CS449 Sprint 2 Report

Michael Cu, Elias Julian Marko Garcia, Samuel Lim

Team **Misael's** Project Submission

November 3, 2019

Contents

1 Micro Charter

1.1 Project Name

N Men Morris

1.2 Vision Statement

Create a extensible framework for board game web apps with scalability and performance.

1.3 Mission Statement

To play Nine Men's Morris on the web browser using a composable web technology stack that allows for future modularity while not foregoing performance.

1.4 Elevator Pitch and Business Value

We are creating Nine Men's Morris on a board game framework using Express.js and Neon for Rust. This allows for a data and type safe application that is capable of composability, scalability, extensibility, and performance.

1.5 Customers and Users

- Customers: Entrepreneurs and ventures that want to deploy board games on the web with low overhead, latency, and maintenance.
- Users: Individuals who are passionate about board games and want a new online experience that they can take and play wherever they go with their friends.

1.6 Metrics

By benchmarking N Men Morris, we can compare our solution to other products on the market on:

1. latency
2. binary size
3. up-time

1.7 Milestones

1. First MVP
2. First Offline N Men's Morris
3. Player versus Player (Offline)
4. Player versus Player (Online)

1.8 Risks

1. Inherent complexity of technology stack.
2. Inability to cooperate with teammates.
3. Plausibility of orphaning project due to development team size.

1.9 Authors

- Michael Cu
- Elias Julian Marko Garcia
- Samuel Lim

2 User Stories

Below you will find a table that makes up our "User Story Board", with some simplifications taken with respect to the total contents of the board. With respect to the final formal documentation, i.e. this paper, we only keep the basic qualitative and quantitative values for each story in the table while giving each user story proper its own section. This makes documenting each story less unruly while also easier to read. Each Story I.D. (SID) value is internally linked to its respective story, which also helps with navigating this section.

SID	Story Name	Priority	Time Est. (hr)	Actual (hr)	Status	Developer(s)
S1	Default Board Layout	high	10	4	DONE	Sam, Michael, Elias
S2	Mills Board Coordinate System	high	10	4	DONE	Sam
S3	Player Selection	medium	10	-	TODO	-
S4	Piece Assignment	medium	10	-	TODO	-
S5	Game Menu Prompt	high	10	2	TODO	Sam, Michael
S6	Start Game Prompt	medium	10	-	TODO	-
S7	Player Turn Assignment	medium	10	-	TODO	-
S8	Position Selection	medium	10	-	TODO	-
S9	Position Placement	medium	10	-	TODO	-
S10	Position Movement	medium	10	-	TODO	-
S11	Elimination Movement	medium	10	-	TODO	-
S12	Mill Definition	medium	10	-	TODO	-
S13	Mill Attack Attribute	medium	10	-	TODO	-
S14	Mill Defense Attribute	medium	10	-	TODO	-
S15	Mill Piece Movement	medium	10	-	TODO	-
S16	Elimination with Mills	medium	10	-	TODO	-
S17	Flying Definition	medium	10	-	TODO	-
S18	Flying Piece Ability	medium	10	-	TODO	-
S19	End Game: Loss	medium	10	-	TODO	-
S20	End Game: Win	medium	10	-	TODO	-
S21	End Game: Tie	medium	10	-	TODO	-
S22	Reset Board	medium	10	-	TODO	-
S23	Start New Game	medium	10	-	TODO	-

2.1 Default Board Layout

Description

As a user, I need a game board with 4 expanded squares, each with 8 equidistant positions, to play a game of Nine Men's Morris.

2.2 Mills Board Coordinate System

Description

As a user, I need a way to navigate and read the board to play a game of Nine Men's Morris.

2.3 Player Selection

Description

As a user, I want to choose a distinct color for my player.

2.4 Piece Assignment

Description

As a user, I want to receive 9 distinct pieces to place on the board.

2.5 Game Menu Prompt

Description

As a user, I am prompted with a GUI that shows the game board along with menu items.

2.6 Start Game Prompt

Description

As a user, I need a GUI to prompt me with the options to start a game with either another human or against the computer.

2.7 Player Turn Assignment

Description

As a user, I want to receive either the first or second player's move at the beginning of the game.

2.8 Position Selection

Description

As a user, I need to be able to select an empty position to choose it for piece placement.

2.9 Position Placement

Description

As a user, I need to be able to place a piece on an empty position to finish my turn.

2.10 Position Movement

Description

As a user, I want to be able to move my pieces to unoccupied positions.

2.11 Elimination Movement

Description

As a user, I want to be able to move my pieces into enemy positions should I qualify.

2.12 Mill Definition

Description

As a user, I need the game to recognize when three of my pieces are placed in adjacent positions in order to form a mill.

2.13 Mill Attack Attribute

Description

As a user, I need the ability to eliminate an enemy piece to attack as a player.

2.14 Mill Defense Attribute

Description

As a user, I need pieces within a recognized mill to be immune from elimination to defend as a player.

2.15 Mill Piece Movement

Description

As a user, I want to move pieces that make up mills into any position not occupied by one of my other pieces.

