Ultimate Chess

Authors: Yoorae Kim, Demi Lee, Anoushka Tiwari: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system that lets a person play chess against an AI on a physical chess board without any software requirements. The system simulates play against a fast human player that makes moves within 24 seconds.**

***Index Terms*— Background Subtraction, Canny, Chess, Computer Vision**

1 INTRODUCTION

Chess is one of the most popular games in the world. Over 600 million people worldwide enjoy the classic game. The elderly population of the world also enjoys chess. According to some studies, it even helps prevent Alzheimer's and mental decline by stimulating the brain. Despite all the benefits Chess has on them, the elderly are not always able to find someone to play with them. They are also often not as comfortable with technology to be able to enjoy a game against an AI on a mobile app. Some feel that a mobile app simply doesn't provide the full chess experience.

Our smart chess board attempts to make chess more accessible to the elderly by providing a simple and easy-to-use way to play chess against an AI opponent. We provide a custom chess board that comes with a camera and LEDs. The camera clicks pictures of the board when the player is done with their move and we detect the move with 99 per cent accuracy. The move of the player is displayed within 10s, after which

Then within 24s, we display the AI player's move on the board using LEDs that light up the initial and final position of the piece.

While there exist automatic chess boards in the market today, most of them require smartphone connectivity and need the player to use apps. Our target demographic is people who do not have access to a smartphone or simply don't want to use it. The simplicity of our product is an advantage over current smart boards.

2 DESIGN REQUIREMENTS

The International chess body FIDE has a standard time limit of 90 minutes for the first 40 moves, followed by 30 minutes for the rest of the game. Since our project is a human vs AI game instead of a human vs human game, we use a faster time control. This is because unlike playing against an actual human player, playing against an AI does not allow the person to talk to the opponent while they are making their move. This would make the wait times between moves boring. A shorter time between moves also lets us accommodate people who want a quick game while still having time to think while the opponent is playing, as they would in a real chess game. One popular speed chess tournament is the FIDE World Rapid Championship, where all moves are played under 15 minutes for each player. We divide this by the average total number of moves for each player 38 to get the timing constraint for a single move i.e. 15 min / 38 move = 24s.

We thus have the following overall timing constraint:

$$tM + tL + tA \leq 24s \tag{1}$$

where tM is the time it takes for move detection (s), tL is the time it takes to determine if the move is legal (s) and tA is the time it takes to get and display the AI's next move from the AI engine (s).

The main areas of the project are Move detection, determining whether a move is legal, coming up with the AI move and displaying it on the LEDs. The requirements based on area are as follows:

2.1 Move detection

Detecting the player's move is a core part of the project. There are 3 quantitative requirements here: the detection time, the detection accuracy, and the number of tries it takes to come up with the correct detection. The detection time must meet the constraint described in equation (1).

2.1.1 Number of Retries before the CV detects the move with the required detection accuracy

We want the user to retry their move at most 2 times before we detect it with 99 per cent accuracy.

This requirement is due to the fact that we want to keep the human in the loop during the move detection to increase accuracy. When we finish detecting the move using the CV pipeline, we display the move the CV detected on the board by lighting up 2 LEDs (one for the initial position of the piece and one for the final). If this move is correct, the user does not need to do anything, and the detected move will go through the rest of the pipeline in 10 seconds. The reason we give the user 10 seconds here is due to the fact that our target demographic is the elderly, who often have a slow reaction time. If the move is wrong, the user has 10 seconds to press a button indicating this, and then make the move again. This helps mitigate issues with the piece not being within the edges of the square or a bad image. We want the user to try 3 times before we end the game. The reason we settled on this is because there are 2 main sources of error: the piece not being placed properly or issues with image quality. 2 extra tries would help mit-

igate these issues. The test for this happens with the test for detection accuracy described below.

2.1.2 Detection accuracy

The target detection accuracy is 99 per cent. This accuracy is with the assumption that the user will let us know if the move we detected is incorrect. We aim for 99 per cent and not 100 per cent because we are limiting the number of retries the user has to make for the sake, so there may be situations in which we do not get the move right even with the human in the loop. The reason we require this high accuracy is because our entire project is dependent on the detection being highly accurate. If we detect the move wrong, our illegal move detection will not be meaningful. The AI move generator will also be provided the wrong state of the board and generate a nonsensical move. All in all, the entire user experience rests on an extremely high detection accuracy.

