ASSISTIVE SHOGI BOARD

By

Maxim Papushin

Rahul Rameshbabu

Final Report for ECE 445, Senior Design, Spring 2019

TA: David Hanley

01 May 2019

Project No. 48
Abstract

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 meant for learning the game, and one can attempt to play online shogi. However, cross-referencing a long guide to verify that every move is correct gets considerably tedious, and playing online shogi first requires the player to at least be moderately capable to play against real opponents. The complexity of the game remains overwhelming due to the sheer variety of pieces, each with their own 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.
1. Introduction

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.

Shogi has a 9x9 board over chess’s 8x8, but the more significant difference is shogi has a heavier emphasis on counter attacking strategies. Unlike in chess, pieces in shogi 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 community helps support both high-level competition and learning players. 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 the process of learning the game simpler, its popularity could rise significantly.

1.1 High-Level Requirements

● Board must recognize when a piece has been lifted and identify the possible moves for that piece.
● Board must keep a log of valid moves which have been made, which should be exportable to the connected computer.
● Board must display a warning if an invalid move is made, which needs to be corrected before play can continue.
● Connected computer 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.

2. Design

Game State/Compute Processing - The goal of the Game State/Compute Processing subsystem is to receive process input from the AtMega 2560 microcontroller in the Game I/O Processing block and also process any board state correctional data provided by the user to the program running on the computer, in the case of pieces being bumped accidentally. Once it receives game state updates, the program updates its UI and sends information to the AtMega on which LEDs should be updated to display valid moves that can be made.

Game I/O Processing - The Game I/O Processing subsystem’s goal is to handle the large number of sensory inputs, and digital indicator outputs needs for the assistive shogi board. The AtMega microprocessor uses the sensor data to detect when pieces are moved, and controls LED illumination based on the commands it receives from the computer, but does not know about the setup of the game being played. The AtMega communicates with the computer in the Game State/Compute Processing subsystem to update the computer program with information about piece movements that lets the program update the game state. Any detection of a lifted piece should be sent over to the computer, which sends signals in response to the AtMega indicating which LEDs should be illuminated, thus indicating to the players which spaces are valid moves given the piece which was lifted from the board.
**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 controlled by the AtMega, which is in turn signalled to set certain LEDs to light up by the computer. In order to save power, the LEDs will be blinking at high frequencies, rather than always being 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 requirements of the entire board.

**Game Board Sensors** - This block contains a series of photoresistors placed at each position on the 9x9 Shogi board. These sensors each contain a simple voltage divider to calculate an output voltage based on the internal resistance of the photoresistor. The sensors are connected to a series of MUXes within the Game IO Processing subsystem, which pass the sensor data to the AtMega microcontroller.

![Block Diagram](image)

**Figure 1: Block Diagram**

### 2.1 Physical Design Description
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 photoresistors 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.
In addition to the photoresistors, each square will have a two-color LED at one corner of the square. This LED will be made to protrude slightly from the top of the board in order to prevent it from being covered up when a piece is placed on the same tile.

All the electronics will be connected underneath the board, with the only externally accessible parts being a single USB connection from the AtMega to the computer.

![Figure 2: Photoresistor placement, side view](image)

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.
At any time during the game, the pieces which are present will block certain photoresistors and reveal others. Each of these photoresistors will be continuously monitored by the AtMega chip, and changes will be sent over serial to the computer program. Once the computer program receives a change from one of the photoresistors, the program will determine what move the player made based on which tiles have been changed, and will determine whether the move is indeed valid, and respond accordingly.

