Use Case & Application

- The strong competitive scene in Go means there is high demand for training
- Our project does this in two ways:
- Real time over-the-board analysis
  - When two players are playing against each other on our physical board, our systems allows both players to see the best move in their position, if desired
- Historical Analysis and Suggestions
  - Historical analysis allows players to examine which of their moves were strong and which had considerably better alternatives.
  - Multiple engine suggestions allows users to explore alternative to their own move even if there's was the engine's top choice.
  - Monte Carlo Tree Simulations allow us to display the expected win probabilities for each of the candidate moves

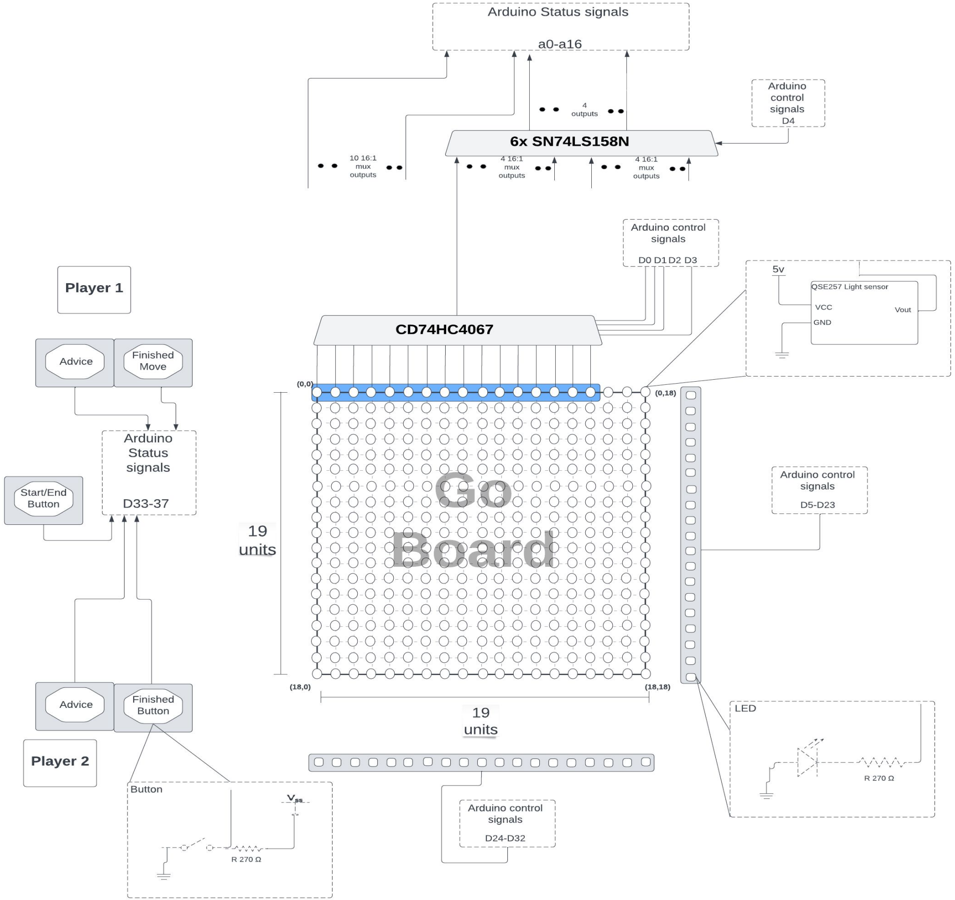
Quantitative Design Requirements

- Mention specifics of hardware
  - 100% accurate identification of game tiles
  - Perform a read of game state in less than 50 microseconds
  - Perform light sensor configuration in less than 50 microseconds
  - Communicate a game state to COM port at most 120ms
  - Communicate advice from COM port to arduino at most 300 Microseconds
- RL Engine
  - Strength level of at least Amateur 5-Dan
  - Display the five best candidate moves in each position on the website
- Website
  - 100% accurate recording of game history
  - Visualization of each move in a specific game’s history