2.16 Elimination With Mills

Description

As a user, I want my pieces to remove opponent pieces from the board individually.

2.17 Flying Definition

Description

As a user, I need the game to recognize when I have less than 4 pieces to gain the ability to "fly" my pieces.

2.18 Flying Piece Ability

Description

As a user, I need the ability to move a piece to any empty position on the map when I only have 3 pieces to fly.

2.19 End Game: Loss

Description

As a user, the game must recognize when I reach less than 3 pieces to declare me a loser.

2.20 End Game: Win

Description

As a user, the game must recognize when my opponent reaches less than 3 pieces to declare me the winner.

2.21 End Game: Tie

Description

As a user, the game must recognize when the board state has repeated the same layout N times after a player has reached less than 4 players in order to declare the game a tie ("Remis").

2.22 Reset Board

Description

As a user, I want to be able to restart a game with a new, empty board.

2.23 Start New Game

Description

As a user, I want to be able to start a new game with the default configuration as before I started the game.

3 Acceptance Criteria

The following section covers the acceptance criteria enumerated in response to the User Stories discovered and documented in §2. In a similar fashion to §2, the table documenting these acceptance criteria is in a simplified form. Every Acceptance Criterion has an Acceptance Criterion ID (`ACID`), which is associated in the table below with its respective `SID`, development status, and the developers responsible for implementing it. Each `ACID` is linked to its respective subsection below for viewing the description of each criterion.

SID & Name	ACID	Status	Developer(s)
S1 Default Board Layout	A1	DONE	Sam, Elias, Michael
S2 Mills Board Coordinate System	A2	DONE	Sam, Elias, Michael
S3 Player Selection	A3	TODO	-
S4 Piece Assignment	A4	TODO	-
S5 Game Menu Prompt	A5	DONE	Sam, Michael
S6 Start Game Prompt	A6	TODO	-
S7 Player Turn Assignment	A7	TODO	-
S8 Position Selection	A8	TODO	-
S9 Position Placement	A9	TODO	-
S10 Position Movement	A10	TODO	-
S11 Elimination Movement	A11	TODO	-
S12 Mill Definition	A12	TODO	-
S13 Mill Attack Attribute	A13	TODO	-
S14 Mill Defense Attribute	A14	TODO	-
S15 Mill Piece Movement	A15	TODO	-
S16 Elimination with Mills	A16	TODO	-
S17 Flying Definition	A17	TODO	-
S18 Flying Piece Ability	A18	TODO	-
S19 End Game: Loss	A19	TODO	-
S20 End Game: Win	A20	TODO	-
S21 End Game: Tie	A21	TODO	-
S22 Reset Board	A22	TODO	-
S23 Start New Game	A23	TODO	-

3.1 Criterion 1

ACID	Description
1.0	Given a User...
1.1	When the board appears, it will render the default layout for Nine Men's Morris
1.2	When the User does not visit our site (IP), the board will not appear.

3.2 Criterion 2

ACID	Description
2.0	Given a User...
2.1	When the board is rendered, it will include a coordinate system along the axis.
2.2	When the board is not rendered, there will be no coordinate system.

3.3 Criterion 3

ACID	Description
3.0	Given a user...
3.1	When a user chooses one of two colors, then their player pieces will be the chosen color.
3.2	When a user chooses one of the two colors, and it is already taken, then they will not be the chosen color.
3.3	When a user does not choose one of the two colors, then they will not be assigned a color.

3.4 Criterion 4

ACID	Description
4.0	Given a user...
4.1	When users chooses one of two players, then the users will be assigned N pieces of their color.
4.2	When a user does not choose one of the two players, then the user will not be assigned N pieces of their color.

3.5 Criterion 5

ACID	Description
5.0	Given a User...
5.1	When the website is loaded, it will show the game along with a menu buttons.
5.2	When the website is not loaded, it will not show the game or the game menu system..

3.6 Criterion 6

ACID	Description
6.0	Given a User...
6.1	When the menu options are rendered, there is a button that displays start game.
6.2	When the menu options are not rendered, it will not show a button to start the game.

3.7 Criterion 7

ACID	Description
7.0	Given a user...
7.1	When a user chooses which turn to play, then a user will recieve the corresponding turn.
7.2	When a user does not choose a turn to play, then a user will not receive the corresponding turn.

3.8 Criterion 8

ACID	Description
8.0	Given a User...
8.1	When it is my turn, I can click on a empty board position.
8.2	When it is not my turn, I cannot click on an empty board position.

3.9 Criterion 9

ACID	Description
9.0	Given a User...
9.1	When it is my turn, I can place a piece on an empty position I have selected.
9.2	When it is not my turn, I cannot place a piece on an empty position.

3.10 Criterion 10

ACID	Description
10.0	Given a user...
10.1	When a user moves a piece to an unoccupied position, then the user's piece assumes the new position.
10.2	When a user moves a piece to an occupied position of their own, then the user's piece is not moved to the new position.

3.11 Criterion 11

ACID	Description
11.0	Given a user...
11.1	When a user moves their piece into an enemy position, then the user's move will not qualify.