2.2 PCB Design

Figure 3: Board layout render with visible photoresistors

Figure 4: AtMega 2560 Component PCB Layout
The AtMega 2560 PCB allows us to use Arduino-like microcontroller functionality on our own custom board. To simplify the design, we use a separate USB-to-Serial FTDI chip module to handle programming the IC as well as reading and writing serial data to the device. The components are spaced out evenly on the PCB layout to make routing traces simple. However, the board is designed to maintain order of the analog and digital I/O to a certain degree on the header holes to make routing the large number of LEDs as well as the MUX I/O to the device simple. To see what requirements the AtMega 2560 PCB meets, please reference the requirements in the R&V table (See Appendix A, Table A.2).
The MUX PCB is designed to share 4 select bits among five 16:1 analog MUXes in order to scan the MUX outputs at the same time using the AtMega 2560 with only four digital outputs for all five MUXes.
2.3 Software Design

As part of the design of the board, it was necessary to create a computer program which would take the part of the Game State/Compute Processing subsystem, to interact with the AtMega in order to update the internal game state, as well as to verify which moves are permitted, allowing for corrections when moves are physically made which should not be allowed by the rules of the game.

The structure of the program is described in Figure 8. Three threads run to interact with the AtMega over serial, to interact with a CLI interface for debug and correction, and a main thread which accepts work from the other two threads and uses it to process updates to the game state and FSM.

In addition to the functionality which is purely necessary to accomplish the requirements listed in the R&V table (See Appendix A, Table A.1), the program also includes several debug features made to assist testing the program independently from the rest of the project. These features include simulating sensor input to mimic the input received from the AtMega, as well as a command to force certain LEDs to be set to a certain state to test the LED matrix without needing to interact at all with the photoresistors.

Aside from the program running on the computer for the Game State/Compute Processing subsystem, there is also a program running on the AtMega for the Game I/O Processing block. This program is much simpler, mostly functioning as a loop which iterates through all 81 tiles, for each one turning on the LED if it should be on, checking the state of the photoresistor and sending it over serial if necessary, then turning the LED back off. This LED blinking is done in order to save power, since only one LED is enabled at a time with this system. Making the LEDs look like they are always on did take some optimization, since the loop needs to spend as little time as possible between different iterations to avoid downtime. This was successfully accomplished, and the final version of the AtMega code works exactly as intended.

Figure 8: Program design diagram
3. Design Verification

The sensors which are used to detect whether a piece is present, photoresistors, also known as 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 $5\,\text{M}\Omega$, while the ‘bright’ resistance is in the range of $50\,\text{k}\Omega$ at 10 Lux [8], which is still well below ordinary ambient room lighting, at roughly 50 Lux [9].

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. The final design used a plain voltage divider for this purpose, and provided sufficient responsiveness and accuracy in a reasonably well-lit room. Because shadows could sometimes cause false positives in a room with only a single ceiling light, the board performed best in a room with multiple lights, such as the design lab itself, so this did not prove to be an issue during the demo.

![Photoresistor voltage divider circuit](image)

Figure 9: Photoresistor voltage divider circuit

Figure 8 shows a 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 $50\,\text{k}\Omega$ to $5\,\text{M}\Omega$, we take $R_1$ to be $500\,\text{k}\Omega$.

Some testing was performed using the photoresistors to determine their approximate range of resistance values in various light conditions, the results of which are shown in Table 1 below. From this testing, it was determined that a $47\,\text{k}\Omega$ resistor would be closest to the ideal value to use for these voltage dividers.

Once actually tested, these photoresistors with voltage dividers worked exactly as intended, and the AtMega was able to reliably read the difference in their values when the photoresistor was either covered or uncovered, with nearly no delay, much less than the 2.5s which was originally permitted, but proved to be an unnecessary concern.
<table>
<thead>
<tr>
<th>Light Level</th>
<th>Resistance (kOhm)</th>
<th>Voltage (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covered</td>
<td>197</td>
<td>4.30</td>
</tr>
<tr>
<td>Dark</td>
<td>19.6</td>
<td>1.493</td>
</tr>
<tr>
<td>Bright</td>
<td>13.5</td>
<td>1.10</td>
</tr>
<tr>
<td>Direct</td>
<td>2.68</td>
<td>0.322</td>
</tr>
</tbody>
</table>