Solution

- Physical Go board that connects to a computer
  - Communicates game state
  - Emits advice from engine
- Website following live gameplay of the Go game
  - Once game is finished, the history can be downloaded
  - Game history can also be visualized on the website
  - Interfaces the engine to the Go board
- Engine will make suggestions for each move if the player wants

System Specification - The Board



Key:

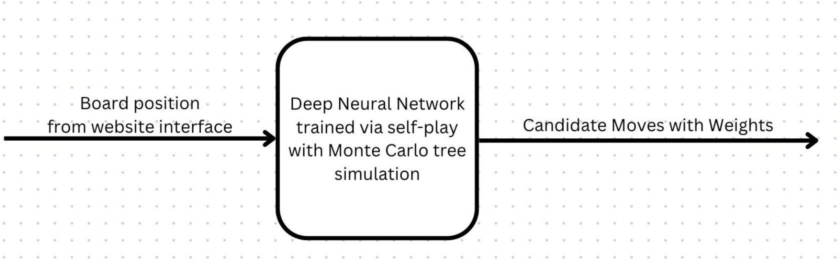
Grey boxes- breadboarded components

Blue boxes- representation of one data Segment for 1 16:1 mux

Dotted boxes- arduino interface signals

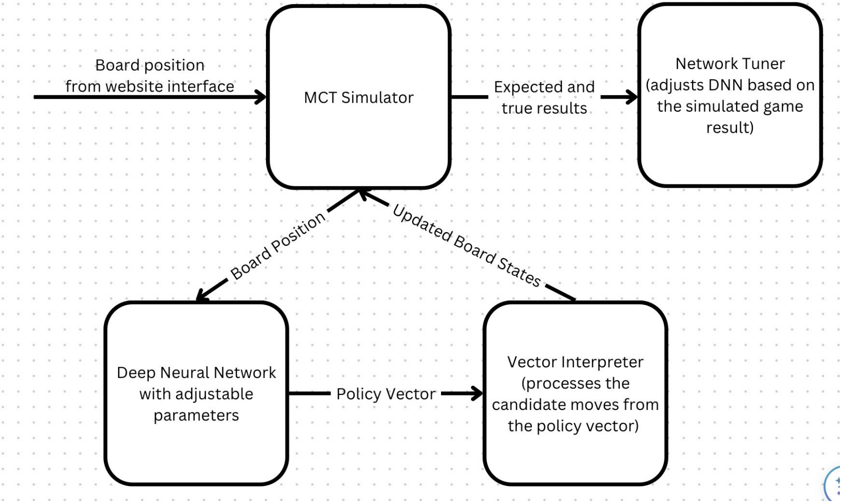
System Specification - RL Engine

System Workflow



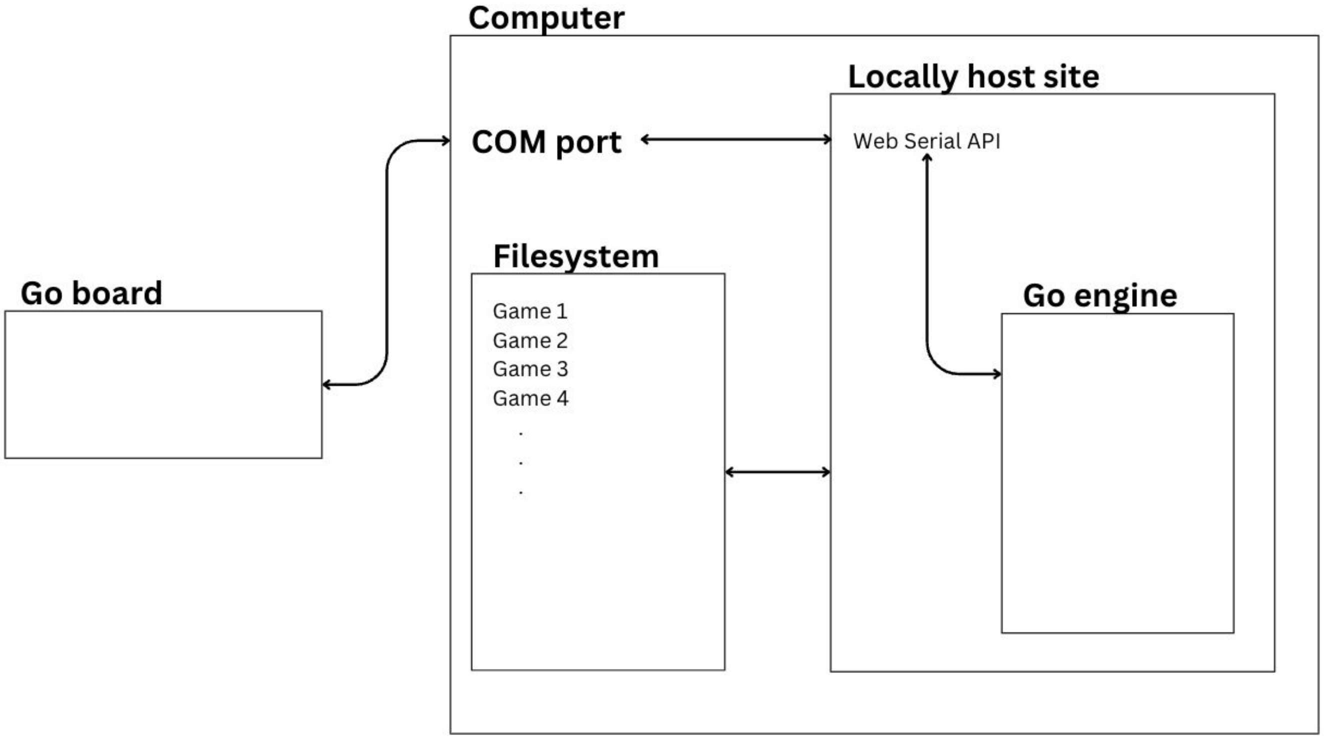
Board position
from website interface
Deep Neural Network trained via self-play with Monte Carlo tree simulation
Candidate Moves with Weights