3.12 Criterion 12

ACID	Description
12.0	Given a User...
12.1	When I place three pieces adjacent to each other, the game recognizes it as a mill.
12.2	When I do not place a piece that forms three adjacent occupied positions, it is not recognized as a mill.

3.13 Criterion 13

ACID	Description
13.0	Given a User...
13.1	When it is my turn, and I have a mill formed, then I have the ability to eliminate an opponent's piece.
13.2	When it is my turn, and I do not have a mill formed, then I do not have the ability to attack.
13.3	When it is not my turn, and I have a mill formed, then I do not have the ability to attack.
13.4	When it is not my turn, and I do not have a mill formed, then I do not have the ability to attack.

3.14 Criterion 14

ACID	Description
14.0	Given a User...
14.1	When it is my turn, and I have a mill formed, then the pieces in my mill are defended from elimination.
14.2	When it is my turn, and I do not have a mill formed, then my pieces are not defended from elimination.
14.3	When it is not my turn, and I have a mill formed, then the pieces in my mill are defended from elimination.
14.4	When it is not my turn, and I do not have a mill formed, then my pieces are not defended from elimination.

3.15 Criterion 15

ACID	Description
15.0	Given a User...
15.1	When a user selects a piece in a mill to move to an open position, then the piece will be moved to that new position outside of the mill
15.2	When a user selects a piece in a mill to move to an invalid position, then the piece will not be moved.

3.16 Criterion 16

ACID	Description
16.0	Given a User...
16.1	When a user removes opponent pieces from the board, then the opponent's piece will no longer appear on the board.
16.2	When a user removes their own piece from the board, then the piece will not be removed from the board.

3.17 Criterion 17

ACID	Description
17.0	Given a User...
17.1	When it is my turn, and I only have three pieces, then I gain the ability to "fly" across the board.
17.2	When it is my turn, and I have more than three pieces, then I do not gain the ability to "fly" across the board.
17.3	When it is not my turn, and I only have three pieces, then I do not gain the ability to "fly" across the board.
17.4	When it is not my turn, and I have more than three pieces, then I do not gain the ability to "fly" across the board.

3.18 Criterion 18

ACID	Description
18.0	Given a User...
18.1	When it is my turn, and I only have three pieces, then I can "fly" a piece I own to any open position on the board.
18.2	When it is my turn, and I have more than three pieces, then I can't "fly" a piece I own to any open position.
18.3	When it is not my turn, and I only have three pieces, then I can't "fly" my piece across the board.
18.4	When it is not my turn, and I have more than three pieces, then I can't "fly" my piece across the board.

3.19 Criterion 19

ACID	Description
19.0	Given a User...
19.1	When I am reduced to less than three pieces, the game must declare me the loser and end the game.
19.2	When I am not reduced to less than three pieces, then the game does not declare me the loser nor end the game.

3.20 Criterion 20

ACID	Description
20.0	Given a User...
20.1	When I reduce my opponent to less than three pieces, the game must declare me the winner and end the game.
20.2	When I do not reduced my opponent to less than three pieces, then the game does not declare me the winner nor end the game.

3.21 Criterion 21

ACID	Description
21.0	Given a User...
21.1	When neither my opponent and I are reduced to less than three pieces and we repeat the same board arrangement more than N times, then the game is declared a "Remi", e.g. a tie, and the game is ended.
21.2	When neither my opponent and I are reduced to less than three pieces and we do not repeat the same board arrangement more than N times, then the game is not declared "Remi" and is not ended.

3.22 Criterion 22

ACID	Description
22.0	Given a User...
22.1	When a user restarts the game, then the board will restart with an empty board.
22.2	When a user does not restart the game, then the board will retain the current layout it contains.

3.23 Criterion 23

ACID	Description
22.0	Given a User...
22.1	When a user starts a new game, then a user's default configuration will be used when a new game is started.
22.2	When a user does not start a new game, then the configuration will remain unchanged.

4 Code Review

4.1 Checklist

Checklist	Item	Findings
Coding Standards		
	Naming Conventions	All naming conventions link directly to their use case and structs have compact naming systems relating to their purpose or game identity. This relates heavily to game component system design patter, i.e. ECS
	Argument ordering	Rust's formal parameter argument ordering is strict, i.e. the order of arguments cannot vary between calls for the same method. For JavaScript equivalent, same restrictions apply due to conversion.
	Comments	Commenting could be improved by adding clarity in areas that handle pattern matching on handling of board inputs and conversions. No verbose comments and existing comments are well placed.
	Code Style	Rustc comes with a component called Rustfmt that automatically formats code upon save to keep a consistent style across the codebase inline with Rust code formatting standards. Javascript is compliant with ES6 Lint.
	Indentation	See above for answer. Both handled by linters provided.
Design Principles		

