Deep Blue and Gold

Sean Scofield, Frank Lu, and Arthur Jeng

The Goal

- Build a smart chess board that can
  1. display legal moves
  2. provide hint

  for the human player by lighting up squares (using neopixel strips under the plexiglass board).
- *Inspiration:* http://www.instructables.com/id/How-to-Build-an-Arduino-Powered-Chess-Playing-Robo/ -- a project by *maxjus*
  - We ended up deviating in a considerably direction than that of this project by deciding to only tackle neopixel lighting for move evaluation rather than tackling piece actuation.

Design hj,n

- Faced with the difficulty of wiring 64 reed switches and 64 LEDs, we decided to use **neopixel strips** for lighting instead
  - *advantages:* 1 GPIO pin per strip (8 strips total, assuming 1 per row on board), as opposed to 1 pin per LED
  - challenge: power consumption
- Instead of using a **chess server**, resolved to use a **native chess program**
  - *advantages:* not reliant on wifi to play game
  - tradeoff: speed vs reliability
- Microcontroller: **Arduino Mega**
  - *advantages:* more support for neopixels and more GPIO pins
  - limited computing power to run chess program
- Big challenge coming out of our mid-semester demo was a power source. Easily solved by cutting a generic 5V, 4A power supply

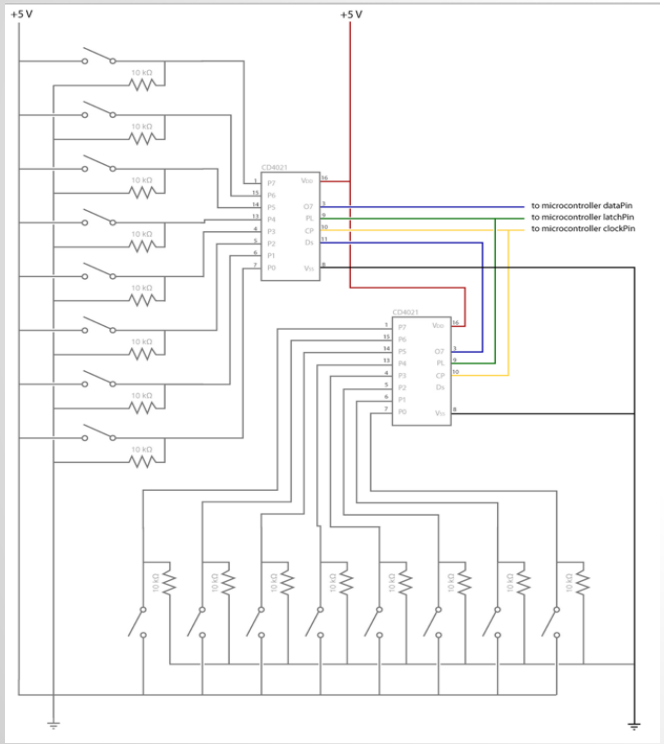
Challenges

1. Scalability
   - Limited GPIO pins
   - Timing is an issue
2. Chess AI
   - Memory allocation in Arduino (run-time vs compile-time)
   - Memory leak
   - Memory fragmented if allocated run-time
3. FSM/Real Time Behavior
   - Accidentally activate reed switches result in state transitions
   - Need to put in delay and calibrate sensitivity

Scalability/ Real-Time Behavior

- Use of multiplexers to solve this problem
  - faster than serial shift register
  - use a lot more pins (for 64 switches we need at least 10 pins)
- Use of shift registers to solve this problem
  - only uses 3 pins, no matter how many bits
  - slower than multiplexers (slightly)
  - Timing constraint:
    - need to share clocking among the 8 shift registers
    - collect data without coordinating with the microcontroller
    - need to output serial data

Scalability/Real-Time Behavior
Input/Output



- Use of Asynchronous Parallel Input to Serial Output shift registers
    - why asynchronous?
    - why parallel input?
    - why serial output?

