Sprint #2 Presentation

CS449

Samuel Lim, Michael Cu, Elias Garcia

2019.11.07

Outline

1. Sprint 2 Goals
2. Sprint 2 Results
3. Sprint 2 Lessons
4. Sprint 3 Goals

Sprint 2 Goals

Finish Front ⟷ Back Communication

- Sprint 1 left us with Rust defined types and a mocked front-end for rendering
- JS had no means of communicating to back-end, same goes for back-end to front-end.
- Neon library allows Rust to compile into actual node module that can be used by JS

Back-end Driven Logic

- Need front-end to take user input and back-end to compute all game logic, including:
  - Checking piece movement
  - Tracking mill formations and destructions
  - Attack validation

Polish Off Front-end

- Improving player piece coloring and position selection/movement
- Menu functionality
- Preparing for themes

Get ready for Web Sockets

- Separate web-server from game logic itself
- Drive game logic based on requests/responses over API endpoints

MVP product

- Be able to actually play Nine Men’s Morris!

Sprint 2 Results

Issues resolved i

The finally issues were finished:

- #20 Cargo Workspaces
- #23 align current rust module types with new requirements
- #24 exporting game types from rust to JS
- #23 Board Generation Utility

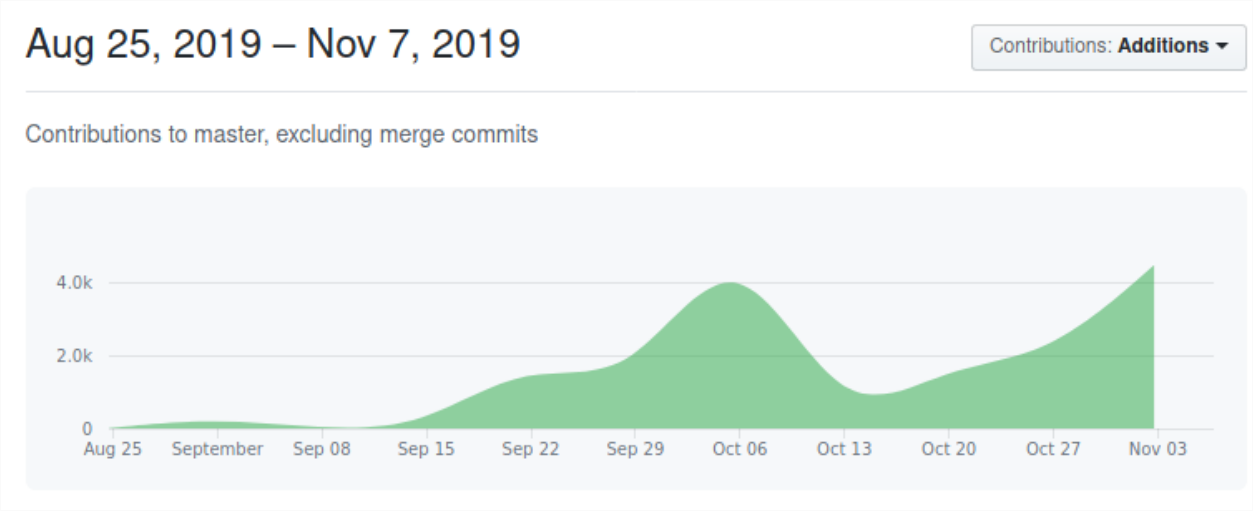
Issues resolved ii

Relevant PR's:

- #21 Workspaces
- #29 aligning types
- #30 enabling board generation
- #32 exporting rust types
- #33 exporting rust types (part 2)
- #36 back-end tests and documentation
- #37 sprint 2 write up deliverable
- #38 express js refactorings

Issues resolved iii

A lot of code was written!




```
💀 ↛  tokei
-------------------------------------------------------------------------------
 Language            Files        Lines         Code     Comments       Blanks
-------------------------------------------------------------------------------
 JavaScript              2           60           50            2            8
 JSON                    2         1316         1316            0            0
 Markdown                1            3            3            0            0
 Org                     1           95           91            0            4
 Rust                    6         1062          897           50          115
 Plain Text              1           24           24            0            0
 TOML                    3           37           30            1            6
-------------------------------------------------------------------------------
 Total                  16         2597         2411           53          133
-------------------------------------------------------------------------------
```

Front ⟷ Back Mostly Done i

API finalized

- `Manager` type makes the API over FFI
  - `poll` the back-end with the `options` and `type` of the move being made
  - checks whether is correct, or not, and returns result
- Internally, `Manager` engages with `GameState` and `GameOpt`, which are the major internal API interfaces of the back-end.
  - Only these entities have direct access to things like `Board`, `Coord`, `PositionStatus`, and other primitive types.

Front ⟷ Back Mostly Done ii

Types cross FFI

- Successfully convert from Rust → JS and JS → Rust
  - neon provides a `define_types!` macro that provides convenient syntactic sugar for defining what gets publicly exposed to Node.
  - everything else defined stays private to node!

Front ⟷ Back Mostly Done iii

Integration Tests

- Use Mocha and Chai within Node to pull in and check the exports and logic provided by the compiled Rust crate.
  - Construct mock values, pass to generated rust code, and check functionality.
- Isolates the testing logic between what the front-end needs to worry about vs what the back-end need to worry about.
  - Runs separate from the unit tests internal to rust module!
  - Truly separated concerns and modulation of program logic.

Front-end finished

- Add endpoints on ‘dev-express-js‘ to specialise client-server communication
- Browser has board receptors that work both on Web Socket and direct server messages
- Separating concerns between client and server, where we can now style and switch out game logic accordingly
- Integration testing from above (Mocha/Chai) is now being extended to serving logic

Ready for Web Sockets

- WS (game server established) -> Node (web server dispatcher)
- we can extend the game to online multiplayer
- Browser/server supports this, clients can play against one another
- Where the three-tier separation scales
    - Faster computation on native module can push directly to WS
    - Clients can interact with each other's GUI without nasty artifacting from server-server talkback

Sprint 2 Lessons

Planning time better

- Kind of a toss up because of other commitments and classes

Stub code is useful

- knowing ahead of time how the different components of our project were going to interact
- would solve developer paralysis from not knowing *what* to expect

Documenting Changes

- Didn’t write sprint 2 writeup artifact as development occurred, rushed all at the end.
- Would be a lot smarter next time to writeup as we develop.
    - stops confusing cross checking between our development board, implemented tasks, tests, etc.

Sprint 3 Goals

Finish Front ⟷ Back

- All the infrastructure is here now, just need to connect.
- Back-end also needs to finalize game logic checking.

Game AI

- Generate back-end logic for a somewhat intelligent game opponent.
- Decide on what this model looks like and how computed.

Final UI tweaks

- Fully implement themes
- Finish off menu/options system

Implement Web Sockets

- Major stretch goal: play across browsers.
- Need to figure out server hosting and communication between possibly multi-threaded processes