Table 1: Photoresistor testing result

\[
\begin{align*}
\text{I}_{\text{out}} &= 0 \rightarrow I_{\text{Rp}} = I_{\text{R1}} \\
V_{\text{out}} / R_p &= (5 - V_{\text{out}}) / R_1 \\
V_{\text{out}} / R_p + V_{\text{out}} / R_1 &= 5 / R_1 \\
V_{\text{out}} * (1 / R_p + 1 / R_1) &= 5 / R_1 \\
V_{\text{out}} &= 5 / (R_1 / R_p + 1)
\end{align*}
\]

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

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

Equation 1: Photoresistor voltage divider calculations

As seen in Equation 1 from the two values for \( V_{\text{out}} \) based on the resistance of the photoresistor, the two voltages should be easily distinguishable as digital low/high, though some testing may be necessary to determine the ideal resistance for \( R_1 \) for the particular photoresistors which end up being used in the design.

As expected, one of the most difficult part of the project’s physical construction was 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 MUX PCB was very large, and the added size constraint of having all electronics fit underneath the board further complicated the wiring. This wiring worked without issue during the demo, however it would be very difficult to repair should one of the wires become loose, due to the physical inaccessibility of the wires.

Each square on the board used a three-color LED, of which two colors were used, unlike the originally considered single-color LED in each corner. Instead of individually connecting the LEDs or using LED control circuits for every single LED under the board, this problem was instead solved by using an LED matrix setup for the LEDs, mapping 9x9 with two colors into an effective 18x9 matrix, which worked without issue using only 27 digital IO ports on the AtMega.

As predicted, the physical construction of the board was not difficult, as the only real requirement was that the LEDs and photoresistors fit through the holes, which was indeed accomplished. Although it was not strictly necessary, some time was spent measuring the holes out to be placed in as close to a perfect grid as possible, followed by drilling them with a household power drill, which proved sufficient as opposed to the original plan to use a drill press.
4. Costs

4.1 Parts

<table>
<thead>
<tr>
<th>Part</th>
<th>Manufacturer</th>
<th>Retail Cost ($)</th>
<th>Bulk Purchase Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shogi Board + Pieces</td>
<td>Generic</td>
<td>22.00</td>
<td>22.00</td>
</tr>
<tr>
<td>100 CdS Photoresistors</td>
<td>MCIGICM</td>
<td>7.99</td>
<td>0.06 per resistor</td>
</tr>
<tr>
<td>100 3-Color LEDs</td>
<td>Generic</td>
<td>8.99</td>
<td>0.07 per LED</td>
</tr>
<tr>
<td>AtMega 2560 IC</td>
<td>Atmel</td>
<td>12.35</td>
<td></td>
</tr>
<tr>
<td>5 16:1 Analog MUXes</td>
<td>Texas Instruments</td>
<td>4.40</td>
<td>0.375 per MUX</td>
</tr>
<tr>
<td>AVR Programmer for Flashing Arduino Firmware</td>
<td>Generic</td>
<td>5.99</td>
<td>5.99</td>
</tr>
<tr>
<td>Mini-USB to Serial</td>
<td>FTDI</td>
<td>5.79</td>
<td>5.79</td>
</tr>
<tr>
<td>Miscellaneous Capacitors, Resistors, LEDs, and Diodes</td>
<td>Generic</td>
<td>~ 5.00</td>
<td>~ 4.60</td>
</tr>
<tr>
<td>Linear Op-Amp</td>
<td>Generic</td>
<td>0.80</td>
<td>0.34</td>
</tr>
<tr>
<td>5V to 3.3V Voltage Regulator</td>
<td>Generic</td>
<td>0.61</td>
<td>0.26</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>73.92</td>
<td>53.86</td>
</tr>
</tbody>
</table>

Just a note that the bulk price of $53.86 is fairly reasonable for a shogi board since moderately decent non-plastic shogi boards have a starting price of around fifty dollars.

