

**PHYSICAL CHESS GAME TRACKING USING RASPBERRY PI**

**BY**

**YEE WEI JUN**

**A REPORT**

**SUBMITTED TO**

**Universiti Tunku Abdul Rahman**

**in partial fulfillment of the requirements**

**for the degree of**

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

**Faculty of Information and Communication Technology**

**(Kampar Campus)**

**JAN 2024**

## REPORT STATUS DECLARATION FORM

**Title:** Physical Chess Game Tracking Using Raspberry Pi  
\_\_\_\_\_  
\_\_\_\_\_

**Academic Session:** Jan 2024

I YEE WEI JUN  
**(CAPITAL LETTER)**

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



\_\_\_\_\_  
(Author's signature)

Verified by,



\_\_\_\_\_  
(Supervisor's signature)

**Address:**

138, Lengkok Wira Jaya Timur 3,

Taman Desa Kebudayaan

31350 Ipoh, Perak

Dr. Teoh Shen Khang

Supervisor's name

**Date:** 25<sup>th</sup> April 2024

**Date:** 25 April 2024

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY  
UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 25<sup>th</sup> April 2024

**SUBMISSION OF FINAL YEAR PROJECT**

It is hereby certified that YEE WEI JUN (ID No: 20ACB01647) has completed this final year project entitled “ *Physical Chess Game Tracking Using Raspberry Pi* ” under the supervision of Dr Teoh Shen Khang (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



YEE WEI JUN

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**PHYSICAL CHESS GAME TRACKING USING RASPBERRY PI**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  \_\_\_\_\_

Name : YEE WEI JUN

Date : 25<sup>th</sup> April 2024

## **ACKNOWLEDGEMENTS**

I would like to express my sincere thanks and appreciation to my supervisor, Dr Teoh Shen Khang has given me this bright opportunity to engage in a project that span various field of study, namely Computer Vision, Machine Learning, Internet of Things, and web application development. With your guidance and support, I am able to explore and learn many things that I never had a chance to explore before. Many thanks to you.

To my family and friends, thank you for staying by my side whenever I feel down. Without you all, I could not stand wherever I am now. Also, many thanks to my parents for your continuous support throughout this course.

## **ABSTRACT**

With the popularity in chess increases exponentially caused by the advancement of information technology where there are many chess players were influenced by online chess streamer and got interest in chess. Due to this fact, there are many chess tournaments being organized worldwide no matter the scale of them. With the use of the current technology in tracking and broadcasting the chess game which is using electric chess board with uniquely manufactured chess pieces with unique patented sensor technology, it is costly to track and broadcast every single chess game begin played in the tournament on a website. Hence, a novel alternative way of tracking and broadcasting a chess game is proposed in this project by using Raspberry Pi with compatible camera module and detection and recognition model installed on it. The Raspberry Pi will detect chess move and relay the move to a web application where it will change the state of the digital chess board with the chess move received. Additionally, it will also store the move in a database for future analysis. By using the approach proposed, the cost of tracking a chess game and broadcasting it will be drastically decreases, thus, the overall cost of organizing a chess tournament will also decreases.

# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>REPORT STATUS DECLARATION FORM</b>	<b>ii</b>
<b>FYP THESIS SUBMISSION FORM</b>	<b>iii</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>iv</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF TABLES</b>	<b>xvi</b>
<b>LIST OF SYMBOLS</b>	<b>xvii</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Background Information	1
1.2 Problem Statement and Motivation	3
1.3 Project Scope	4
1.4 Project Objective	4
1.5 Impact, Significance, and Contributions	4
1.6 Report Organization	5
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>6</b>
2.1 Object Recognition	6
2.2 Chessboard Detection	8
2.3 Chess Piece Detection and Recognition	14
2.4 Chess Move Detection	21
2.5 Limitation of Previous Studies	23
<b>CHAPTER 3 SYSTEM METHODOLOGY/APPROACH</b>	<b>25</b>
3.1 System Architecture Diagram	25
3.1.1 Amazon Web Services(AWS) Used	26
3.1.2 General Flow and Process	27

3.2	Use Case Diagram and Description	28
3.2.1	Use Case Diagram	28
3.2.2	Use Case Description	29
3.3	Activity Diagram	35
3.4	Timeline	41
<b>CHAPTER 4 SYSTEM DESIGN</b>		<b>43</b>
4.1	System Design Overview	43
4.2	System Design for Raspberry Pi	45
4.2.1	Image Acquisition	46
4.2.2	Image Pre-processing	46
4.2.3	Image Thresholding and Finding Contours	47
4.2.4	Finding largest contour to discover the location of the chessboard	47
4.2.5	Approximate polygonal curve	48
4.2.6	Transform chessboard into top-view perspective image	48
4.2.7	Find chessboard square grid corners	49
4.2.8	Square grid masks construction	49
4.2.9	Detect changes	50
4.2.10	Image segmentation	51
4.2.11	Chess piece detection and Classification	51
4.2.12	Compute Source and Destination Squares	52
4.2.13	Send Detected Move to Backend of the Web Application	52
4.3	System Design for Web application	53
<b>CHAPTER 5 SYSTEM IMPLEMENTATION</b>		<b>55</b>
5.1	Hardware Setup	55
5.2	Software Setup	57
5.3	Settings and Configuration	58
5.3.1	Visual Studio Code Installation	58
5.3.2	YOLOX Installation	59



5.3.3	OpenCV Installation	60
5.3.4	AWS Free Tier Account Creation	60
5.3.5	Frontend Creation Using Command Line	62
5.3.6	Backend Creation Using Amplify CLI	62
5.3.7	Database Creation Using Amplify Studio	64
5.3.8	Application Protocol Interface(API) and Lambda Function Creation Using Amplify CLI	65
5.4	System Operations	68
5.5	Implementation Issue and Challenges	78
<b>CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION</b>		<b>79</b>
6.1	System Testing and Performance Metrics	79
6.2	Testing Setup and Result	80
6.2.1	Testing Setup	80
6.2.2	Result	80
6.2.2.1	Result for First Test Case	80
6.2.2.2	Result for Second Test Case	88
6.2.2.3	Result for Third Test Case	90
6.2.2.4	Result for Fourth Test Case	93
6.2.3	Error Analysis	95
6.3	Objective Evaluation	96
<b>CHAPTER 7 CONCLUSION AND RECOMMENDATION</b>		<b>97</b>
7.1	Conclusion	97
7.2	Recommendation	97
<b>REFERENCES</b>		<b>98</b>
<b>APPENDIX A</b>		
A.1	Weekly Report	A-1
A.2	Poster	A-7
A.3	Plagiarism check result	A-8

## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 1.1.1	Average number of chess players over the years	3
Figure 2.2.1	Equations used in predicting the possible locations and appearance of the chess pieces [1].	9
Figure 2.2.2	The chessboard preprocessing result. The board boundaries are marked by green lines and the normal vector of each square is indicated using a blue stick[2].	9
Figure 2.2.3	a) Contour of chessboard, b) Warped perspective of chessboard ,c) Canny edge detection, d) Hough Lines, e) Square vertices[4].	10
Figure 2.2.4	Board corner points in white[6].	10
Figure 2.2.5	(a) The original chessboard image, (b) the detected chessboard grid corners and (c)the related points to chessboard grid corner detection[7].	11
Figure 2.2.6	The chessboard before geometric rectification step, (b) The chessboard after geometric rectification step[7].	11
Figure 2.2.7	Straight line detection[11].	12
Figure 2.2.8	Example results of the geometric detector. Green color represents rhomboids that were identified by the detector. Only the rightmost matrix will be classified as a lattice point because it is the only one that contains four rhomboids[11].	13
Figure 2.2.9	Polyscore values calculated for various frames. The frame with the highest score is marked with the red border. Higher scores are highlighted with darker blue backgrounds[11].	14
Figure 2.2.10	Equations for polyscore function P(F)[11].	14
Figure 2.3.1	Standard modern Staunton wood chess set[14].	15

Figure 2.3.2	Classifier hierarchy used to recognize chess pieces during a game[6].	16
Figure 2.3.3	Object Detection Structure of YOLOv4[16].	17
Figure 2.3.4	Overview of volumetric CNN[5].	17
Figure 2.3.5	Architecture of the 1-Step Model[9].	18
Figure 2.3.6	Architecture of the Binary classifier in the 2-Step model[9].	19
Figure 2.3.7	Architecture of the Shape-based classifier in the 2-Step model[9].	20
Figure 2.3.8	The overall structure of AlexNet[10].	21
Figure 3.1.1	System Architecture Diagram.	25
Figure 3.2.1.1	Use case diagram of the system.	28
Figure 3.3.1	Activity diagram for Access Home Page use case.	35
Figure 3.3.2	Activity diagram for Login use case.	36
Figure 3.3.3	Activity diagram for Sign Up use case.	37
Figure 3.3.4	Activity diagram for Select Chess Game use case.	38
Figure 3.3.5	Activity diagram for Display Move Using Buttons use case.	39
Figure 3.3.6	Activity diagram for Create Game use case.	40
Figure 3.3.7	Activity diagram for Update Move use case.	40
Figure 3.4.1	Gantt chart for FYP 1	41
Figure 3.4.2	Gantt chart for FYP 2	41
Figure 4.1.1.	System design overview diagram.	43
Figure 4.2.1.	System design diagram for Raspberry Pi.	45
Figure 4.2.1.1	Sample result for image acquisition.	46
Figure 4.2.2.1	Sample result for image pre-processing.	46
Figure 4.2.3.1	Sample result image thresholding and finding contours.	47
Figure 4.2.4.1	Sample result for finding largest contour.	47
Figure 4.2.5.1	Sample result approximating polygonal curve of an image.	48
Figure 4.2.6.1	Sample result for warpPerspective().	48
Figure 4.2.7.1	Sample result for findChessboardCornersSB() with additional calculation	49

Figure 4.2.8.1	Sample result for square grid masks construction.	50
Figure 4.2.9.1	Sample result for finding changes using absdiff().	50
Figure 4.2.10.1	Sample result for image segmentation.	51
Figure 4.2.11.1	Sample of training images	52
Figure 4.2.11.2	Sample of validation images	52
Figure 4.2.12.1	Sample result of model inference	52
Figure 4.3.1.	System design diagram for Web Application	53
Figure 5.1.1	Hardware Setup	57
Figure 5.3.1.1.	Visual Studio Code Installation.	58
Figure 5.3.2.1.	YOLOX documentation	59
Figure 5.3.2.2	YOLOX installation guideline.	59
Figure 5.3.2.3.	Code to load and use the model in performing inference for object detection.	59
Figure 5.3.3.1	OpenCV Installation.	60
Figure 5.3.4.1.	AWS main page.	60
Figure 5.3.4.2.	Account sign up page.	61
Figure 5.3.4.3.	AWS console management page.	61
Figure 5.3.5.1.	Creating a frontend using command line.	62
Figure 5.3.6.1.	Creating backend using Amplify CLI.	62
Figure 5.3.6.2.	AWS console management page.	63
Figure 5.3.6.3.	AWS Amplify main page..	63
Figure 5.3.6.4.	Setconfdemo app backend environment page.	63
Figure 5.3.7.1.	Enabling Amplify Studio	64
Figure 5.3.7.2	Creating database in Amplify Studio.	64
Figure 5.3.7.3.	Tables successfully created in DynamoDB.	65
Figure 5.3.8.1.	Creating API and lambda functions using Amplify CLI.	65
Figure 5.3.9.1.	Project in GitHub repository.	66
Figure 5.3.9.2.	Selecting GitHub in hosting environment.	66
Figure 5.3.9.3.	Adding the github repository to Amplify.	67
Figure 5.3.9.4.	Web application successfully deployed.	67
Figure 5.4.1.	ChessEyes app hosting environment page.	68
Figure 5.4.2.	Web application home page.	68

Figure 5.4.3.	Sign in page of the web application.	69
Figure 5.4.4.	Home page content after user signed in.	69
Figure 5.4.5.	Login GUI in Raspberry Pi.	70
Figure 5.4.6.	Home page GUI in Raspberry Pi.	70
Figure 5.4.7.	Create game popup GUI in Raspberry Pi.	70
Figure 5.4.8.	Updated home page GUI in Raspberry Pi.	71
Figure 5.4.9.	Updated home page in web application.	71
Figure 5.4.10.	Chess page in web application.	72
Figure 5.4.11.	Chess game tracking GUI in Raspberry Pi.	72
Figure 5.4.12.	Threshold image obtained from processed raw image.	73
Figure 5.4.13.	Contour of chessboard obtained from threshold image.	73
Figure 5.4.14.	4 corners points found using approxPolyDP() from contour.	74
Figure 5.4.15.	Top-view perspective image obtained using the 4 corner points.	74
Figure 5.4.16.	Square grid corner points found from the top-view perspective image.	74
Figure 5.4.17.	Comparison of reference and latest snapshot image using absdiff().	75
Figure 5.4.18.	Segmented image obtained using segmentation mask.	76
Figure 5.4.19.	Chess game page displaying real-time move.	76
Figure 5.4.20.	Comparison of reference and latest snapshot image using absdiff() for second move.	77
Figure 5.4.21.	Segmented image obtained using segmentation mask for second move.	77
Figure 5.4.22.	Chess game page displaying real-time move for second move.	78
Figure 6.2.2.1.1.	White pawn move from e2 to e4 detected by Raspberry Pi.	81
Figure 6.2.2.1.2.	Detected move displayed correctly in web application.	81
Figure 6.2.2.1.3.	White castling move detected by Raspberry Pi.	82
Figure 6.2.2.1.4.	White castling move displayed correctly in web application.	82

Figure 6.2.2.1.5.	Black castling move detected by Raspberry Pi.	82
Figure 6.2.2.1.6.	Black castling move displayed correctly in web application.	83
Figure 6.2.2.1.7.	White knight moved from c3 to d5 detected by Raspberry Pi.	83
Figure 6.2.2.1.8.	White knight moved from c3 to d5 displayed correctly in web application.	84
Figure 6.2.2.1.9.	Black knight moved from f6 to d5 detected by Raspberry Pi.	84
Figure 6.2.2.1.10.	Black knight capturing move displayed correctly in web application.	84
Figure 6.2.2.1.11.	Black player moved king from h8 to h7 as displayed in web application.	85
Figure 6.2.2.1.12.	White player promote the pawn to queen by moving pawn in b7 to b8 as detected by Raspberry Pi.	85
Figure 6.2.2.1.13.	The pawn promotion move displayed correctly in web application.	86
Figure 6.2.2.1.14.	Black player moved the queen from e1 to f1 displayed in web application.	86
Figure 6.2.2.1.15.	White player's capturing move detected by Raspberry Pi.	87
Figure 6.2.2.1.16.	White player's latest move and result of the game displayed correctly in the web application.	87
Figure 6.2.2.2.1.	Chess game state for test case two shown in web application.	88
Figure 6.2.2.2.2.	Both kings is detected to be moved from their source square to the center of the board on light squares by Raspberry Pi.	89
Figure 6.2.2.2.3.	Result displayed in the web application that white is victorious.	89
Figure 6.2.2.3.1.	Chess game state that enable white to plays en passant shown in web application.	90
Figure 6.2.2.3.2	En passant move detected by Raspberry Pi.	91

Figure 6.2.2.3.3.	En passant move by white displayed correctly in the web application	91
Figure 6.2.2.3.4.	Both kings is detected to be moved from their source square to the center of the board on dark squares by Raspberry Pi.	92
Figure 6.2.2.3.5.	Result displayed in the web application that black is victorious.	92
Figure 6.2.2.4.1	Starting position of the game in fourth test case.	93
Figure 6.2.2.4.2.	Both kings is detected to be moved from their source square to the center of the board on light and dark squares by Raspberry Pi.	94
Figure 6.2.2.4.3	Result displayed in the web application that both players had drew.	94
Figure 6.2.3.1.	Raspberry Pi mistakenly detected 3 squares where it should detect 2 squares only.	95
Figure 6.2.3.2.	Raspberry Pi mistakenly detected 3 squares where it should detect 2 squares only.	95
Figure 6.2.3.3	Raspberry Pi mistakenly detected 3 squares where it should detect 2 squares only.	96

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 1.1.1	Average number of players over the years	3
Table 2.1.1	State-of-the-art object recognition method[1].	7
Table 2.3.1	The recognition accuracy of various methods[2].	16
Table 2.3.2	Confusion matrix of classification results[5].	17
Table 2.3.3	Accuracy and model precision[10].	22
Table 3.2.2.1	Use case Description for Access Home Page use case.	29
Table 3.2.2.2	Use case Description for Login use case.	30
Table 3.2.2.3	Use case Description for Sign Up use case.	31
Table 3.2.2.4	Use case Description for Select Chess Game use case.	32
Table 3.2.2.5	Use case Description for Display Move Using Buttons use case.	33
Table 3.2.2.6	Use case Description for Create Game use case.	34
Table 3.2.2.7	Use case Description for Update Move use case.	34
Table 5.1.1	Specifications of laptop	55
Table 5.1.2	Specifications of Raspberry Pi[22]	56
Table 5.1.3	Specifications of Raspberry Pi camera[27].	56



## LIST OF ABBREVIATIONS

<i>CNN</i>	Convolutional Neural Network
<i>R-CNN</i>	Region-based Convolutional Neural Network
<i>SSD</i>	Single-Shot Detector
<i>YOLO</i>	You Only Look Once
<i>FPS</i>	Frame Per Second
<i>SVM</i>	Support Vector Machine
<i>ReLU</i>	Rectified Linear Unit

# CHAPTER 1

## Introduction

In this chapter, the background information, problem statement and motivation of the project, project scope, project objective and the impact, significance and contribution of the project to the community will be presented.