Continued on next page

Continued from previous page

Checklist	Item	Findings
	Abstraction and Interfaces	Due to ECS, each component has high cohesion and loose coupling natively. Each entity as defined in the rust module is single purpose, and is incrementally composed from smaller entities defined for the game. Js mimics this, with abstraction mostly occurring in the module. Browser interactions make for event based components.
	Proper Encapsulation	JS x Rust interop makes for high modularity within types and their methods. Only that which is necessary to be exposed publicly is exposed as such, and FFI wrappings enforce a minimally exposed API to the server for handling game logic.
	Command Query Separation Principal	JS calls to rust module's methods, which in turn have a minimal API exposed through Manager. Manager, in turn, only have mutators or accessors for its private GameState object. These call to GameState itself through basic getters and setters.
	Design by Contract	The strict typing over the possible game states representable allow for exhaustive matching over the game variants. The JS poll's input objects are strict to their parameters passed to Rust.
	Reasonable pre and post conditions	Yes. The typing models the domain space so we can keep track of pre/post conditions at an abstract level.
	Open Closed Principal	JS can directly inherit class models from Rust, extending their functionality while not disturbing native code and implementation logic. This allows for modification of input logic while the game logic remains consistent and unaffected by irrelevant mutation.

Continued on next page

Continued from previous page

Checklist	Item	Findings
	Single Responsibility Principal	Due to our usage of the ECS design model/pattern, our classes, structure, and types, both in JS and Rust, are highly cohesive yet maintain individual responsibility for their logic.
Code Smells		
	Magic Numbers	None.
	Unnecessary Globals/Class vars	None.
	Duplicate Code	Due to the nature of entity component systems (ECS), which our program uses as a design pattern for the game, there are no instance of duplication, only interactions with types.
	Long Methods	None.
	Long parameter list	None, due to ECS.
	Over-complex expression	None.
	Unnecessary Branching	None, where branching occurs through match statements, it is exhaustive of the game state without being enumerable.
	Bad method/variable naming	None, see Coding Standards: Naming Conventions.
	Similar methods in other classes	None, due to ECS.

4.2 Bugs Discovered

Bugs	Status	buggy code snippet	bug summary	bug logic
#1	Fixed	calcPlayer(1)	The board text displays the player who last played as the current player.	Is nullified by the piecesLeft decrement that precedes it, resulting in referring to the previous colour as the new colour.
#2	Fixed	pub fn poll	Manager's hidden gamestate moves instead of copy/clone when using basic accessors.	Ownership defined for accessors in GameState resulted in unnecessary moves of Manager's hidden Gamestate

5 Implementation Tasks

This section summarizes the details of implementation tasks for the project.
You will find in each subsection a table similar to those found in §2 and §3.

5.1 Summary of Production Code

SID & Name	ACID	Class Name(s)	Status
S3 Player Selection	A3	Window, Board	Done
S1, S8, S9 Default Layout and Positions	A1, A8, A9	Manager	Done
S2 Coordinate System	A2	Coord	Done
S3, S4, S8, S9, S10	A3, A4, A8, A9	Board (Rust)	Done

5.1.1 Class `Window, Board`

Method	Notes
1. `eventPress` 2. `at`	These functions relate to a pseudo-epic, and thus the testing will be generic.

5.1.2 Class `Manager`

Method	Notes
1. `poll`	Called by JS front-end to query, check, and get the next game state.
2. `get_board`	Returns the current game board layout..

5.1.3 Class `Coord`

Method	Notes
1. `as_string`	String representation of `Coord` value.

5.1.4 Class `Board` (Rust)

Method	Notes
1. `len`	Gets the length of the board.

5.2 Automated Test Code

There were no automated tests for this sprint.

SID & Name	ACID	Class Name(s)	Method Name(s)	Description	Status	Developer

5.3 Manual Test Code

All manual tests before M7 are from the previous sprint and no longer accurately link to their issues earlier in the documentation due to refinement changes.

All missing tests are to still be found within the source code provided with the Sprint 2 Submission. Lack of documentation in this artifact is not lack of existence

SID & Name	ACID	MTID	Status	Developer(s)
S2 User Input and Selection	A2.1	M1	DONE	Samuel, Michael
S2 User Input and Selection	A2.2	M2	DONE	Samuel, Michael
S5 Piece Placement	A5.1.1	M3	DONE	Samuel, Michael
S5 Piece Placement	A5.1.2	M4	DONE	Samuel, Michael
S5 Piece Placement	A5.2.1	M5	DONE	Samuel, Michael
S5 Piece Placement	A5.2.2	M6	DONE	Samuel, Michael
S3 Player Selection	A3	M7	DONE	Samuel, Michael
S4 Piece Assignment	A4	M8	DONE	Samuel, Michael
S7 Player Turn Assignment	A7	M9	DONE	Samuel, Michael
S10 Position Movement	A10	M10	DONE	Samuel, Michael

5.3.1 Manual Test 1

Test Input	Test Oracle	Notes
document .getElementById("A1") .onclick	function onclick()	Checks if element clickable.

5.3.2 Manual Test 2

Test Input	Test Oracle	Notes
document .getElementById("container") .onclick	"undefined"	Checks if element clickable.

5.3.3 Manual Test 3

Test Input	Test Oracle	Notes
"A1"	elem.style.backgroundColor !== undefined	This is a GUI test. GUI will show piece placed in bottom left corner.