While 99 per cent may be too high of an accuracy given that we are relying on canny edge detection which is only 91.56 per cent accurate, the retry mechanism should help in case there is noise in the picture. Additionally, the accuracy of all the edges detected does not matter for our case as we are only use the edge detection algorithm to determine the grid of the chessboard. We will apply the Hough transform to the edges detected to get the horizontal and vertical lines of the grid. We will essentially ignore all the edges other than the ones making up the grid. The edges making up the grid are simple edges, which should further improve the accuracy.

To verify this metric, we have to ensure that we achieve 99 per cent accuracy within 3 tries. The test plan is to simulate a real game by providing configurations from the beginning, middle and end of the game. Then, we simply make moves at most 3 times to get the right detection. If the detection is still wrong after the final try, it is counted as an error.

2.1.3 Distance of center of piece from center of square

This requirement is essentially a measure of how close the center of the piece could be to the edge of the square it is placed in for it to still be detected correctly. This is a measure of how robust our CV is and is required because we cannot expect the player to perfectly place the piece on the center of the square every time. The requirement here is that as long as the piece is fully inside the square it should be detected correctly in that square 99 per cent of the time. This means that

$$d \leq l - r \tag{2}$$

where d is the distance of the center of the piece from the center of the square, l is the length of the side of each square on the board, and r is the radius of the piece. For our case, l = 15/8 inches, so we get:

$$d \leq 1.875 - r \tag{3}$$

where d is the distance of the center of the piece from the center of the square in inches and r is the radius of the piece in inches.

2.1.4 Detection time

This requirement stems from Equation (1) that constrains the total time that the CV, illegal move detection and game playing AI can take. Since the AI takes at most 10 seconds and the illegal move detection at most 2 seconds, it leaves us with 12 seconds for detection time. The way this is tested is similar to the tests in 2.1.2, except we time each run through the move detection pipeline. For each set of up to 3 tries, the maximum time across all tries is taken as the time of the detection.

2.2 Game playing AI

We will be integrating an already existing Chess AI engine to our project. There are multiple options available, but our major requirements for Chess AI engine was
1. Available for public use (open source)
2. Fast response time (<10s)
3. Offers varying levels of difficulty (optional)

Our final decision stood on Stockfish engine, which offers 8 varying levels of difficulty, response time ranging from 50ms to 400 ms, and publicly posted on github as an open source.

2.3 LED

Given coordinate pairs from the game software, the LEDs corresponding to the coordinates should light up. Additionally, to address the ambiguity of castling, the LEDs should be able to light up in two different colors. Based on the requirements, we need 64 RGB individually addressable LEDs for each square of the chessboard. To verify the correctness of LEDs, we will visually inspect that the appropriate LEDs light up given a set of random coordinate pairs.

3 ARCHITECTURE OVERVIEW

Figure 1 shows a block diagram of the architecture. Figure 2 is a more detailed interaction diagram of the same.

The player will play on a physical chessboard that has an 8x8 LED matrix installed under it. A camera will be mounted directly above the board. The Stockfish engine is initialised when the user is ready to begin the game.

When it is the player's turn, they will make a move by moving their piece on the chessboard. Once they make their move, they will press a button mounted on the chessboard. This will signal the camera to take an image of the board. The image passes as an input to the CV pipeline implemented on the Raspberry Pi.

The CV thread determines the user's move and the Raspberry Pi lights up the LEDs corresponding to the initial and final square. In case the CV was not able to detect the move, the raspberry Pi lights up all LEDs red to tell the user to retry the move twice. After lighting up the moves, the system waits 10 seconds for the user to click a button if the move was incorrect. If the user does not click a button, the move is assumed to be correctly detected and the move detector implemented on the Raspberry Pi passes control to a legal move detector. The legal move detector gets the coordinates of the current move and the current state of the board. The state is maintained as an 8 by 8 matrix where each element $\in$ [-6, 6]. 0 indicates the presence of no piece in the square. 1 and -1 indicate the presence of a black or white pawn. 2 and -2 indicate the presence of a black or white rook. Similarly, we have representations for Knight, Bishop, Queen and King.