### 1.1 Background Information

First and foremost, computer vision, it is a branch of artificial intelligence focused on allowing machines to interpret and understand the visualized world. It equips computers with the ability to perceive, analyze, and extract information from images or video, replicating human vision processes. By adopting sophisticated algorithms, machine learning, and image processing techniques, computer vision systems can identify and comprehend patterns, objects, and even context within visual data. Thus, it empowers machines to perform tasks like object detection, image classification, facial recognition, and scene understanding. Additionally, the applications span across diverse fields such as in healthcare, assisting diagnoses through medical imaging analysis; in autonomous vehicles, enabling them to perceive surroundings, enhancing quality control processes, facilitating augmented reality experiences, identifying anomalies for surveillance and many more. As the advancement of technology in the recent years in computer vision, particularly with deep learning models and neural networks, have propelled its capabilities. As machines gain the ability to interpret visual information more accurately, efficiently, and contextually, the impact of computer vision across industries continues to expand, revolutionizing how we interact with technology and perceive the world.

Besides, machine learning, a branch of artificial intelligence that focuses on constructing algorithms and systems that enable computers to acquire and make predictions or decisions without explicit programming. Overall, it involves training machines on data, allowing them to recognize patterns, making predictions, or gaining insights from new information. At its core, machine learning employs various techniques and algorithms, such as supervised learning that involve labeled data, unsupervised learning that involve discovering patterns in unlabeled data, and reinforcement learning that learn from feedback, to enable computers to learn and improve from experience. In addition, the applications of machine learning vary across industries such as aiding diagnoses and drug discovery, fraud detection

and risk assessment, personalized recommendations, enabling navigation and decision-making and many more.

Moreover, IoT, also known as Internet of Things, is refers to a network of interconnected devices embedded with sensors, software, and other technologies, allowing them to collect, exchange, and act upon data. In short, IoT can increases the efficiency, convenience, and insights of common things by giving them the ability to interact, connect to the internet, and carry out automated operations. Additionally, IoT devices span various domains, from smart homes with connected appliances, thermostats, and security systems to industrial settings integrating sensors in machinery for predictive maintenance and optimization of operations. IoT adoption causes revolutionary shifts in a variety of businesses such as wearable technology is used in healthcare to track vital signs and provide remote patient care, sensors monitor soil moisture content in agriculture to enable accurate watering, tracking systems improve supply chain management in logistics, and most importantly, can be used in tracking over the board chess game.

Finally, Chess game, a two-player strategic game where the players will battle it out with their pieces either in white or black typically. It is played on a 8x8 alternating black and white square grid chess board and each player will have a total of 16 pieces which is 8 pawns, 2 rooks, 2 knights, 2 bishops, 1 queen and 1 king. The primary objective of the game is to checkmate the opponent or strategically outplay the opponent, resulting in resignation and claiming the victory.

As time goes from 1990s to 2000s and to 2010s and finally 2020s, the popularity of chess has been increasing drastically especially during the period of Covid-19 pandemic, caused by the influence of online chess streamer, resulting in more people taking interest in chess, no matter the age. With reference to the data from Chess Ratings[26], the average number of chess players over the years is increasing steadily from 2019 to 2023 as shown in Figure 1.1.1. As it is now in the post pandemic period, where chess game can be played physically again, there are many over-the-board chess competitions and tournaments begin held all over the world, participated by hundreds of player for every single chess competition. In addition, every chess games begin played in the competitions and tournaments is begin broadcasted online. Consequently, there will be tens or hundreds of chess game begin played and broadcasted simultaneously. Hence, it is important to track every single chess game, record down every move begin played by every player and broadcast them online in the competition.

Year	Average number of player
2019	339073
2020	360351
2021	367797
2022	389768
2023	419782

Figure 1.1.1. Average number of chess players over the years.

## 1.2 Problem Statement and Motivation

As a continuation to the previous sub-chapter, the method commonly used nowadays to track a chess game and broadcast it online is by using an electric chess board. Additionally, in order to detect the chess pieces on the board, the chess pieces is uniquely manufactured with unique patented sensor technology. Normally, an electric chess board will cost around \$400 to \$1500. Due to the fact that not every chess competition is organized by the official, FIDE, it is costly to track every game using the electric chess board for every chess game being played at the same time. It will cost tens of thousands of dollars just for a chess competition. Additionally, the electric chess board and the unique chess pieces require constant maintenance, which will cause another fortune. Consequently, there exists difficulty in increasing the scale of a tournament or competition due to limited budget.

Thus, the primary objective of this project is to propose a novel and alternative way in chess game tracking. With only \$50 to \$100, a chess game can be tracked easily without any complex manufacturing of chess board and chess pieces. In this project, chess game tracking by using Raspberry Pi is proposed as the novel alternative way in doing so. The Raspberry Pi will be installed with a well-trained model in tracking chess game with the help of compatible camera module mounted on it in capturing the real-time images of the chess game. The chess move will be detected and the related information such as the chess piece that had moved, color, new position will be sent to a web application dedicated in displaying, potentially broadcasting and saving the move in the database for future analysis. By scaling down the cost in tracking every chess game, the size of the competition can be scaled up without any heavy concern for the budget. Additionally, the organizer now have more options in tracking and broadcasting chess game online. The ultimate and the most optimistic goal of this project is the novel method can eventually replace the traditional way of tracking chess game.

### **1.3 Project Scope**

The scopes of this project include preparing an optimal environment such as adequate lighting and dark-colored background table for the chessboard. Besides, training and validation data will be collected manually by taking the photos from the top-view of the chessboard. After that, the images will be labelled using DataTorch and labelled images will be used in model training. The trained model will then deployed on the Raspberry Pi for chess piece detection. To increase the accuracy of the model, model fine tuning will also be done to improve the performance of the model in detecting, recognizing chess pieces and eventually capable of detecting the chess pieces more accurately. Furthermore, a web application dedicated in displaying, potentially broadcasting and saving chess game will be constructed and hosted using Amazon Web Service(AWS). The web application will receive information such as chess piece that had moved, the new position and the player that plays it from the Raspberry Pi and update the state of the digital board display on it. A database will also be designed and hosted on the cloud to store all the moves from the all the game that had been played and is tracked by Raspberry Pi.

### **1.4 Project Objectives**

The primary objective of this project is to provide a novel and alternative way in tracking, saving and potentially broadcasting physical chess game on the web application using Raspberry Pi. Aside from that, one of the objectives of this project is to train a model that is capable of detecting and recognizing chess pieces that will be deployed on Raspberry Pi. Moreover, a web application that is capable of displaying real-time data for the game being played and saving chess game will be also constructed. Besides, the other objective of this project is tracking a physically played chess game from the start to the end without error and without human intervention.

### **1.5 Impact, Significance and Contribution**

This novel method will provide an alternative option for the chess competition and tournament organizers in tracking, saving and potentially broadcasting the chess games being played simultaneously. Additionally, chess game begin played privately such as between friends and chess club members can also be tracked, broadcasted and saved for further analysis. By using this method, the community no longer have to key in the chess move from the

scoresheet manually to the computer to perform analysis on their games as this method are able to save the chess move automatically and simultaneously as they are playing the game.

### **1.6 Report Organization**

There are in total 7 chapters in this report that comprise of Chapter 1 Introduction where the background information, problem statement and motivation of the project, project scope and objective and the contribution of the project to the community. Additionally, several previous studies are reviewed that are related to objection recognition, chessboard detection, chess piece detection and recognition and chess move detection in Chapter 2 Literature Review. Besides, the methodology and system architecture will be thoroughly explained in detailed in Chapter 3 System Methodology/Approach. Moreover, system design with detailed explanation will be introduced in Chapter 4 System Design. In addition, step-by-step implementation procedure will be provided in Chapter 5 System Implementation. Other than that, the system performance will be discussed in Chapter 6 System Evaluation and Discussion. Lastly, the report will be concluded with recommendations for future work in Chapter 7 Conclusion and Recommendations.

## CHAPTER 2

### Literature Reviews

In this chapter, several research papers, articles and systems were reviewed to better understand the latest technology, techniques and approaches used in object recognition. Furthermore, previous studies related to detecting and recognizing chessboard, chess piece and chess game were also reviewed to acknowledge the approaches in doing so. There are 5 sub-chapters comprised in this chapter that talk about different topics which are 2.1 object recognition, 2.2 chess board detection, 2.3 chess piece detection and recognition and 2.4 chess move detection. Additionally there are also 2 sub-chapters which talk about the limitations of the previous studies discussed in all previous sub-chapters and the proposed solutions in sub-chapters 2.5 limitation of previous studies and 2.6 proposed solutions.

#### 2.1 Object Recognition

Object recognition is a process that involves both classification and localization tasks[1]. It must recognize the intended object in the image and tell it apart from the complex backdrop. Each object instance, its associated label, and its confidence score are precisely localized using multi-object pixel masks and bounding boxes. In year 2022, H. M. Ahmad and A. Rahimi[1] had done a survey that summarized object recognition techniques using deep learning approaches. As shown in Table 2.1.1, many state-of-the-art object recognition methods and techniques had been reviewed and summarized. In general, object recognition using deep learning approach consists of a deep backbone architecture and region selection model.

In accordance with the survey, deep backbone architecture reviewed are VGGNet, INCEPTION Net, XCEPTION Net, RESNet, RESNEXT, DARKNet, MOBILENet, SENet, EfficientNet, Visual transformers and Lightweight deep learning models. Backbone is known as the feature extraction network employed in the deep learning architecture. This feature extractor encodes the network's input into a particular feature representation. Most of them are the improved version of the previous state-of-the-art methods that applied CNN as their deep backbone architecture. For example, INCEPTION Net is the improved version of VGGNet with various convolution layers with smaller filter size used, XCEPTION Net that built upon Inception blocks with reference to INCEPTION Net and many more.

As for the methods and techniques used in region selection, the survey had reviewed various of them such as R-CNN, FASTER R-CNN, MASK R-CNN, SSD, RefineDet, YOLO, EfficientDet and DETR:detection-transformer. Excluding DETR:detection-transformer, all of the region selection models reviewed are built on CNN as the deep backbone architecture. With reference to Table 2.1.1, YOLOX has the highest FPS in object recognition in real-time.

Source	Region Selection	Backbone Architecture	Image Size	MS COCO IoU @0.5 (mAP)	FPS
[20], 2014	R-CNN	VGG-16	–	–	–
[27], 2015	Faster R-CNN	VGG-16	–	35.9	5
[30], 2016	YOLOv1	VGG-16	–	–	–
[70], 2017	Mask R-CNN	ResNeXt	640	62.3	–
[36], 2017	SSD	ResNet	512	31.2	8
[32], 2017	YOLOv2	DarkNet-19	608	44	–
[35], 2018	YOLOv3	DarkNet-53	608	33	19
[38], 2018	RefineDet	ResNet101	512	62.9	24.1
[42], 2020	YOLOv4	CSPDarknet53	608	65.7	62
[44], 2020	YOLOv5	CNN	640	68.8	43.4
[40], 2020	EfficientDet-D1	EfficientNet-B1	640	58.6	74.1
[77], 2020	PP-YOLO	ResNet50-vd-dcn	608	65.2	72.9
[81], 2020	Scaled-YOLOv4	Modified-CSPDarknet53	640	66.2	73
[46], 2020	DETR	ResNet50	–	62.4	28
[75], 2021	YOLOS	DeiT-B [89]	800	42.0	2.7
[73], 2021	YOLOX	DarkNet53	640	67.3	90.1
[45], 2021	YOLOF	ResNet-50	–	60.5	32
[78], 2021	PP-Yolov2	ResNet101-vd-dcn	640	69	50.3
[74], 2021	YOLOr	CNN	1280	73.3	30

Table 2.1.1 State-of-the-art object recognition method[1].



## 2.2 Chessboard Detection

Chessboard, the most important component in a game of chess, is a square, checkered game board consisting of 64 alternating light and dark squares. It is typically divided into an 8x8 grid, with each row and column being labeled with letters (a to h) and numbers (1 to 8), respectively. Additionally, the contrasting colors of the squares, usually black and white, create a visually appealing pattern. Furthermore, the chessboard serves as the arena for the ancient game of chess, where two players strategically maneuver own pieces to win the game. Thus, detecting and locating the chessboard is important. In order to detect and locate the chessboard, line-detection-based methods were used by Y. Xie et al.[2] and G. Ranganathan [8], combination of image processing techniques used by C. Belshe[4], detecting the chessboard's corner points approach was used by C. Matuszek et al.[6] and C. Koray and E. Sumer[7], simple binary image classifier was used by A. Mehta[10] and a single process stage was proposed by M. A. Czyzewski et al.[11].

In the method proposed by G. Ranganathan [8], only edge detector such as Canny Edge Detector were used and the acquired image is further segmented to detect pieces. In contrast, the techniques and algorithm[3] used by Y. Xie et al[2]. produces a high success rate for chessboard recognition, and more crucially, the range of their workable viewing angles corresponds to how a player would naturally look at the chessboard while playing. Prior to grouping the potential lines into two groups that represent two orthogonal sets of lines on the board depending on where they are placed in a scaled Hough transform space, all potential lines in an image of a chessboard are found using the Canny edge detector and the Hough transform. By examining the relationship between the detected lines, outlier lines in the same space are filtered out. Two more groupings of lines' crossings are calculated and noted next. Lastly, a reference model of a chessboard is compared to all possible candidates for chessboard sites after they have been changed. The location with the highest number of correctly matched corners and the lowest matching residual error determines the system output.

After the chessboard lines are found, the equations in Figure 2.2.1 may be used to determine the board's posture with regard to the camera. The optical center of the picture, the camera focal length in pixels, and the coordinates of the vanishing points on the image plane are represented by the symbols  $c_x$ ,  $c_y$ , and  $f$ , respectively. After  $R_x$  and  $R_y$  are found,  $R_z$ , the rotation matrix can also be found by taking cross product of both of them. Furthermore, a hyperplane using the chessboard coordinate system's  $R_z$  basis as the support vector and a fixed constant is defined. Posterior of performing both the operations stated previously, possible

locations and appearance of the chess pieces can be found and size scale factor can be controlled automatically. The result of the preprocessed chessboard is shown in Figure 2.2.2 with color markers.

$$R_x = K^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \quad R_y = K^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \quad (1)$$

$$K^{-1} = \begin{bmatrix} \frac{1}{f} & 0 & -\frac{c_x}{f} \\ 0 & \frac{1}{f} & -\frac{c_y}{f} \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$\langle R_x, R_y \rangle = 0 \quad (3)$$

Figure 2.2.1 : Equations that forecast where the chess pieces could end up and how they might look [1].

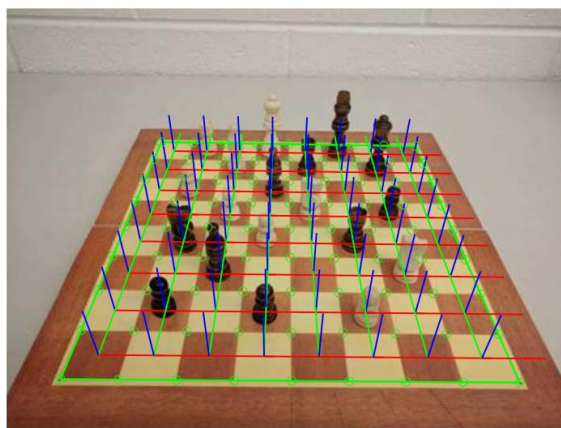


Figure 2.2.2 : The outcome of the chessboard preprocessing. Green lines denote the board's bounds, while blue sticks show each square's normal vector. [2].

Besides, C. Belshe[4] uses combination of image processing techniques in detecting the chessboard. The chessboard is first preprocessed such as image resizing and grayscale conversion and followed by contour detector such as findContours() method provided in OpenCV. The corners of the chessboard will then be found and the chessboard will then warped to obtain an image containing only the chessboard to reduce noises in the background. Next, the grids which contain pieces will be detected by using canny edge detection, followed by, Hough Line Transform and finally, the points of the square grids will be determined the intersection points of all the lines found previously. The results of each image processing techniques is shown in Figure 2.2.3.

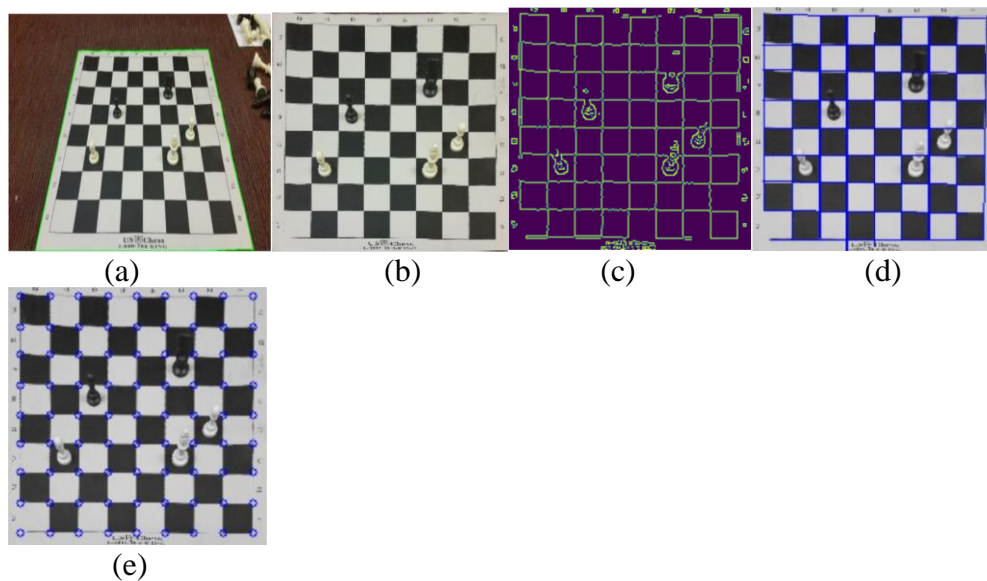


Figure 2.2.3 a) Contour of chessboard, b) Warped perspective of chessboard ,c) Canny edge detection, d) Hough Lines, e) Square vertices[4].