5.3.4 Manual Test 4

Test Input	Test Oracle	Notes
"A1", "A1"	"elem.style.backgroundColor = previousColor"	This is a GUI test. GUI will show piece placed in bottom left corner.

5.3.5 Manual Test 5

Test Input	Test Oracle	Notes
"D6"	"board = previousBoard"	This is a GUI test. GUI will show piece placed in bottom left corner.

5.3.6 Manual Test 6

Test Input	Test Oracle	Notes
"A1"	"board = previousBoard"	This is a GUI test. GUI will show piece placed in bottom left corner.

5.3.7 Manual Test 7

language=js,label= ,caption= ,captionpos=b,numbers=none document.getElementsByClass("turn-button") [...].includes(pieceElement) Choose the turn order

5.3.8 Manual Test 8

language=js,label= ,caption= ,captionpos=b,numbers=none document.getElementById('E4') (setupFullBoard() move('E4', 'F4')) document.getElementById('F4').style.backgroundColor === 'blue'

5.3.9 Manual Test 8

language=js,label= ,caption= ,captionpos=b,numbers=none document.getElementById('E4') (setupFullBoard() formMill('red') move('E4', 'F4')) document.getElementById('F4').style.background === 'red'

5.3.10 Manual Test 9

5.4 Other Manual Test Code

There were no other manual tests for this sprint.

ID	Test Input	Expected Result	Class Name	Method Name of Test	Status	Developer

6 Meeting Minutes

6.1 Meeting 2019.09.04

- Duration: 1 Hour

- Location: Miller Nichols Library

Agenda

- going over project pdf as group
  - discussing tech stack
  - going over sprint assignments
  - going over normal assignments
- discussing the actual structure of sprint 1
  - requirements
  - user stories
  - what submission might look like
  - discussion of who gets to do what
  - discussion of when to meet, general availability
    - Sam will be gone from 9th through 19th
    - Elias will be gone through the 12th - 14th

project 1 report

- want to get scrum documentation done
- get general idea down by end of this friday (2019.09.06)
  - structure of the project
  - how to use the frameworks/libraries involved (personal research/reading per individual)
    - Neon for rust
    - node.js
    - potentially express.js
- generating cards, user stories

6.2 Meeting 2019.09.06

- Duration: 1 Hour
- Location: Miller Nichols Library

Agenda

- discussing game rules
- discussing/writing user stories
- discussing tooling
- discussing design

Game Rules

- watched a video demo'ing the game
- discussed/clarify mechanics
  - whether or not to include coin flip
  - terms of loss
  - flying mechanic

User stories

- elias wrote user stories in a new org mode file called kanban.org on the repository
- discussed problem of documentation given requirements from the pdf for sprint 1
- discussed alternative means of documenting, carrying out execution of our cards for the project

Tooling & Design

- did not achieve agenda, did not get to these topics because of the time it took to discuss our epics/user stories.

TODO

- ☐ discuss tooling
  - need to finalize what our stack will look like and frameworks to be used.
  - elias has experimented with Neon and reports that it works well, seems viable for the product.

- ☐ discuss design
  - need to discuss how the actual product will be packaged and its architecture.

6.3 Meeting 2019.09.27

- Duration: 1 Hour, 30 minutes
- Location: Miller Nichols Library

Agenda

- discuss project structure
- acceptance criteria
- work assignment
- remaining TODOs

Project Structure

- express.js has a lot of dependencies, only really need connect.js
  - might try just using connect.js, which would be a lot simpler
  - will continue with using Neon
- board
  - gui
    - js renders the fontend
    - logic/data is all handled on back
  - data structure/representation
    - two choices:
      1. one big board object that includes methods for both resolving where players are **and** where things like mills are
      2. two object entities, one is purely for the GUI (tracking positions on the board), the other would be some kind of graph structure that allows position nodes to check peers for occupation and whether it is the same or opposing players

- movement and move validity
    - need to track flying
        - proposition: flying is a universal property, merely constrained until player count is reduced.
        - need some kind of getter/setter between board and entity management system
        - mill detection
        - if going with entity system, would merely be a graph traversal from any given node
        - another idea: create a mill entity system that tracks active mills and checks each mill upon each turn(?) and modifies or destroys the mill as necessary.
        - could save a lot of checking
        - as for organization/logical membership, would keep such a mill entity system independent of other objects in the system for simplicity, at least for now.
        - Checking for attack
        - if a mill entity system is used, we natively have a means to detect valid attacks. so long as the node is not in one of the mills, do not attack **unless** all available nodes are in mills.
- game driver
    - Will have some kind of Game entity/manager object that drives the game event loop.
        - will take inputs from players, run them as game moves
        - however, internal logic to the entity management system is what will ultimately validate moves
        - game manager will have no logic for why this happens, only passes back and for game inputs and the results of moves.
        - consequentially, need to codify where and how game validation logic happens
- validation logic
    - as of now, think it will be handled by the main entity management system

    * will have a set of logic checking methods defined over the system that verify whether a given move is allowed

Acceptance Criteria

- realized we need to add numbering to the board GUI (a-g, 1-7)
- (deferred, Sam will begin working on before next meeting)

Work Assignment

- elias will begin on exploratory work for the backend (board, entity management, etc)
- sam, michael will begin exploratory work for the frontend (GUI, communicating with backend)

TODO

- ☐ kanban board setup, finalization of workflow for documentation
  - can probably just use github for real time management, but keep organizational and notes in `kanban.org` file on the repo.
- ☐ defining test cases for stories and acceptance criteria
- ☐ refining stories
  - same case applies with above: refine stories, and put them on github's project management board accordingly; actual refinement can be delegated to within `kanban.org` file.

