Chess Teacher

Author: Michael Cai, Joseph Chang, Jee Woong Choi: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of teaching people how to play chess. The system is able to play chess against a human being in various levels and also recommend moves to help them improve their performance. The system is built on a real chessboard with a camera to detect its moves and the computing is done using a Raspberry Pi and FPGA. By having a real chessboard, the system allows for a realistic environment for the users. By playing against an artificial intelligence, the system allows solo play.**

***Index Terms*—Artificial Intelligence, Design, Chess, Computer Vision, FPGA, Game Tree, Open CV, PyGame, Raspberry PI, Real Time Board Analysis, Stockfish**

I. Introduction

The Chess Teacher is motivated from learning how to play chess in this pandemic and the chess boom after the Netflix's historical drama "The Queen's Gambit" became famous. In the COVID era, it is hard for people to learn how to play chess from a human coach or play over the board with others. We wanted a way to solve this issue when people are desperate to learn how to play chess after the boom from the Netflix drama. Our group aims to create a Computer Vision based Chess Artificial Intelligence (AI) which can compete against a human player and teach various moves while a user is playing the game.

Computer Vision based Chess AI can give the following advantages. First, people can save money from the in-person tutoring. Because the Chess Teacher will recommend various moves and play against the human player, the user does not have to learn from a human coach which can cost quite a lot. In addition, our approach definitely helps social distancing in the pandemic since people do not have to meet each other to play chess. Lastly, it will be easy for the users to develop their chess skills since we are planning to create different levels of Chess AI which users can try competing against the AI in different levels. Most importantly, users can play with physical chess boards and chess pieces. There are chess games available online, but users often feel like they are not playing chess but some computer game. However, because our Computer Vision chess board detector can detect changes on the board, users can play with actual chess sets which help them feel like they are in more competitive settings such as tournaments.

Our goal for this project is to let people enjoy playing and learning chess with physical chess boards and pieces without any help from other people. In order to achieve this goal, we need to detect the board accurately and efficiently through a Computer Vision Algorithm. We are aiming to detect changes on the board within 400 ms with 99 percent accuracy. Furthermore, we would like to keep our Chess AI to generate all possible moves in 300ns which we are planning to minimize the latency of each move the AI makes.

II. Design Requirements

A. Move Detection

The main requirement of the move detection is split into two parts: the accuracy and the processing time. The accuracy of detecting the board will be done through a scene of 10 unique moves. To test the accuracy, after 20 unique moves, we will have the software indicate which move it is detecting by indicating the piece, the position to and where it is moving. To achieve 99% accuracy, we want to ensure none of the moves are detected incorrectly, since not detecting the move will have a significantly negative impact on the user experience. As for the processing time, the software will also have print statements to indicate how long it took to detect the move in milliseconds after receiving two frames.

B. Legal Move Generation

For the main requirements of the legal move generation, we want to ensure 100 percent correctness on all generated moves. We will test the correctness of legal move generations on 10 unique board states. On these 10 unique board states, we will check all the valid moves by going through each piece and see if all the moves have been generated correctly.

C. Transfer Protocol

The main requirements of the transfer protocol are accuracy and speed. We want to ensure that each of the packets we send is not corrupted and 100 percent correct. We also want to ensure that the speed of the Transfer Protocol to be under 1 second. This is important because communication will be the main bottleneck of the FPGA subsystem. The user should not experience any large latency or noticeable lag after making their move

D. User Interface

The main requirement of the user interface is its correctness. The interface should show the correct state of the board. The correctness will be tested through visual confirmation on whether the board and the interface match. The test will be conducted on 20 board states, and we want it to be correctly representing the board state 100 percent of the time.

III. Architecture and/or Principle of Operation

The overall architecture is divided into two main subsystems. The computer vision software, and the game logic software/hardware system. These subsystems are further divided into the following components:

A. OpenCV
B. Image Processing
C. Raspberry Pi
D. Pygame
E. UART Communication Protocol
F. FPGA Custom Hardware
G. Hard Processor System
H. Stockfish AI

We will now describe in detail these components and interconnects and demonstrate how they fit into our overall architecture. For a full diagram of how all components work and connect, see the appendix (Fig. 4)

A. OpenCV

We are using OpenCV for our Computer Vision part of the project, which provides a real-time optimized Computer Vision library and tools. Since OpenCV supports Python, it easily soothes into our program which is mainly based on Python. OpenCV provides conversion to HSV scale, board detection, background subtraction, and etc. Above are some of the functions that we need for our Computer Vision based Chess AI.