Additionally, in order to localize the chessboard and the square grids, C. Matuszek et al.[6] locate the chessboard by recognizing every corner point present on the 2D(RGB) picture of the chessboard. The corner points' depth data will then be included, and a plane will be fitted to the points using RANSAC. This creates the plane on which the board surface is located, but it does not precisely determine where the board will be placed. In order to localize the board, the best match of all 3D points on the board plane will be found using a template of 8 by 8 contiguous chessboard cells. The result can be seen in Figure 2.2.4. By identifying the board itself and its transformation formation with regard to the camera, this method allows the board posture with respect to the camera to be updated continually. Furthermore, this method is resistant to partial corner occlusions caused by hands or other objects.



Figure 2.2.4 Board corner points in white[6].

Moreover, C. Koray and E. Sumer[7] uses the grid corner point information to determine the location of the board. Firstly, all grid corner points of the chessboard will be found using a snapshot of the camera. Next, detectCheckerboardPoints method of MATLAB

will be used to find the grid corners. As a preprocessing phase, the color saturation of the acquired picture is gradually enhanced before all of the grid corners are found. After all the grid corners are located as shown in (b) in Figure 2.2.5, chessboard corners will now be located. By using diagonal closest inner point of the pivot points, the chessboard corners can be determined by reflecting the diagonal closest inner point to the opposite side as shown in (c) Figure 2.2.5. Upon locating the chessboard corners, geometric rectification will now be applied to the chessboard in order to separate it from its surroundings and correct perspective distortion in preparation for further procedures.. By utilizing the corner points located previously, the chessboard is warped from them to coincide with the predetermined 480x480px square corners as shown in the left image and the warped chessboard can be seen in the right image in Figure 2.2.6 respectively.

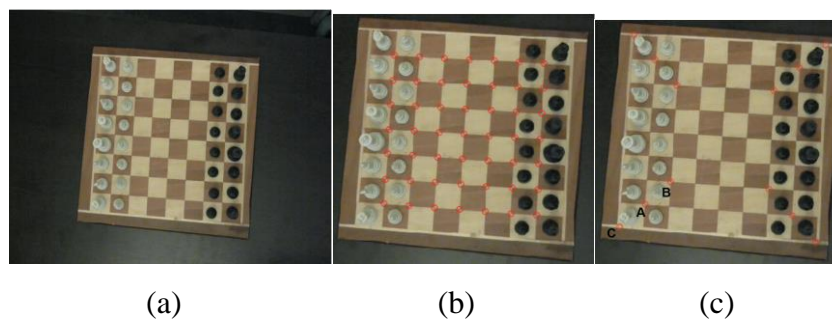


Figure 2.2.5 (a) The original chessboard image, (b) the detected chessboard grid corners and (c) the related points to chessboard grid corner detection[7].

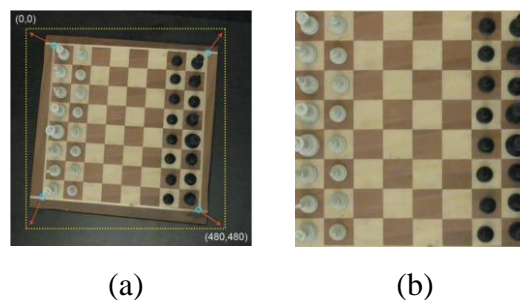


Figure 2.2.6 (a) The chessboard before geometric rectification step, (b) The chessboard after geometric rectification step[7].

Other than that, A. Mehta[10] had used a single binary classifier in chessboard detection. Prior to detecting the chessboard, a simple binary classifier is developed and trained using 95 images with a ratio of 8:2 training/test split. Upon detecting the chessboard, in order to combine all short, almost colinear segments into a single, long straight line and to find the corresponding parallel lines, parallel as well as straight lines will be discovered, and the bounding box will be determined next to insulate the chessboard from the environment and background noises. In detecting straight and parallel lines, operations such as edge detection,

line detection, grouping of segments, merging and filtering non-parallel lines are used, results in potential lines that represent the chessboard. Posterior to detecting the lines, the intersections of those lines will be found and will merged together using K-cluster. Upon getting the point of intersections, the bounding box will be obtained by getting the lowest and maximum values at which those points cross. Finally, the chessboard will be segmented into 8 by 8 squares, outputting an series of 8x8 photos for later processes.

Moreover, a single process stage was proposed by M. A. Czyzewski et al. [11] which uses relatively novel approach in discovering an improved estimation of the image's chessboard location. This approach employs a hybrid method that modifies popular computer vision techniques based on neural network-augmented algorithms in order to increase the approach's efficacy. Furthermore, the approach will identify the features of structures in a picture, such as lattice points and lines, whose forms and positions are assessed using a scoring function known as polyscore. The polyscore value is then used in defining the temperature of each point in the heat map and identifying section that representing a single chessboard. The overall processes in generating a heatmap is as follows :1)straight line detector(SLID) , 2)lattice points search(LAPS), 3)utilizing neural network to ease the geometric detector and 4)chessboard position search(CPS).

In the first step in the approach proposed by M. A. Czyzewski et al.[11], which is detecting the straight lines, the authors used a SLID algorithms that comprises three main steps, which are boosting, grouping and merging as shown in Figure 2.2.7. The boosting step will use the gradiental threshold approach and different contrast limited adaptive histogram equalization (CLAHE) masks, we locate all potential segments by doing repeated analyses on the same image, whereas the grouping step will use a grouping function to divide segments into groups of almost collinear segments, and finally the merging step will utilize M-estimator to examine and combine segments found in previous step in each group, producing a single normalized straight line. This algorithm will be executed until no more lines are detected.

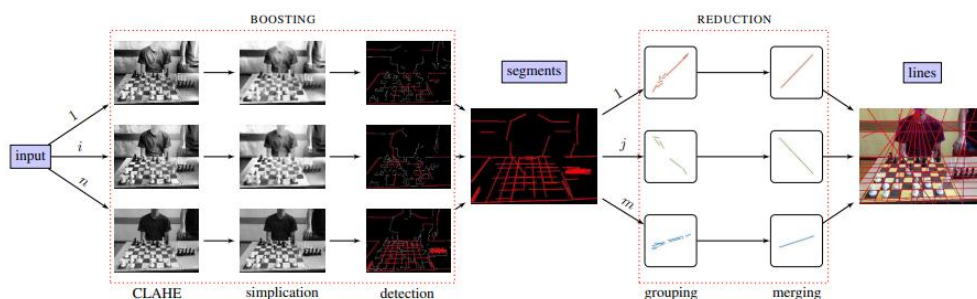


Figure 2.2.7 Straight line detection[11].

Following SLID, the LAPS algorithm will be run to process each intersection of a pair of lines that SLID has identified as likely lattice points. This will include using both geometric and neural detectors, and the result will be a list of points that have been identified as likely lattice sites. The algorithm acquire 21x21 matrix, which will be preprocessed and handled by the detectors next. The geometric detector will recognizes perfect cases only whereas the neural network detector will be used in identifying twisted and malformed patterns. It is presumed that the geometric detector's positive output indicates a lattice point on a chessboard. If not, a neural network detector based on a convolutional neural network (CNN) will be employed. It has two layers: a flattened layer with a 0.5 dropout function and a 2D layer with 12 filters. The results can be seen in Figure 2.2.8.

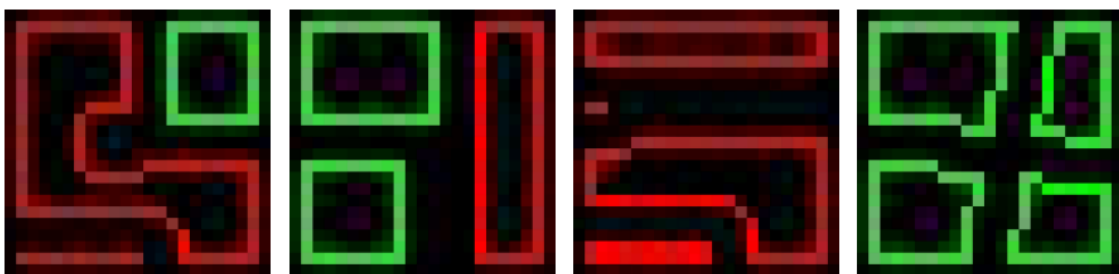


Figure 2.2.8 Examples of the geometric detector's findings. The color green denotes rhomboids that the detector detected. Since it is the sole matrix with four rhomboids, only the rightmost matrix will be categorized as a lattice point. [11].

After LAPS algorithm, a CPS algorithm will then be used in searching for positions of chessboard by analyzing a four-side frame that can enclose a chessboard. The steps of the algorithm are as follows : 1)select a cluster of lattice points consists of largest number of points generated by LAPS, 2) calculate width of chessboard square by square rooting surface area of group G(cluster selected) divided by 7, 3) Discover lines generated by SLID for each chessboard lattice point in G that meet conditions stated, 4) Separate the lines identified in the previous step into two groups which is horizontal and vertical lines group, 5) Compute polyscore of each frame where the frame is formed by taking pair of lines from each group, finally, select the frame that has the maximum polyscore. As a result frame with highest polyscore will be obtained to draw the heat map. The result of the polyscores is shown in Figure 2.2.9.

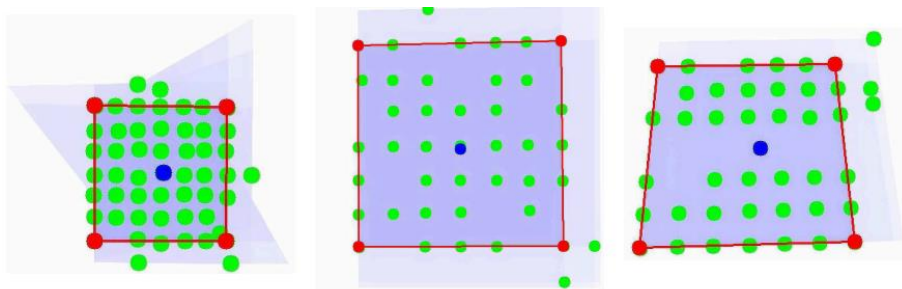


Figure 2.2.9. Values of the polyscore are computed for different frames. The red border designates the frame with the highest score. Darker blue backgrounds emphasize higher scores.[11].

The function of scoring  $P(F)$ , also known as polyscore, is used to determine the probability that a frame  $F$  consisting of four lines would have a chessboard, as shown in Figure 2.2.10.  $L$  is the total amount of points within the frame;  $k$  is the mean distance between the points inside the frame and the nearest side of this frame, and  $l$  is the distance between the group  $G$  centroid and frame  $F$  centroid. The weight function  $W_i(x)$  adds a weight  $i$  on the area of frame  $F$ , or  $A_F$ , as a function of  $x$ . The goal of the function  $P(F)$ 's highest value for a frame  $F$  is to precisely represent a trimmed chessboard.

$$W_i(x) = \frac{1}{1 + \left(\frac{x}{A_F}\right)^{\frac{1}{i}}} \quad (10)$$

$$P(F) = \frac{L^4}{A_F^2} \cdot W_3(k) \cdot W_5(l) \quad (11)$$

Figure 2.2.10. Equations for polyscore function  $P(F)$ [11].

### 2.3 Chess Piece Detection and Recognition

In a game of chess, each player controls a total of 16 pieces, which are 8 pawns, 2 rook, 2 knight, 2 bishop 1 queen and 1 king. Additionally, each player will control different color in adverse of their opponent such as white and black. Furthermore, each of the pieces has unique features that make them differentiable. According to the standard chess set such as the standard modern Staunton wood chess set[14] as shown in Figure 2.3.1, a pawn typically has a spherical top and it is the smallest piece in a chess set. Moreover, a rook has a top that resemble the castle wall and is slightly taller than a pawn. Besides, a knight has the most distinguishable feature which is the head that resemble a horse head and has the same height as a rook. Next, a bishop has a top which resemble the hat that a pope wears and is slightly taller than a knight. Furthermore, a queen, has a top that resemble a queen crown and is the second tallest piece in a chess set and finally, a king is the tallest piece in a chess set and the top resemble the imperial

state crown wear by the British king/queen. On the other hand, all the pieces has a similar bottom part which is round in shape.



Figure 2.3.1 Standard modern Staunton wood chess set[14].

There are several proposed methods and approaches in detecting and recognizing the chess pieces, including Oriented Chamfer Matching[2], machine learning approach such as SVM[6] and deep learning approach such as CNN [2],[4],[5],[9],[10].

As proposed by Y. Xie et al.[2], Oriented Chamfer Matching is used to detect and classifier the chess pieces. An established contour matching method that compares items in the input image to templates is defined as Oriented Chamfer Matching. A chamfer matching score for each candidate item position is computed. The template and the region with the lowest chamfer matching score define the object's class and position. Following template matching, the location of the template with the lowest oriented chamfer matching score for each AOI will be indicated on the input picture. High-scoring templates are disregarded.

Apart from that, C. Matuszek et al.[6] uses a binary linear SVM using LIBLinear in detecting and recognizing the pieces on the board. The machine is trained with images with resolution 240x240 – 300-300px padded and cropped images obtained from square detector. In accordance with Figure 2.3.2, square detector will spot square on the board and crop it. The cropped square image will go through a piece/background detector which is the SVM to detect whether there is piece currently occupying the square or otherwise. If a piece is detected, the piece will be classified, which followed by cropping the image to undergoes color detection on the piece. Finally, a piece will be detected and classified with the correct piece type and color.



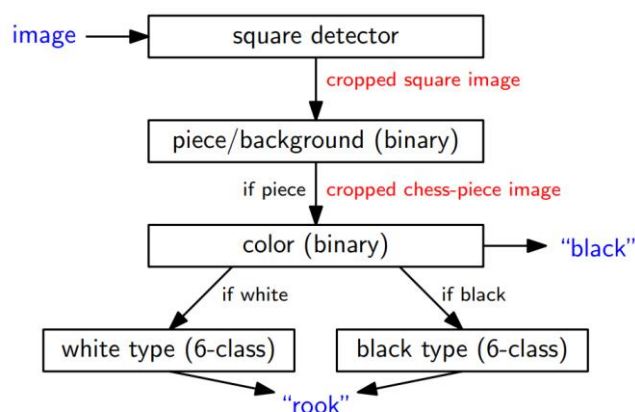


Figure 2.3.2 Classifier hierarchy used to recognize chess pieces during a game[6].

Other than that, deep learning approach also used in performing detection and recognition task. In comparison of approaches by Y. Xie et al.[2], various most popular CNNs had been used such as GoogleNet[17], ResNet[18] and VGG[19]. The networks' final layers are changed to a softmax regression with six output nodes to accommodate the piece recognition application, and all test images are shrunk to 223x223x3 pixels as a result. The Adam optimization method [20] is used, with a maximum number of 1000 iterations and a learning rate of 0.001. As a result of comparison, most of the CNNs had outperformed Oriented Chamfer Matching in terms of recognition accuracy as shown in Table 2.3.1

	King	Queen	Bishop	Knight	Rook	Pawn	Overall
GoogleNet	97.67%	100.00%	100.00%	100.00%	97.96%	96.53%	98.14%
VGG-16	100.00%	90.63%	97.37%	98.41%	87.76%	99.42%	96.08%
ResNet50	88.37%	100.00%	100.00%	100.00%	81.63%	97.69%	94.43%
Oriented Chamfer	90.70%	90.63%	85.53%	100.00%	95.92%	100.00%	95.46%

Table 2.3.1 The recognition accuracy of various methods[2].

Besides, C. Belshe[4] had uses YOLOv4, a CNN that uses 53 convolutions and CSPDarknet53 for its backbone as a feature extractor. In order for the chess piece detection to be useful in a real chess game, the model must be able to run quickly enough to recognize objects in real time after training. It was picked as the best neural network-based object detection method since it is both fairly accurate and quick. The overall structure of YOLOv4 consists of 2 detectors which are one-stage detector that comprises the input image, backbone which are the convolution layers, the neck that uses SPP and a path aggregation network to enhance the receptive field of the backbone by dividing the feature map into equal-sized blocks, or "spatial pyramids," and integrating the pyramids into a CNN while employing max pooling to lower the size (max pooling eliminates non-max features), and dense prediction step and two-stage detector that consists one-stage detector and a sparse prediction as shown in Figure

2.3.3. In comparison with others state-of-art object detectors, YOLOv4 runs twice faster than EfficientDet with comparable performance.

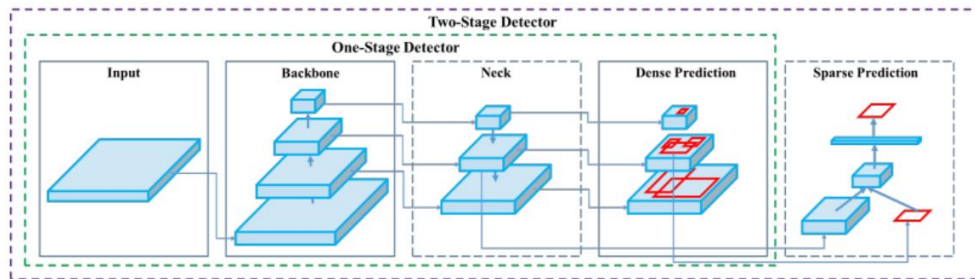


Figure 2.3.3 Object Detection Structure of YOLOv4[16].

