Introduction

Objective

Shogi is a very difficult game for beginners to learn because of its complex set of rules. While anybody could find a rulebook and start trying to play Shogi, it would be very simple to violate one of its rules, or not to be able to take advantage of some more complex rules. There are a number of guides out there and one can even attempt to play online Shogi; however, the complexity of the game remains overwhelming due to the sheer number of types of pieces, each with their own unique moveset, compounded with added complexity from the ability to replace captured pieces.

To rectify this issue, we propose a Shogi board which helps to interactively teach a player all the valid moves that can be made using a piece that has been lifted from the board. This will help players know all possibilities of where to move the pieces and prevent any illegal moves from being made, as the board will keep track of its own game state, and thus be able to point out any errors to the players.

Background

While most Americans are at least vaguely familiar with the rules of Chess, if not able to fully play it, far fewer are familiar with Shogi. Whereas Chess has an established community of players and mentors willing to teach the game, Shogi has no such following, with few organized groups, if any. Thus, prospective players are left to either play unbalanced games online against far more experienced players, to try learning alone through a guide, or to start learning with other beginners, where neither player has enough experience to notice when a rule has been violated.

Chess is already a game where some of its more intricate strategies such as forks (simultaneously threatening two or more pieces with a knight) and pins (a similar strategy using a bishop, rook, or queen) are less utilized by beginner players, and even a decent number of the basic maneuvers are less known to many. Shogi has a 9x9 board over Chess’s 8x8, but the more significant difference is Shogi has a heavier emphasis on counter attacking strategies [1]. Also, in Shogi, unlike Chess, pieces are never out of play, and may be replaced on the board by the capturer, creating an immense amount of possible moves compared to Chess. Finally,
Shogi has some obscure rules which beginners may easily forget about, such as optional
promotion, except under certain conditions where it becomes mandatory. Even though Shogi
and Chess pieces share similar names, they behave quite differently, which makes adapting
from Chess to Shogi quite difficult [1].

Chess has quite an astonishing number of people playing the game, 35 million in the US
[2]. This growing community helps support the rise in popularity of the game. However, Shogi
does not have such a strong community, especially outside East Asia. While it is difficult to get
an accurate count of the number of Shogi players in the United States, an estimate can be
made by comparing the number of players well-known enough to have articles on Wikipedia,
which would suggest roughly 100 times as many Chess players (315) as Shogi players (3)
[3][4].

By making learning the game simpler, its popularity could rise significantly.

High-level requirements

- Board must recognize when a piece has been lifted and identify the possible moves for
  that piece.
- Board must update the internal game state after a valid move is made on the board, and
display a warning if an invalid move is made, which needs to be corrected before play
can continue.
- Touchscreen must allow for manual error correction both for physical errors such as
  pieces moving after bumping the board, as well as manually inputting a move if the
  board misreads what has been played.
Design

Figure 1: Block diagram for implementation of Shogi board

Legend
- Sensor Data
- Digital I/O
- Analog I/O
- Power
- Touchscreen I/O + Power over GPIO

Figure 1: Block diagram for implementation of Shogi board
The design of the augmented Shogi board has six major components. We have a Game State/Compute Processing unit that handles maintaining the game state for each move made and making the master logical decisions behind which LEDs should be illuminated and processing the sensor I/O. This unit is responsible for back and forth communication with the AtMega 328Ps in the Game I/O block.

The Game I/O block is responsible for handle I/O between the board electronics of the LEDs and photoresistors, as well as for sending data over serial to the Raspberry Pi Zero in the GS/CP unit.

The HID for Game Control/Error Correction is used to allow the user to correct any piece detection errors, as well as to view the game state from the touchscreen display, and to provide additional information to the players, such as warning them when an invalid move has been made.

The Game Board Sensor block provides information about what pieces have been moved with the use of photoresistors. The sensors will require some additional logic to convert resistance into a digital voltage, and will include some MUXes to reduce the number of I/O pins required, which require some control lines back from the AtMega chips.

The Game Board Indicator states what moves are valid to make with the piece selected. They are controlled by signals from the AtMega chips in the Game I/O Processing unit. The indicators will store some internal state logic in order to allow all LEDs to be individually controlled with a minimal number of I/O pins, likely in the form of shift registers and individual comparators.

