  return new Promise((resolve, reject) => {
    _event.watch((error, log) => {
      _event.stopWatching();
      if (error !== null)
      reject(error);
      resolve(log);
    });
  });
}
```



```
  assert.equal(tx.logs[0].event, 'Play');
  assert.equal(tx.logs[0].args.betNumber, tx.logs[0].args.winningNumber);

  const playEvents = tx.logs.filter((ev) => {
    return ev.name == 'Play';
  });
  assert.isNotEmpty(playEvents);
```

Overcoming these issues

- Don’t go through logs manually
- Libraries
  - contract-events
  - truffle-assertions



Quick Usage Guide

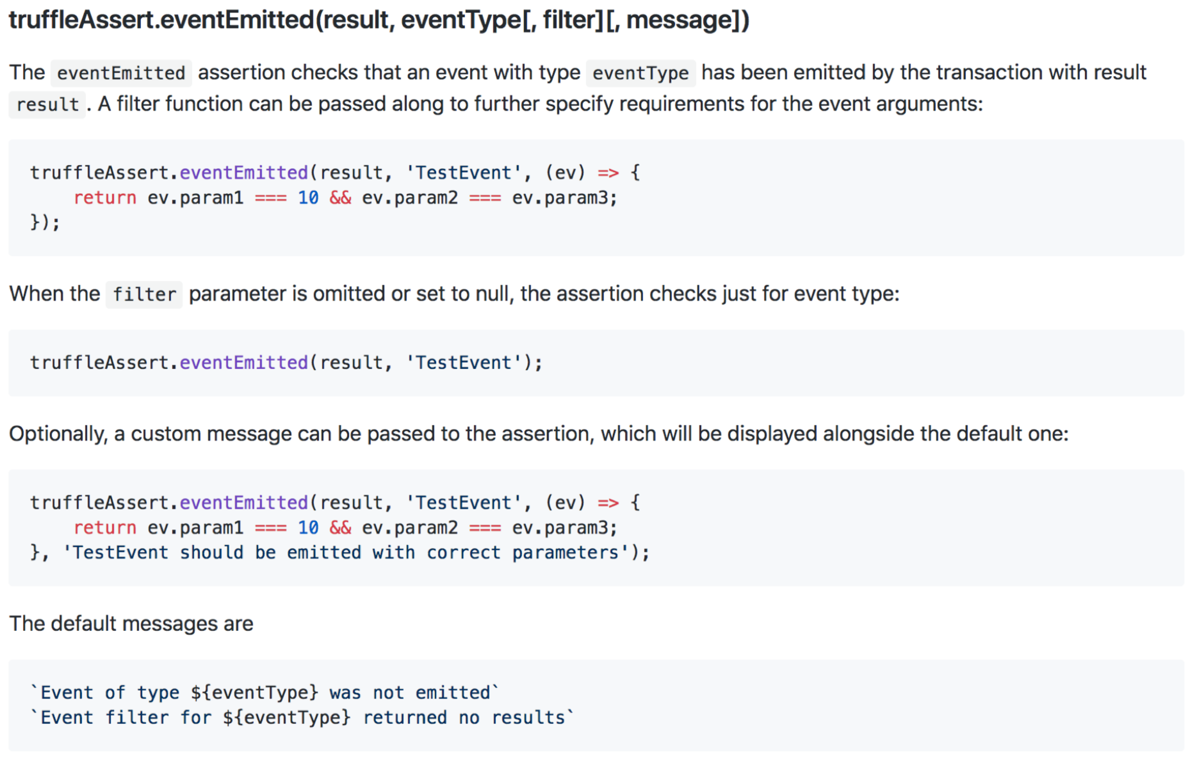
Install from NPM: `npm i contract-events` .

Import and initalise object:

```
let DebugEvents = require('contract-events');
let debugEvents = new DebugEvents(Flight);
```

Actual Usage in code

```
const tx = await flight.book(1, {from: customer});

debugEvents.setTx(tx);

let bookingEvents = debugEvents.getEvent('SeatBooked'); // or
let bookingEvents = debugEvents.setTx(tx).getEvent('SeatBooked');
```



truffleAssert.eventEmitted(result, eventType[, filter][, message])

The `eventEmitted` assertion checks that an event with type `eventType` has been emitted by the transaction with result `result` . A filter function can be passed along to further specify requirements for the event arguments:

```
truffleAssert.eventEmitted(result, 'TestEvent', (ev) => {
    return ev.param1 === 10 && ev.param2 === ev.param3;
});
```

When the `filter` parameter is omitted or set to null, the assertion checks just for event type:

```
truffleAssert.eventEmitted(result, 'TestEvent');
```

Optionally, a custom message can be passed to the assertion, which will be displayed alongside the default one:

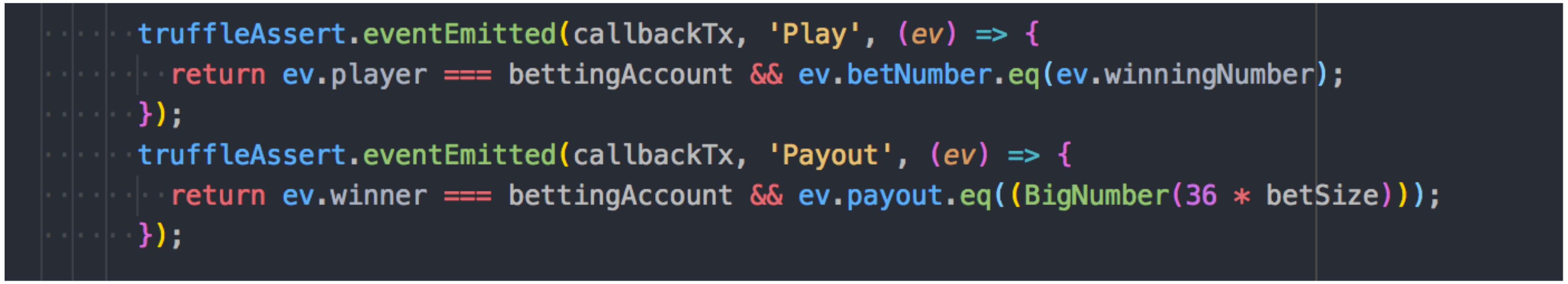
```
truffleAssert.eventEmitted(result, 'TestEvent', (ev) => {
    return ev.param1 === 10 && ev.param2 === ev.param3;
}, 'TestEvent should be emitted with correct parameters');
```

The default messages are

```
`Event of type ${eventType} was not emitted`
`Event filter for ${eventType} returned no results`
```

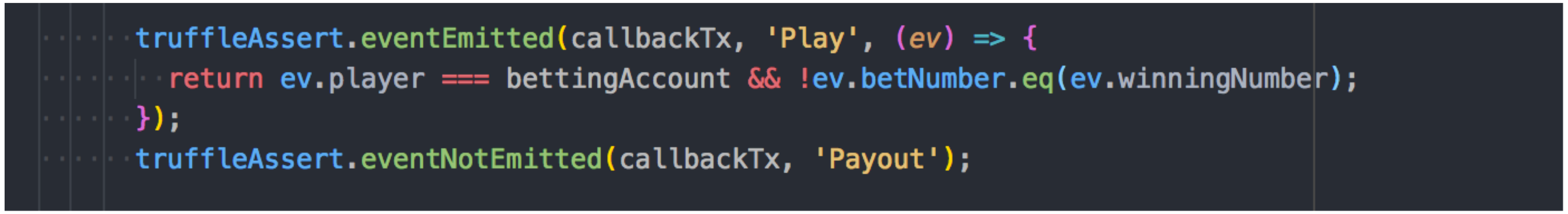
Making assertions on emitted events

- Check that events have been emitted