4.2 Labor

Labor Cost = Average Electrical Engineer Salary ($33) x Estimated Amount of Hours Worked (No multiplier is used since these are final estimates)

- Took approximately 25 hours to design and solder the AtMega PCB. This was mainly due to the complexity of getting the AtMega 2560 to work in comparison to its 328P counterpart plus soldering a TQFP100 package is non-trivial
- Coding and optimizing took approximately 2 hours for the Arduino code
- Soldering the LED matrix took approximately 12 hours over 2 sessions
- Soldering the photoresistors took approximately 8 hours over 2 sessions
- To verify, make, and prep the MUX PCB for fabrication through PCBWay took approximately 2-3 hours.
- Connecting all wires to MUX PCB took 10 additional hours in total
Therefore, total labor cost per teammate is $1980.00.

4.3 Total Production Cost
The total production cost for the prototyping of the assistive shogi board took a total of $4033.92.

5. Conclusion

5.1 Accomplishments
The entire board works as intended, being effectively fully playable, though at times unintuitive due to misdetections of certain movements, and the requirement of manually inputting commands to make moves which play pieces from the hand is somewhat inconvenient. That being said, the board is effective in allowing beginner players to get used to the piece movements, which would gradually lead to learning the types of pieces by repetition.

The physical construction also matched the original goal of fitting all wires underneath the board. Although this was never a formal requirement for the project, it does contribute to the completed board looking much neater than if there were loose wires around the board. Overall, the board looks surprisingly neat for what is effectively a hobby project, with all wiring being laid out on even grids. If a single circuit board were used for both the AtMega and MUXes, it would be possible to further clean it up, however modularity was necessary for making this project prototype, so multiple circuit boards were used instead.

5.2 Uncertainties
Due to the photoresistors’ reliance on adequate lighting, the board does not work reliably under certain lighting conditions. In particular, having a single ceiling light may cause shadows to trigger false positives as the player’s hand blocks out some photoresistors, which the board attempts to interpret as a move. Possible solutions would include putting several lights at each corner of the board, elevated roughly 20cm, aimed down towards the photoresistors. This would likely improve the reliability compared to having only a single light source, since any square without a piece would still have at least one light source hitting it, making shadows less of a concern.

The internal wiring, particularly that of the LED matrix, is very delicate, and may be prone to shorts. While these shorts should not be able to cause any damage to the board, they may cause certain LEDs to be enabled when they should not be, or vice versa.

5.3 Ethical considerations
Although the board will be designed to record the game which is being played by keeping track of the moves made, all game data is made available only to the players themselves, and are not distributed or saved if the players themselves do not want to keep said data.
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.

Our project operates under a low voltage environment, minimizing many electric safety concerns, though not removing them completely. Short circuits could possibly occur in the event of faulty wiring, however this is unlikely to cause any practical hazard aside from flickering LEDs or misreading sensors due to the power limitations built into USB.

One possible issue is the issue of undervoltage in the case of large loads [6]. 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 [7].
References


# Appendix A  Requirement and Verification Table

## Table A.1: R&V for Computer Device Block

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Must continuously store the game state (piece positions and their validity), including both pieces on the board and whether a piece is currently being moved. | 1. Run the game program on the computer while connected to the board.  
2. While making several moves, use debug features in the program to dump the game state between each action (lifting or placing a piece).  
3. Verify that the state output correctly reflects the physical state of the board.                                                                 |
| Program must allow the user to save or load a game state to/from a file on the computer.                                                   | 1. Run the game program on the computer while connected to the board.  
2. Save the game state to a file using the program.  
3. Make several valid moves in the game.  
4. Use the program to load the previous state.  
5. Move pieces on the board as indicated by the program to restore the previous state.  
6. Dump the game state to verify that the layout matches the physical layout of the board, and the layout at the time of the save. |
| Program must allow for captured pieces to be selected and replaced on the board during play.                                                 | 1. Run the game program on the computer while connected to the board.  
2. Load a game state with a captured piece.  
3. Use the program to select which piece to play, then physically play the piece.  
4. Dump the game state to verify that the state matches the physical layout with the played piece on the board. |
| Must detect invalid moves made during play and inform the user, requiring correction before allowing play to continue.                        | 1. Run the game program on the computer while connected to the board.  
2. Pick up a piece, then move it to an invalid tile for that piece.  
3. Verify that the program displays a warning indicating the invalid move.  
4. Move the piece to a valid location from its initial position.  
5. Verify that the error goes away and play is resumed. |
## Table A.2: R&V for AtMega 2560 Block