Furthermore, a volumetric CNN had been trained by Y. -A. Wei et.al[5] to classify the chess pieces. The network receives a tensor with dimensions of 50x50x100px as input. Two convolutional layers with pooling size of 5x5x5px and one max-pooling layer with pooling size of 2x2x2 are placed after the input layer. The pooling result is shaped after pooling and add two fully connected (FC) layers. The dimension is shrunk by the first FC layer to 128 and by the second FC layer to six. Between the first three layers, there is ReLU and dropout layers. The overview of the volumetric CNN can be seen in Figure 2.3.4. Additionally, the network is implemented in Torch[21]. The classification result of the network is shown in Table 2.3.2 which considered good.

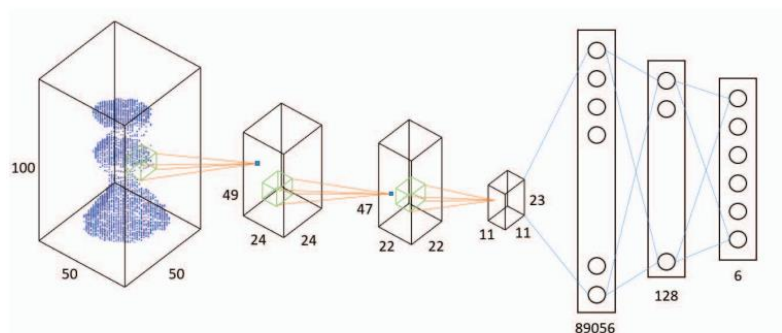


Figure 2.3.4 Overview of volumetric CNN[5].

	bishop	king	knight	pawn	queen	rook
bishop	0.96	0	0.032	0.008	0	0
king	0	0.568	0.01	0	0.422	0
knight	0	0	0.953	0.005	0	0.042
pawn	0	0	0	0.946	0	0.054
queen	0	0	0.001	0	0.999	0
rook	0	0	0	0.053	0	0.947

Table 2.3.2 Confusion matrix of classification results[5].

Aside from that, A. Indreswaran[9] had proposed 2 models based on CNN which are 1-step model and 2-step model where the former utilizes color image to predict the results whereas the latter is a unsupervised learning model that has no prior knowledge that requires

the model breakdown the logics and find underlying patterns and features. In the 1-step model as shown in Figure 2.3.5, the feature constructor receives an image with a size of 256x256 pixels, which it first sends through two convolution + ReLU layers in succession, each having 60 convolution filters of size 5x5. To ensure that no significant information is lost, only one stride was employed for the second convolution layer as opposed to two for the first. While the second convolution layer was intended for a higher feature like a line or an edge, the first convolution layer was designed to function as an edge detector. With a filter of size [2x2] and a stride of 1, the output is then pooled for the largest possible amount. The output of the pooling layer is then fed into the convolution+ReLU layer using a convolutional layer made up of 60 filters with a stride of one and a size of 5x5. It is followed by stride 1, which also pools for the maximum, then the final layer of pooling of size [2x2]. The image size was decreased and only the crucial details were filtered using the pooling layers. Additional layer pooling reduces over-fitting of the data.

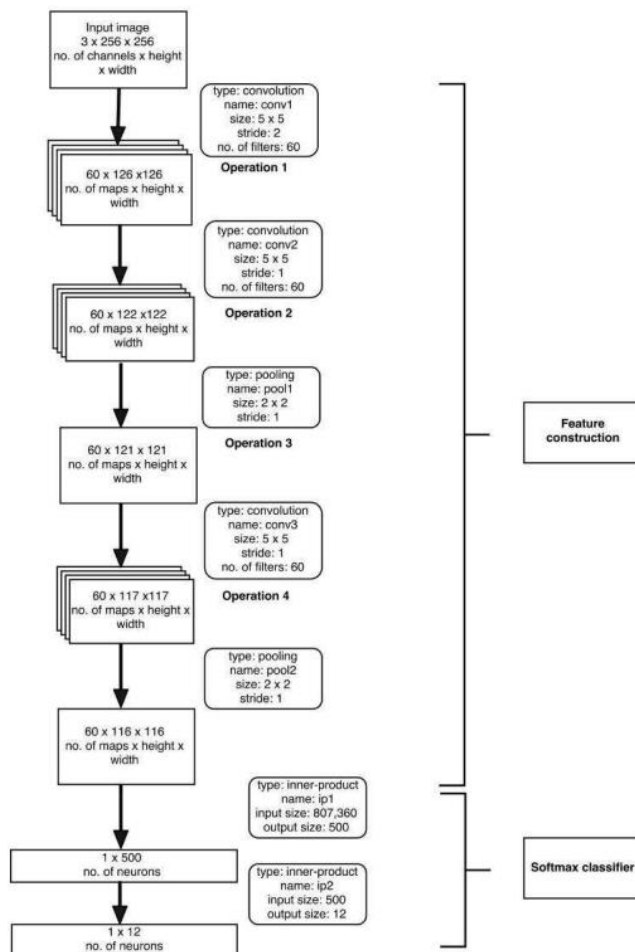


Figure 2.3.5 Architecture of the 1-Step Model[9].

As for 2-step model, it consists of a binary classifier that processes image in HSV color-space and estimate the chess piece color and a piece-wise classifier that processes gray-scale image and classifies the input image according to piece geometry. In accordance to Figure 2.3.6, the model includes four layers in total, with two levels for features extraction and two layers for classification. A convolution layer with a [3x3] size, a two-filter stride, and 20 filters serves as the layer for feature extraction. Using a higher stride in a low filter size was appropriate because only the piece's color information was needed. The pooling layer has a maximum pooling stride of one and a size of [3x3]. 16,820 pixels were inputted into a Softmax classifier with a hidden layer of size 200 neurons. The output layer, which contained 2 neurons in accordance with the number of classes, black and white, receives the output from the 200 neurons.

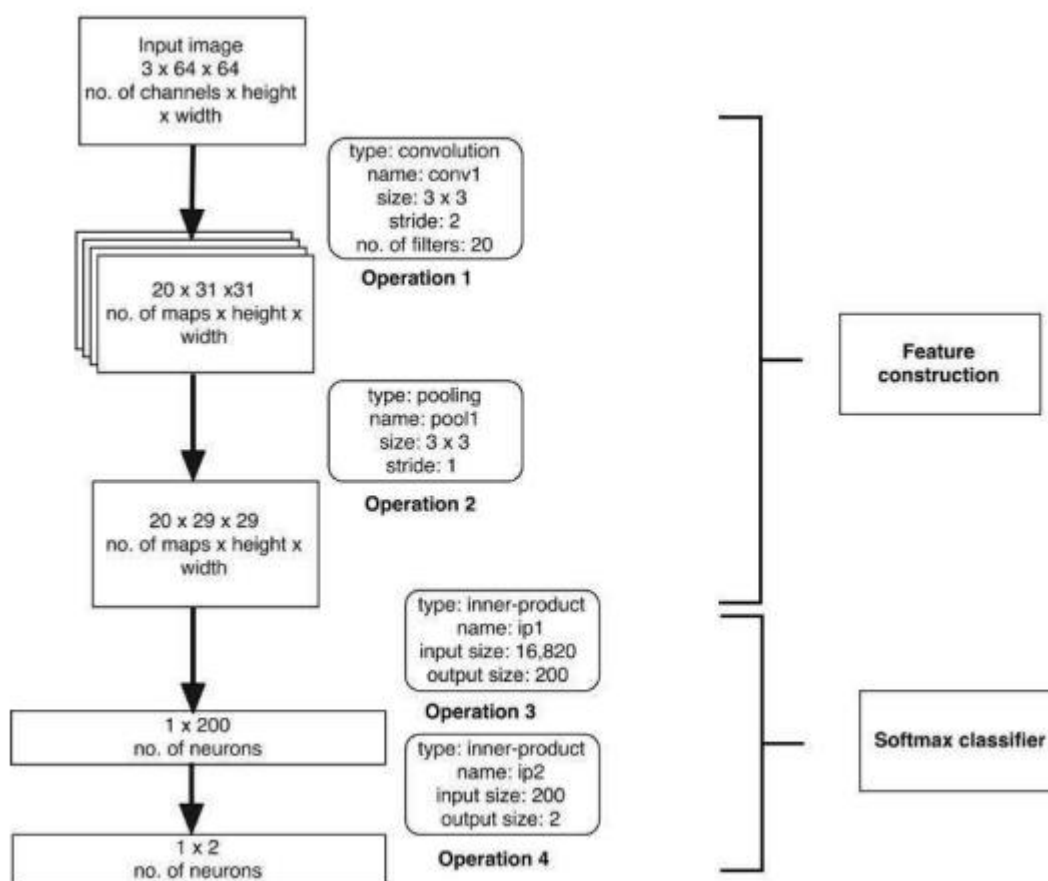


Figure 2.3.6 Architecture of the Binary classifier in the 2-Step model[9].

Whereas for piece-wise classifier in the 2-step model, according to Figure 2.3.7, the network comprises six layers in total, counting the ReLU layers as a part of the layer it activates and excluding the input layer. The first four levels are used to build features, and the final two layers are used to classify data. Two consecutive convolution+ ReLU+pooling layer

combinations make up the feature constructor, which takes as input a grayscale image with a 256 x 256 pixel size. The first convolution layer has 20 filters, the last one has 50 filters, and each convolution layer has a filter size of [5x5] with a stride of 1. In order to reduce the amount of data that needs to be processed and hence shorten the network's computation time, the pooling layers are of filter size [2x2] with stride two. The output layer, which comprises six neurons for each of the six item classes, receives information from the hidden layer's 500 neurons with an input size of 186,050 pixels.

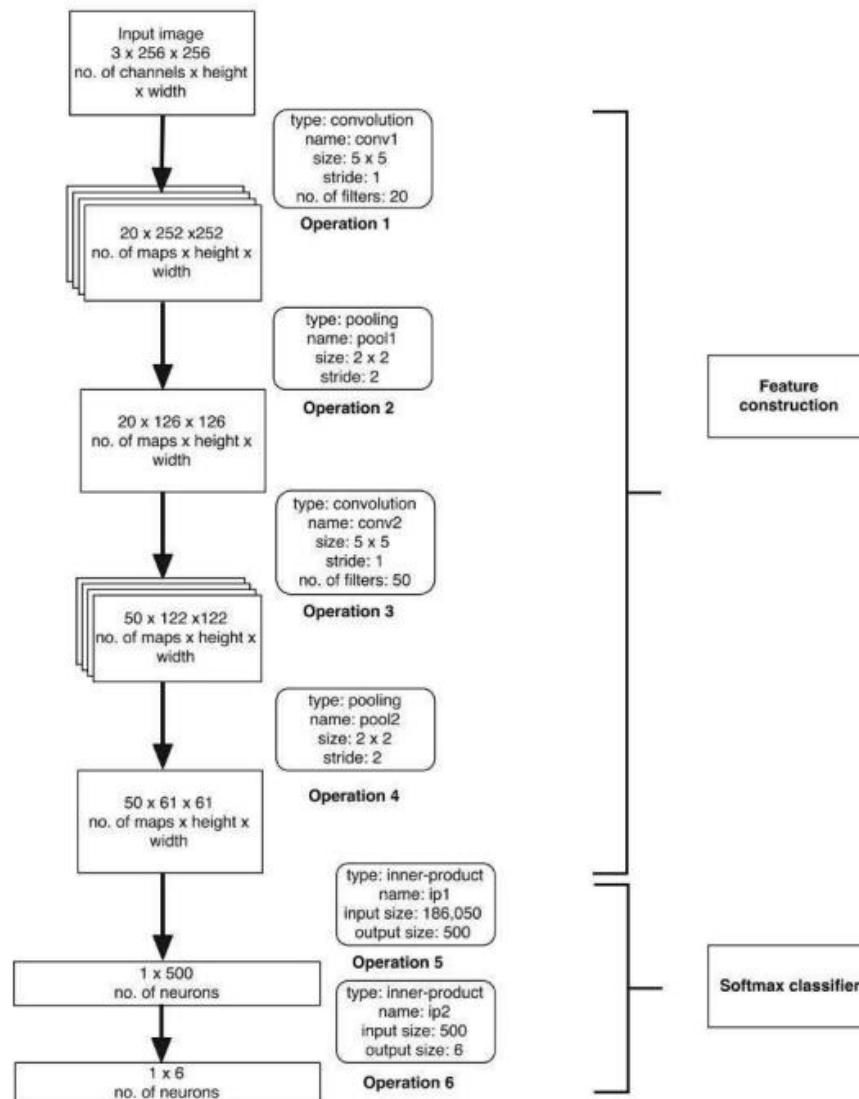


Figure 2.3.7 Architecture of the Shape-based classifier in the 2-Step model[9].

Moreover, A. Mehta[10] also uses CNN approach in chess piece detection and recognition, specifically, AlexNet. The training dataset used by the author is manually constructed from one tournament chess set and every image of individual piece is manually labelled with one of the 13 classes. The training dataset is splitted into 80/20% training and test

set to validate the performance of the model. The training dataset will then fed into AlexNet to training the model via transfer learning. Transfer learning uses low level features that have already been mastered (such as lines, edges, and curves) and takes less data to produce a good CNN. As shown in Figure 2.3.8, eight layers make up AlexNet; the first five were convolutional layers, some of them were followed by max-pooling layers, and the final three were fully linked layers, using a SoftMax activation function. In general, the model perform quite well with maximum accuracy of 94.23% at FP32 bit and a minimum of 93.15% at FP8 bit as shown in Table 2.3.3.

AlexNet model (bit)	Size	Accuracy
FP32	221MB	94.23%
FP16	106MB	93.45%
FP8 .	48MB .	93.15%

Table 2.3.3 Accuracy and model precision[10].

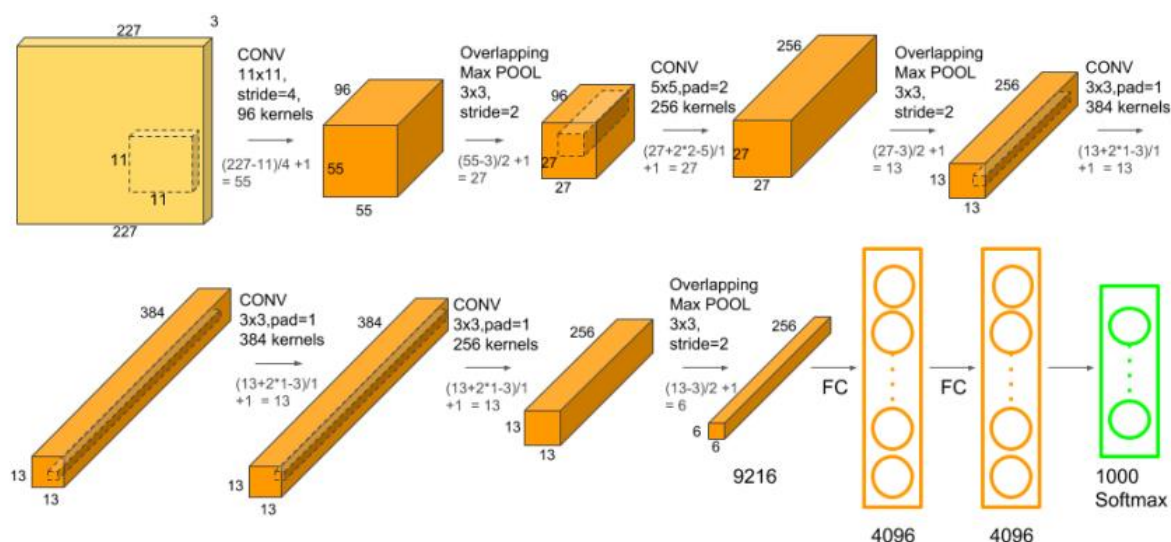


Figure 2.3.8 The overall structure of AlexNet[10].

## 2.4 Chess Move Detection

The move detection methods and approaches proposed by C. Koray and E. Sumer[7], G. Ranganathan[8] and G. Petkov[12] is similar in concept, which is the comparison between the two snapshots where one of them is the reference image and the other one is the one made after a move is played by the player. The first photo that is taken following each legal motion is the reference image. The initial reference photo is thought of as the first still from the video. Throughout the game, the mean color difference between the reference image and the snapshots is calculated. When the outcome of the computation goes above a specific threshold, it is safe

to say that the player has moved, presuming that the player completed the move once the result fell below the threshold.

Next, the last snapshot is interpreted to find out chess piece colors and positions. Prior to this procedure, the last snapshot is warped, and then the improvements are applied to the chessboard's warped picture. The four reference colors established are compared to the ROI within each square of the image. By calculating the deltaE value, which reflects the Euclidean distance between the associated items in this comparison, the color differences are determined in Lab color space. The reference color that yields the lowest deltaE value in the comparisons decides whether a grid cell is a square or a segment of bright or dark hue. It is possible to determine the chessboard's condition as of the last snapshot by using this process on every square. To determine the piece's move, the state of last snapshot and the preceding chessboard state are compared. The last legal move's chessboard state is represented by the preceding chessboard state. Additionally, the first state of the game is saved at the start of play as the previous chessboard state.