```
truffleAssert.eventEmitted(callbackTx, 'Play', (ev) => {
  return ev.player === bettingAccount && ev.betNumber.eq(ev.winningNumber);
});
truffleAssert.eventEmitted(callbackTx, 'Payout', (ev) => {
  return ev.winner === bettingAccount && ev.payout.eq((BigNumber(36 * betSize)));
});
```

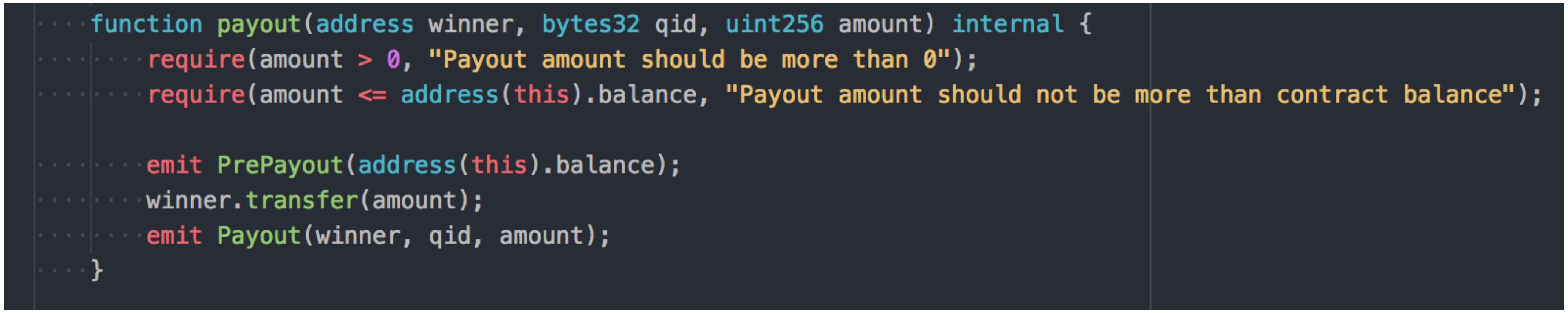
- Check that events have not been emitted



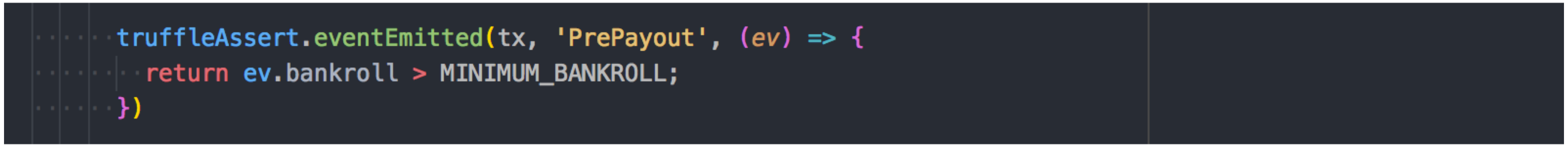
```
truffleAssert.eventEmitted(callbackTx, 'Play', (ev) => {
  return ev.player === bettingAccount && !ev.betNumber.eq(ev.winningNumber);
});
truffleAssert.eventNotEmitted(callbackTx, 'Payout');
```

Strategically placed events

- Test for certain preconditions



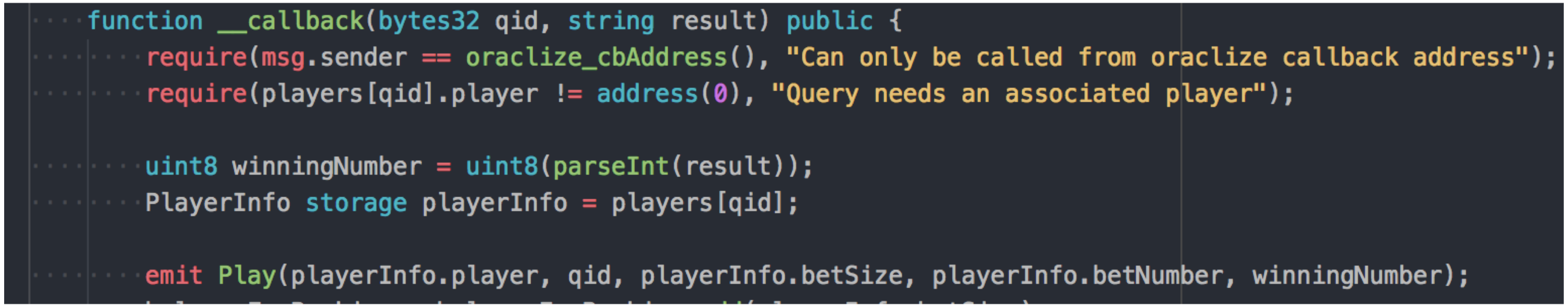
```
function payout(address winner, bytes32 qid, uint256 amount) internal {
    require(amount > 0, "Payout amount should be more than 0");
    require(amount <= address(this).balance, "Payout amount should not be more than contract balance");

    emit PrePayout(address(this).balance);
    winner.transfer(amount);
    emit Payout(winner, qid, amount);
}
```



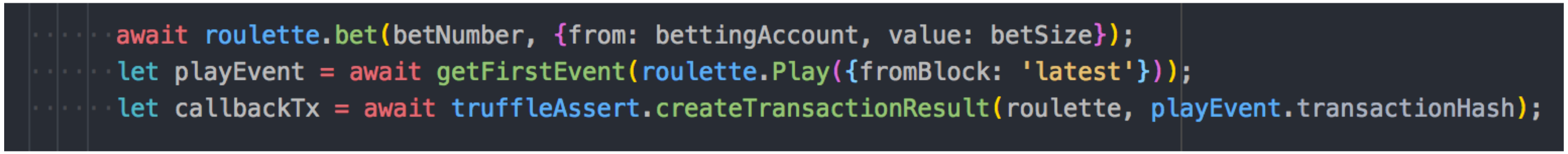
```
truffleAssert.eventEmitted(tx, 'PrePayout', (ev) => {
  return ev.bankroll > MINIMUM_BANKROLL;
})
```

Awaiting callback execution

- Oraclize callback or callback from a different contract
- Would happen automatically inside dapp
- Await the event & retrieve the corresponding transaction result



```
    function __callback(bytes32 qid, string result) public {
        require(msg.sender == oraclize_cbAddress(), "Can only be called from oraclize callback address");
        require(players[qid].player != address(0), "Query needs an associated player");

        uint8 winningNumber = uint8(parseInt(result));
        PlayerInfo storage playerInfo = players[qid];

        emit Play(playerInfo.player, qid, playerInfo.betSize, playerInfo.betNumber, winningNumber);
```



```
      await roulette.bet(betNumber, {from: bettingAccount, value: betSize});
      let playEvent = await getFirstEvent(roulette.Play({fromBlock: 'latest'}));
      let callbackTx = await truffleAssert.createTransactionResult(roulette, playEvent.transactionHash);
```

Questions & Discussion

How would you use events in your unit tests?

Information & Resources

kalis.me
incode.org
github.com/rkalis
twitter.com/RoscoKalis
kalis.me/check-events-solidity-smart-contract-test-truffle
github.com/rkalis/truffle-assertions
github.com/smallbatch-apps/contract-events
github.com/zulhfreelancer/truffle-events