6.4 Meeting 2019.10.02

- Duration: 1 Hour, 40 Minutes
- Location: Miller Nichols Library

Agenda

- addressing tagged issues generated on GitHub
- settling on how front-end talks to back-end
- documentation/design stuff

Issues on GitHub

- issue #3: determine communication channel between js and rust
  - event polling seems overkill for what we need
  - even handler on front-end which speaks to an entity Manager-Glue, which will be the JS that talks to rust backend
    * There will be a manager in the back-end, which will generate game state, and return that to the front-end
    * back-end will also have triggers (flags? Enums?) which signal to front-end when certain actions are no longer needed or valid, i.e. button inputs or game state continuation
  - JSON seems like a good enough medium for message passing between front and back components
  - issue #4 is largely tagged to #3, so this resolves that
    * `State`: Input Handle + BoardStruct + Trigger)
    * `BoardState`
      · this is what gets sent back to the JS
      · 1D array of the `State` struct
      · this array will be handed off as a NeonJS object, whatever it's called in neon
      ·

- issue #6: Front-end/GUI Skeleton, Basic Design

  1. Neon builds a node module
  2. This is sent to express.js
     - accepts it as a bunch of js functions

3. Express takes this, as a bunch of objects, and then saves as strings to JS files, in turn statically served to end user (i.e. browser)
    - express.js interaction is a one-off affair
4. Stretch goal: being able to set different themes on the front-end

- issue #8: CI/CD
    - GitHub has native CI/CD now via it's Action's service.
    - can impl for both Rust and Node.js

TODO

- ☐ Design docs(?)
    - at least 3 needed:
        1. event diagram
        2. general UML diagram for total project
        3. class hierarchy/component diagram

6.5 Meeting 2019.10.03

- Discord: 1 Hour, 53 Minutes
- Location: Video Call

Agenda

- how to branch
- branching $\text{basic}_{\text{gui}}$
- GitHub PR format
- styling format

GitHub PR format

- Show Michael how to create a branch
- name the branch and pull from remote
- push the branch from local
- sync branches
- checkout a branch

Branching `basic_gui`

- created a branch `basic_gui`
- set up an issue with the branch for PR
- push a commit from local to remote branch

GitHub PR format

- went through how to form a PR from different branches
- how to further commit to the compare branch

Styling Format

- no bootstrap, no jquery
- setup proper layouts for the GUI
- discussed how we want to handle events onclick

TODO

- push scaffolds for the website GUI
- handle basic logic for pushing items to back-end storage
- create mock of Rust functionality in TypeScript for further discussion

6.6 Meeting 2019.10.05

- Duration: 1 Hour, 16 Minutes
- Location: Video Call

Agenda

- CSS Grid
- SASS
- TypeScript
- build script compilation and runtime
- proper layout for GUI

CSS Grid

- teach Michael about CSS Grid
- pure CSS, not bootstrap (Elias)
- use columns properly
- no need for floats / flexbox

SASS

- transpiler for CSS
- allows nested functionality
- separate compiled/uncompiled folders
- use `watch` script to sync changes

TypeScript

- better able to handle equivalence mocking to Rust
- easy to push onto browser
- separate folders (see above)
- push to public folder for site access

Build Script Compilation and Runtime

- use watch and start scripts to build site
- separate scripts will be run for Rust beforehand
- build scripts allow for synced changes between folders (see above)

Proper Layout for GUI

- use column areas in CSS Grid
- main column for game
- nested grid for board layout (tentative)
- proportion text for board side-by-side

TODO

- design docs
- microcharter
- mocking TS => Rust
- event keys on front-end (browser)

6.7 Meeting 2019.10.06

- Duration: 7 hours
- Location: Video Call

Agenda

- Tying up loose ends with respect to documentation and write up
- Tying up loose ends with respect to UI/JS end of the application
- Discussing what is left to do with the project

Documentation and Write-up

- Figured out how to format the tables given that many of the ones provided do not play well with latex/org-mode markdown
- Similarly, decided on how to interconnect documentation components between sections
- Discussed the remaining things left undocumented, particularly pair ratings.

UI/JS Loose Ends

- Complete manual testing of interacting elements
- Finalize positions of clickable elements on the board grid.
- Alternating player logic for placement of pieces.
- Limiting piece placement to nine.