B. Image Processing

Image Processing is done using OpenCV libraries. First, we are obtaining frames from the video sent from the webcam set up on top of the board. As you can see from Fig 1, this figure is the example image of the top view of a Chessboard. Then, conversion to HSV scale is required to binarize the image and make the image more clear to be able to detect chessboard and changes on the board. Fig 2 is the HSV converted version of the original Chessboard image. After HSV conversion is done, the image is used for board corner detection and move detection using the background subtraction algorithm.

C. Raspberry Pi

Raspberry Pi is a series of small single-board computers which support Python, OpenCV, and various libraries which use Python as a programming language. For our design choice, we chose to use Raspberry Pi over a laptop computer because it is much smaller and cheaper. Most importantly, we believe that Raspberry Pi will give us a good enough performance on the simple algorithms we use.

Fig 1. Original Image of the Chessboard

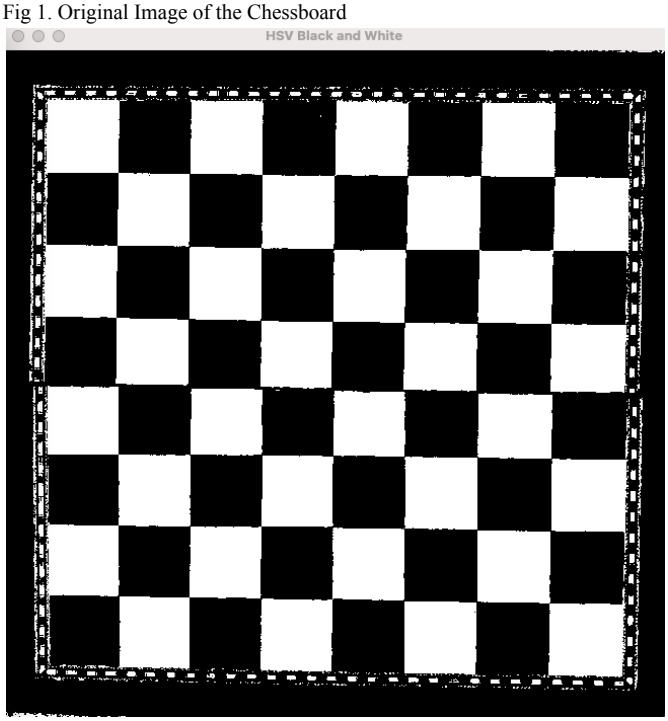

Fig 2. Chessboard image converted into HSV space and black and white

D. Pygame

Pygame is a cross platform set of Python modules designed for writing video games. A Chess game written with Pygame will mainly handle the user interface part of the game. The user interface will show the status of the board, moves that the Chess Artificial Intelligence makes, and recommendation of moves for the user.

E. UART Communication Protocol

UART will be used to communicate between the Raspberry Pi and FPGA. We chose UART over other serial communication protocols such as I2C or SPI because of the ease of implementation. The FPGA has a dedicated UART to microUSB port, thus allowing an easier time to implement the serial communication protocol. I2C or SPI would require use of general purpose IO pins and possibly a very specific cable to connect to the Raspberry Pi. Moreover, the UART protocol is estimated to fulfill the communication latency requirements as demonstrated in equation 1.

$$Assumption\ 1:\ 921,600\ baud\ rate$$
$$Assumption\ 2:\ 20,000\ bits\ /\ board\ state$$
$$20,000/921,600\ =\ 0.022s\ communication\ latency$$

F. FPGA Custom Hardware

The FPGA custom hardware will be computing the legal move set for any given board state. This is the compute heavy portion of any chess engine because of the sheer number of squares and possible moves for any given square. However, the task is easily parallelizable by generating legal moves for all squares. The FPGA excels at this task as it can generate application specific hardware for each square on the chess board.

