Objectives

- Hands on with people writing code, fixing bugs, adding features, testing, etc.
- Documentation
- Moving each of two top-level projects forward
- Proposed topics can be found at the bottom of this doc -- please feel free to add to the list

Prerequisite:

- Dan Middleton: *"No Prereqs required. However for sawtooth it would be helpful to walk thru this: http://intelledger.github.io/tutorial.html . Gets dev environment setup."*

WG Meetings and Maintainer Office Hours During Hackathon

- Identity - scheduled
- Whitepaper - scheduled
- Architecture - 12 pm ET, Friday 6/17
- CI
- Requirements
- Fabric (issues)
- Sawtooth Lake

Wednesday (6/15)

- 10am ET: Kick-off (hosted by Dan Middleton)
  - Coordination of what people want to work on and establish some leaders to help drive forward various effortsWebEx
  - Join
    - +1-415-655-0001 US TOLL
    - Access code: 192 845 883
- noon ET: Identity WG
  - Join WebEx
    - +1-408-525-6800 Call-in toll number (US/Canada)
    - +1-866-432-9903 Call-in toll-free number (US/Canada)
    - Access code: 190 609 401
- 1pm ET: Whitepaper WG
  - +1-866-215-2642 US Toll free
  - +1-617-597-5043 (international dial-in)
  - Access code: 2814 2917
- 6pm ET: 30 min check-in with group
  - Join WebEx
    - +1-415-655-0001 US TOLL
    - Access code: 194 367 225

Thursday (6/16)

- 10am ET: TSC Meeting (hosted by TSC/Chris Ferris)
  - Join GoToMeeting
    - United States (toll-free): 1 877 309 2070
    - United States: +1 (312) 757-3119
    - Access Code: 613-310-429
- 6pm ET: 30 min check-in with group
  - Join WebEx
    - +1-415-655-0001 US TOLL

        - Access code: 194 540 465

Friday (6/17)

- 12pm ET Architecture WG Meeting
    - Join WebEx
        - Meeting number: 203 730 029
        - Meeting password: ntrMWDE7 (68769337 -- On Phone)
        - +1-408-525-6800 Call-in toll number (US/Canada)
        - +1-866-432-9903 Call-in toll-free number (US/Canada)
        - Access code: 203 730 029
- 2pm ET -- Regroup and recap
    - Join WebEx
        - +1-415-655-0001 US TOLL
        - Access code: 197 164 974

Tools

- Slack (if you need an invite, please click here)
    - https://hyperledgerproject.slack.com/messages/fabric-test-hackathon/
    - https://hyperledgerproject.slack.com/messages/sawtooth-hackathon/search/sawtooth/
- IRC #hyperledger on freenode.net
- Etherpad - http://etherpad.org/
- Screenhero - https://screenhero.com/
- WebEx (tbenzies@linuxfoundation.org can set up for any group meetings people want to host)
- Google Hangouts
- Skype
- Mailing Lists

Proposed Topics

- Transaction Family Tutorial posted in relation to the Sawtooth Projects below
  http://intelledger.github.io/txn_family_tutorial.html
- Fabric Bug Buster:
  https://github.com/hyperledger/fabric/issues?q=is%3Aissue+is%3Aopen+label%3A%22help+wanted%22
- Fabric Test Coverage: Using Go test, Behave and/or Ginkgo to write tests
  For detailed Membership Services (CA) server tests, pick up issues from this master issue
- Fabric channel for hackathon: Go to slack #fabric-test-hackathon
- Creating a Participant transaction family would be a good use of time at the upcoming hackathon. Sawtooth employs a concept of “Transaction Families” for composing business logic. The Marketplace Transaction Family is an example transaction family capable of supporting a variety of trading usages. Included in Marketplace is a Participant object. We’ve found there’s a common need for a Participant concept in most transaction families (including the arcade examples and in txn families others have created). The goal of this hackathon project would be to carve off the participant from Marketplace and generalize it to be included by other transaction families in lieu of copying it directly into each txn family. Here’s a link to the code we can pull out of marketplace and generalize.
  https://github.com/hyperledger/sawtooth-mktplace/blob/master/mktplace/transactions/participant_update.py
- SawtoothLakeJs: Port SawtoothLake to nodejs