Besides, there will be six outcomes using the mentioned approach, which are no change, detected chess move, castling, "en passant", piece capturing and invalid move. All six outcomes are detected differently with conditions such as :

- 1) If there is no difference between the prior and most recent states, then the game has not changed, thus not a move and no changes
- 2) If there is only a single difference between an occupied and an unoccupied square with the same piece color, then it is a move
- 3) When there are two occupied and two vacant squares that differ with the same piece color, a special move called castling is played,
- 4) Another unique move known as "en passant" is played if there are one empty square difference with the opposing piece color and one occupied and one unoccupied square difference with the same piece color.
- 5) The comparison's outcome is not a move for any other circumstance.

Upon determining the outcome, if there is a valid move, the move will be recorded using standard algebraic notation which is a notation standardized by World Chess Federation(FIDE)[11]. For instance, king is abbreviated by the letter "K," the queen is abbreviated by the letter "Q," the rook is abbreviated by the letter "R," and the bishop is abbreviated by the letter "B." The knight, a special case, is abbreviated by the letter "N" since "K" is already taken by the king. The pawn is the only piece that has no abbreviation. If a pawn

is moved, only the name of the square where the pawn moves will be recorded[15]. Additionally, as an example, the notation “Nf3” means knight to f file(column) rank(row) 3, “Bxc6” means bishop captures piece on c file rank 6, 0-0 means castling kingside and 0-0-0 means castling queenside.

### 2.5 Limitation of Previous Studies

In accordance with the approaches proposed by Y. Xie et al.[2], although in some cases that Oriented Chamfer Matching had a better performance as compared to CNN, labeled training images collection is more time consuming and is a huge difficulty for the user. As mentioned in above section, Oriented Chamfer Matching matches the input image with the template image provided by the user. In order to obtain an acceptable accuracy, the user needed to collect images either by themselves or obtain them in open sources. Further, the user also needed to draw a bounding box containing the target object and label them with the correct class. Hence, collecting the template images for the use of the model is a challenge for the user.

Besides, the method proposed by C. Belshe[4] has a decrease of accuracy as compared to the predicted accuracy and not resistant to changes in image layout. If there is a major changes in the image detected by the system, there is a high chance that the system will not be able to detect the chess pieces correctly as the model is overfitted to the trained model.

Moreover, the approach proposed by Y. -A. Wei[5] has major flaw which is the almost half of the king piece is classified as a queen piece. This is because the camera angle that supposed to be mounted on the chess playing robot is almost orthogonal to the chess board, making the king piece look almost the same as a queen piece, hence, lowering the accuracy of the system. Other than that, the training data are generated from a CAD model, which may not represent the real-world scenario, which may affect the accuracy of the system when the system is implemented in it.

Besides, the limitation that can found in the method proposed by C. Matuszek et al.[6] is similar to the approach proposed by Y. -A. Wei[5], which is the camera angle is almost orthogonal to the chessboard, hence, it is difficulty for the system to detect useful features that can be seen when the camera angle is 45 degrees, which is height, unique feature from each piece and many more. Other than that, lower accuracy will be obtained when the piece detector failed to detect the piece such that the piece color is the same as the background color.

Furthermore, the limitation that exists in the method proposed by C. Koray and E. Sumer[7] which is the move detection may be faulty when there is some random movement



above the chessboard. The system may incorrectly identify that random movement as a valid chess move, making the chess game faulty which require human intervention.

Aside from that, there exists computation time problem in method and approach proposed by A. Indreswaran[9]. The computation time of the system designed by the authors from loading image, resizing image, converting image color, preprocessing image, inputting image and classifying image takes around 3.82 seconds, which is much higher than the expected 1 second classification time in the 1-step model proposed. Other than that, the classification accuracy of the model is 70% in average which is considered low as compared to another model. As a result, the robustness of the model decreases when there is changes in scale of the object.

Furthermore, in the method and approach proposed by A. Mehta[10], it is intolerant to roll, yaw and pitch and detect hidden points and lines that are used to recognize partially hidden pieces. Additionally, there is only one chess set used which make the system less sensitive to other chess set, causing the system having difficulty in detecting the chess pieces. Other than that, the system only uses StockFish as the only chess engine, causing difficulty in desiring other chess engines based on the user preference or engine strength.

Aside from that, there exists flaws in the proposal mentioned by M. A. Czyzewski et.al.[11] which is it is slower than other alternative approaches. As speed is one of the factors in measuring the strength of a system, it is undisputed that the strength of the system is much weaker as compared to other system in a certain extend. Additionally, there is lack of comprehensive labeled datasets of images containing chessboard, causing more uncertainty for the system when recognizing different chess set.

Finally, the limitation exists in the system proposed by G. Petkov[12] is using few variation of chess set when training the model in the system. As mentioned previously, when the is not trained with different set of chess, the tolerance and accuracy of the system will drop drastically when begin used in different kind of scenario containing chess set that might not trained by the system.

## CHAPTER 3

### SYSTEM METHODOLOGY/ APPROACH

In this chapter, system methodology/approach used in this project will be addressed and well explained. In the following sub-chapters, explanations will be provided for the general system design as illustrated in the system architecture diagram, use case diagram and activity diagram.

#### 3.1 System Architecture Diagram

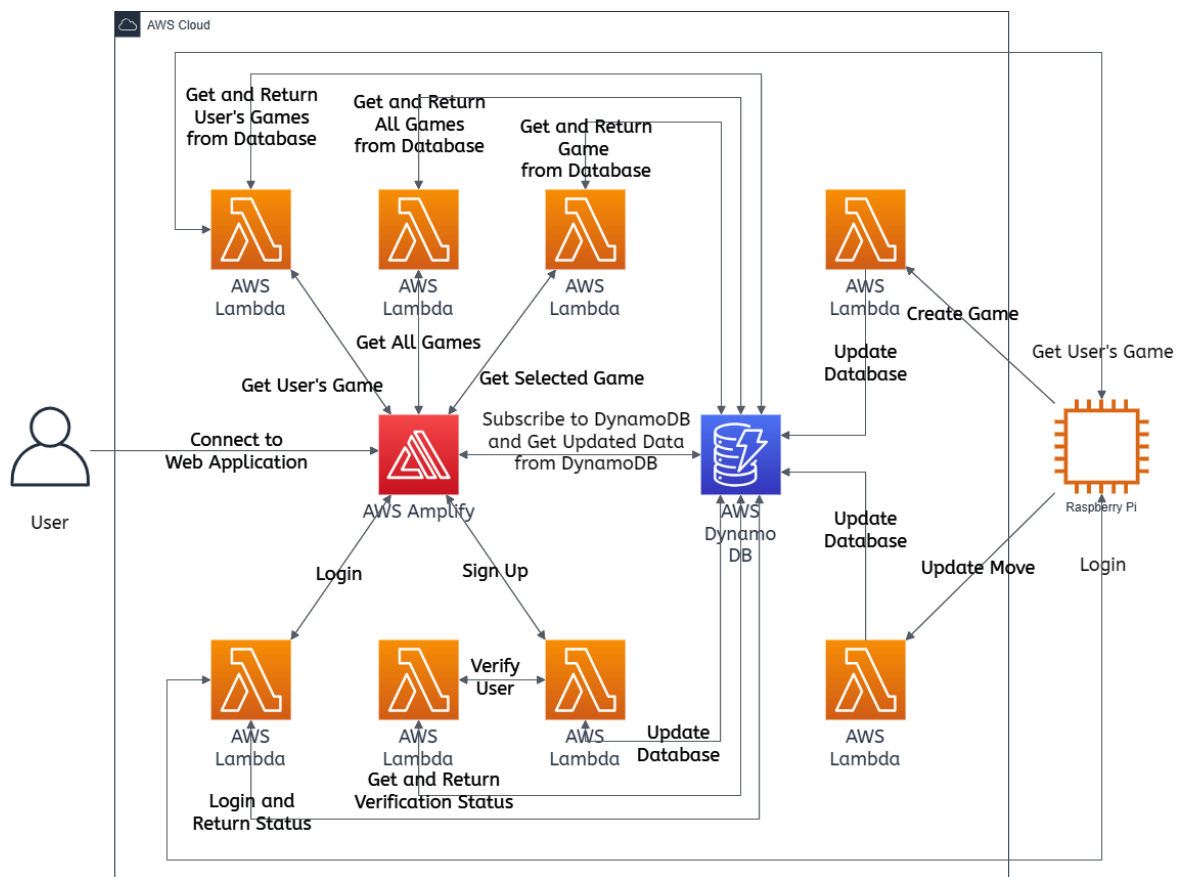


Figure 3.1.1 System Architecture Diagram.

In accordance with Figure 3.1.1, the framework or the system architecture is built using Amazon Web Services(AWS) services provided by Amazon cloud platform. There are 3 major AWS services used for the system architecture including AWS Amplify, AWS DynamoDB and AWS Lambda. In total, 1 AWS Amplify , 1 AWS DynamoDB and 8 AWS Lambda is used for the system. In the following sub-chapters, brief introduction for the AWS services and the explanation of the general flow and process of the system will be provided.

### 3.1.1 Amazon Web Services(AWS) Used

AWS Amplify, a robust development platform from AWS, streamlines the creation and deployment of full-stack applications. It provides an array of tools and services to expedite development, covering frontend and backend development, authentication, storage, APIs, and analytics. Amplify enables developers to swiftly build scalable and secure applications, utilizing cloud resources without the hassle of infrastructure management. Its compatibility with common frameworks and libraries, along with features such as continuous deployment and automatic scaling, renders it an effective choice for crafting contemporary web and mobile apps.

AWS DynamoDB, offered by Amazon Web Services (AWS), is a fully managed NoSQL database service renowned for its rapid and adaptable document and key-value storage capabilities. Its exceptional speed, delivering data access within single-digit milliseconds, caters to applications demanding swift response times regardless of scale. DynamoDB seamlessly adjusts to accommodate surges in traffic and accommodates both simple key-value access and intricate query operations through secondary indexes. By handling tasks like hardware setup, configuration, replication, and maintenance, DynamoDB liberates developers from infrastructure management, empowering them to concentrate on application development. Its pay-per-request pricing structure ensures cost efficiency across various applications such as web and mobile platforms, gaming, IoT, and beyond.

AWS Lambda, a service provided by Amazon Web Services (AWS), allows developers to execute code without the need to provision or oversee servers. With Lambda, developers can upload their code and set up triggers for events like modifications to data in Amazon S3 buckets, updates to DynamoDB tables, or HTTP requests via API Gateway. Upon the occurrence of these events, Lambda automatically runs the code, adjusting to the volume of incoming requests and billing based solely on the compute time utilized. Developers can write code in different languages like Node.js, Python, or Java, enabling them to use their preferred language and concentrate on writing business logic rather than managing infrastructure. This serverless approach brings advantages such as decreased operational burden, enhanced scalability, and cost-effectiveness, making Lambda a favored solution for constructing event-driven and microservices architectures in cloud environments.

### 3.1.2 General Flow and Process

With reference to Figure 3.1.1, first of all, user can access the web application hosted by AWS Amplify by using the unique link generated by it, which will redirect user to the home page of the web application. Upon accessing the home page, the web application will send request to or trigger AWS Lambda to get all games that are set accessible to public and are live games. The AWS Lambda will get all the relevant games from AWS DynamoDB, that is the database of this system and return it to the web application for user to select upon.

Moreover, user may login using their registered account .When user submits the login form, the web application will trigger another AWS Lambda to verify the identity of the user by checking the existent of such account in AWS DynamoDB and return the status of the authentication back to web application. If user is authenticated, the web application will redirect user back to home page with the authenticated account. If the user does not have an account, the user may sign up an account by providing username and password. Upon submitting the sign-up form, the web application will send request to AWS Lambda that handle the sign-up request which subsequently trigger another AWS Lambda that check the existent of account such that it is registered with the same username in AWS DynamoDB. If there is no such username registered in the database, the sign-up process will be completed and new account will be added to the database. Subsequently, the user will be redirected back to home page upon the completion of sign-up process with the registered account.

When user successfully login or signing up, user will be redirected back to the home page. Different from first accessing the web application, user now accessing the home page with an account. Thus, the web application will trigger an AWS Lambda to get games created under the user's account from AWS DynamoDB and return it to the web application for display and selection purposes. Posterior to selecting the desired game to watch or review, the web application will redirect user to the chess page. At the same time, the web application will send request to the AWS Lambda to get the details of the selected game such as moves, opponent name, result and many more for display purposes. If the game is not complete, the web application will utilize the subscription function of GraphQL that allow the AWS DynamoDB to return any updated data to the chess page, hence, achieving real time display of chess game state.

On the other hand, user can login with the registered account and get all the games created under the account. Besides, user can also create game by utilizing the GUI implemented

in the raspberry pi . Upon filling the required details such as which side is the user playing(white or black), opponent name, whether wants the game to be accessible to public and more and submitting the form, the raspberry pi will send a request to AWS Lambda to create the game with the details and update AWS DynamoDB. After creating game, user can now start to track the game. Posterior of detecting a move, the raspberry pi will send the move to AWS Lambda to update the chess game state to AWS DynamoDB. When there is an update to the database, AWS DynamoDB will send the updated data to the web application when it successfully subscribed to the database. After obtaining the updated data, the updated move will now be displayed on the page in the web application.

### 3.2 Use Case Diagram and Description

#### 3.2.1 Use Case Diagram

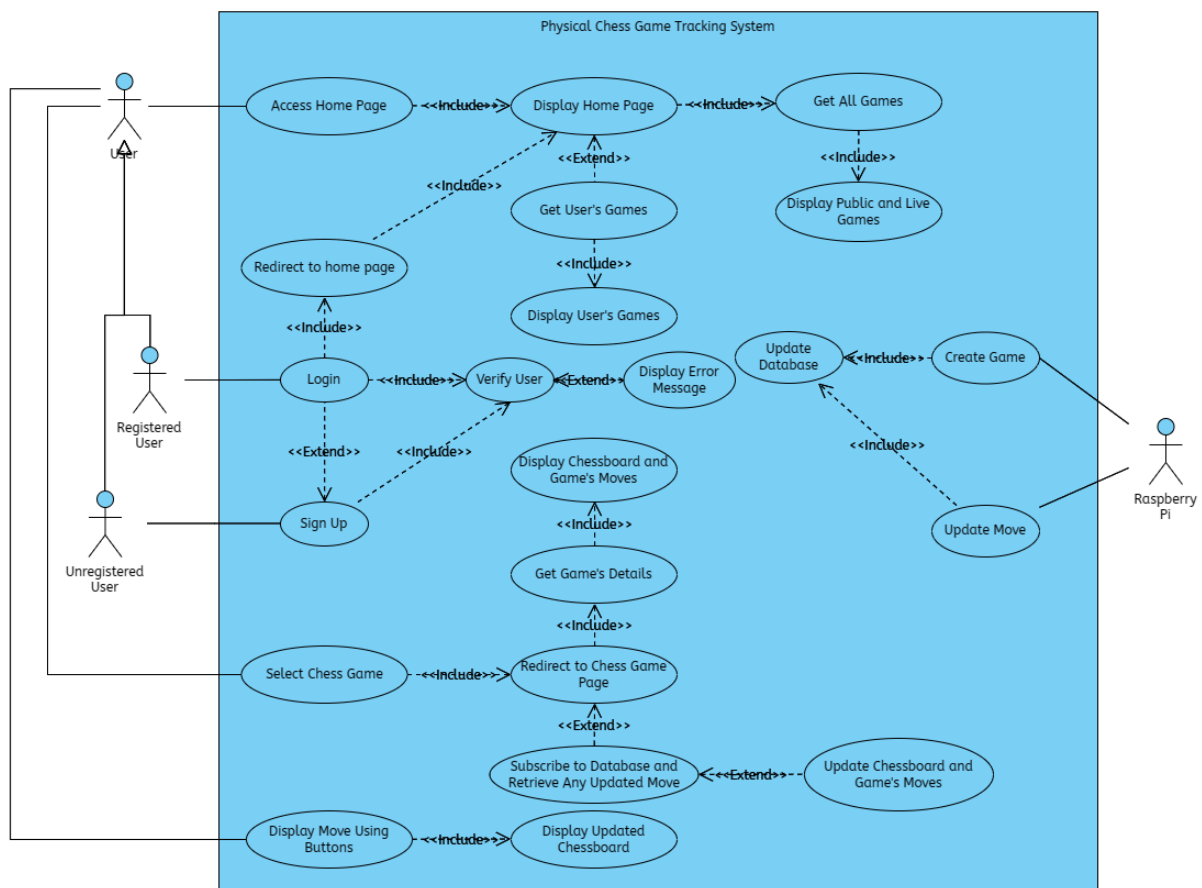


Figure 3.2.1.1. Use case diagram of the system.

In the use case diagram illustrated in Figure 3.2.1.1, there are a total of 7 major use cases which are Access Home Page, Login, Sign Up, Select Chess Game, Display Move Using Buttons, Create Game and Update Move associated with the actors, that are, User and Raspberry Pi. In the following sub-chapter, the use case description will be focused on these 7 use cases.

### 3.2.2 Use Case Description

Use Case ID:	CE_UC_1
Use Case Name:	Access Home Page
Description:	This use case allows user to access the system to view the games saved in the database no matter the user is registered or unregistered.
Primary Actor:	User
Preconditions:	1. User access the web application with the unique link
Postcondition:	1. The system displays the home page with game list with details describing the particular game.
Main Flow:	<ol style="list-style-type: none"> <li>1. User enter the unique link of the web application.</li> <li>2. The system trigger Display Home Page use case.</li> <li>3. The system trigger Get All Games use case.</li> <li>4. The system trigger Display Public and Live Games use case.</li> <li>5. The system check user login session.</li> </ol>
Alternative Flows:	<p>5a Login session found.</p> <ol style="list-style-type: none"> <li>1. The system trigger Get User's Games use case.</li> <li>2. The system trigger Display User's Games use case.</li> </ol> <p>5b Login session not found.</p> <ol style="list-style-type: none"> <li>1. The system do nothing.</li> </ol>

Table 3.2.2.1 Use case Description for Access Home Page use case.