- Daisy chain them on the serial outputs and share the clock and serial/parallel control pins
    - clock: metronome of the conversation between the shift register and micro controller
    - latch pin: parallel to serial control

Scalability/Real-Time Behavior Detection and Debugging

- Piece detection
  - Accidentally knock over one piece
  - Sensitivity
- Approach:
  - Unit test each component before testing the system.
  - Setup user constraint while moving pieces
  - Add more states in FSM for error checking
  - Immediate feedback

Chess: External Research

- Most existing chess libraries don’t have helpful, modular functions to support the actions in each state of the state machine.
  - complicates unit testing

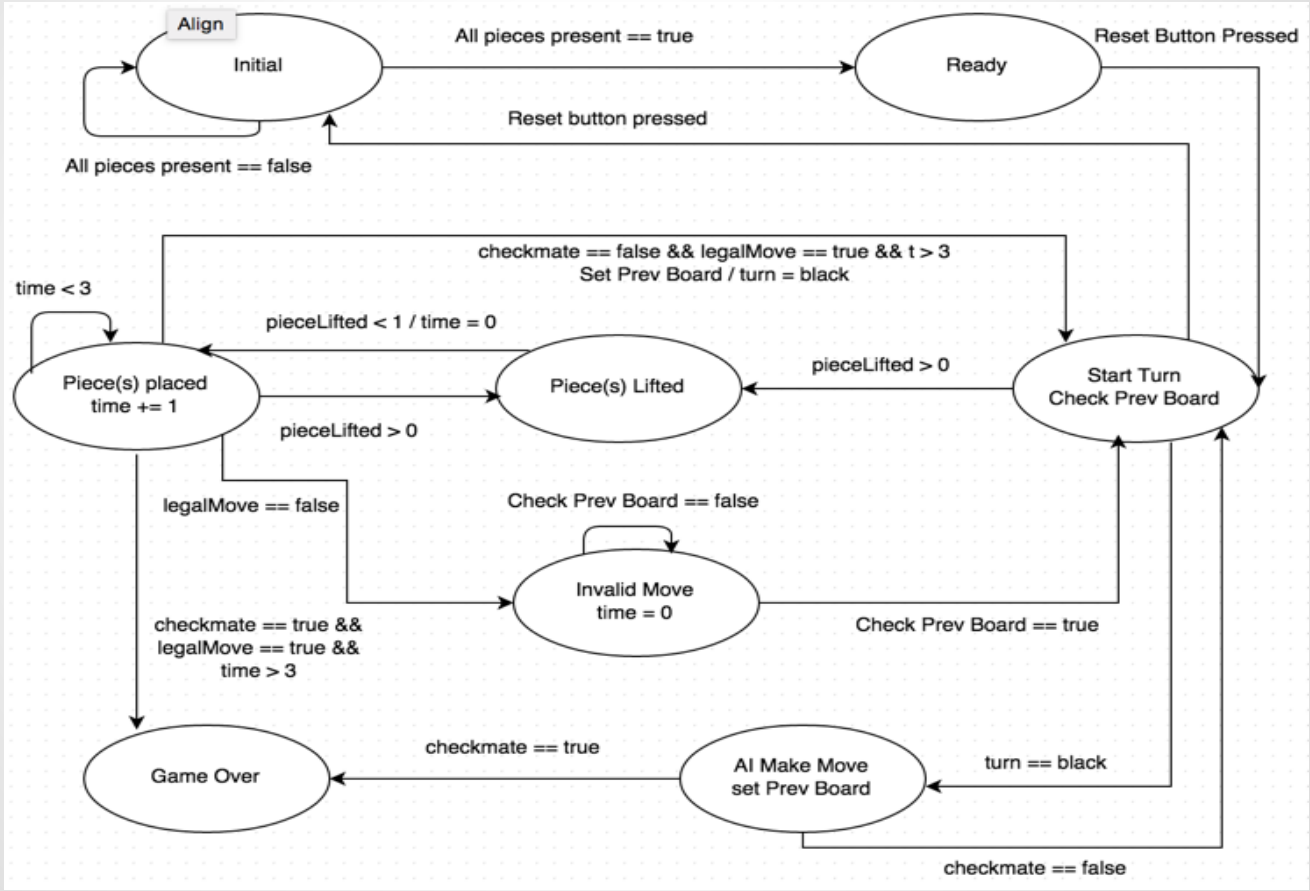
- Not arduino compatible:
  - requires a computer or another microcontroller to run the computation and transmit responses serially.