The Power Supply Block is responsible for handling power distribution to the electronics. The PCB with the AtMega chips and the Raspberry Pi Zero will be individually powered, while most other peripherals will receive power from either of these two boards due to their relatively individual low current requirements.
Physical Design Description

Shogi Board Layout [5]

The board will be modified to include photoresistors underneath each of the 81 squares on the board. For each square, a hole will be drilled through the center of the piece, and a photoresistor will be attached into the hole, facing upwards. When a piece is present on that square, the photoresistor is covered; without a piece, the photoresistor is lit by the ambient light of the room. This difference in lighting will be used to detect when a piece is moved. All the electronics will be connected underneath the board, except for the touchscreen, which will be placed off to one side of the board, accessible to both players.
Figure 2: Photoresistor placement, side view

This simple render is intended to show a cross-section of one tile of the board, with a piece present on this square. The photoresistors are slightly inset into the surface of the board, such that the pieces smoothly sit on the board, blocking the ambient light to the photoresistor.

Figure 3: Board layout render with visible photoresistors

At any time during the game, the pieces being present will block certain photoresistors and reveal others. Each of these is being monitored by the Arduino chips and translated into a game state. Once a change is detected in one of the photoresistors, the system will determine whether the move which the player has made is indeed valid, and respond accordingly.
Tolerance Analysis

The sensors used for detecting whether a piece is present, photoresistors, or light-dependent resistors (LDRs), are nonlinear devices which vary in resistance by many orders of magnitude with a change in lighting. Using one example of a datasheet taken from an arbitrarily selected photoresistor, the resistance in darkness, such as one of the photoresistors which is being covered by a piece, is in the range of 5MΩ, while the ‘bright’ resistance is in the range of 50kΩ at 10 Lux \([9]\), which is still well below ordinary ambient room lighting, at roughly 50 Lux \([10]\).

Because of this wide variation in resistance, the tolerance for individual differences between photoresistors is quite large, as anything within an order of magnitude should be easily distinguishable. A simple analog-digital converter should be sufficient for this application, or even a plain voltage divider.
This extremely simple circuit should be sufficient to convert the LDR’s resistance into a voltage which can be read by the microprocessor chips. Using the above example, with a resistance range of 50kΩ to 5MΩ, we take R1 to be 500kΩ.

\[
I_{\text{out}} = 0 \rightarrow I_{\text{R}_p} = I_{\text{R}_1} \\
V_{\text{out}} / R_p = (5 - V_{\text{out}}) / R_1 \\
V_{\text{out}} / R_p + V_{\text{out}} / R_1 = 5 / R_1 \\
V_{\text{out}} * (1 / R_p + 1 / R_1) = 5 / R_1 \\
V_{\text{out}} = 5 / (R_1 / R_p + 1)
\]

Case 1: No piece present, bright light, Rp = 50kΩ
\[V_{\text{out}} = 5 / (500k / 50k + 1) = 5 / 11 = 0.4545V\]

Case 2: Piece present, dark light, Rp = 5MΩ
\[V_{\text{out}} = 5 / (500k / 5M + 1) = 5 / 1.1 = 4.545V\]

These two voltages should be easily distinguishable as digital low/high, though some testing may be necessary to determine the ideal resistance for R1 for the particular photoresistors which end up being used in the design - This value is only appropriate for this particular example of a photoresistor.
Functional Overview

**Game State/Compute Processing** - The goal of the Game State/Compute Processing block is to receive process input from the Game I/O Processing block AtMega 328P microcontrollers and also process any board state correctional data provided through the touchscreen interface from the HID For Game Control/Error Correction block. Once it receives game state updates, the Raspberry Pi Zero then processes the updates to be reflected both on the touchscreen and sent to the AtMega 328Ps to update the respective LEDs for the valid moves that can be made.

**Game I/O Processing** - The Game I/O Processing blocks goal is to handle the large number of sensory inputs and digital indicator outputs needs for the assistive Shogi board. This block sub-divides the work of piece detection and LED illumination but does not know about the big-picture setup of the overall game board. Therefore, the AtMega 328Ps used in this block must use Digital I/O serial communication with the Raspberry Pi Zero in the Game State/Compute Processing block to update the Pi with sensor information that lets the Pi update the overall game state. Any detection of a lifted piece should be sent over the the Pi Zero, which in turn sends a signal back to the appropriate AtMega chips to tell them, which LEDs to illuminate, thus indicating to the players which spaces are valid moves given the piece which was lifted from the board.