The legal move detector checks if the move is valid. If not, the Raspberry Pi makes all LEDs light up red and the game is over. If it is, the state of the board is updated internally. This is done by simply making the move that the player made. The new move is then fed to the Stockfish chess engine which comes up with the next move. The internal state of the chess board is again updated. The move is then displayed to the user using the LEDs. The total time taken for determining the users move and determining the next move of the AI is 24s. The retries and 10 second wait time are not counted in these 24s.

4 DESIGN TRADE STUDIES

4.1 Stockfish Chess AI Engine

We are integrating already existing chess AI 'Stockfish' into our project. Currently there are numerous options available for chess AI engines, but Stockfish has suited our needs the best. Our major requirements for the chess engine were availability to public use, and fast response time to decrease latency. Our optional requirement was offering various levels of difficulty to meet the needs of a wider range of users. Considering these requirements, open source Stockfish AI with 8 varying levels and response time ranging from 50 to 400ms was the optimal option for us.

4.2 WS2812B LED Strip

An 8x8 LED matrix display will be constructed and installed under the chessboard. The 64 LEDS would need to be individually programmable to correctly display the AI's move. To overcome the ambiguity of castling, the LEDs should be able to light up in at least two colors.

We chose to use an LED strip over individual LEDs because LED strips are generally cheaper. Additionally, it is less circuit work for us to use the LED strips since the LEDs come in wired together.

The main factor that determines the cost of LED strips is how well it handles the loss of color accuracy due to voltage drop. 5V WS2812B is the cheapest most common type of LED strip. For WS2812B, voltage drop happens after 2.5m or 150 LEDs. Since we are using a relatively small number of LEDs, only 64 LEDs for each square, voltage drop will not be an issue for our case. Therefore, we are able to use the most cost effective 5V WS2812B LED strip for the project.

4.3 Board LED Integration

In our project, AI moves are communicated to the player by lighting up the appropriate LEDs on the board. There were two possible ways to achieve this. The first way was to buy a standard chessboard set and drill small holes in each of the squares to allow LED lights to go through. The second way was to create our own custom chessboard using translucent acrylic sheets allowing the LED lights to pass through. We chose to go with the second approach because the drilled holes in the first approach may affect the performance of computer vision when detecting the squares and pieces. Moreover, there may be a case where the chess pieces block the drilled holes causing the players not able to see which LEDs have lit up.

Using the second approach means that we have to construct our own chessboard. The top of the board is made using translucent acrylic sheets that is non see-through but allow light to pass through. 64 squares are laser cut using white and green acrylic sheets. The squares are welded into one piece using acrylic weld-on. An 8x8 LED matrix is installed under the chessboard. LEDs are separated from each other using grids which are laser cut from plywood.

4.4 Computer Vision for move detection

4.4.1 Piece detection v.s. change detection

There are 2 main approaches to determining the move the user made. The first is detecting every piece on the board and figuring out which piece changed position. This provides both the move information and information about which piece is at which square directly. The other approach, i.e. the one we employed, is simply detecting changes in the board and treating pieces as nothing but a set of edges. The edges themselves do not give us any significant information about the piece and we do not try to identify the piece based off of what it looks like. Instead, we identify pieces by tracking their positions through time. Since we know the position of each piece at t=0 (i.e. before any move is made), we are able to determine the new position of a particular piece at t=T simply by combining the position of the piece at t=T-1 with the move made by the human and the AI at t=T to determine the new position of the piece.

There are 3 benefits to this approach:

1. Invariance to the exact shape of the piece: Since we do not rely on the shape of the piece to determine its position, we are able to support different kinds of piece shapes. This is useful because it lets the user play with their own

