Ownable.sol

```
/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic
authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;

  event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);

  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to
the sender
   * account.
   */
  function Ownable() public {
    owner = msg.sender;
  }


  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }


  /**
   * @dev Allows the current owner to transfer control of the contract to a
newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

}
```

Safemath.sol

```
pragma solidity ^0.4.18;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater
than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

Zombiefactory.sol

```
pragma solidity ^0.4.19;

import "./ownable.sol";
import "./safemath.sol";

contract ZombieFactory is Ownable {

  using SafeMath for uint256;

  event NewZombie(uint zombieId, string name, uint dna);

  uint dnaDigits = 16;
  uint dnaModulus = 10 ** dnaDigits;
  uint cooldownTime = 1 days;

  struct Zombie {
    string name;
    uint dna;
    uint32 level;
    uint32 readyTime;
    uint16 winCount;
    uint16 lossCount;
  }

  Zombie[] public zombies;

  mapping (uint => address) public zombieToOwner;
  mapping (address => uint) ownerZombieCount;

  function _createZombie(string _name, uint _dna) internal {
    uint id = zombies.push(Zombie(_name, _dna, 1, uint32(now + cooldownTime), 0,
0)) - 1;
    zombieToOwner[id] = msg.sender;
    ownerZombieCount[msg.sender]++;
    NewZombie(id, _name, _dna);
  }
```

```
  function _generateRandomDna(string _str) private view returns (uint) {
    uint rand = uint(keccak256(_str));
    return rand % dnaModulus;
  }

  function createRandomZombie(string _name) public {
    require(ownerZombieCount[msg.sender] == 0);
    uint randDna = _generateRandomDna(_name);
    randDna = randDna - randDna % 100;
    _createZombie(_name, randDna);
  }

}
```

Erc721.sol

```
contract ERC721 {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256
_tokenId);

  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function transfer(address _to, uint256 _tokenId) public;
  function approve(address _to, uint256 _tokenId) public;
  function takeOwnership(uint256 _tokenId) public;
}
```

Zombiefeeding.sol

```
pragma solidity ^0.4.19;

import "./zombiefactory.sol";

contract KittyInterface {
  function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
```

```
    uint256 sireId,
    uint256 generation,
    uint256 genes
  );
}

contract ZombieFeeding is ZombieFactory {

  KittyInterface kittyContract;

  modifier onlyOwnerOf(uint _zombieId) {
    require(msg.sender == zombieToOwner[_zombieId]);
    _;
  }

  function setKittyContractAddress(address _address) external onlyOwner {
    kittyContract = KittyInterface(_address);
  }

  function _triggerCooldown(Zombie storage _zombie) internal {
    _zombie.readyTime = uint32(now + cooldownTime);
  }

  function _isReady(Zombie storage _zombie) internal view returns (bool) {
      return (_zombie.readyTime <= now);
  }

  function feedAndMultiply(uint _zombieId, uint _targetDna, string _species)
internal onlyOwnerOf(_zombieId) {
    Zombie storage myZombie = zombies[_zombieId];
    require(_isReady(myZombie));
    _targetDna = _targetDna % dnaModulus;
    uint newDna = (myZombie.dna + _targetDna) / 2;
    if (keccak256(_species) == keccak256("kitty")) {
      newDna = newDna - newDna % 100 + 99;
    }
    _createZombie("NoName", newDna);
    _triggerCooldown(myZombie);
  }

  function feedOnKitty(uint _zombieId, uint _kittyId) public {
    uint kittyDna;
    (,,,,,,,,,kittyDna) = kittyContract.getKitty(_kittyId);
    feedAndMultiply(_zombieId, kittyDna, "kitty");
  }
```