**HID For Game Control/Error Correction** - This block consists of a touchscreen that interfaces back to the Raspberry Pi Zero and used to as mechanism to both provide information about the current board state perceived by the Raspberry Pi Zero and let the user update it due to any errors, such as pieces being accidentally bumped or moved.

**PSU** - The PSU block is used to distribute power to the various electronics involved in the system. However, power will be provided to most of the individual components through the AtMega 328P microcontrollers and the Raspberry Pi Zero, which are able to provide power as needed to the rest of the electronic systems. For instance, the Raspberry Pi Zero will use the power it is given from this block and provide power to the touchscreen through its GPIO.

**Game Board Indicators** - This block consists of LEDs used to indicate when each position is valid to move to on the 9x9 Shogi board. These indicators are physically updated by the AtMega 328Ps, which are in turn signalled to enable or disable certain LEDs to light up by the Raspberry Pi Zero.

In order to save power, the LEDs will be blinking at high frequencies, rather than being always-on. This will require some logic internal to each LED, but should allow each of the AtMega chips to draw considerably less power to power peripherals, thus reducing the power draw of the entire board, and allowing for a smaller PSU, compared to the possibility of fully powering 20 LEDs at once (the theoretical maximum for a promoted rook with no obstructions).
**Game Board Sensors** - This block contains a series of photoresistors placed at each position on the 9x9 Shogi board. These sensors send their input into an analog-digital converter within the Game IO Processing unit subsystem, and the converted sensor data is passed into one of the AtMega 328P microcontrollers.
# Block Requirements

**Table 1: Functional Requirements for Individual Blocks of Board Design**

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Technical Requirement</th>
<th>Quantitative Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game State/Compute Processing</td>
<td>Must be able to interface with both a touchscreen and multiple AtMega microcontrollers to update the game state automatically, while remaining open to manual user error correction.</td>
<td>Needs to utilize a single board microprocessor such as a Raspberry Pi Zero to interface with four AtMegas over serial and use GPIO to interface to a touchscreen.</td>
</tr>
<tr>
<td>Game I/O Processing</td>
<td>Must be able to provide sensor input to a microprocessor as well as to drive LEDs On and Off based on signals provided from the microprocessor over serial.</td>
<td>81 photoresistor inputs and 81 3-color LED outputs fully controlled by four AtMega chips through logic circuitry.</td>
</tr>
<tr>
<td>HID For Game Control/Error Correction</td>
<td>Must be an interface compatible with the microprocessor that allows for both visualizing how the game state is currently interpreted by the microprocessor and allowing for any errors in that state to be corrected by the user through a touchscreen display</td>
<td>Piece movements must be recorded with at least 95% accuracy, and interface must allow for all errors to be corrected such that play can continue.</td>
</tr>
<tr>
<td>PSU</td>
<td>Must be able to power the 4 AtMega 328Ps and the Raspberry Pi Zero, including the power these devices provide to their peripherals, such as the touchscreen and LEDs.</td>
<td>One 12±0.5V, 5A (or greater AMP) power supply to the 7805 Voltage Regulators, which will power all individual components.</td>
</tr>
<tr>
<td>Game Board Indicators</td>
<td>LEDs must be controlled from each AtMega 328P such that they indicate to the user the valid moves for the currently lifted piece.</td>
<td>81 LEDs which are individually controlled to allow all possible combinations.</td>
</tr>
<tr>
<td>Game Board Sensors</td>
<td>Photoresistors under each square whose variable resistance can be read by the AtMega 328Ps in order to determine when a specific Shogi piece has been lifted or replaced.</td>
<td>81 Photoresistors which can accurately determine with 99% accuracy whether a piece is present while placed in an ECEB classroom, within 1 second of the piece being moved.</td>
</tr>
</tbody>
</table>
Block Verification

Game State/Compute Processing:
- A game can be played by following the recorded moves of an existing game without obstruction such as having to wait for the computer to catch up.
- An error can be intentionally made at any point during play, which the system should detect. Play should not continue until the warning is resolved.

Game I/O Processing:
- All inputs and outputs are being forwarded to the microprocessor for controlling game state, this block has no visible response in a fully working system
- If individual testing is necessary, chips can be substituted for independent Arduino boards and add extra debug displays.