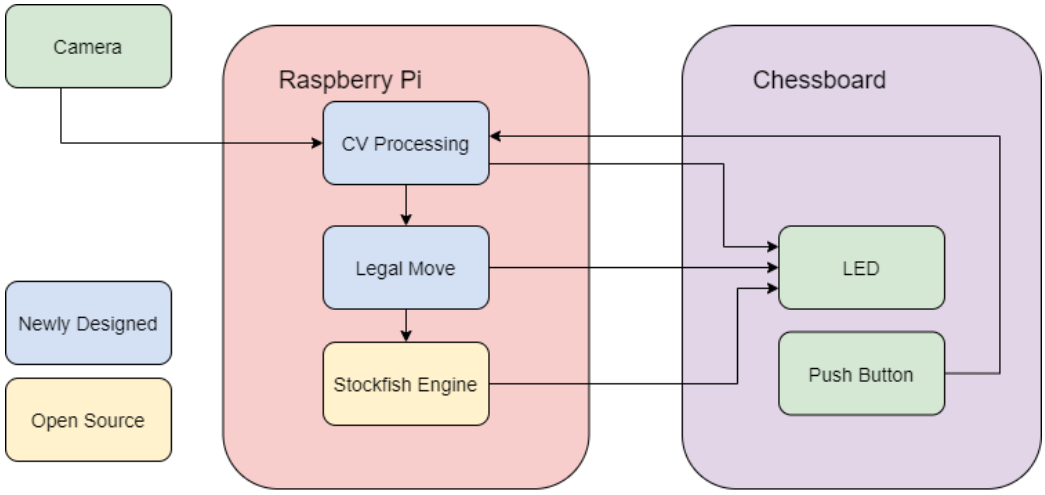

Figure 1: Block Diagram

chess pieces instead of us having to provide them. This cuts costs for us because we do not have to provide the user chess pieces. This also makes the set less expensive if the user already owns a regular chess set.

2. Only one top view required: Since we do not need to detect the identity of the piece using its shape, we can simply mount a camera on the top of the chessboard to get a top-down view of it. This is less intrusive than, for example, having to mount 3 or 4 cameras all around the chessboard to ensure that we have enough views of a piece to be able to get an unoccluded image that lets us detect its shape.

3. Less computation: Since we are not using the edge profile of a piece to figure out which piece it is, we save on computation that helps us stay within the time constraints described in equation (1).

4.4.2 Edge detection algorithms

The edge detection algorithm is what gives us the edges of the chessboard, the edges of the squares, and the edges of the piece inside the square. Here, we want extremely high accuracy, even if it comes at the cost of time complexity. This is because the results of edge detection feed into the entire move detection pipeline, so we need as high an accuracy as we can get here as the mistakes cannot be made up for later in the pipeline. Therefore, we picked the Canny edge detector which has the highest accuracy.

A table comparing the edge detectors is shown in Figure 3 [1]. As seen in the figure, the Canny operator has the least sensitivity to noise and the least number of false edges. Since we are dealing with real world images, a low noise sensitivity is very important to us. Additionally, since we use edges to discretize the chess board into squares, we must not have false edges as that would lead to the wrong grid being formed which will be disastrous for the entire move detection system.

5 SYSTEM DESCRIPTION

5.1 Move Detection

5.1.1 Edge detection algorithms

The flowchart for edge detection is shown in Figure 4. First, the image is blurred to make edge detection more accurate and remove noise. Then, we use Canny edge detection to get the edges of the image. After that, we apply the Hough transform to get the edges of the grid. Once we have the chess board discretized, we begin the process to determine the user's move. We do this by iterating through each of the squares on the board and comparing the state of the square at current time t to that at time t-1. We do this by subtracting the image at time t from that at t-1. There are 2 possible cases for moves:

1. The player moved a piece from one square to another unoccupied square. This means that we must be able to find a square that did not have edges present inside it at time t-1 and does at time t. We must also be able to find a square that had edges present inside it at time t-1 and does not at time t. This would indicate a move from the square that does not have edges at time t to the square that has edge at time t.

2. The player moved a piece from one square to another square that was previously occupied by the opponent. In this case, there will be at least one square where the edge profile changed between time t-1 and time t (i.e. the square at which the piece being moved initially was). To determine the position this piece moved to, first the strategy of trying to figure out if the edge profile of another square changed is used. However, this strategy may not return a square in the situation where the piece captured is the same piece type as the opponent's piece that was already there (for e.g. if a pawn captures a pawn) as the edges would be very similar in both cases. In this case, we depend on color changes between time t-1 and time t. Since the piece captured was a different color than the capturing piece, we simply determine which square had a change in color inside it between