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Must detect a piece being lifted or placed at any one of 81 spaces within 2.5 seconds of the move. | 1. Modify the computer program to output debug information to a console whenever a piece movement is received.  
2. Perform several piece movements:  
   a. Lift or place a piece, and simultaneously start a stopwatch  
   b. Verify that the debug message appears within 2.5 seconds.  
   c. Verify that the debug message contains the correct piece location, and the correct direction (placed or lifted) |
| Must act on any LED state commands to update the board LEDs within 0.5 seconds of the command being sent. | 1. Modify the computer program to send arbitrary LED state commands to the AtMega.  
2. Use the program to send a pattern of LEDs to the AtMega, and simultaneously start a stopwatch.  
3. Verify that within 0.5 seconds, the LEDs on the board match the state which was sent in the command. |

## Table A.3: R&V for LED Array Block

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Each of 81 squares must be individually set as either green, blue, or off, independent of all other squares. | 1. Connect LED control signals to a standalone Arduino board with equivalent chip.  
2. Run a program which provides control signals to the LEDs in order to draw some pattern on the board.  
3. Verify that the LEDs match intended patterns. |
| Total average power consumption for all LEDs on board should not exceed 250mW for any configuration of enabled LEDs. | 1. Connect GND and VCC to a metered DC power supply as 0V and +5V.  
2. Connect LED control signals to a standalone Arduino board with equivalent chip.  
3. Run a program which provides control signals to the LEDs in order to set all tiles to green.  
4. Verify that power usage recorded on power supply does not exceed required maximum value. |

## Table A.4: R&V for Photo-resistors + Voltage Divider Blocks

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor output voltage records greater than 4V</td>
<td>1. Connect the Vout from one photoresistor on the board to a voltmeter.</td>
</tr>
</tbody>
</table>
when a piece is present on the tile, and less than 1V when no piece is present on the tile.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Sensor output can be accurately recorded as digital high (1) when a piece is present on the tile, and digital low (0) when no piece is present on the tile. | 1. Connect the Vout from one photoresistor to a standalone Arduino board.  
2. Run a program which reads the sensor input on the connected pin and outputs it to the console as either 1 or 0.  
3. Verify that console output matches the actual state of the piece on the board. |

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Digital value of a sensor is correctly updated within 2 seconds of a piece being either placed or removed. | 1. Connect the Vout from one photoresistor to a standalone Arduino board.  
2. Run a program which reads the sensor input on the connected pin and outputs it to the console as either 1 or 0.  
3. Place a piece on an empty tile, and run a timer at the same time the piece is placed.  
4. Verify that the console output becomes a steady 1 within 2 seconds.  
5. Repeat steps 3-4, removing a piece instead of placing it. |

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Should be able to update value of MUX between two photoresistors with a 1-3 ms delay on the analog MUX. | 1. Have one photoresistor covered and another exposed photoresistor connected to a MUX wired to some microcontroller for testing.  
2. Start by reading the value from one photoresistor through the MUX.  
3. Have the microcontroller switch over to reading the other photoresistor and measure the time elapsed for the switch over to occur. Since measuring time itself is hard, using delays in the code can be another way to verify this requirement. |

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Should be able to update the value of MUX between 16 photoresistors in under 3 ms | 1. Have half the photoresistors covered and another half exposed connected to a MUX wired to some microcontroller for testing in an alternating fashion.  
2. Start by reading the value from one photoresistor through the MUX.  
3. Have the microcontroller switch over to reading the next |
photoresistor and measure the time elapsed for the switch over to occur. Since measuring time itself is hard, using delays in the code can be another way to verify this requirement. Keep cycling through the photoresistors till all sixteen are read.