HID For Game Control/Error Correction:
- If an error is intentionally made during demonstration, the system should allow that error to be corrected, then resume play.
- Following an ordinary game's move list, at least 95% of valid moves should be made without the system incorrectly detecting an error.

PSU:
- Entire board is powered off the shared external power supply.
- No part of the board overheats even during an extended period of ordinary play.

Game Board Indicators:
- During the course of a game, LED indicators on board indicate all moves a piece is capable of making every time one is picked up to play.
- LED color variation is used to differentiate moves and captures.
- A debug mode may be introduced to manually control LEDs during development.

Game Board Sensors:
- Every one of the 81 squares should independently be able to detect pieces being moved.
- Pieces being moved are correctly registered by the system within 1 second of the move having been made.
Risk Analysis

The most difficult part of the project construction will likely be the circuitry needed to read from the photoresistors. Since there are 81 inputs, the amount of wiring needed to connect each photoresistor to an input on the PCB will be extensive, and will likely cause errors. Furthermore, the amount of inputs needed are more than any single AtMega chip is able to support. As such, it will be necessary to add input multiplexers in order to reduce the number of inputs to a manageable amount, which will in turn add additional complexity to the wiring, as well as require larger amounts of work on each AtMega to poll all inputs consecutively.

Similarly, each square on the board will have to show one or more LEDs, which will also have to be controlled individually by the AtMega chips. In order to avoid having to use so many output lines, it may be necessary to create simple LED controller circuits with latches to save a state, so that they can be programmed with fewer data output lines, instead of creating a 81-wide one-hot bus to control LEDs, which would further complicate wiring.

Ideally, a circuit could be designed in such a way that only one or two AtMega chips are necessary. This will not only reduce the cost of the entire board, but should also simplify the code needed for the controlling Raspberry Pi to interact with each of the AtMega chips in turn. According to the AtMega 328P datasheet, each chip has 23 programmable I/O pins [6], which would have to be shared between the input photoresistors and the output LEDs.

The physical construction of the board is not expected to be difficult, as a relatively large margin of error is acceptable for the photoresistors to fit in the holes while still being entirely covered by the pieces. This is not expected to require any more precision than what could be achieved with a simple drill press. However, depending on what sort of board is used for the design, some effort may be needed to fit electronics into a relatively small space, or otherwise to lift the board up with some supports in order to create more space underneath the board.
Cost And Schedule

Cost

Labor Analysis

Rahul - $7 x 2.5 x 44 hrs = $770
Max - $7 x 2.5 x 42 hrs = $735
Total - $1505

Parts Cost

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Quantity</th>
<th>Price</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMega 328P</td>
<td>4</td>
<td>$4.99</td>
<td>$19.96</td>
</tr>
<tr>
<td>Raspberry Pi Zero</td>
<td>1</td>
<td>$5.00</td>
<td>$5.00</td>
</tr>
<tr>
<td>Photoresistors (100 pieces)</td>
<td>1</td>
<td>$8.95</td>
<td>$8.95</td>
</tr>
<tr>
<td>3-color LED Diodes (100 pieces)</td>
<td>1</td>
<td>$8.96</td>
<td>$8.96</td>
</tr>
<tr>
<td>FT232RL USB to Serial Cable</td>
<td>1</td>
<td>$7.99</td>
<td>$7.99</td>
</tr>
<tr>
<td>16 MHz Clock Crystal</td>
<td>1</td>
<td>$7.99 for pack of 20</td>
<td>$7.99</td>
</tr>
<tr>
<td>10 uF capacitors</td>
<td>8</td>
<td>$6.39 for a pack of 50</td>
<td>$6.39</td>
</tr>
<tr>
<td>220 Ohm resistors</td>
<td>8</td>
<td>$7.03 for a pack of 100</td>
<td>$7.03</td>
</tr>
<tr>
<td>Red LEDs</td>
<td>4</td>
<td>$6.68 for 100 pack including both red and green LEDs</td>
<td>$6.68</td>
</tr>
<tr>
<td>Green LEDs</td>
<td>4</td>
<td>Shared with above</td>
<td>Shared with above</td>
</tr>
<tr>
<td>10k Ohm resistors</td>
<td>4</td>
<td>$7.03 for a pack of 100</td>
<td>$7.03</td>
</tr>
<tr>
<td>Part Description</td>
<td>Quantity</td>
<td>Price for 1</td>
<td>Price for Pack</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>22 pF Capacitors</td>
<td>8</td>
<td>$6.39</td>
<td>$6.39</td>
</tr>
<tr>
<td>7805 Voltage Regulators</td>
<td>4</td>
<td>$6.58</td>
<td>$6.58</td>
</tr>
<tr>
<td>74HC595 Shift Registers</td>
<td>6</td>
<td>$6.98</td>
<td>$6.98</td>
</tr>
<tr>
<td>LM393 Comparator</td>
<td>6</td>
<td>$7.99</td>
<td>$7.99</td>
</tr>
<tr>
<td>TFT Touchscreen for Raspberry Pi</td>
<td>1</td>
<td>$31.99</td>
<td>$31.99</td>
</tr>
<tr>
<td>12 V 5 AMP Power Supply</td>
<td>1</td>
<td>$24.95</td>
<td>$24.95</td>
</tr>
<tr>
<td>Perf Boards (20 pieces)</td>
<td>1</td>
<td>$6.49</td>
<td>$6.49</td>
</tr>
</tbody>
</table>