Use Case ID:	CE_UC_2
Use Case Name:	Login
Description:	This use case allows user to login into the system to view the games created and tracked under the registered account that user is going to use in login in this use case. In order to login into the system, user has to enter username and password.
Primary Actor:	Registered User
Preconditions:	1. User accessed the home page.
Postcondition:	1. The system save login session and display the home page with game list with details describing the particular game.
Main Flow:	<ol style="list-style-type: none"> <li>1. User enter username and password.</li> <li>2. The system perform authentication by triggering Verify User use case.</li> <li>3. The system trigger Redirect to Home Page use case.</li> <li>4. The system trigger Display Home Page use case.</li> <li>5. The system trigger Get All Games use case.</li> <li>6. The system trigger Display Public and Live Games use case.</li> <li>7. The system check user login session.</li> </ol>
Alternative Flows:	<p>2a Invalid username and/or password</p> <ol style="list-style-type: none"> <li>1. The system trigger Display Error Message use case.</li> <li>2. The system prompts for username and password.</li> <li>3. Use case resumes at main flow step 1.</li> </ol> <p>7a Login session found.</p> <ol style="list-style-type: none"> <li>1. The system trigger Get User's Games use case.</li> <li>2. The system trigger Display User's Games use case.</li> </ol> <p>7b Login session not found.</p> <ol style="list-style-type: none"> <li>1. The system do nothing.</li> </ol>

Table 3.2.2.2 Use case Description for Login use case.

Use Case ID:	CE_UC_3
Use Case Name:	Sign Up
Description:	This use case allows user to sign up an account to start creating and tracking game. In order to sign up an account, user has to enter new username and password.
Primary Actor:	Unregistered User
Preconditions:	1. User accessed the home page.
Postcondition:	1. The system login user with the newly registered account and redirect user back to home page.
Main Flow:	<ol style="list-style-type: none"> <li>1. User enter username and password.</li> <li>2. The system perform authentication by triggering Verify User use case.</li> <li>3. The system trigger Login use case.</li> <li>4. The system trigger Redirect to home page use case.</li> <li>5. The system trigger Display Home Page use case.</li> <li>6. The system trigger Get All Games use case.</li> <li>7. The system trigger Display Public and Live Games use case.</li> <li>8. The system check user login session.</li> </ol>
Alternative Flows:	<p>2a Username already exists.</p> <ol style="list-style-type: none"> <li>1. The system trigger Display Error Message use case.</li> <li>2. The system prompts for username and password.</li> <li>3. Use case resumes at main flow step 1.</li> </ol> <p>8a Login session found.</p> <ol style="list-style-type: none"> <li>1. The system trigger Get User's Games use case.</li> <li>2. The system trigger Display User's Games use case.</li> </ol> <p>8b Login session not found.</p> <ol style="list-style-type: none"> <li>1. The system do nothing.</li> </ol>

Table 3.2.2.3 Use case Description for Sign Up use case.



Use Case ID:	CE_UC_4
Use Case Name:	Select Chess Game
Description:	This use case allow user to select and display the desired chess game stored no matter the user is registered or unregistered. Upon selecting the game from the list of games displayed in home page, the system will redirect user to chess game page.
Primary Actor:	User
Preconditions:	1. User accessed the home page.
Postcondition:	1. The system redirect user to chess game page and display chessboard and the moves played for the game.
Main Flow:	<ol style="list-style-type: none"> <li>1. User select the chess game from the list of games displayed.</li> <li>2. The system trigger Redirect to Chess Game Page use case.</li> <li>3. The system trigger Get Game's Details user case.</li> <li>4. The system trigger Display Chessboard and Game's move.</li> <li>5. The system check game status.</li> </ol>
Alternative Flows:	<p>5a The game is not finish.</p> <ol style="list-style-type: none"> <li>1. The system trigger Subscribe to Database and Retrieve Any Updated Move use case.</li> </ol> <p>5a.1 The system receive updated data</p> <ol style="list-style-type: none"> <li>1. The system trigger Update Chessboard and Game's Move user case</li> </ol>

Table 3.2.2.4 Use case Description for Select Chess Game use case.

Use Case ID:	CE_UC_5
Use Case Name:	Display Move Using Buttons
Description:	This use case allows user to display move using various buttons provided by the system.
Primary Actor:	User
Preconditions:	1. User accessed chess game page.
Postcondition:	1. The system display the corresponding move of the game on the chessboard.
Main Flow:	1. User select buttons provided by the system. 2. The system verify buttons selected.
Alternative Flows:	<p>2a User selected “Go to first move” button.</p> <p>1. The system display the first move of the game on the chessboard.</p> <p>2b User selected “Previous move”.</p> <p>1. The system display the first move of the game on the chessboard.</p> <p>2c User selected “Play” button.</p> <p>1. The system start to display the next move of the game automatically until the last move.</p> <p>2d User selected “Next move” button.</p> <p>1. The system display the next move of the game on the chessboard.</p> <p>2e User selected “Go to last move” button.</p> <p>1. The system display the last move of the game on the chessboard.</p>

Table 3.2.2.5 Use case Description for Display Move Using Buttons use case.

Use Case ID:	CE_UC_6
Use Case Name:	Create Game
Description:	This use case allows raspberry pi to create game after the user has enter the required information.
Primary Actor:	Raspberry Pi
Preconditions:	1. User log in to the raspberry pi
Postcondition:	None
Main Flow:	1. User enter color about to play, opponent name, public status 2. The system trigger Update Database use case.
Alternative Flows:	None

Table 3.2.2.6 Use case Description for Create Game use case.

Use Case ID:	CE_UC_7
Use Case Name:	Update Move
Description:	This user case allow raspberry pi to update move for the game that it is tracking to the database.
Primary Actor:	Raspberry Pi
Preconditions:	1. Raspberry pi is tracking the game
Postcondition:	None
Main Flow:	1. Raspberry pi detect move. 2. Raspberry pi send detected move to system. 3. The system trigger Update Database use case.
Alternative Flows:	None

Table 3.2.2.7 Use case Description for Update Move use case.

### 3.3 Activity Diagram

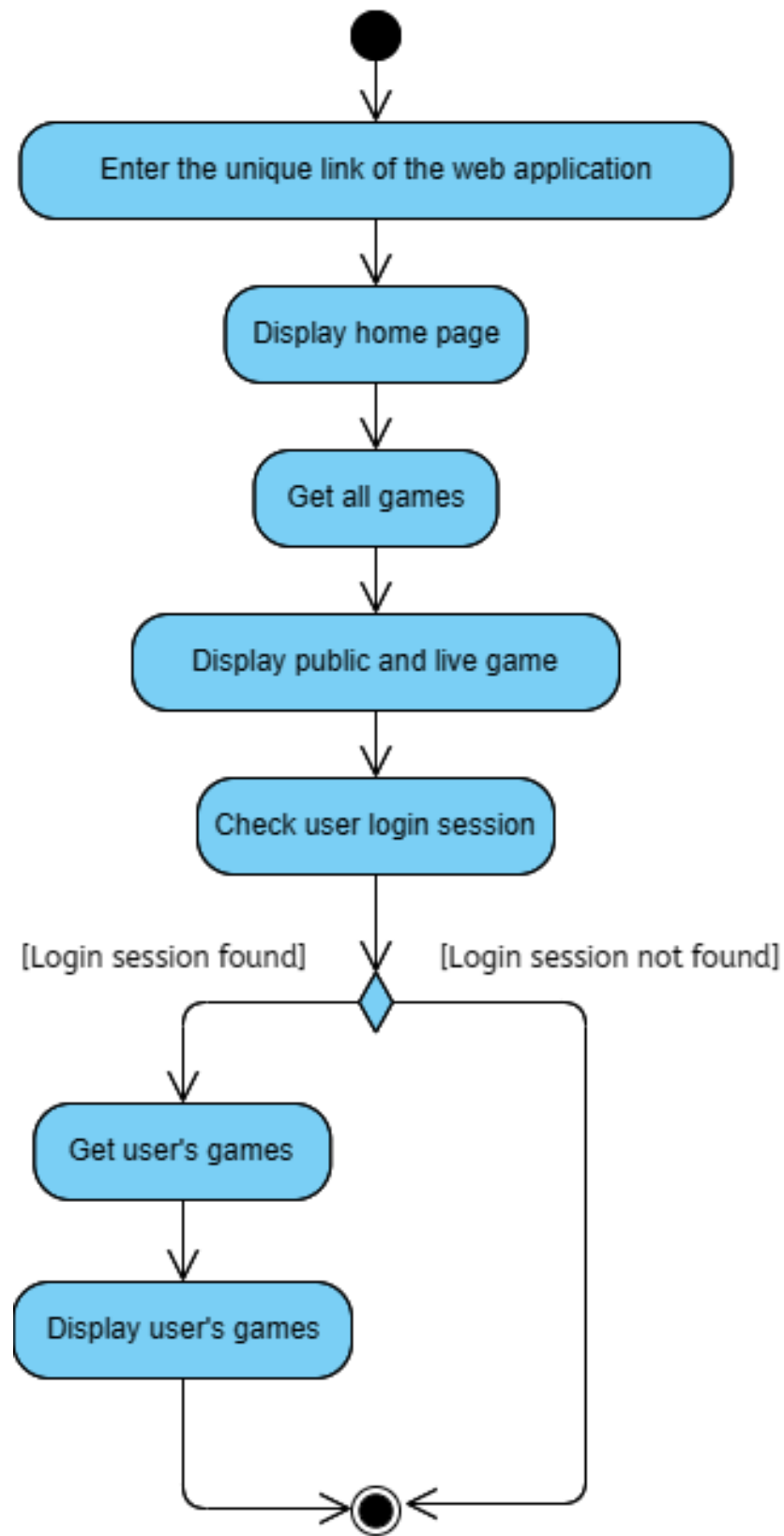


Figure 3.3.1 Activity diagram for Access Home Page use case.

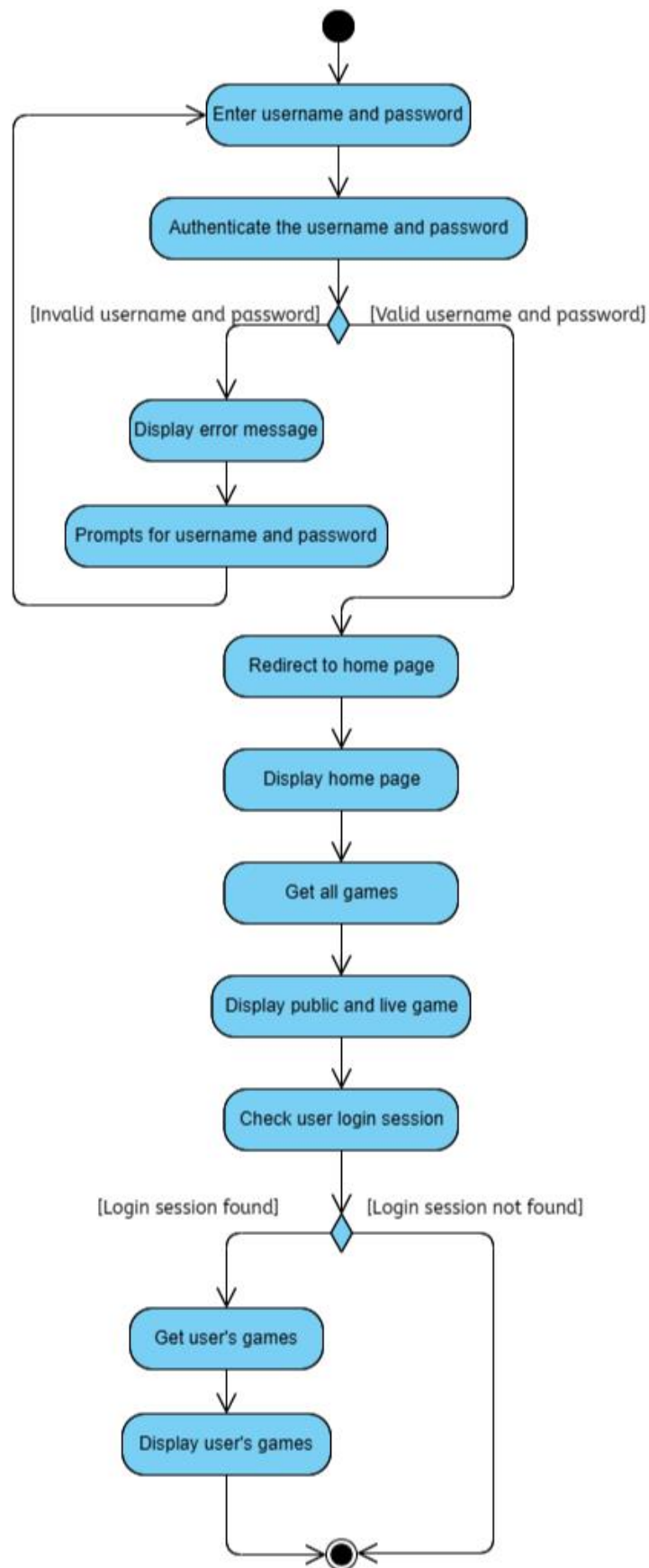


Figure 3.3.2 Activity diagram for Login use case.

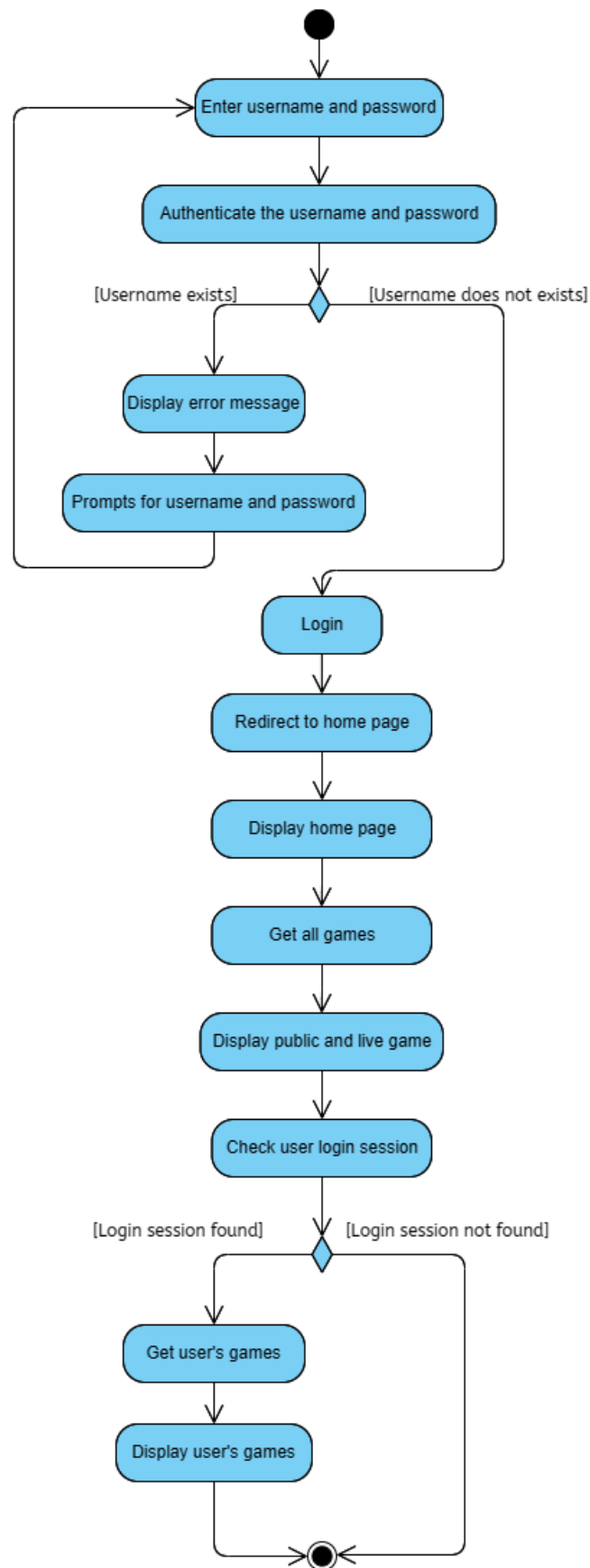


Figure 3.3.3 Activity diagram for Sign Up use case.