Chess: Our own AI

- wrote our own chess library with minimax game tree and alpha beta pruning. Good moves are determined by point values of pieces

- having our own library has given us greater control over the information our board can obtain, and the syntax in which we provide it
  - i.e. can obtain legal moves (for FSM) by a call to *genMoves(String square, char color, String board)*

FSM: Modeling



FSM: Visibility

Initial State:

111111111111111**000**….0001111111111111111

a1, b1, c1, … a2, b2, c2….

Next State:

101111111111111**001**0..0001111111111111111

Board State (AI keeps track of)

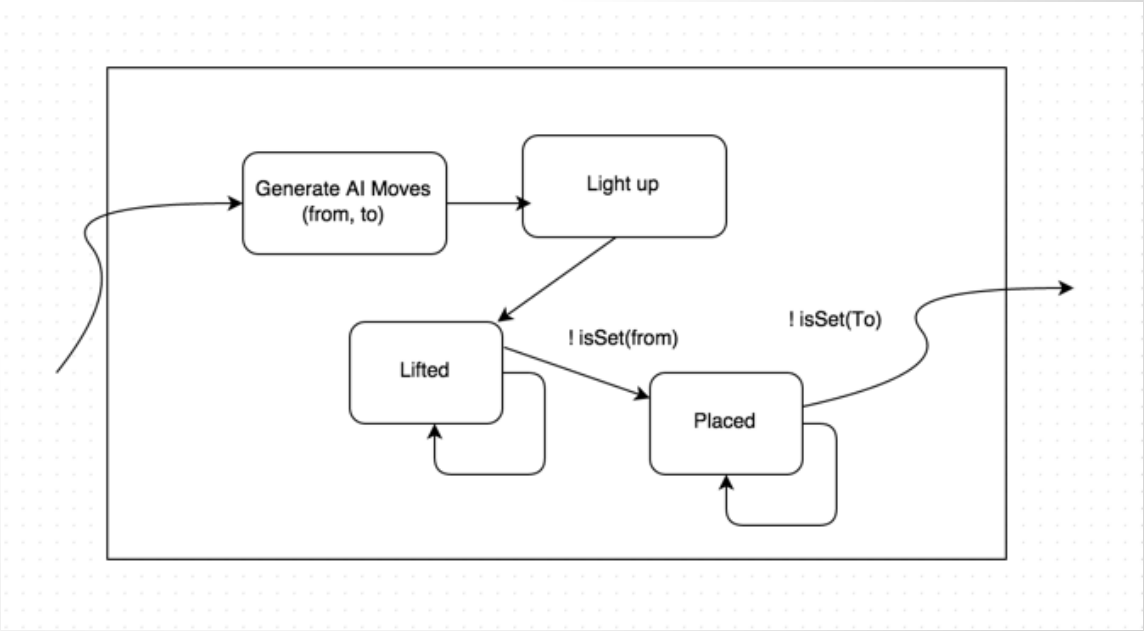
RKBQKBKRPPPPP……pppppppprkbqkbkr

FSM: Composite State Machine and more Modeling..

We realized many of our states had many substates.

Again much of the flow, was to account for “real world” -> CS domain problems.

For a simple example, ...



Chess: Challenges for our AI

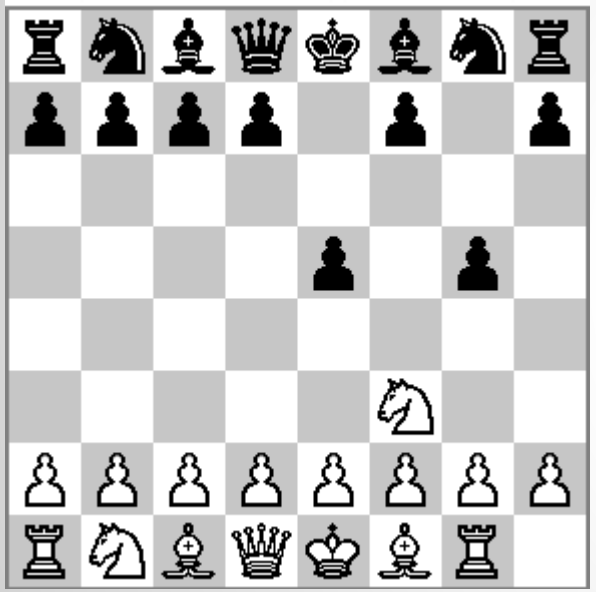
- When there is more than one legal move, the board will have problem detecting which move is taken.