```
}
```

Zombiehelper.sol

```
pragma solidity ^0.4.19;

import "./zombiefeeding.sol";

contract ZombieHelper is ZombieFeeding {

  uint levelUpFee = 0.001 ether;

  modifier aboveLevel(uint _level, uint _zombieId) {
    require(zombies[_zombieId].level >= _level);
    _;
  }

  function withdraw() external onlyOwner {
    owner.transfer(this.balance);
  }

  function setLevelUpFee(uint _fee) external onlyOwner {
    levelUpFee = _fee;
  }

  function levelUp(uint _zombieId) external payable {
    require(msg.value == levelUpFee);
    zombies[_zombieId].level++;
  }

  function changeName(uint _zombieId, string _newName) external aboveLevel(2,
_zombieId) onlyOwnerOf(_zombieId) {
    zombies[_zombieId].name = _newName;
  }

  function changeDna(uint _zombieId, uint _newDna) external aboveLevel(20,
_zombieId) onlyOwnerOf(_zombieId) {
    zombies[_zombieId].dna = _newDna;
  }

  function getZombiesByOwner(address _owner) external view returns(uint[]) {
    uint[] memory result = new uint[](ownerZombieCount[_owner]);
    uint counter = 0;
    for (uint i = 0; i < zombies.length; i++) {
```

```
      if (zombieToOwner[i] == _owner) {
        result[counter] = i;
        counter++;
      }
    }
    return result;
  }

}
```

Zombieattack.sol

```
pragma solidity ^0.4.19;

import "./zombiehelper.sol";

contract ZombieAttack is ZombieHelper {
  uint randNonce = 0;
  uint attackVictoryProbability = 70;

  function randMod(uint _modulus) internal returns(uint) {
    randNonce++;
    return uint(keccak256(now, msg.sender, randNonce)) % _modulus;
  }

  function attack(uint _zombieId, uint _targetId) external onlyOwnerOf(_zombieId)
{
    Zombie storage myZombie = zombies[_zombieId];
    Zombie storage enemyZombie = zombies[_targetId];
    uint rand = randMod(100);
    if (rand <= attackVictoryProbability) {
      myZombie.winCount++;
      myZombie.level++;
      enemyZombie.lossCount++;
      feedAndMultiply(_zombieId, enemyZombie.dna, "zombie");
    } else {
      myZombie.lossCount++;
      enemyZombie.winCount++;
      _triggerCooldown(myZombie);
    }
  }
}
```

Zombieownership.sol

```
pragma solidity ^0.4.19;

import "./zombieattack.sol";
import "./erc721.sol";
import "./safemath.sol";

contract ZombieOwnership is ZombieAttack, ERC721 {

  using SafeMath for uint256;

  mapping (uint => address) zombieApprovals;

  function balanceOf(address _owner) public view returns (uint256 _balance) {
    return ownerZombieCount[_owner];
  }

  function ownerOf(uint256 _tokenId) public view returns (address _owner) {
    return zombieToOwner[_tokenId];
  }

  function _transfer(address _from, address _to, uint256 _tokenId) private {
    ownerZombieCount[_to] = ownerZombieCount[_to].add(1);
    ownerZombieCount[msg.sender] = ownerZombieCount[msg.sender].sub(1);
    zombieToOwner[_tokenId] = _to;
    Transfer(_from, _to, _tokenId);
  }

  function transfer(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    _transfer(msg.sender, _to, _tokenId);
  }

  function approve(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    zombieApprovals[_tokenId] = _to;
    Approval(msg.sender, _to, _tokenId);
  }

  function takeOwnership(uint256 _tokenId) public {
    require(zombieApprovals[_tokenId] == msg.sender);
    address owner = ownerOf(_tokenId);
    _transfer(owner, msg.sender, _tokenId);
  }
}
```

Cryptozombies_abi.js

```
var cryptozombiesABI = [
  {
    "constant": false,
    "inputs": [
      {
        "name": "_to",
        "type": "address"
      },
      {
        "name": "_tokenId",
        "type": "uint256"
      }
    ],
    "name": "approve",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_zombieId",
        "type": "uint256"
      }
    ],
    "name": "levelUp",
    "outputs": [],
    "payable": true,
    "stateMutability": "payable",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_zombieId",
        "type": "uint256"
      },
      {
        "name": "_kittyId",
        "type": "uint256"
```