H. Stockfish AI

The chess engine we are using is Stockfish, which is an open source chess engine. The majority of the engine code will run on the FPGA's Hard Processor System (HPS). Move generation will be repurposed to make use of the custom FPGA logic. Otherwise, the majority of Stockfish code base will remain the same as its current release, Stockfish 13.

IV. Design Trade Studies

A. Background Subtraction Algorithm

To detect the movement of the pieces, we chose to use the background subtraction algorithm. This choice was made because we want to have one camera with a top down view. With a top down view, it might be hard to detect the pieces especially because they are mostly in the same color. To mitigate this, we simply use the background subtraction algorithm to find out which pieces moved rather than detecting where each piece is at every board state. This even reduces the computation since we only need to subtract two frames rather than try to identify what each of the pieces are in each of the squares.

B. PyGame

We wanted to use Python to develop the User Interface because we are also using OpenCV which is written in Python, which makes it easier for the User Interface to interact with the Computer Vision algorithms. There are mainly two possible options for us to create the User Interface. First, we can use tkinter which is a built-in Graphical User Interface toolkit. However, tkinter only supports simple features, so we thought it would be a good idea to use a different toolkit. The second option was to use Pygame. Pygame is explicitly built for creating a game using Python programming language. Thus, it will be a better idea to use Pygame over tkinter to create the User Interface for our Chess game.

C. FPGA

We decided to use an FPGA for move generation. This was mainly inspired by previous chess engines which handled move generation on FPGA. Such engines include Hydra and Brutus. We hope that running Stockfish on the integrated HPS system will ensure that latency costs are minimized and there is a total compute time decrease as compared to Stockfish's base move generation.

D. Terasic DE-10 Standard FPGA

We chose the DE-10 Standard mainly because of its IO capacities and its relatively simple layout and design process. The DE-10 Nano has no switches and only 2 push buttons. We figured there may be a possibility that we use FPGA IO to toggle different modes in Stockfish. The DE10-Standard has sufficient switches and push buttons to allow for this possibility. We also considered Xilinx Kintex-7 boards but ultimately decided to use an Altera compatible board due to our experience with the Altera workflow as compared to Xilinx's Vivado software.

E. Raspberry Pi

Needing a user interface as well as an operating system to run the computer vision algorithm, we have decided to use the Raspberry Pi for these two purposes. This is especially helpful working in remote environments to allow different people to work on the FPGA and others to work on the Raspberry Pi. With the accessibility Raspberry Pi provides, it is easy to connect it to a monitor as well as download the necessary OpenCV libraries on the Raspbian operating system.

V. System Description

A. Raspberry Pi

The Raspberry Pi starts the image processing through receiving images from the camera. The images received form the camera will be converted from a RGB image into a HSV space. This will allow the image to be less affected by glare. Then the algorithm will detect the board. The board detection is done through detecting corners of each of the squares of the chessboard. By obtaining the coordinates of the chessboard corners, we can define each of the squares. After we define each of the squares to its coordinates, we can place the pieces and map each of the squares to its pieces. After all of these steps, we can use the background subtraction algorithm on the board to see which pieces have moved by comparing two frames. The two frames are simplified by the fact that a game uses a timer mechanic to signify the end of a turn. Hence, by

simply comparing two relevant frames, we can achieve a much higher accuracy. After the background subtraction algorithm, the board keeps track of the board state to update the board and communicate with the UI to show the new board state.

B. FPGA + Hard Processor System

The FPGA and Hard Processor System are used to generate the legal moves given a specific board state. The Hard Processor System is tasked with running Stockfish, an open source chess engine/AI, with slight modifications to make use of the FPGA hardware. Custom logic will be described in SystemVerilog and synthesized onto the FPGA board. This custom logic will handle the legal move generation of a specific board state. This will aid stockfish to make faster decisions and also quickly verify whether or not the human user has made a legal move.

C. User Interface

User Interface is mainly written in Pygame, which allows the user to interact with the overall system. The user can see recommendations of moves which are sent from the Chess AI so that the user can learn various moves from the recommendations. Also, the user needs to place pieces for the Chess AI. Once the AI's turn is over and the User Interface shows the AI's move, the user needs to move the piece for the AI so that the new board representation can be recognized by the Computer Vision Algorithm. This way, the user might cheat on the AI's move, so we will double-check if the user made the correct move for the AI or not.