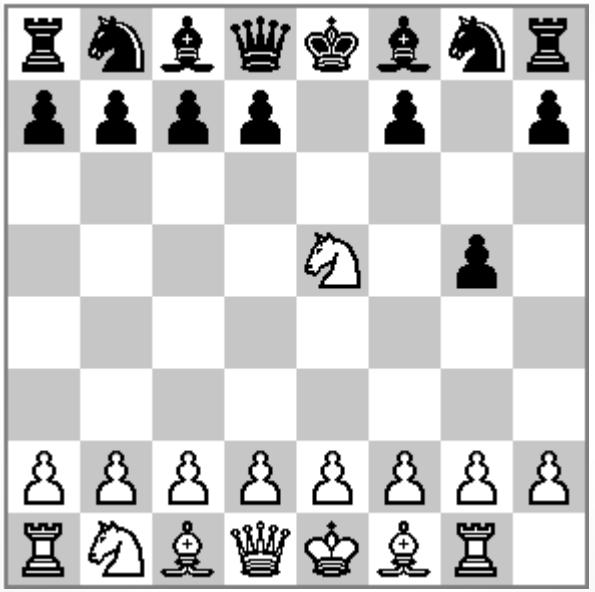
Possible solutions:

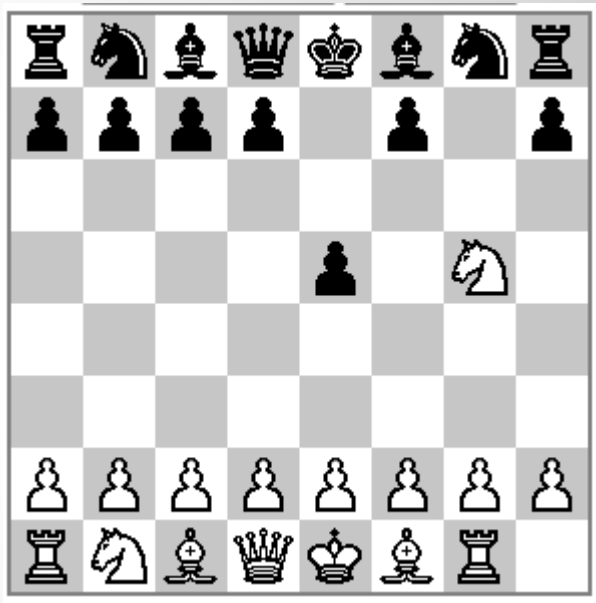
Hard -- try to detect which piece is captured by detecting which one gets lifted

Ideal solution -- have a better hardware setup (i.e. use RFID tags to uniquely identify pieces). In other words, many times it's better to solve the root "real world problem" as close to the to "real world" as possible.

Continued….







Results/Analysis

- Can play game against AI (AI lights up squares for its moves)
- Can switch between *easy* and *normal* mode (*easy* mode provides legal moves)
- Hardware sensitivity has made implementing our FSM more difficult than we originally intended:
  - i.e. when board gets bumped slightly, pieces move a small amount
    - often causes pieces to no longer be positioned correctly above reed switches
    - may cause pieces to move near enough to each other that the magnetic pull between them becomes a problem
    - one solution might be to make the squares bigger

Takeaway

Overall, while modeling this project on paper was fairly straightforward, we discovered that the difficulty of implementing it was magnified greatly by some of our hardware decisions.

If we were to iterate upon this project, we would put more time into making sure these hardware decisions in order to make the implementation of our model more feasible.