Parts Cost Total - $177.35

These costs are calculated under the assumption that none of these parts can be freely acquired in small quantities. If simple ICs, resistors, and capacitors can, price goes down dramatically, as these are parts usually purchased in bulk, but only a few are required.

**Total Cost Analysis**

The total cost is $1505 in labor plus $177.35 in cost, which leads to $1682.35. The vast majority of the cost is dominated by labor cost, while the parts are relatively inexpensive. If the individuals working on this project were actually getting paid, the cost would increase dramatically over the current real cost.
Table 3: Predicted Schedule for Project Work

<table>
<thead>
<tr>
<th>Week</th>
<th>Tasks Completed (Member, Time Taken)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/18</td>
<td>Revise Design Document - (Rahul/Max, 3 hrs) Begin ordering basic components for simple prototyping - (Rahul/Max, 1 hour)</td>
</tr>
<tr>
<td>2/25</td>
<td>Begin prototyping and debugging basic components of the design - (Rahul/Max, 5 hours)</td>
</tr>
<tr>
<td>3/4</td>
<td>Formalize PCB designs - (Rahul, 6 hrs) Construct Perf circuits for voltage dividers - (Max, 4 hrs)</td>
</tr>
<tr>
<td>3/11</td>
<td>Confirm PCB design - (Rahul, 3 hrs) Initial software implementation working - (Max, 3 hrs)</td>
</tr>
<tr>
<td>3/18 (Break)</td>
<td>Construct physical board prototype - (Max, 4 hrs) Begin working on touchscreen interface - (Rahul, 4 hrs)</td>
</tr>
<tr>
<td>3/25</td>
<td>Integrate pieces together and begin unified testing - (Rahul/Max, 8 hrs)</td>
</tr>
<tr>
<td>4/1</td>
<td>Debug and make needed design revisions - (Rahul/Max, 8 hrs)</td>
</tr>
<tr>
<td>4/8</td>
<td>Work On presentation material - (Rahul/Max, 6 hrs)</td>
</tr>
<tr>
<td>4/15</td>
<td>(Nothing currently scheduled for this week)</td>
</tr>
</tbody>
</table>
Ethics And Safety

Due to our project’s lack of data collection from the users, the ethics concerns involved seem to be negligible. Our project also operates under a low voltage environment, minimizing many electric safety concerns, though not removing them completely.

If this product were to be marketed, some experimentation would be necessary to determine the how effective the board is to assisting learning, however this is far outside the scope of this class, and is not a consideration at this time.

One possible issue is the issue of undervoltage in the case of large loads [7]. We need to be careful when developing the PSU to be sure we accurately accounted for the power consumption needs of the entire design. If not, we might damage our hardware over time due to operating electronics at inconsistent voltages. If we did so, we would be delivering a product that is not capable of sustaining itself, which would be unacceptable as a consumer product.

We should also be very of the case of using incorrect or unregulated power supplies that may result in an overvoltage and thus burn or damage the ICs and other electronic components used in this project. However, aside from power supply concerns, this project operates using mostly low voltage systems and does not take in an special input or sensitive data, reducing the amount of ethical and safety concern significantly. Even though our project itself does not seem to violate any ethical or safety concerns, we as developers should be sure to follow a code such as the IEEE Code of Ethics [8].
References