VI. Project Management

A. Schedule

We have currently have been trying out the different computer vision algorithms to figure out how to coordinate them together. Specifically, we have worked on detecting the chessboard corners, edge detection to figure out the squares, and background subtraction to find the moves. Our next steps will be to work on coordinating these Computer Vision algorithms together as well as add some details to improve the performance such as converting the images to HSV scale.

As we are coordinating the Computer Vision algorithms, it is crucial to debug whether the algorithms demonstrate expected behaviors. The main way to debug is to use board representation from the user interface. Visualization of the board status will give us a clear idea how the Computer Vision algorithms are performing. So, the first step for implementing the user interface is to show the status of the Chessboard. After we confirm that the Chess board representation is correct, we can move on to the next step which handles more user interface to make the overall system intuitive.

As for the FPGA, currently we have flashed a Linux distribution onto the HPS, and have UART communication between PC and FPGA. This will later be integrated with the Raspberry Pi to have UART communication between Raspberry Pi and FPGA. Our next steps are to design a legal move generation module in SystemVerilog and integrate the module into Stockfish.

The schedule is included at the end.

B. Team Member Responsibilities

Joseph is in charge of the computer vision algorithms. This includes the implementation of the background subtraction algorithm, board recognition, and the whole chess board setup such as each individual piece and squares to represent the board state.

While collaborating with Joseph on computer vision algorithms, Jee Woong will implement the user interface of the Chess game. User interface of the game, written in Pygame, is the main communication program for the user, which allows the user to interact with the program, see recommendations of moves generated by the Chess AI, and moves for the Chess AI.

Michael is in charge of the FPGA and HPS subsystem. This includes pipelining the legal move generation, integrating stockfish code to utilize the FPGA custom logic, and integrating UART communication with the Raspberry Pi. The FPGA custom logic will be designed in SystemVerilog while modifications to Stockfish will be in C++.

Budget

Table 1 is the list of materials we have acquired so far. Since our teammates are working from different places, we needed three sets of Chess sets, webcams and webcam stands in order to work individually. We have budgeted only $395.28 out of the $600 budget, which gives us more flexibility to change some of the components if we encounter any issues.

Material	Quantity	Cost
Incurred Expenses		
DE10-Standard FPGA	1	ECE owned
Chess sets	3	$25.43
Webcam	3	$42.39
Webcam Stand	3	$23.31
Planned Future Expenses		
Raspberry Pi	1	$121.89
Total		$395.28

Table 1. Budget

C. Risk Management

One of the risk factors that we found is the coordination of all the algorithms as well as the components. It might be easy to detect the corners of the chessboard and also detect changes in each board state using the OpenCV library, but it is another

thing to coordinate them together. We need to identify which piece moved through the background subtraction algorithm and communicate with the board detection algorithm to figure out where it moved to. We need to set up a chess system in our code to coordinate all of these safely. To mitigate this risk, we have set up enough time for coordination and we think it will be enough time to coordinate them together.

Another risk factor is detection of moves when pieces are taken. It might be easy to notice which piece has moved when there is only one piece moving, but it might be harder to figure out which piece has moved when one piece takes another piece. To mitigate this, we will use the HSV color space to reduce glare and binarize the image and then use the fact that a piece has changed from black to white or white to black to find which piece has been taken.

Integration of all hardware components is another risk we must consider. The FPGA and Raspberry Pi must communicate with each other and ensure that this communication happens without large latency or lost packets. We will mitigate this by beginning integration as soon as possible and working on components that can be integrated earlier on in our schedule. These details are specifically listed in the Gantt chart and schedule section.

VII. Related Work

There are similar projects to ours, but none of them seem to be exactly the same as ours. One example we found was DecodeChess. DecodeChess is a virtual chess teacher that goes in depth about recommending what the best move would be in each of the situations. In comparison to ours, this environment does not give a realistic environment, but it does give more feedback than our game because it also provides reasoning behind each move. Another example is a Fall 2020 capstone project from the Blokus group. They simulated a virtual environment for playing Blokus, which is also kind of similar to our project in that they detect the moves using a camera to upload it to a computer. The difference with ours is that our product goes against artificial intelligence while the Blokus group used a player to player interaction when playing the game.