- Sawtooth Lake Arcade is a collection of games written to demonstrate different features of the Sawtooth Lake platform. It is available at: https://github.com/hyperledger/sawtooth-arcade. The first game, Sawtooth Tac Toe (sawtooth_xo), was written to facilitate an upcoming transaction family tutorial we plan to add to the developer's guide. It should be a good starting place for anyone writing a new transaction family. The second game, Sawtooth Ethereum Guess, was written during our last Hyperledger F2F to demonstrate how to use an external service (in this case, Ethereum's API) as a database within a transaction family. I'd like to encourage community contributions to the arcade. I think it is a great place to start learning about Sawtooth Lake, and additions of unique games will serve as examples for development of more complex real-world transaction families and clients. A few ideas:
    - Rock-paper-scissors. This game could demonstrate how to solve problems where two untrusted participants play a game with fairness enforced.
    - Battleship. This game could demonstrate secrecy of the game boards for each players. The boards are slowly revealed to the other player.
    - Mafia/Werewolf. This game could demonstrate participant anonymity on some transactions.

Battleship game

How to participate?

- Slack channel: https://hyperledgerproject.slack.com/archives/sawtooth-hackathon
- Branch added: https://github.com/hyperledger/sawtooth-arcade/tree/battleship

Section 1 - Intro

Overview of the Business Problem or Opportunity

This game could demonstrate secrecy of the game boards for each player. The boards are slowly revealed to the other player.

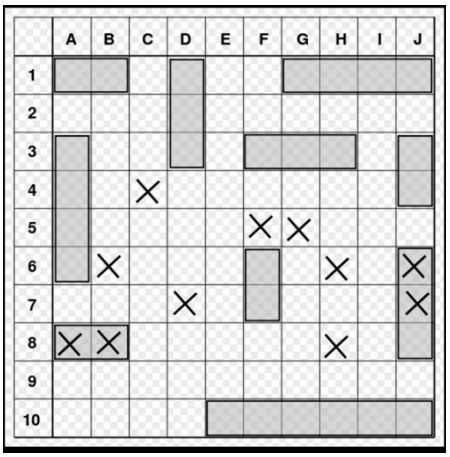
Section 2 - User Stories and Requirements

- Technicalities
    - Sawtooth Tutorial
      https://intelledger.github.io/tutorial.html
    - **Transaction Family Tutorial**
      https://intelledger.github.io/txn_family_tutorial.html
    - **Source code on Git:**
      https://github.com/hyperledger/sawtooth-arcade/tree/battleship
    - **TL;DR on Setting up your development area:**

        - You need 4 repositories from git:
            - sawtooth-core, sawtooth-dev-tools, sawtooth-validator, and sawtooth-arcade
            - From sawtooth-arcade, clone the “battleship” branch . This is the branch we will be iterating over during this hackathon.
    - **Submitting a Pull Request (PR):**
        - Fork your own copy of https://github.com/hyperledger/sawtooth-arcade/tree/battleship (you do this from the web UI from github - just press the fork button -- see upper right corner in github page)
        - On your development system, clone the fork you just created (e.g. git clone <url>) The url can be copied from the github page … find the clone button. URL will look like this:git@github.com:<your-github-id>/sawtooth-arcade.git
            - *git clone git@github.com:joe-hackathon/sawtooth-arcade.git*
        - get into the battleship branch
            - *git checkout battleship*
        - Create a feature branch locally:
            - *git checkout -b my_bs_feature*
        - Make your changes and commit locally
            - *git commit -a -m “added a new ship”*
        - To submit the pull request, just
            - *git push -u origin my_bs_feature*
        - If you refresh your github web page there will be a button to submit the Pull Request.
        - This method is called a “Forking Workflow” - you can read an article about it here : https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
        -

    - **Hangout:** https://hangouts.google.com/call/b52rygf7orapdcotq53d7pzsbae
    - **Current document:** https://docs.google.com/document/d/1ROq44KpxuelkOp8GpceFNcv8K6XfWdaOnknzBk73OMc/edit?pref=2&pli=1
- Background
    - Rules of the game: http://www.hasbro.com/common/instruct/Battleship.PDF
    - a paper and pencil version of the game : https://upload.wikimedia.org/wikipedia/commons/e/e4/Battleships_Paper_Game.svg
    - Tic-tac-toe

        - for sawtooth_xo (tic-tac-toe), we inferred player1 and player2 from the initial two actions (players just identified by their signing key), which we could also do here.
- Business requirements
    - Create_game
    - add_board (each player adds their board state)
    - fire/~~guess~~
        - Might be able to combine fire & reveal in the same transaction. In order to fire you must reveal whether opponent's fire was a hit. Just need to handle initial case on first fire.
    - reveal_space