Discussing Future Sprint/Direction of Project

- Current User Stories are pseudo-Epics and need to be refined into better User stories aside from S1. As they stand, discussing the current user stories makes for overly generic/abstract discussion and doesn't meaningfully translate into logic/behavior to implement and actual engineering tasks.
- Currently, the front end mocks all of the behavior/functionality that would otherwise be provided by the backend. In sprint 2, this is where the real meat of programming will come in as we learn to make the back-end and front-end interface, particularly with translating data types across the FFI boundary through Neon.
- We need to improve the current state documentation massively.
  - Design diagrams.
  - Docstrings across software code base.
  - Event diagrams.
- Translate the above issues into their proper documentation for the master documentation and write-up file
- How to test more of the functionality given that a major component of this application is running directly on the browser.

6.8 Meeting 2019.10.11

- Duration: 2 hour
- Location: Miller Nichols Library

Agenda

- review sprint 1
  - bugs
- roadmap sprint 2
  - outline what is needed
  - setup project board
  - issues
  - interface between front/end

Review of Sprint 1

- GUI
  - coordinate system
  - testing was not fun w/ current setup
- backend
  - types, not much else
- documentation
  - implemented a decent workflow for putting together and compiling a documentation artifact for submission
- complains
  - mostly just not having enough meetings despite getting 6 of them in before deadline.
    - had one right before submission, was clearly heavily crunch oriented.
    - documentation was a major pain point given the ratio of code/work to actual required documentation.

Sprint 2 Roadmap

- bugs
  - Wrong current player display
  - Bad formatting of org mode files on GH
    - won't fix bc final artifact pdf, that looks fine
  - Bad formatting of code inputs/outputs on manual tests within tables as currently.
- submission requirements of Sprint 2
  - full human player vs human player functionality
    - piece movement
    - piece elimination
    - mill formation
    - (optional) piece flying
    - end game
  - All proper documentation wrt to above new functionality and its implementation
    - User story refinements
      - side note: this is actually more sizeable then what it appears
    - AC and respective refinements
      - at least one manual **or** automatic
    - all test code and documentation
      - aside from manual/automatic requirements, all units of code should have a accompanying unit test.

- project board (backlog/requirement refinement)

    1. Front End -> Backend communication

        - JS speaks to rust

            * Struct will have a position struct and an event enum that dictates to backend what to do next.
            * let's call this object a `PushEventStruct`
            * This represents the attempted inputs/moves of the current player

    2. Backend -> Front end communication

        - backend runs actual logic and validation
        - upon receiving a `PushEventStruct`, the back-end:

            (a) Does a validation:

                * Checks whether requested action is valid in the current game state
                * When not valid, immediate returns:

                    i. Original `BoardStruct`

                    ii. `InputHandle` -> Error that rust handles, and returns "none" to JS

                    iii. `Trigger` dependent on game state transition

            (b) Game Manager handles event:

                * Processes game events and generates resulting new game state

            (c) Back-end returns new `State`, where:

                * `InputHandle` is `Success`

        * `BoardStruct` is the new `State`
        * `Trigger` of the condition of the phase is satisfied
            · `none`
            · `elimination`
            · `flying`
            · `gameover`
    3. Proper GUI styling
        – Add Game meta information
            * how many pieces left for each player
            * current phase (turn)
            * (optional/idea) GUI indicates existing mills
        – game menu
            * theme options
            * game options
            * how to play
        – mobile/responsive design

- snow plow form sprint 1
    1. design docs
    2. Development Workflow
        – CI/CD with GitHub Actions
            * CI
                · need front-end GH Action

            - need back-end GH Action
            - figure out action for integration between front/back components
    - Development environment documentation
        * How to setup local machine to build both the front and back-end

6.9 Meeting 2019.10.25

- Duration: 2 hours
- Location: Miller Nichols Library

Agenda

- new project format
- testing
    - server
    - gui
    - unit testing
- interop/ffi/interfaces
    - managerGlue
- general development/programming

Project Format

- rust projects using neon have a default layout
    - not friendly for testing the rust specific code
    - similarly, not friendly for testing the node/js/front-end code not involved with rust exported module
- solution: rust workspaces

  - simply change top level to have cargo file with workspace member declarations
  - well documented here: `https://github.com/neon-bindings/examples/tree/master/workspace`

- benefits:
  - can now run rust tests independent of node/js via traditional `cargo test` and `cargo check`
  - furthermore, now have backend code organized as:

```
.
 Cargo.lock
 Cargo.toml
 lib
ăă  index.js
 native
            artifacts.json
            base
           ăă  Cargo.toml
           ăă  src
            build.rs
            Cargo.toml
            index.node
            src
           ăă  lib.rs
            target
```

  - This allows us to move all logic into a sub-crate, `base`, which is private and internal to the backend. An API for the module that provides the backend functionality is the only thing exposed in the main `native/src/lib.rs` file, which is what is compiled and exported to node.
    - i.e. we keep our code enapsulated and enforce demeter's law
  - finally, means all integrations tests on node side only test the actual interopability logic and functionality rather than any of the rust specific code and logic.