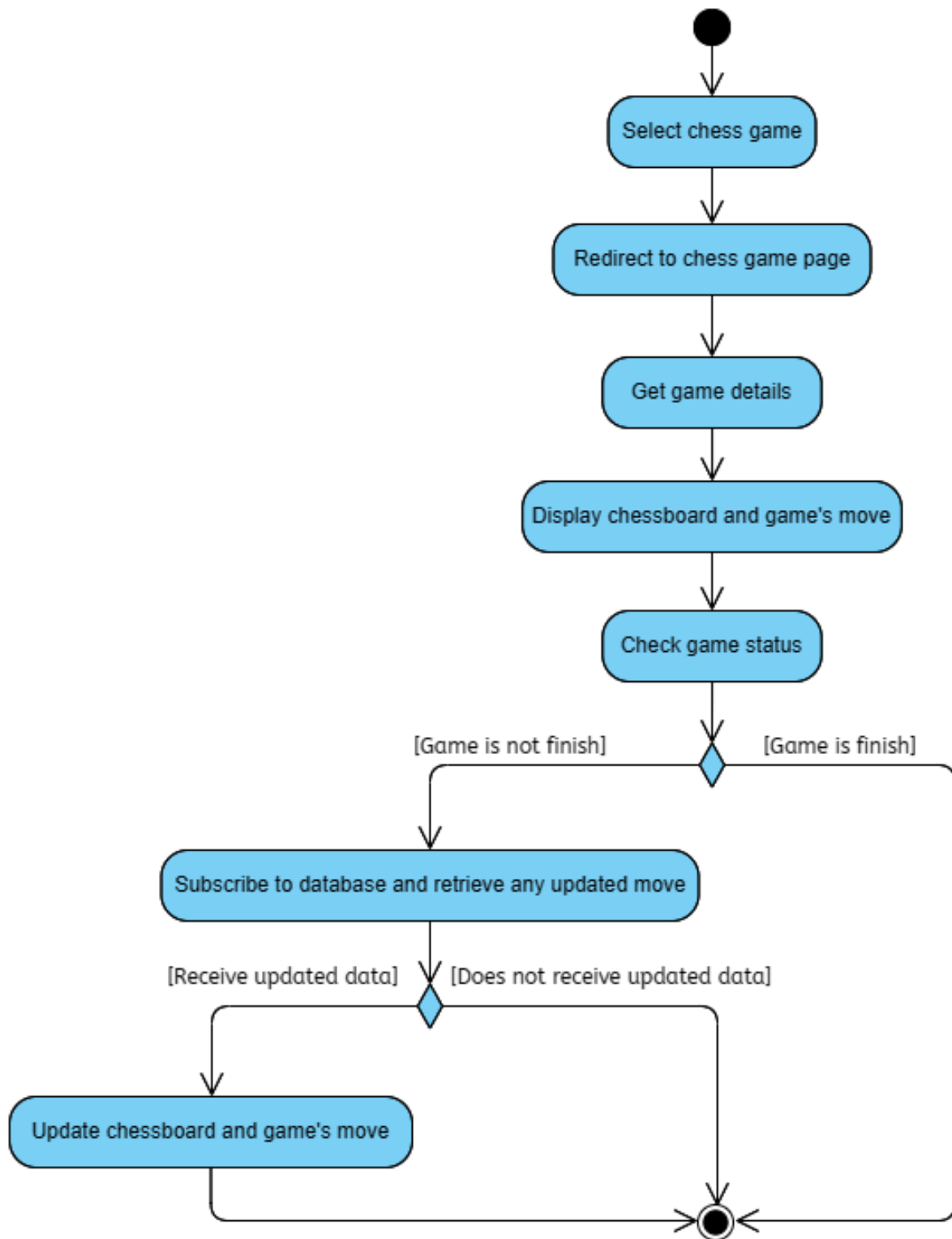


Figure 3.3.4 Activity diagram for Select Chess Game use case.

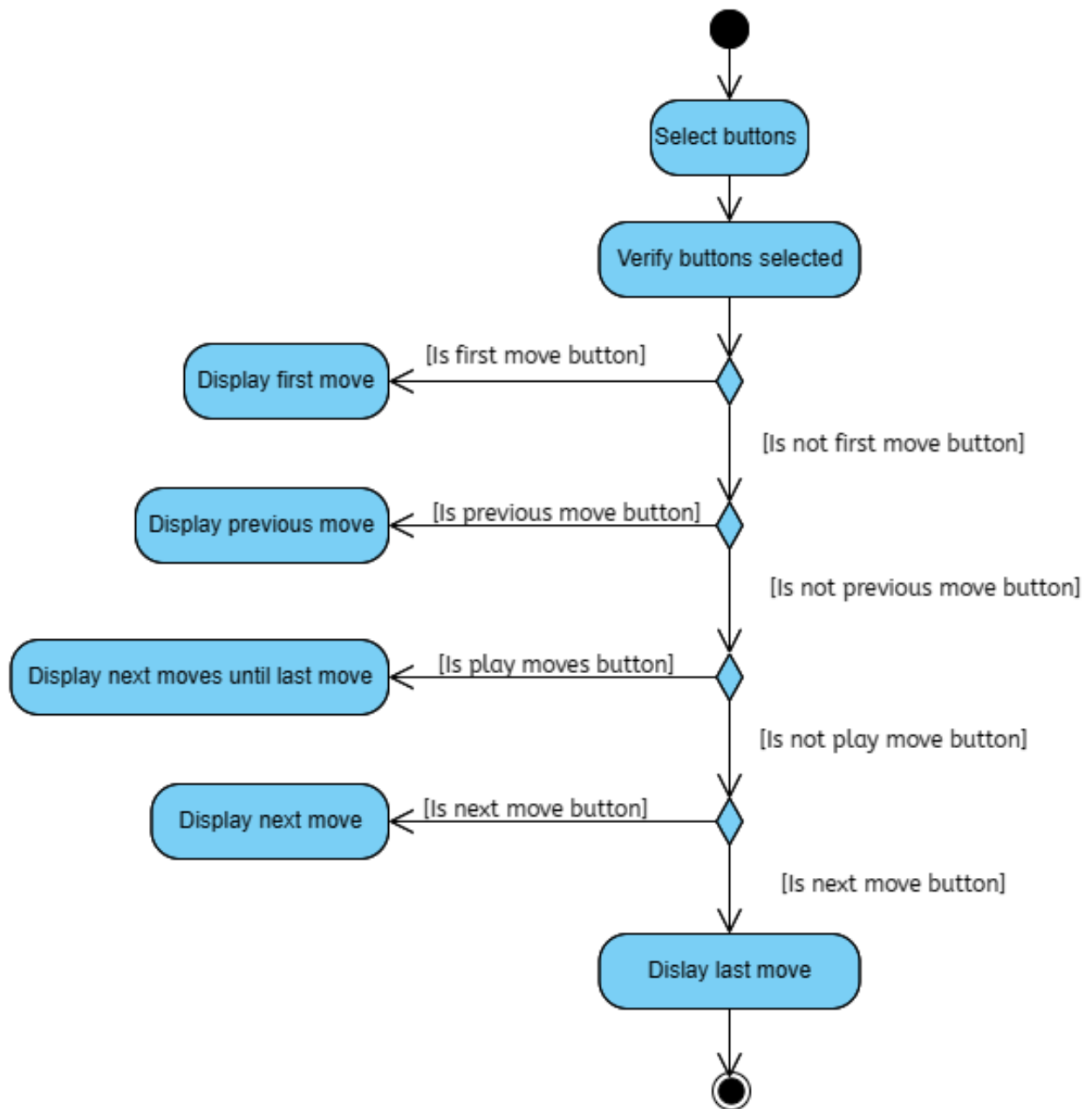


Figure 3.3.5 Activity diagram for Display Move Using Buttons use case.



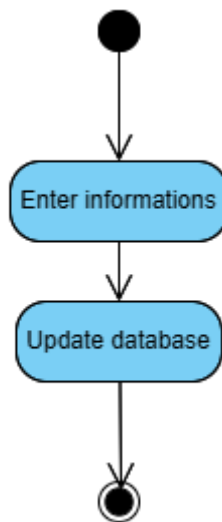


Figure 3.3.6 Activity diagram for Create Game use case.

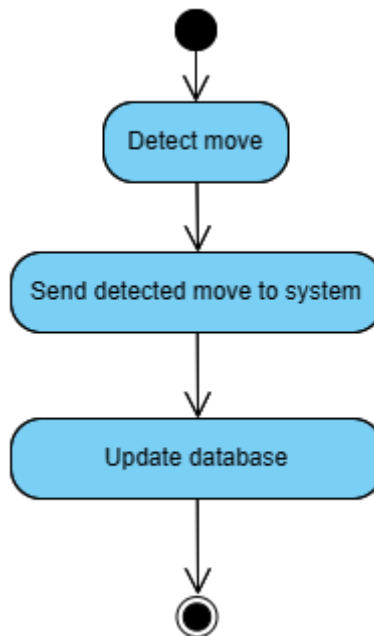


Figure 3.3.7 Activity diagram for Update Move use case.

### 3.4 Timeline

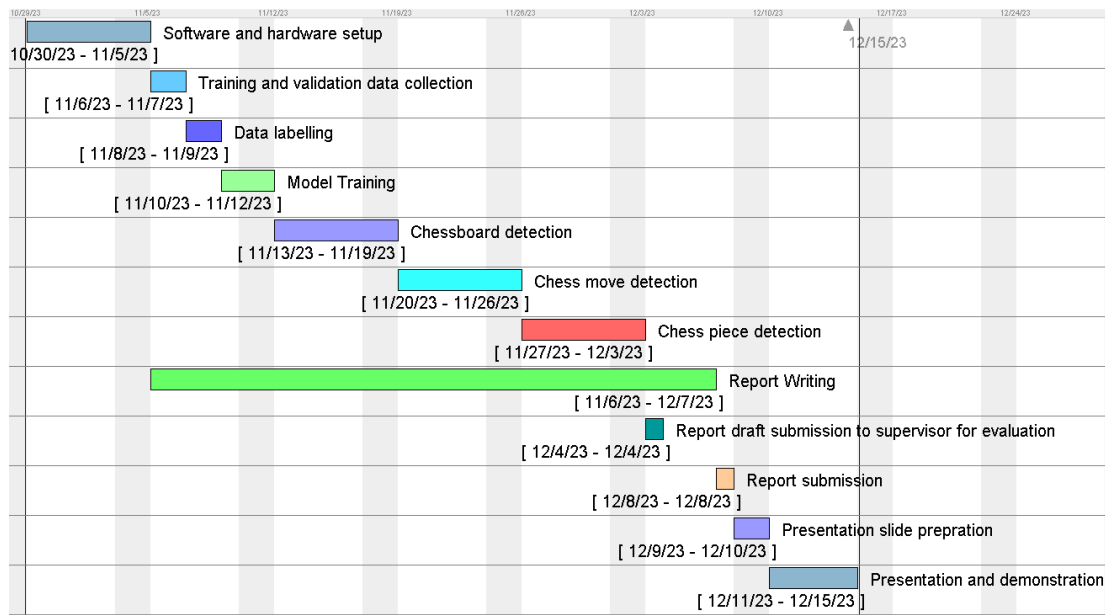


Figure 3.4.1 Gantt chart for FYP 1.

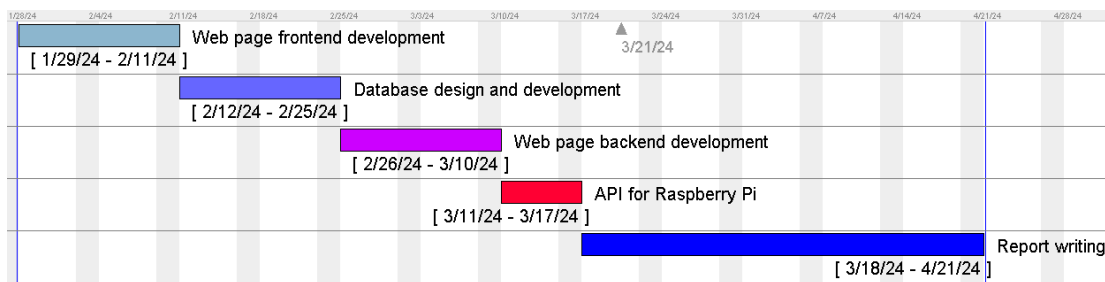


Figure 3.4.2 Gantt chart for FYP 2.

With reference to Figure 3.4.1, the project is started by setting up the software and hardware such as OpenCV, YOLOX, Raspberry Pi and many more in the first week. In the second week, training and validation data specifically the chess image taken from the top-view is collected and labelled. After labelling the images using DataTorch, the images is begin trained on by using YOLOX model specifically yolox-s model with a duration of 3 days. After the model training is finished, the priority now is to focus on detecting the chessboard in the third week. With the accomplishment of detecting the chessboard, the chess move detection can now be worked on to detect the changes between the reference image and the latest image with changes in the fourth week. Finally, the fifth week will be focus on chess piece detection by deploying the model trained to detect the chess piece. As for the report writing, it is started since the second week and end on the sixth week. On 4<sup>th</sup> December 2023, the draft of the report will be submitted to supervisor for evaluation. Finally, the report will be submitted on 8<sup>th</sup>

## CHAPTER 3

December 2023 and the preparation of presentation slide will be started right away, and on the seventh week, the project will be presented and the prototype will be demonstrated.

According to Figure 3.4.2, the development of the web page will be started on 29<sup>th</sup> January 2024. Overall, 2 weeks will be allocate for the development of web page frontend, 2 weeks for designing and developing database, 2 weeks for developing the backend of webpage, 1 week for building the API for Raspberry Pi and 7 weeks for report writing.

## CHAPTER 4

### SYSTEM DESIGN

In this chapter, the system design start from zero to the end, in particular, from the chess game tracking algorithm on the side of raspberry pi to displaying the real-time chess state of a game on the web application. In the following sub-chapters, a system design overview diagram will be provided to explain the abstract concept of the whole system, followed by detailed explanation of the system design on the side of raspberry pi and web application.

#### 4.1 System Design Overview

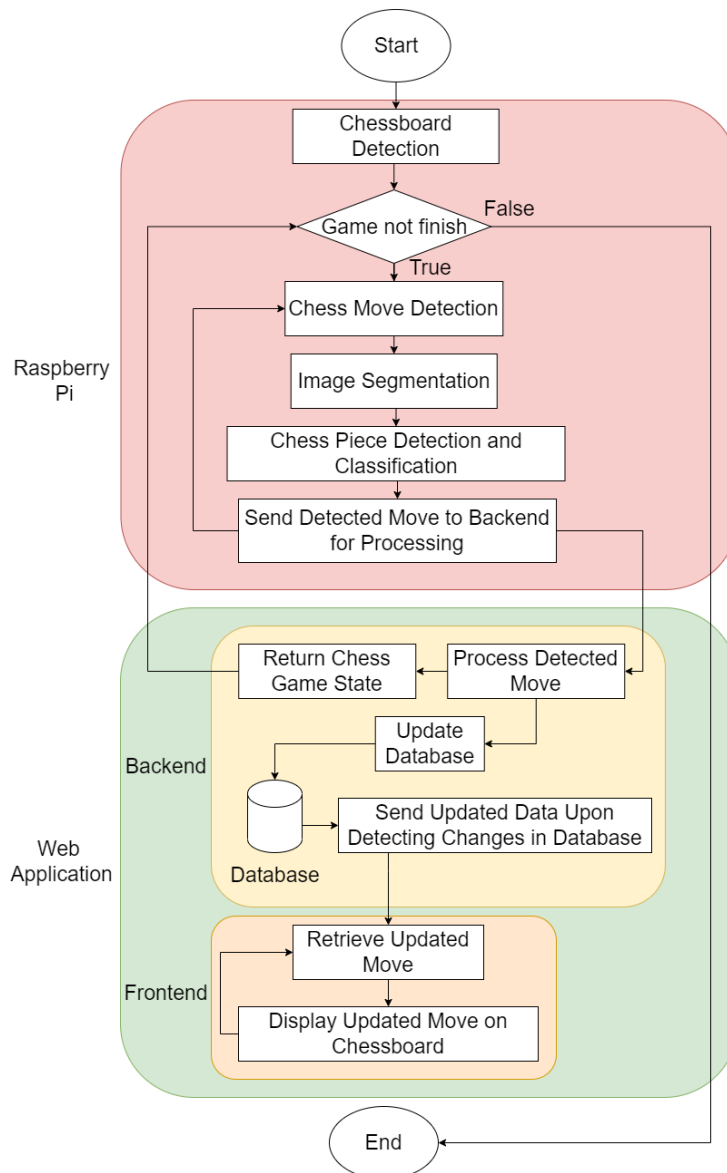


Figure 4.1.1. System design overview diagram.

The overall system design that comprises the core components of the whole system is illustrated in Figure 4.1.1. First of all, Raspberry Pi mounted with camera, will detect the chessboard placed on a dark-colored table by thresholding the image captured by the camera in real time using binary thresholding. By using the result of the previous process, the contour of the chessboard, which is the largest contour will be found by comparing the area of the contours found. Next, the 4 corner points of the chessboard will also be found by approximating the polygonal curve of the contour for the chessboard and by using the 4 corner points, the particular segment of the image that is containing the chessboard will be transformed into a top-view perspective image.

Besides, in order to detect the chess move accurately, the system will compare between the reference image, which is the image that containing the latest chess move made and the latest image captured. To facilitate the system in detecting the changes accurately, the corner points of each square grid will be found and to be used in creating masks, specifically, 64 masks to be used in detecting changes in every square on the chessboard. If there are 2,3 or 4 changes found in the 64 masks, there will be a chess move. After detecting a chess move, the system will use the masks that found the changes to perform image segmentation to obtain the particular segments of the latest image captured. Moreover, the segmented image will be used in model inference to detect and classify the piece that had move. After the completion of model inference, the result of it is used in computing the location of source square and destination square and send the information to the web application backend to process the detected move.

In web application backend, the detected move is used in determining the next chess game state. For example, will the game ended in checkmate for the player on the next turn, ended in draw by insufficient material, 50-move rule, stalemate and many more. Upon determining the chess game state, the backend function will return the chess game state to raspberry pi. If the game is ended, the program will then be ended as well. Moreover, the backend function will also update the database with the latest chess game state with the detected move. Now, if the web application frontend, specifically the chess game page had subscribed to the database for any changes, the database will then send the updated data after changes had made to the database, specifically, the game details including the moves for the ongoing game. Upon receiving the updated move from the game details sent by the database, the frontend will display the updated move on the chessboard and wait for move updated move from the database until the game has ended.