- Technical requirements
    - Gif of the boardgame: https://hyperledgerproject.slack.com/files/mack/F1H5SQ761/pasted_image_at_2016_06_15_09_50_am.png
    - to encrypt each space on the game board with a unique key. as turns are taken, the private key for each space is revealed by the player.
    - there are some potentially hard problems around cheating (similar to rock-paper-scissors), but if we ignore those edge cases initially we can probably get the framework for the game in place fairly quickly
    - rather than a separate key for each board space, couldn't the user assign a couple values to each space, (ship_present, nonce) and commit that with a hash and then when challenged on the space, could reveal whether ship was present or not, by revealing (ship_present, nonce) that matches the hash commit
        - or perhaps, just a hash commitment could be brute forced depending on the nonce range
        - Proposal : store the nonce rather than a key
        - so maybe you could just sign the (ship_present, nonce), and then you could provide the message that was signed and the other user could verify?
        - Eventually, blockchain stores encrypted committed positions, player stores key/nonce required to reveal committed data

- Data model
    - Global State = {Game {Board-Player1, Board-Player2}, ???}
    - Local (Client/Wallet) State = {MyBoard}
    - Board
        - Decide on a linear array like in tic tac toe or a 2D array to represent the grid naturally.
        - Blockchain will record hits / misses, unknown
            - Initial development will also store Boat on blockchain but eventually this will be hidden and stored only on client state
        - Potential linear structure example for 5x5 board (actual board 10x10)
            - -M--- HBBB- -M--B ---MB ----B
        - Representing graphical board:

unknown	Miss	unknown	unknown	unknown
HIT	Boat	Boat	Boat	unknown
unknown	Miss	unknown	unknown	Boat
unknown	unknown	unknown	Miss	Boat
unknown	unknown	unknown	unknown	Boat

    - LocalBoard (MyBoard) = Boat locations
- Board Data Structures
    - Random thoughts here
    - Each player needs to submit their 10x10 board, with the ship types during the PHASE-SETUP portion of the game. These boards should only be visible to the player who owns their board. This is currently mocked up in the code on ablek's code.
    - Based on: https://docs.google.com/presentation/d/1bKMUQbes6LbTNtTEK7eI-OTA3_hCCOEk0EZ7V1yKnmg/edit#slide=id.p
      We will have a total of 4 boards to store.
        - Each player's client submits an encrypted version of their board/ship locations to the ledger. So two boards on the ledger.
        - Each player maintains a local copy of their board.
        - As each turn progresses, the targeted cells are revealed as either a hit (H) or miss (M).

    - Shawn noted that there may be limitations on which data structures will be serialized using the key/value store. is a hard limit on the size and type of data structures which can be put into the transaction. TODO: find out what these are. For Tic-Tac-Toe, the list[] was suitable. But there may be some upper limit

maybe bounded by the UDP packet. Mic should be able to be authoritative on this.
    -
    -
- CLI
- Ship Numbers and Types (can use for marking which ships occupy which space on boards):
    - 1 x A - Aircraft Carrier (length = 5)
    - 1 x B - Battleship (length = 4)
    - 1 x C - Cruiser (length = 3)
    - 2 x D - Destroyer (length = 2)
    - 2 x S - Submarine (length = 3)
- Board Space Encodings
    - -: Unknown
    - H: Hit
    - M: Miss
    - {A,B,C,D,S}: Ship / type of ship

- Abdelkrim Mission: <DONE>
    - edit txn_family.py
        - In __init__ add Name, Action as in Xo
        - in check_valid(..) add handling for create
- Craig Mission: <DONE>
    - Delete reveal(); add comment to fire() that it should provide reveal functionality.
    - Submit PR as above.
- apply() pseudo-code:
    - Input transactions:
        - CREATE, name
        - JOIN, name, board
        - FIRE, name, board, column, row
    - If transaction is CREATE, then update state with new board
        - game = {state: 'NEW'}
        - store[name] = game
    - If transaction is ADD_BOARD, then add the new board to the game; if it is the first board, set player1; if it is the second board, set player2
        - …
    - If transaction is FIRE…
        -
        - Check State of board
            - If board if "game over" then end

(should we check state of square? If already fired on?)

Board concept for client and ledger states:
https://docs.google.com/presentation/d/1bKMUQbes6LbTNtTEK7eI-OTA3_hCCOEk0EZ7V1yKnmg/edit?usp=sharing