```
      }
    ],
    "name": "feedOnKitty",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "name": "zombies",
    "outputs": [
      {
        "name": "name",
        "type": "string"
      },
      {
        "name": "dna",
        "type": "uint256"
      },
      {
        "name": "level",
        "type": "uint32"
      },
      {
        "name": "readyTime",
        "type": "uint32"
      },
      {
        "name": "winCount",
        "type": "uint16"
      },
      {
        "name": "lossCount",
        "type": "uint16"
      }
    ],
    "payable": false,
    "stateMutability": "view",
```

```
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [],
    "name": "withdraw",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "_owner",
        "type": "address"
      }
    ],
    "name": "getZombiesByOwner",
    "outputs": [
      {
        "name": "",
        "type": "uint256[]"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "name": "zombieToOwner",
    "outputs": [
      {
        "name": "",
        "type": "address"
      }
    ],
```

```
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "name": "_address",
      "type": "address"
    }
  ],
  "name": "setKittyContractAddress",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "name": "_zombieId",
      "type": "uint256"
    },
    {
      "name": "_newDna",
      "type": "uint256"
    }
  ],
  "name": "changeDna",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": true,
  "inputs": [
    {
      "name": "_tokenId",
      "type": "uint256"
    }
  ],
  "name": "ownerOf",
```

```
    "outputs": [
      {
        "name": "_owner",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "_owner",
        "type": "address"
      }
    ],
    "name": "balanceOf",
    "outputs": [
      {
        "name": "_balance",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_name",
        "type": "string"
      }
    ],
    "name": "createRandomZombie",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
```

```
    "inputs": [],
    "name": "owner",
    "outputs": [
      {
        "name": "",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "name": "_to",
        "type": "address"
      },
      {
        "name": "_tokenId",
        "type": "uint256"
      }
    ],
    "name": "transfer",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "getAllZombies",
    "outputs": [
      {
        "name": "",
        "type": "uint256[]"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
```

```
  "constant": false,
  "inputs": [
    {
      "name": "_tokenId",
      "type": "uint256"
    }
  ],
  "name": "takeOwnership",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "name": "_zombieId",
      "type": "uint256"
    },
    {
      "name": "_newName",
      "type": "string"
    }
  ],
  "name": "changeName",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "name": "_fee",
      "type": "uint256"
    }
  ],
  "name": "setLevelUpFee",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
```

```
{
  "constant": false,
  "inputs": [
    {
      "name": "_zombieId",
      "type": "uint256"
    },
    {
      "name": "_targetId",
      "type": "uint256"
    }
  ],
  "name": "attack",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "name": "newOwner",
      "type": "address"
    }
  ],
  "name": "transferOwnership",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "name": "_from",
      "type": "address"
    },
    {
      "indexed": true,
      "name": "_to",
      "type": "address"
    },
```

```
      {
        "indexed": false,
        "name": "_tokenId",
        "type": "uint256"
      }
    ],
    "name": "Transfer",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": true,
        "name": "_owner",
        "type": "address"
      },
      {
        "indexed": true,
        "name": "_approved",
        "type": "address"
      },
      {
        "indexed": false,
        "name": "_tokenId",
        "type": "uint256"
      }
    ],
    "name": "Approval",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "name": "attackResult",
        "type": "bool"
      },
      {
        "indexed": false,
        "name": "winCount",
        "type": "uint16"
      },
      {
```

```
        "indexed": false,
        "name": "lossCount",
        "type": "uint16"
      }
    ],
    "name": "AttackResult",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "name": "zombieId",
        "type": "uint256"
      },
      {
        "indexed": false,
        "name": "name",
        "type": "string"
      },
      {
        "indexed": false,
        "name": "dna",
        "type": "uint256"
      }
    ],
    "name": "NewZombie",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": true,
        "name": "previousOwner",
        "type": "address"
      },
      {
        "indexed": true,
        "name": "newOwner",
        "type": "address"
      }
    ],
    "name": "OwnershipTransferred",
```

