IEEE format examples, resources, and guides used… Second one is overkill, but useful...
http://dreuarchive.cra.org/2009/Jaramillo/DREU/files/final_report.pdf
https://www.unlv.edu/sites/default/files/page_files/27/Engineering-PreparingPaperIEEE-Sept15.pdf

Risk-Chain Formal Report

Sean Thunquest
Luke Burford
David Wells

Computer Science Department, Colorado State University

- Abstract

***Abstract*— This paper describes a group project experience in the Blockchain course at Colorado State University. The team project consists of creating a web platform game based on the classic board game of Risk. The problem the team worked to find a solution for is to accurately track the state of a traditional board game built into a web app with a smart contract on the Ethereum blockchain to handle transacting with the state of the board. The project relied heavily on React for the interface, while the logic for the game was built using Ethereum and Solidity to develop a Blockchain solution for tracking the state of the board, ensuring immutable game conditions. With all of these technologies converging together came the proposed solution dubbed "Risk-Chain". Although the team decided to deviate from the original game rules to serve a web based variant, the solutions primary aims are consistent with the theme of the original game.**

I. Introduction

Risk is a game of world domination. Through the ups and downs of the game power is gained and lost at the peril of other players. Anyone who has played this classic board game know that it is cutthroat with triumph, failure, and sometimes emotional outbursts of rage, often resulting in boardflipping. This is where a blockchain can improve the board game world. Creating a Dapp (decentralized application) on a blockchain was a natural solution to these problems. Blockchains are immutable, so there is no flipping of boards, or cheating by fudging rolls or sneaking any extra troops in Argentina. Using react, web3 and Solidity this has become a reality with Risk-Chain.

II. Modifications

Risk-Chain was developed to be a true turn based game. Risk-Chain is based on Risk, but with some distinct differences. Risk-Chain can be played by performing your turn whenever you have time throughout the day. To accomplish this, Risk-Chain deviated from traditional Risk rules. Risk-Chain does not require defending players to perform actions during other players turns. A turn consists of A deployment phase, an attack phase, and a troop distribution phase. We considered adding an opportunity for players to play cards like Risk allows, but have not fully implemented it for this version.

III. Implementation

Using smart contracts to handle the state of a game is great, but making an user interface that utilizes the logic of that contract is essential. For this purpose Risk-Chain relies on React JS and a large interactive SVG map for a user to see and interact with. To communicate with the contract web3 was utilized to make calls to the contract, and access information on the blockchain.

(First technical section as second and third paragraph)

IV. Issues

(Second technical section…)

As with learning any new technology, going from demos and examples to scalable projects, issues will present themselves. One such issue involves the limitations to using solidity contracts for robust applications. One such issue scaling projects in solidity involves dealing with a hard gas limit deploying contracts. Although in theory gas limits can be set by the contract owner, in practice there is a hard gas limit set by the evm. Since contracts are compiled into EVM bytecode, at the cost of 200 gas per byte, there are really two ways at looking at size limits. First there is a hard gas limit of 8 million wei. This in theory would mean your bytecode can be up to 40,000 bytes. In practice however the actual limit is clearly lower. There are extra bytes needed for deployment and transaction data, and there is a separate limit for individual transactions which is much lower, and the actual limit that we ran into while working with the Risk-Chain contract. [insert source related to evm gas limit]

https://ethereum.stackexchange.com/questions/47539/how-big-could-a-contract-be

Risk Contract constituents:

- Deploying & Initializing
  - Constructor deploys contract with the number of players defined and seed for random rolls.
  - Call addPlayer(address) for each player
  - Call createContinents() to initialize the structures for the continents
  - Call createRegions() to initialize each region structure and place into corresponding continent
  - assignPlayers() distributes ownership of regions to players and gives them their starting troops on those countries.
- Viewing the game state
  - getGameState() returns a JSON object representing ownership of all countries, troop counts, and turn status information.
  - getAllPlayers() returns all players addresses of registered players.
  - getCurrentPlayer() returns the player whose turn it currently is.
  - getPlayerIncome() Gets the players income for how many troops they can reinforce with.
- Interacting on your turn

        - PlaceTroopsDriver(uint[]) is the public function to place troops from your income in the first phase of a turn.
        - AttackDriver(uint[]) is the public function to launch the attack for the turn.
        - TrfArmiesDriver(uint[]) is the public function to launch the troop movement phase.
    - Internal functions
        - Phase driver helpers
            - Attack() is called from the AttackDriver in a loop and attacks one country to another with a set number of troops
            - TrfArmies() is called from the TrfArmiesDriver in a loop and moves armies to the desired locations
            - PlaceTroops() is called from the PlaceTroopDriver and
        - Attack Helpers
            - atckFavoredCompare() Drives dice roll when the attacker's dice outnumbers the defenders.
            - EvenCompare() Drives dice roll when the attacker and defender have the same number of dice.
            - defFavoredCompare() Drives dice roll when the defender outnumbers the attacker.
            - Rolldie() Invokes a roll on the dice.
            - sort() invokes quicksort for the dice rolls.
            - quickSort() sorts the list of results of the dice rolls for easy comparison.
        - View Helpers
            - checkContinentOwnership() Checks if a continent is owned by a player for adjusting bonus income.
            - uintToString() toString used for viewing game state.
            - addressToString() toString used for viewing gamestate
            - getCurrentPlayerOpponents() used to help construct opponents list for viewing on the board.

Skeleton…

- One or more technical sections on results, issues encountered, etc.

<u>Technical section ideas…</u>

1. <u>Countries svg, and highlighting/ previewing issues, and results.</u>
2. <u>Handling game state in Solidity contract. Migrating from react functions to using "on chain" functionality.</u>

3. Issues implementing web3 in the app.
4. Solidity issues and results?
5. Game adaptations to fit the platform.
6.

- Conclusion

- References

Report should be 4-5 pages in length.

-