Testing

- ibid project format for overall structure's effect on testing approach
- server (node):
  - no browser tests
  - check if all functions are translated properly
    - node functions get translated to plaintext js functions
  - request testing
    - extra assets (GET stuff)
    - further research/parking lot:
      - coop and multiple clients
- gui (client stuff):
  - test functions within test module:
    - master functions that assert all tests at instantiation
    - i.e. when browser pulls the plain js
    - pull from dev endpoint
  - manual testing:
    - automated testing on browser is pretty bad/gross
    - getting setup not really worth it, so opting for manual tests as necessary.
  - three layer testing (`eventPress`):
    - log input events from divs
    - validation testing on newly retrieved `BoardState`
- unit testing:

  - **reminder**: 1:1 ratio between functions and tests at minimum, 2:3 ideally
  - rust is pretty straight forward, built in as first class feature.
    * managerGlue
      · dedicated interop testing
      · mutation of state from rust
      · this is probably going to mocking

Interop/FFI/Interface

- rust -> neon -> index.node -> yields an array of exports, i.e. `register_module!`
- managerGlue
  - pulls rust functions into script source, i.e. require
  - completely server side
  - browser does not support at current level
    * node has a virtual dom representation that browsers simply don't support
    * this is why node gets trans to strings, saved as asset/js, and pulled into browser

TODO action items

- refine user stories generally
  - currently are epic-y, need to be more specific and decomposable
- Generate acceptance criterions from new refinements + issues discussed in today's meeting
  - add as items to issues/sprint 2 board on GH

6.10 Meeting 2019.11.01

- duration: 2 hours
- Location: Miller Nichols Library

Agenda

- updates on back-end
- discuss change of server structure
- code review
- refinement user stories
- refinement acceptance criteria
- matching browser to server requirements

Back-end Updates

- made a tracking issue on GH for all requirements of sprint2 with respect to back-end
- consists of multiple tasks derived from acceptance criterion and user story relevant to game board
  - #23: align back-end types with front-end types due to requirement changes (DONE)
  - #28: add file reader utility (DONE)
  - #24: exporting rust types to js equivs (Nearly done)
  - #25: importing js types to rust equivs (TODO)
  - #26: game validation logic (#)
- **NOTE about game movement**
  - No longer have PieceSwitch/PieceMove enums.
  - Instead, option will yield a player and a position
  - backend will know from gamestate whether this is a piece moving OUT or INTO a position
  - will check accordingly

Server Structure Changes

- Four changes coming out:
    1. break out static files from server
        - tightly coupled with server currently
        - as if local app
    2. adding 3 endpoints for:
        - logic
        - theme
        - setup: (reach stuff for sprint3)
    3. new structures and classes (issue #31)
        - client connection class
            * browser & networking validation
            * putting in place design necessary to allow program to be extended to client-client interactions
            * separation of game and web-server
        - onclick request (Browser side)
            * setup structure for binding between browser and server, 1:1
            * each event calls back to browser's main function for getRequest
                · send off payload to server, which sends to rust module export
        - browser needs its own board manager, separate from ManagerGlue

        * allows for state to be accepted on browser side
    - BoardReceptor (server)
        * scaffold for endpoint components
        * allows browser to hit endpoints/communicate with server

Code Review

- documented in sprint2$_{\text{writeup.org}}$

Matching Browser-Server Reqs

- Reference: server structure changes list item 3
- triggers are sent from server to browser
    - new components are entered on display
        * menu item
        * removal of input during player's turn, on opponent
        * game over
        * restart

TODO

- refinement (grooming) of US, AC, and tasks.

6.11 Meeting 2019.11.02

- Duration: 2 hour
- Location: Voice Chat

Agenda

- refactor server templating
- add endpoints
- revise UI

Refactor Server Templating

- board grid elements can be iterated
- colors can be templated
- tests can be separated
- scripts can be separated

Add Endpoints

- see last meeting (endpoint list)
- logic receives payload, returns native response
- BoardReceptor now implied, soon to be implemented (sprint #3)
- error component templates on bad payload

Revise UI

- Figma for design mockups
- Get menu bar up and running (Michael)
- Reconfigure game site layout
- information page?

TODO

- add final endpoints for talkback
- browser's 'Window'/'Board' needs HTTP adhesion to server
- design reconfiguration

6.12 Meeting 2019.11.03

- Duration: 7 hour
- Location: Voice Chat

agenda

- Blitzing remaining issues
- Blitzing documentation

Remaining Issues

- Mocking final game logic with typescript
- Adding tests
- Definition of Done: Hit all story points that have been listed thus far.

Blitzing Documentation

- Updating all tables
- Fixing formatting on code review tables
- Adding new meeting minutes

7 Team Ratings

Submission document does not specify scale, so it is assumed out of 5 with 1 being "Worst" and 5 being "Excellent".

	Elias Julian Marko Garcia	Michael Sy Cu	Samuel Chia Ern Lim
Elias Julian Marko Garcia	-	5	5
Michael Sy Cu	5	-	5
Samuel Chia Ern	5	5	-
Average	5	5	5