References

[1] DecodeChess, https://decodechess.com/first-ai-chess-tutor/
[2] https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-5505271707235-de10-standard-user-manual-sm.pdf
[3] https://www.chessprogramming.org/Board_Representation
[4] https://www.chessprogramming.org/Move_Generation

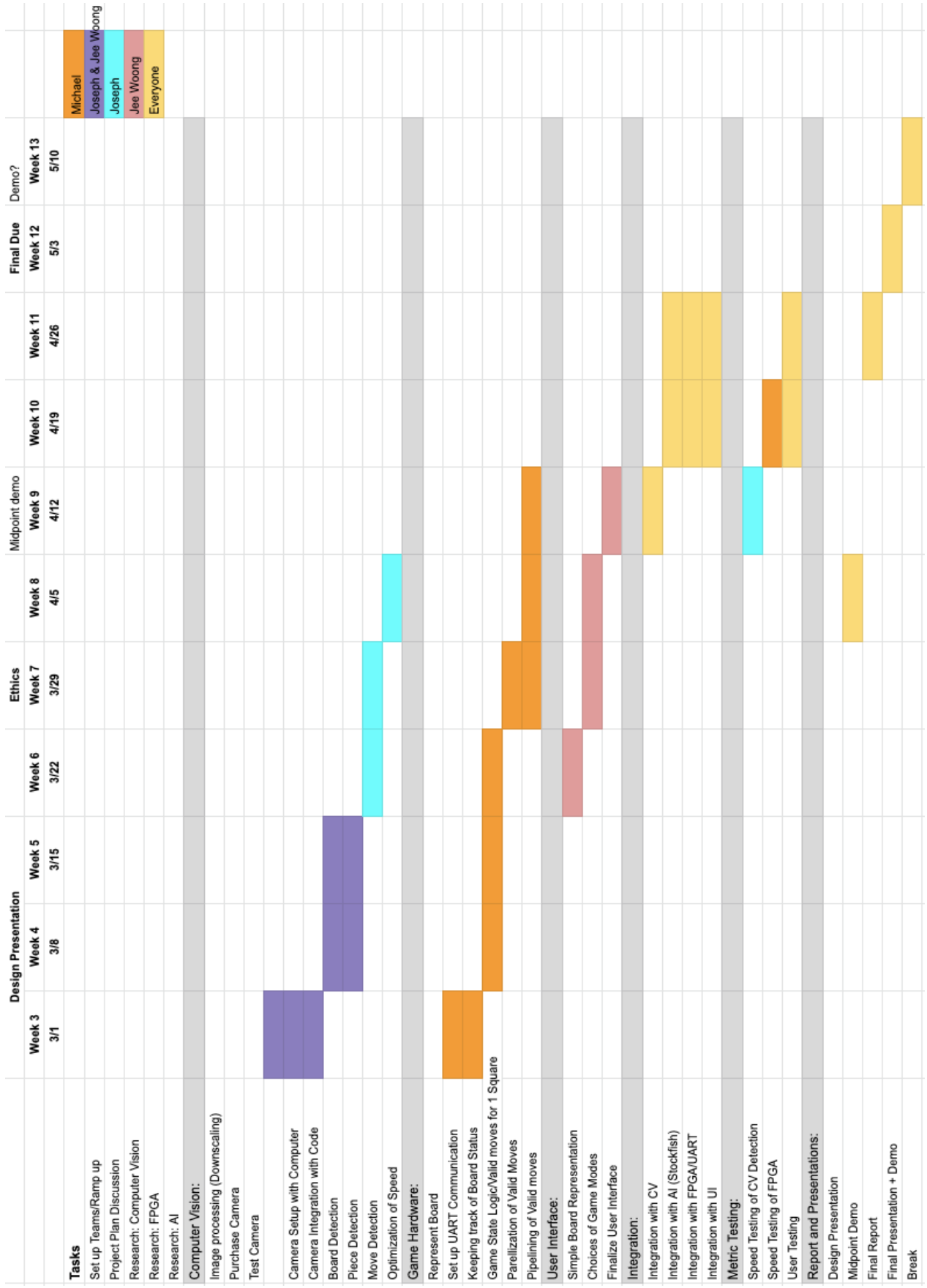

Fig 3. Schedule (Gantt Chart)