Training Workflow

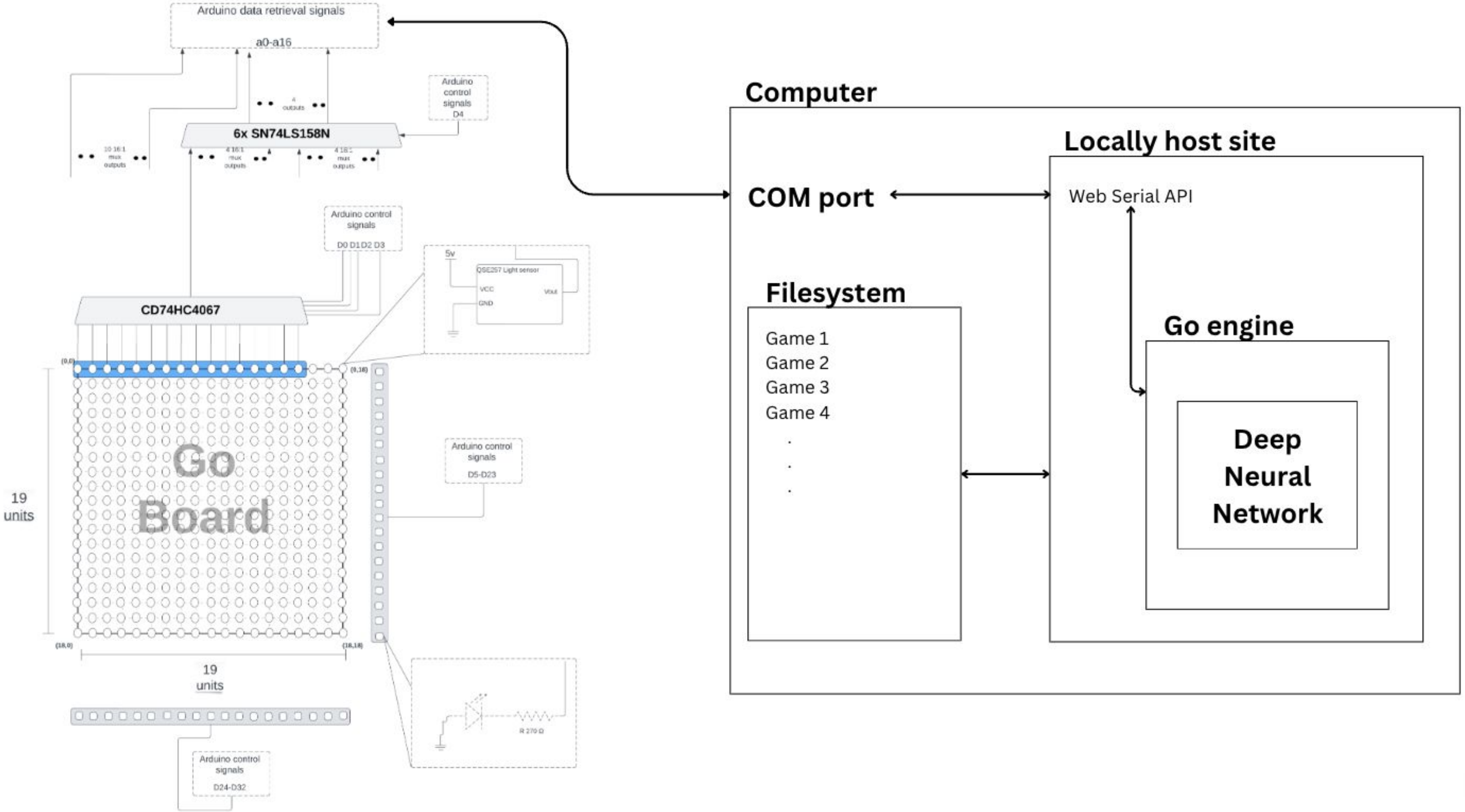


Board position
from website interface
MCT Simulator
Expected and true results
Network Tuner (adjusts DNN based on the simulated game result)
Board Position
Updated Board States
Deep Neural Network with adjustable parameters
Policy Vector
Vector Interpreter (processes the candidate moves from the policy vector)

System Specification - Site



Block diagram of entire system



Implementation Plan - Hardware

- Make custom adjustments to physical GO board to mount electrical components
- Embedded the light sensors output and multiplexers with the arduino to allow for multi-cycle data retrieval
  - Requires automatic configuration for environment
  - Edge cases needing to be more thorough for muxes 16-21
- Develop software for retrieving and formatting game state data retrieval
- Develop LED software for optional advice mid game

Implementation plan - RL

- Randomly initialized deep neural network parameters
- Monte Carlo Tree Simulations used to determine each move
    - Simulations are computed with the deep neural network playing the game against itself.
- Parameters adjusted post-game result via gradient descent
    - Minimize expected result post-move vs. actual game result
    - Maximize similarity between policy vector and search probabilities
- Once trained, weights can be stored locally
    - Given input of board state, it will output the best available moves, along with their expected win percentages
    - If queried, can transmit the best available move to be displayed on board