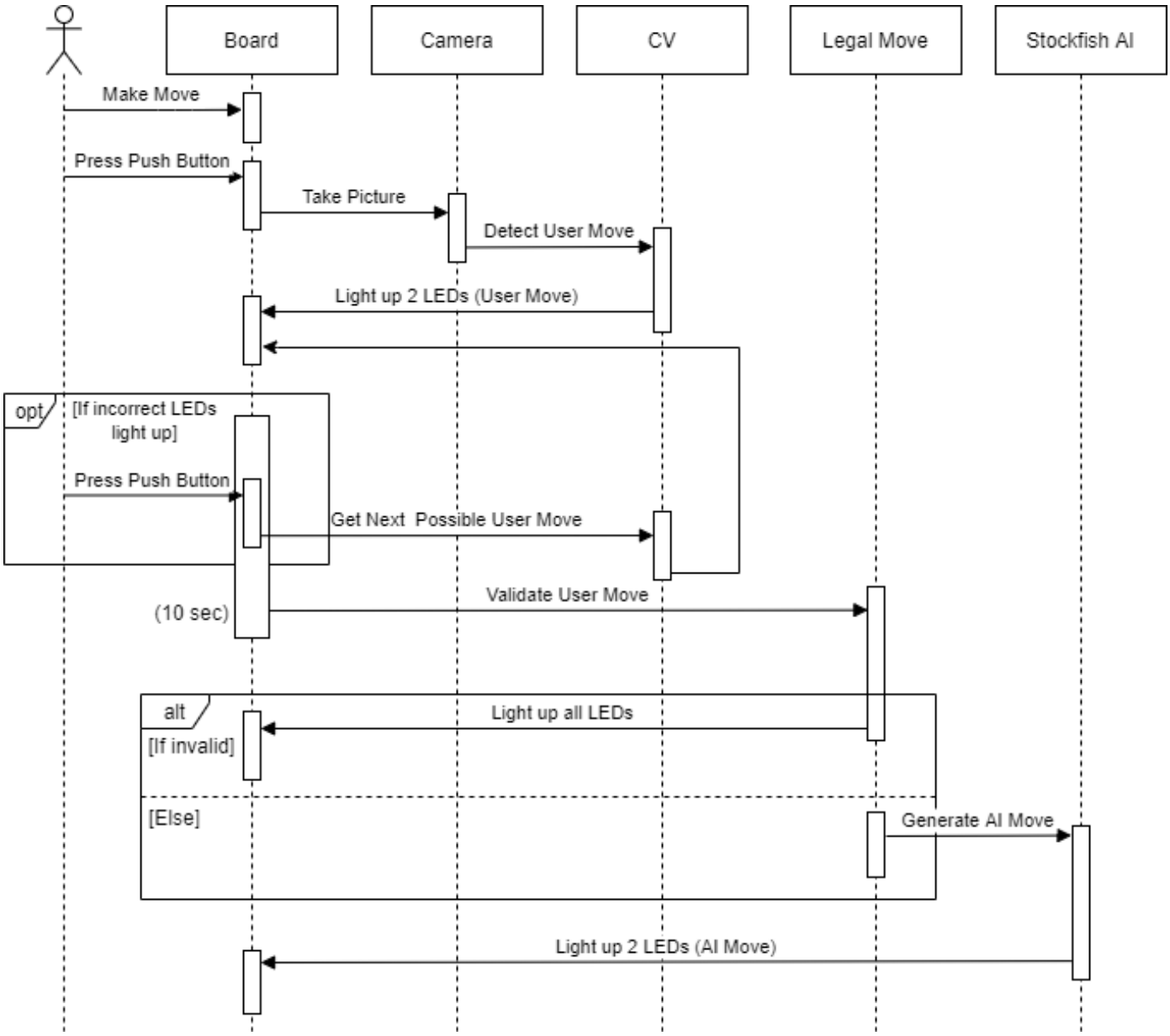

Figure 2: Sequence Diagram

S.NO	Opertor	Complexity		Noise sensitivity	False Edges
		Time	Space		
1	Sobel	lower	high	Less Sensitivity	More
2	Canny	high	high	Least Sensitivity	Least
3	Robert	high	high	Sensitivity	More
4	Prewit	low	lower	Least Sensitivity	More
5	Laplacian of Gausian	low	least	Least Sensitivity	More
6	Zero crossing	low	less	Least Sensitivity	More

Figure 3: Comparison of edge detectors

time t-1 and time t.

