Sprint #3Presentation

CS449

Samuel Lim, Michael Cu, Elias Garcia

2019.12.06

Outline

1. Project Summary
2. Sprint 3 Goals
3. Sprint 3 Results
4. Sprint 3 Lessons
5. Future Improvements

Project Summary

Top Level

- Nine Men’s Morris in the browser
- Front-end delivers static HTML/CSS and handles user input
- Back-end drives game logic

Front-End

- Delivered statically via:
  - Node.js sever
  - Browser events target API endpoints
  - Express.js pulls the engine
  - Use EJS for templating

Back-End

- Back-end is game logic and heavy lifting
  - Written entirely in Rust, a systems programming language
  - Use Neon library to interop with Node.js
  - Compile time guarantees for:
    - Memory safety
    - Data race safety
    - Type Safety
    - FFI behavior
  - In addition to being as fast as C++!

Sprint 3 Goals

Finish Front ⟷ Back Communication

- Sprint 1 left us with Rust defined types and a mocked front-end for rendering
- Sprint 2 left us with a working back-end and front-end that could talk
  - 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
- Needed to put together

AI

- Create basic functionality of a game AI
  - take turns
  - at least somewhat intelligently, preferably

Polish Off Front-end

- UI Theme
- Board layout improvement

Get ready for Web Sockets

- Separate web-server from game logic itself
- Drive game logic based on requests/responses over API endpoints
- Structure of project allows game to be played over web, over any device

Sprint 3 Results

Issues resolved

The final issues were finished:

- Proper back-end game validation
  - Piece Placement
  - Moves
  - Attacking

Relevant PR's:

- #42 Game Logic, Part 1 #43 Change Poll API
- #45 Game Logic, Part 2
- #46 AI Logic

Front ⟷ Back 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 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 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 3 Lessons

Planning time better

- Kind of a toss up because of other commitments and classes
- Particularly with sprint 3, brutal for some of us

Documenting Changes

- Didn’t write sprint 3 writeup artifact as development occurred, rushed all at the end.
- Would be a lot smarter next time to writeup as we develop.
  - Lesson was not learned from sprint 2
  - said same thing last time

Massive churn can still happen

- Back-end saw probably around ~1.5K LOC change over sprint 3
  - I am not happy
  - I am not well rested
  - I am not chill

Future Improvements

Optimizations

- Lots of low hanging fruit:
  - back-end: excessive heap allocations
  - front-end: more player options, back-menu

Game AI

- Behavior
  - simple search currently
  - search state space for better moves in future

Implement Web Sockets

- Major stretch goal: play across browsers.
- Need to figure out server hosting and communication between possibly multi-threaded processes
- Have all infrastructure in place between rust and node