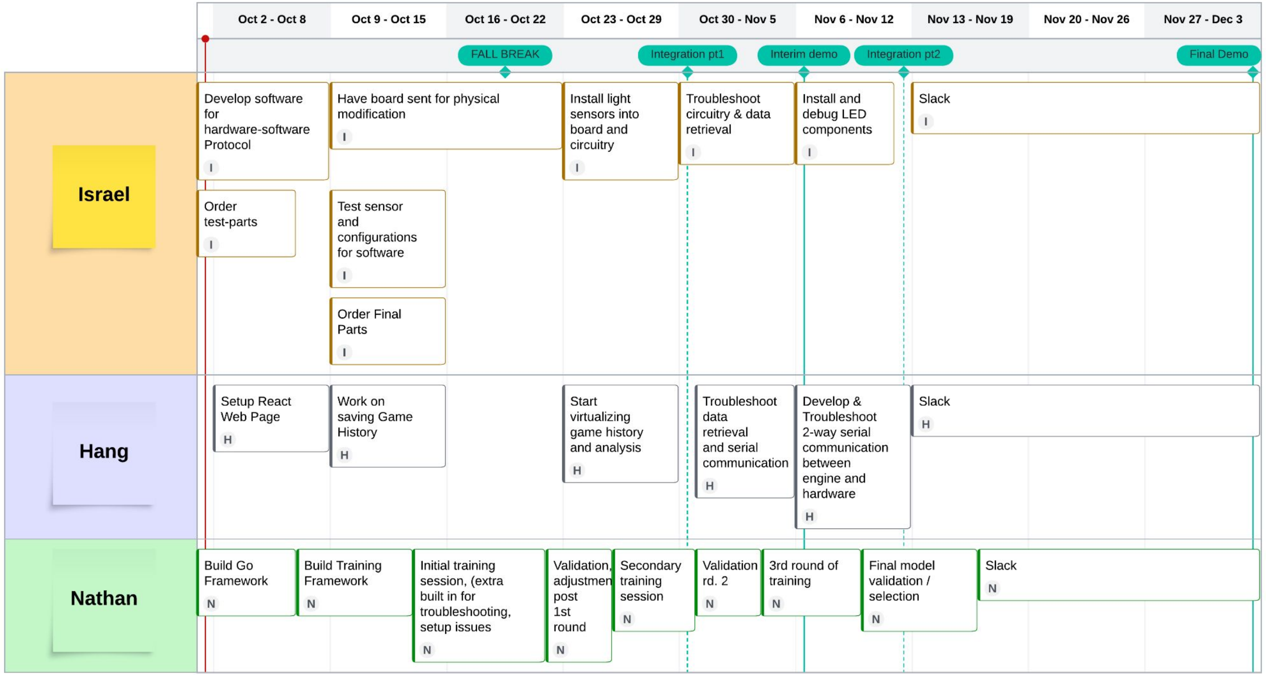
Implementation plan - software/site

- Built on React + Javascript
- Interfacing the board and engine
    - Web serial API to read and send signals to the COM port
    - Serialize signal from board into an input for the go engine
        - 361 element array
    - Send the output of the engine back to the Arduino board
- Saving game history
    - Saved into text file → download onto computer
- Displaying game history
    - Import txt file
    - Visualization of the board for each move made in the game
        - Convert each game state/move into a board visualization
        - Each move will also show the engine's suggested move

Test, Verification, and Validation

- Hardware
  - Initial parts
    - Develop circuitry for light sensors and multiplexer performance through breadboarding
  - Final parts
    - Make dummy physical game states for retrieval
    - Push serial requests to LEDs to test correct provided insight
- Game History Analysis
  - Compare engine suggestions to high-level open source engines to ensure strong suggestions
  - Use dummy Game state in CSV format for ensuring game state is recorded and displays
- RL
  - Ensure engine performs at 50+% record against a 5-Dan level engine available online
- Website
  - Jest unit testing for each page in the site to ensure correctness of code

Project Management



Oct 2 - Oct 8 | Oct 9 - Oct 15 | Oct 16 - Oct 22 | Oct 23 - Oct 29 | Oct 30 - Nov 5 | Nov 6 - Nov 12 | Nov 13 - Nov 19 | Nov 20 - Nov 26 | Nov 27 - Dec 3

FALL BREAK | Integration pt1 | Interim demo | Integration pt2 | Final Demo

Israel

- Develop software for hardware-software Protocol I
- Order test-parts I
- Have board sent for physical modification I
- Test sensor and configurations for software I
- Order Final Parts I
- Install light sensors into board and circuitry I
- Troubleshoot circuitry & data retrieval I
- Install and debug LED components I
- Slack I

Hang

- Setup React Web Page H
- Work on saving Game History H
- Start virtualizing game history and analysis H
- Troubleshoot data retrieval and serial communication H
- Develop & Troubleshoot 2-way serial communication between engine and hardware H
- Slack H

Nathan

- Build Go Framework N
- Build Training Framework N
- Initial training session, (extra built in for troubleshooting, setup issues N
- Validation, adjustmen post 1st round N
- Secondary training session N
- Validation rd. 2 N
- 3rd round of training N
- Final model validation / selection N
- Slack N