5.2 LED with Raspberry Pi

The WS2812B RGB LED strip is controlled by the Raspberry Pi using the rpi_ws281x library, which is also known as Neopixel library. In addition, a 5V power supply is required. The single data line of the LED strip is connected to the GPIO pin on the Raspberry Pi. The ground of the power supply is connected to both the ground wire from the strip and the ground pin on the Raspberry Pi. The 5V output of the power supply is connected to the 5V voltage wire from the strip but not to the voltage pin on the Pi.

6 PROJECT MANAGEMENT

6.1 Schedule

The major requirement of our project is to be able to detect user's moves through CV image processing. Thus, after finishing the construction of the physical chess board, the first several weeks of the schedule is dedicated to constructing our CV algorithm and testing it. As soon as we conclude that our CV algorithm correctly detects moves, we will begin integrating Stockfish engine and valid / invalid move logic into our project. Once we are done with the final integration, we will focus on testing our project's functionality, and its responses to possible edge cases. Our detailed schedule is in Figure 5 at the end of the document.

6.2 Team Member Responsibilities

The areas of major concentration are physical board construction and hardware assembly for Demi, CV image processing for Anoushka, and integration of Chess AI and logic for Yoorae. So far, Demi has laser cut sample square pieces that we will be using for the construction of the final physical chess board. Anoushka had experimented with different edge detection algorithms for sample images of chess boards, and Yoorae wrote the logic for valid / invalid moves. Anoushka and Yoorae are now working on background subtraction algorithms to detect changes of a board state while Demi focuses on hardware assembly including communication of RaspberryPi and camera, circuiting user input button, and timing display on LCD board.

6.3 Budget

The details of purchased components and their cost is attached in the table below. So far, we have ordered every component listed in the document and all of them have arrived. We will be using all of the components listed for the project. It is possible to make additional purchases on acrylic board and wooden sheets for the final construction of the physical chess board, but our total cost would be well managed under budget limit considering the possible additional purchases.

6.4 Risk Management

Most of the risks that might arise in our project are related with image detection. CV might not accurately detect edges of the squares on the chess board. A possible mitigation strategy would be to redesign our physical board with more contrasting alternating colors of squares. There is a chance that CV might not accurately represent the board state after the user has made a move. The cause of this risk might be inconsistent lighting, or interruption in a photo such as the user's hand. To mitigate this risk factor, we included a push button that the user can alert the camera to capture the board so that the photo will be taken in a more consistent state. There is a chance that CV image processing and AI's move generation might cause a latency bottleneck. Again, we are aiming for a response time that is short enough for a user to enjoy the game. To mitigate this risk factor, we are using an efficient chess engine with a fast response time. We are also down-scaling 720p HP

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Chess board and pieces set	N/A	Chess Armory	1	$28.99	$28.99
Logitech C270 Webcam	C270	Logitech	1	$26.92	$26.92
Raspberry Pi 4 (Capstone Inventory)	B	LABISTS	1	$0	$0
LED Strip	WS2812B	BTF-LIGHTING	1	$22.88	$22.88
5V 10A Power Adapter	N/A	ALITOVE	1	$23.99	$23.99
Cast Acrylic Sheet White	N/A	McMaster	1	$31.91	$31.91
Cast Acrylic Sheet Green	N/A	McMaster	1	$31.91	$31.91
Weld-on 4	N/A	WELD-ON	1	$17.85	$17.85
					$184.36

photos from our logitech camera before image processing to reduce response time from CV.

7 ETHICAL ISSUE AND USE CASE

The current COVID-19 pandemic has excluded the elderly and others who are uncomfortable with modern technology from society, since social interactions based on physical contact have immensely decreased. Even before COVID-19, exclusion of older adults from fastly developing technology has been an ethical issue. Our project aims to provide leisure to anyone who is experiencing such exclusion from modern society by developing a system that allows users to play a chess game without any physical contact with another individual and any software-based interaction.