```
    "type": "event"
  }
]
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>CryptoZombies front-end</title>
    <script language="javascript" type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script language="javascript" type="text/javascript"
src="web3.min.js"></script>
    <script language="javascript" type="text/javascript"
src="cryptozombies_abi.js"></script>
  </head>
  <body>
    <div id="txStatus"></div>
    <div id="zombies"></div>

    <script>
      var cryptoZombies;
      var userAccount;

      function startApp() {
        var cryptoZombiesAddress = "YOUR_CONTRACT_ADDRESS";
        cryptoZombies = new web3js.eth.Contract(cryptoZombiesABI,
cryptoZombiesAddress);

        var accountInterval = setInterval(function() {
          // Check if account has changed
          if (web3.eth.accounts[0] !== userAccount) {
            userAccount = web3.eth.accounts[0];
            // Call a function to update the UI with the new account
            getZombiesByOwner(userAccount)
            .then(displayZombies);
          }
        }, 100);

        // Start here
      }

      function displayZombies(ids) {
```

```
        $("#zombies").empty();
        for (id of ids) {
          // Look up zombie details from our contract. Returns a `zombie` object
          getZombieDetails(id)
          .then(function(zombie) {
            // Using ES6's "template literals" to inject variables into the HTML.
            // Append each one to our #zombies div
            $("#zombies").append(`<div class="zombie">
              <ul>
                <li>Name: ${zombie.name}</li>
                <li>DNA: ${zombie.dna}</li>
                <li>Level: ${zombie.level}</li>
                <li>Wins: ${zombie.winCount}</li>
                <li>Losses: ${zombie.lossCount}</li>
                <li>Ready Time: ${zombie.readyTime}</li>
              </ul>
            </div>`);
          });
        }
      }

      function createRandomZombie(name) {
        // This is going to take a while, so update the UI to let the user know
        // the transaction has been sent
        $("#txStatus").text("Creating new zombie on the blockchain. This may take
a while...");
        // Send the tx to our contract:
        return CryptoZombies.methods.createRandomZombie(name)
        .send({ from: userAccount })
        .on("receipt", function(receipt) {
          $("#txStatus").text("Successfully created " + name + "!");
          // Transaction was accepted into the blockchain, let's redraw the UI
          getZombiesByOwner(userAccount).then(displayZombies);
        })
        .on("error", function(error) {
          // Do something to alert the user their transaction has failed
          $("#txStatus").text(error);
        });
      }

      function feedOnKitty(zombieId, kittyId) {
        $("#txStatus").text("Eating a kitty. This may take a while...");
        return CryptoZombies.methods.feedOnKitty(zombieId, KittyId)
        .send({ from: userAccount })
        .on("receipt", function(receipt) {
```

```
          $("#txStatus").text("Ate a kitty and spawned a new Zombie!");
          getZombiesByOwner(userAccount).then(displayZombies);
        })
        .on("error", function(error) {
          $("#txStatus").text(error);
        });
      }

      function levelUp(zombieId) {
        $("#txStatus").text("Leveling up your zombie...");
        return CryptoZombies.methods.levelUp(zombieId)
        .send({ from: userAccount, value: web3.utils.toWei("0.001") })
        .on("receipt", function(receipt) {
          $("#txStatus").text("Power overwhelming! Zombie successfully leveled
up");
        })
        .on("error", function(error) {
          $("#txStatus").text(error);
        });
      }

      function getZombieDetails(id) {
        return cryptoZombies.methods.zombies(id).call()
      }

      function zombieToOwner(id) {
        return cryptoZombies.methods.zombieToOwner(id).call()
      }

      function getZombiesByOwner(owner) {
        return cryptoZombies.methods.getZombiesByOwner(owner).call()
      }

      window.addEventListener('load', function() {

        // Checking if Web3 has been injected by the browser (Mist/MetaMask)
        if (typeof web3 !== 'undefined') {
          // Use Mist/MetaMask's provider
          web3js = new Web3(web3.currentProvider);
        } else {
          // Handle the case where the user doesn't have Metamask installed
          // Probably show them a message prompting them to install Metamask
        }

        // Now you can start your app & access web3 freely:
```

```
        startApp()

      })
    </script>
  </body>
</html>
```