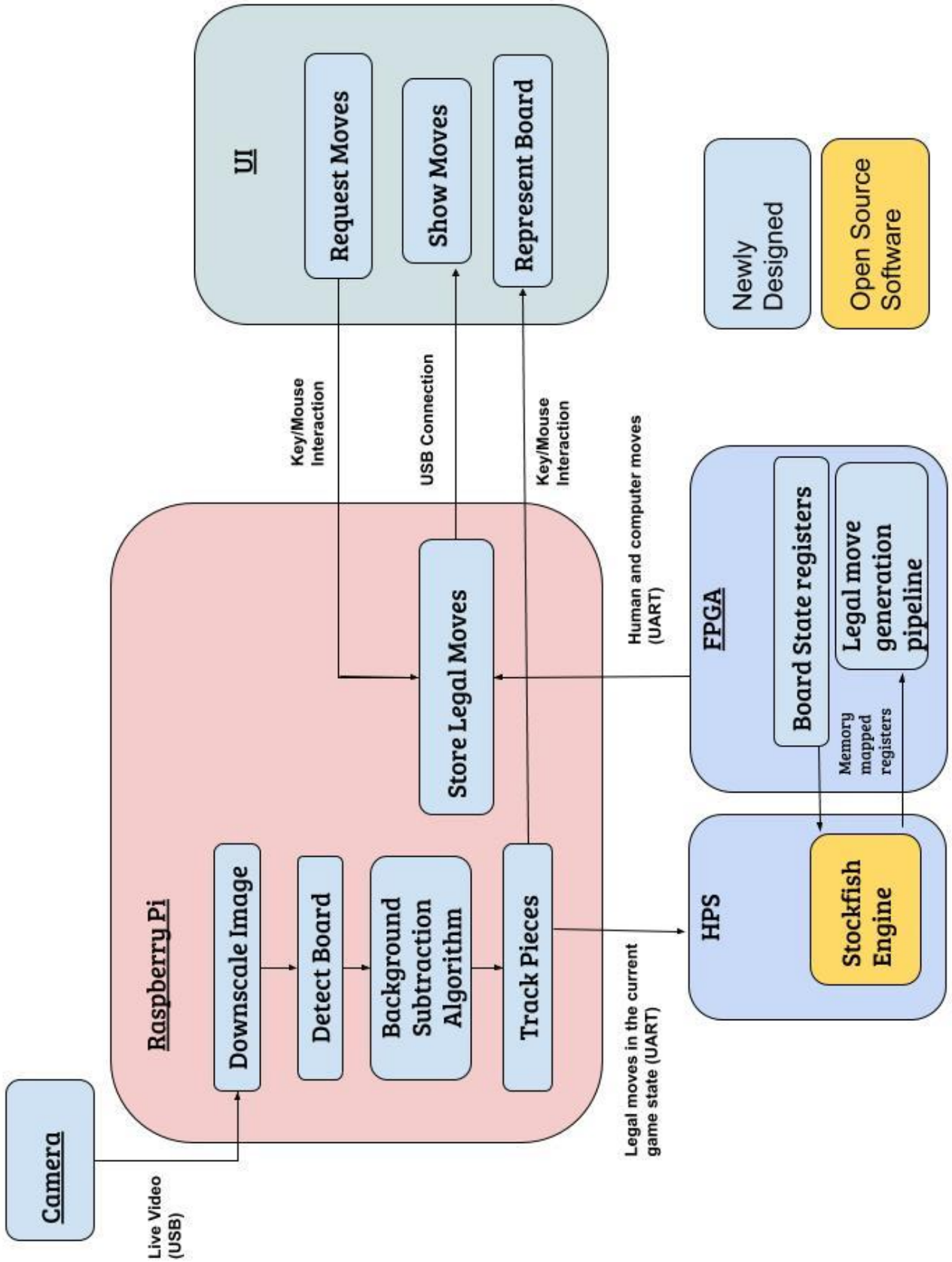

Fig 4. System Diagram