8 RELATED WORK

Our project was initially inspired by the smart chess board 'Square off' when formulating ideas of the project. 'Square off' is an automated chess board that users can make a move on a physical chess board, and AI will respond with an automated move of pieces through circuited magnets inside the board. To reduce the complexity of the hardware model, we pivoted to the idea of updating user's moves by image processing from a top view camera and displaying AI's move on LEDs of the board. The group 'Chess Teacher' from last semester's ECE Design Experience course had a similar project with us. They detect user's moves through image processing of images taken from a top view camera, and display the AI's moves through their front end UI. The biggest difference between our project and 'Chess Teacher' will be the display of AI's moves. Our major goal is to get rid of any software components from user experience to remove exclusions from any users, such as elderly, who are having difficulties with software components. The users of ultimate chess will not be required any interaction with front-end UIs, and every communication in the game will be fully physically visible.

References

[1] S.K. Katiyar. "Comparitive analysis of common edge detection techniques in the context of object extraction". In: *IEEE TGRS Vol.50 no.11* (2012), pp. 77–78.

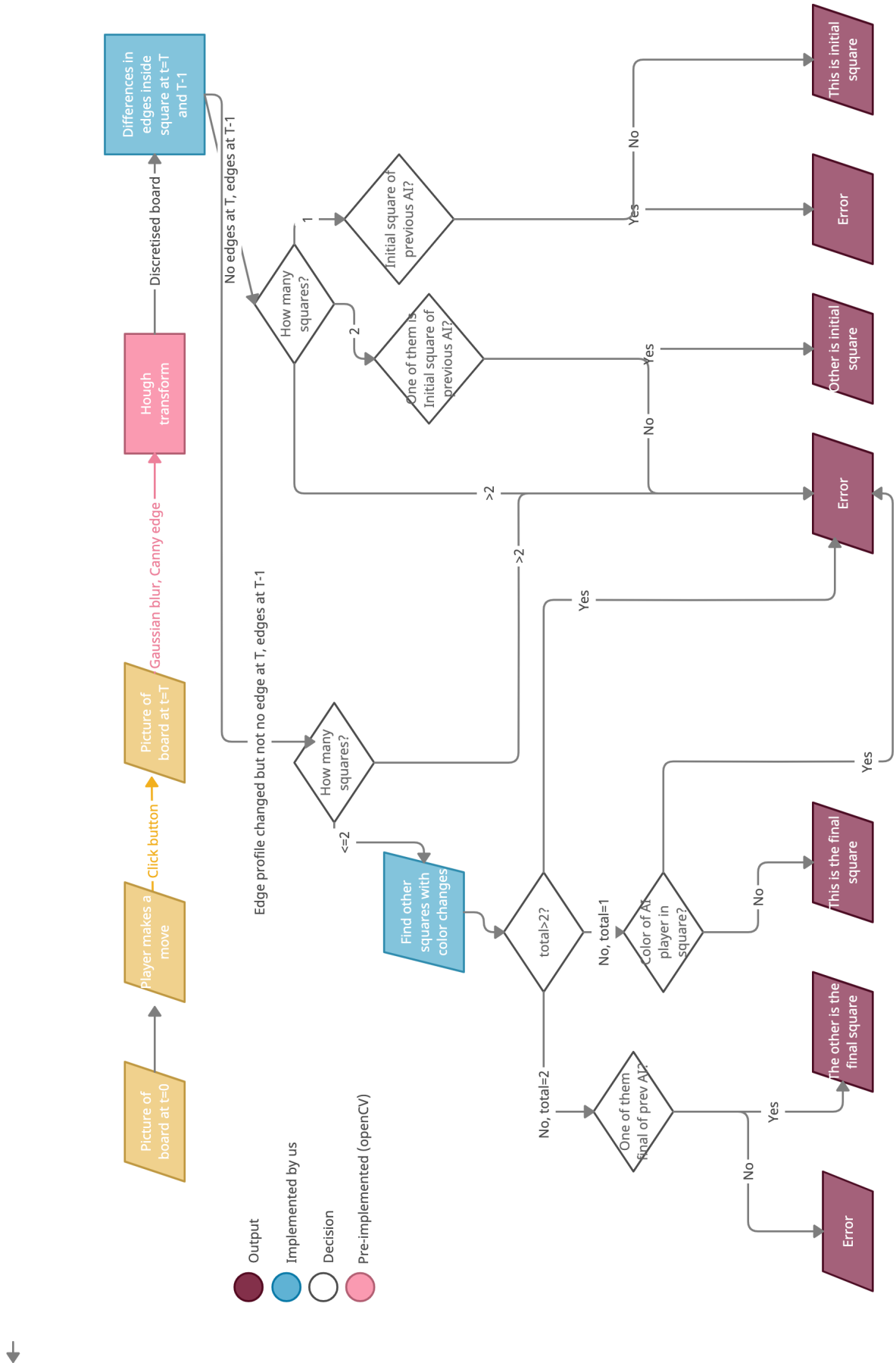

Figure 4: CV Flow Chart

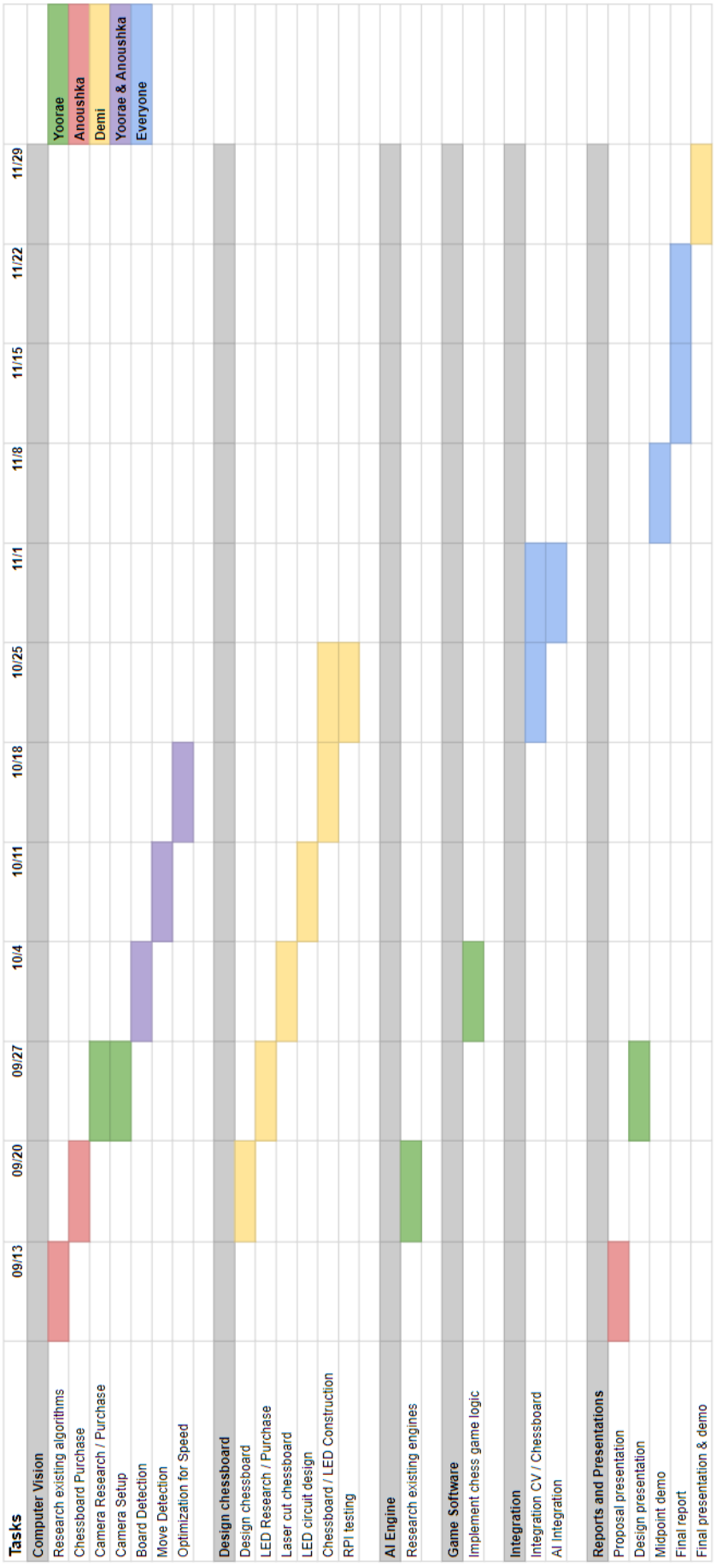

Figure 5: Schedule