## 4.2 System Design for Raspberry Pi

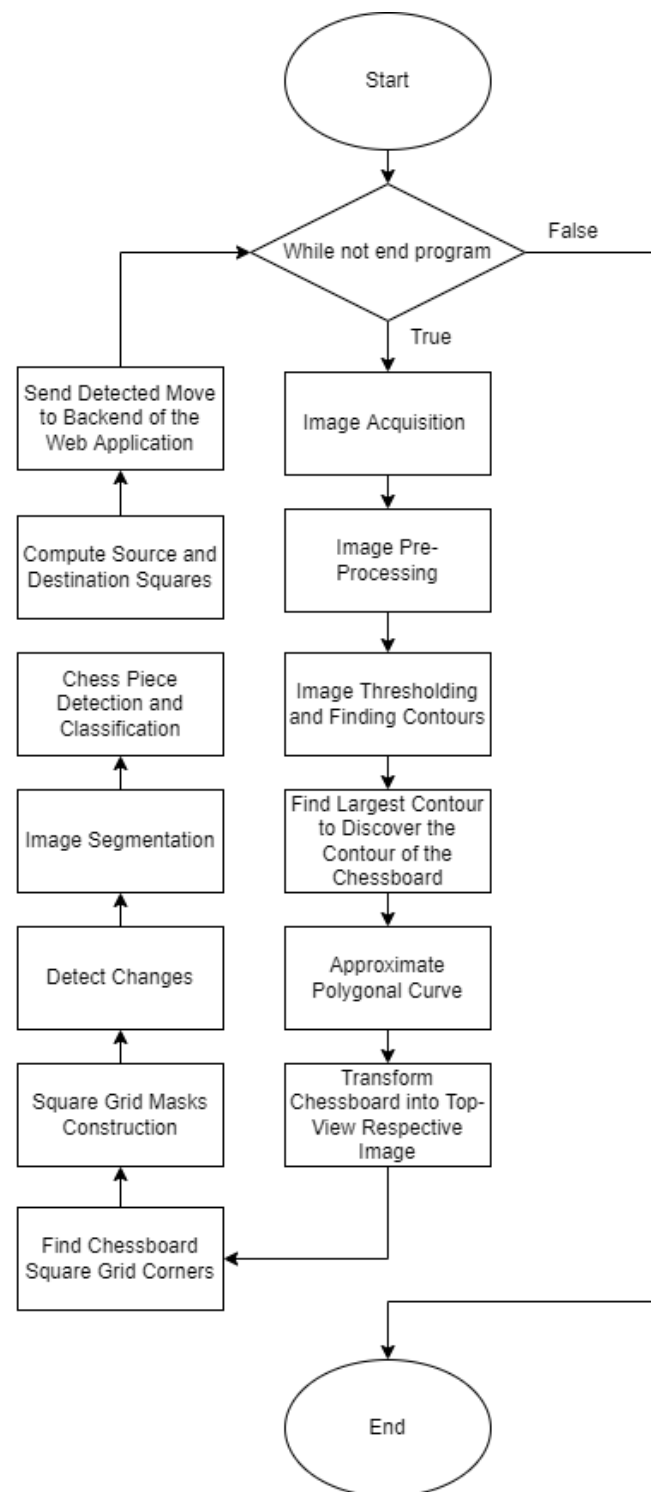


Figure 4.2.1. System design diagram for Raspberry Pi.

With reference to Figure 4.2.1, there are several processes involved from the start to the end in detecting chessboard, detecting chess move, classifying the piece and finally sending the information to the web application. In the following sub-chapters, each of the process will be explained thoroughly.

### 4.2.1 Image Acquisition

In this process, picamera2 library will be utilized to initiate the camera mounted on Raspberry Pi. After creating an instance of Picamera, the camera will start capturing image with RGB format which is compatible with the format needed by OpenCV to process image using the function `capture_array()` as shown in Figure 4.2.1.1. Additionally, every captured image by the camera will be processed to perform chessboard detection, chess move detection and chess piece classification.



Figure 4.2.1.1 Sample result for image acquisition.

### 4.2.2 Image Pre-processing

Prior to image thresholding and finding the contours in the image, image pre-processing will be performed on the image captured to reduce noises and increase accuracy in detecting chessboard. First of all, the image captured will be converted to gray image using OpenCV function `cvtColor()` as shown in image on the left in Figure 4.2.2.1. Next, the gray image will undergoes Gaussian filtering to reduce noises by utilizing the function, `GaussianBlur()` in OpenCV as shown in image on the right in Figure 4.2.2.1. Upon pre-processing the captured image, the processed image with reduced noises is now ready for image thresholding.



Figure 4.2.2.1 Sample result for image pre-processing.

### 4.2.3 Image Thresholding and Finding Contours

The pre-processed image will now undergoes image thresholding, specifically, binary thresholding to find all the points in the image that surpass the threshold using `threshold()` function provided by OpenCV as shown in image on the left in Figure 4.2.3.1. The result of binary thresholding will now be used in finding the contours, line that can surround an area of white bits, using the function `findContours()` in the OpenCV library as illustrated in image on the right in Figure 4.2.3.1.



Figure 4.2.3.1 Sample result image thresholding and finding contours.

### 4.2.4 Finding Largest Contour to Discover the Location of the Chessboard

The contours that are found using `findContours()` function in the previous process as shown in the image at the left in Figure 4.2.3.1 will now be used in finding the largest contour among all the contours found. In order to find the largest contour, the area of all the contours will be calculated using `contourArea()` and will be used in comparison among each other, where the contour with the largest area will be the largest contour. Normally, chessboard will be the largest object in the captured image, hence, the largest contour found will belong to the contour of the chessboard as shown in image in Figure 4.2.4.1.

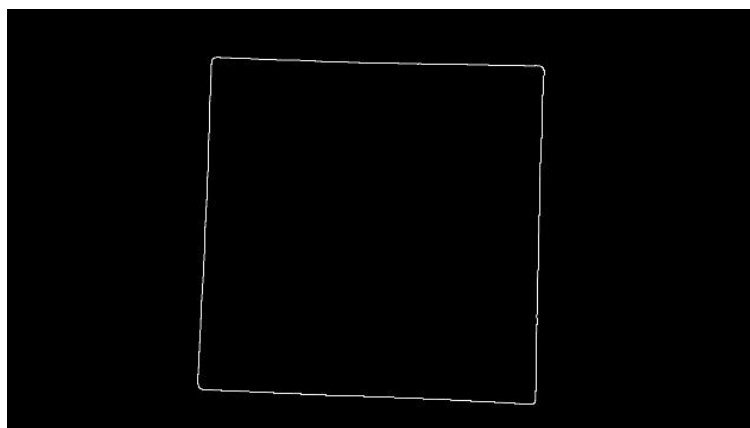


Figure 4.2.4.1 Sample result for finding largest contour.



### 4.2.5 Approximate Polygonal Curve

The largest contour found from the previous process, which is also the contour of the chessboard, will now be used in approximating the polygonal curve of the contour to obtain the 4 corner points of the chessboard using `approxPolyDP()` function in OpenCV library. If the number of points return by `approxPolyDP()` is not equal to 4, then the system will display the reference image and will not detect any changes. The reasoning behind the previously stated logic is when a player wants to make a move, he/she will have to extend his/her hand to the piece he/she want to play, hence, the camera will capture the hand that will hinder the view of the camera to the chessboard. As a result, all subsequent processes will most likely end in error. Additionally, as this process require high computational resource, the 4 corner points will be saved as soon as they are found and they will be used until the end of the program.



Figure 4.2.5.1 Sample result approximating polygonal curve of an image.

### 4.2.6 Transform Chessboard into Top-view Perspective Image

After obtaining 4 corner points of the chessboard successfully from the previous process, the corner points will be used in transforming the chessboard into top-view perspective image with dimension of 1000x1000 using `warpPerspective()` function provided in OpenCV library. With image dimension of 1000x1000, calculations can be made more easily. If the transformed image is the first image begin captured by the camera, then it will become the reference image that will be used in change comparison for chess move detection.

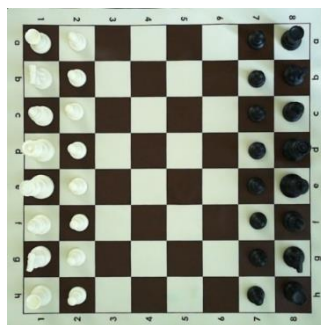


Figure 4.2.6.1 Sample result for `warpPerspective()`.

### 4.2.7 Find Chessboard Square Grid Corners

By utilizing the transformed image from previous process, square grid corners of the chessboard will be found using `findChessboardCornersSB()` function provided by OpenCV library. As `findChessboardCornersSB()` unable to find all corner points of all 64 squares using the parameter (8,8), which is the dimension of chessboard corners wished to be found, due to the algorithm requires 2 white squares and 2 black squares on each diagonal to determine the point in between the squares, (7,7) is used instead to find all inner square grid corners first, then utilize them to find all outer corner points by adding or subtracting the coordinates of the points found. Hence, all the square grid corners are found as shown in Figure 4.2.7.1. In addition, as this process also consuming high computational resource, as soon as all square grid corners points are found, they will be stored and used until the end of the program.

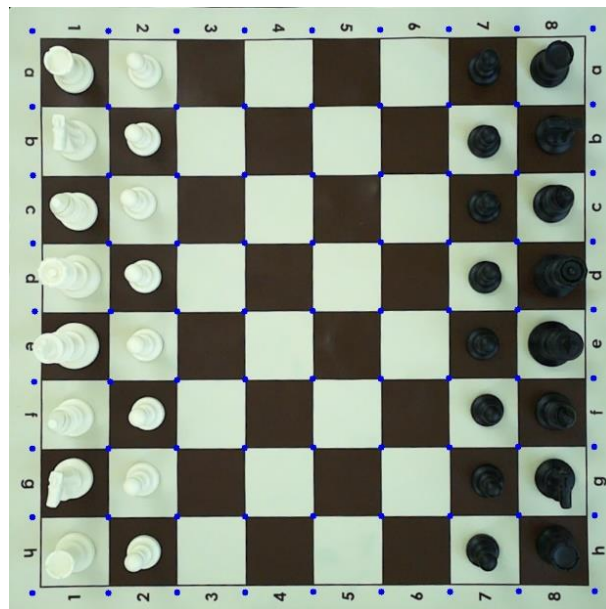


Figure 4.2.7.1 Sample result for `findChessboardCorners()` with additional calculation.

### 4.2.8 Square Grid Masks Construction

The square grid corner points found in previous process will now be used in constructing the masks, specifically, 64 masks that will detect changes in every individual square on the chessboard. First of all, a black images with the same size of the transformed image will be created. Next, masks will be created by converting the particular segment of each black image to white points/bits using the values of x-coordinate of current point and the point on the right of it and the values of y-coordinate of the current point and the point below it in loops. During the loops, the centre, rank and file of the masks will also be determined and saved. Hence, 64 masks are now constructed.

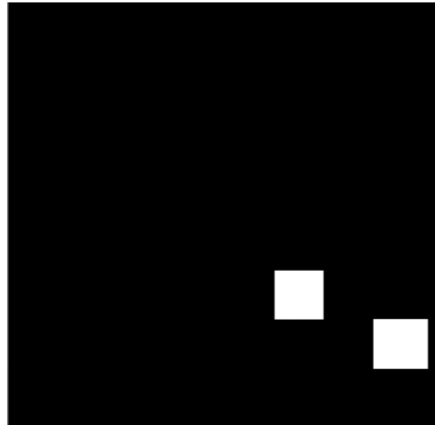


Figure 4.2.8.1 Sample result for square grid masks construction.

#### 4.2.9 Detect Changes

The changes can be detected by comparing the reference image and the latest captured image to determine if there is any change between them using `absdiff()` function provided by OpenCV. The function will return an image with the same size as the input images that contain black points/bits for those points that has no change in comparison and retain the points/bits for the otherwise. Next, the masks created in the previous process will now be used in detecting the changes in each individual square on the chessboard. If there is significant change detected in a square by the mask, specifically, more retained points/bits, the corresponding mask will be added to a black image. Eventually, the black image that containing the masks will become the mask for the next process, image segmentation. If the black image does not contain 2, 3, 4 masks, the system will declare that there is no change between the reference image and the latest captured image as it impossible to move more than 1 piece in a single move. When a move is made, there can be only 2 changes can be detected. Moreover, there will be 3 changes be detected for the special move “en passant”. In addition, when a castling move, resignation and draw by agreement is made, there can be only 4 changes can be detected .

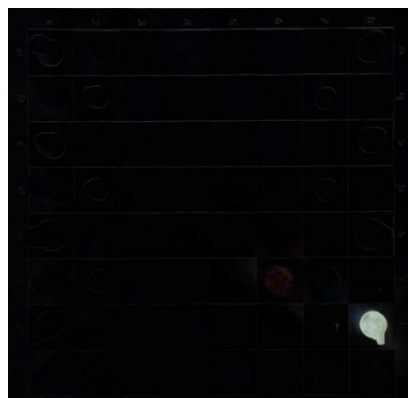


Figure 4.2.9.1 Sample result for finding changes using `absdiff()`.

#### 4.2.10 Image Segmentation

By using the output of the previous process, which is black image containing masks that corresponding to the squares that have significant changes, it will be used in segmenting the latest captured image by using `bitwise_and()` function that will segment the image according to the white points/bits contained in the black image. Hence, a segmented image will be obtained.

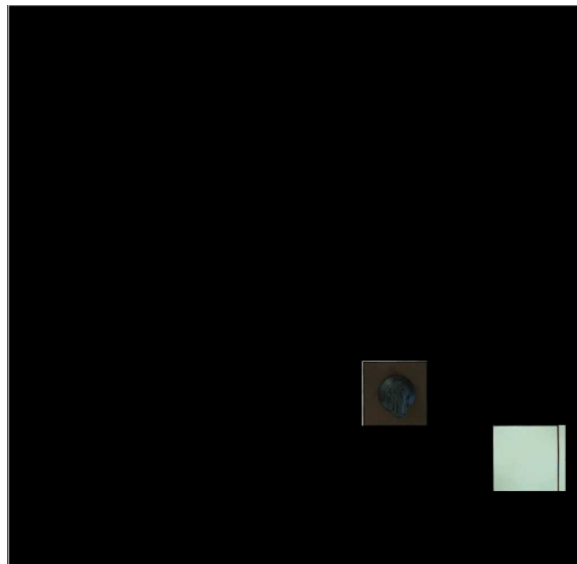


Figure 4.2.10.1 Sample result for image segmentation.

#### 4.2.11 Chess Piece Detection and Classification

The segmented image from the previous process will now be used in model inference. The model chosen to be used is YOLOX as it contain various model that can be chosen according to the user application. Among all the model provided by YOLOX, `yolox-s` will be used in this project as it is suitable for device that has relatively less computational power. The model will be trained with 2976 labelled training images and validated with 280 labelled validation images. Sample of training and validation images is shown in Figure 4.2.11.1 and Figure 4.2.11.2. As Raspberry Pi has a lower computational power as compared to other normal computer, model inference is performed on a separate thread by utilizing multithreading capability of the CPU to prevent the program from stopping at the point of model inference and not detecting any more chess move potentially made by the players.

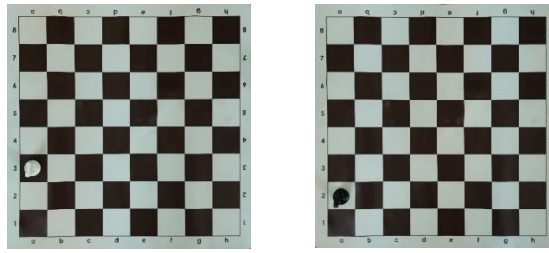


Figure 4.2.11.1. Sample of Training images.

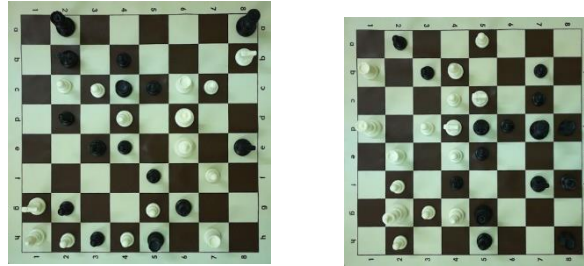


Figure 4.2.11.2. Sample of validation data.

#### 4.2.12 Compute Source and Destination Squares

There will be 7 outputs as the result of the previous process, that is, model inference, where the first four output will be the value of coordinates of the top left point and bottom right point of the bounding box containing the chess piece, fifth and sixth will be the score when multiplied together and the last will be the class of the chess piece. Next, the source and destination square of a chess piece can be determined by calculating the distance between the centre of the masks used in image segmentation and the value of the coordinates of the bounding box. The destination square will be the mask that has a centre nearer to the bounding box and source square for the otherwise. The sample result of completion of model inference and source and destination squares computation can be seen in Figure 4.2.11.3.

```
2023-12-03 10:11:14.204 | INFO | tools.demo:inference:165 - Infer time: 7.2952s
Black Knight moved from g8 to f6
█
```

Figure 4.2.12.1 Sample result of model inference and source and destination squares calculation.

#### 4.2.13 Send Detected Move to Backend of the Web Application

Finally, the information such as chess piece that had moved, source and destination square or special move such as castling, resignation or draw by agreement will be send to the web application backend to process the information, update the database and display it in the web application frontend.

4.3 System Design for Web Application

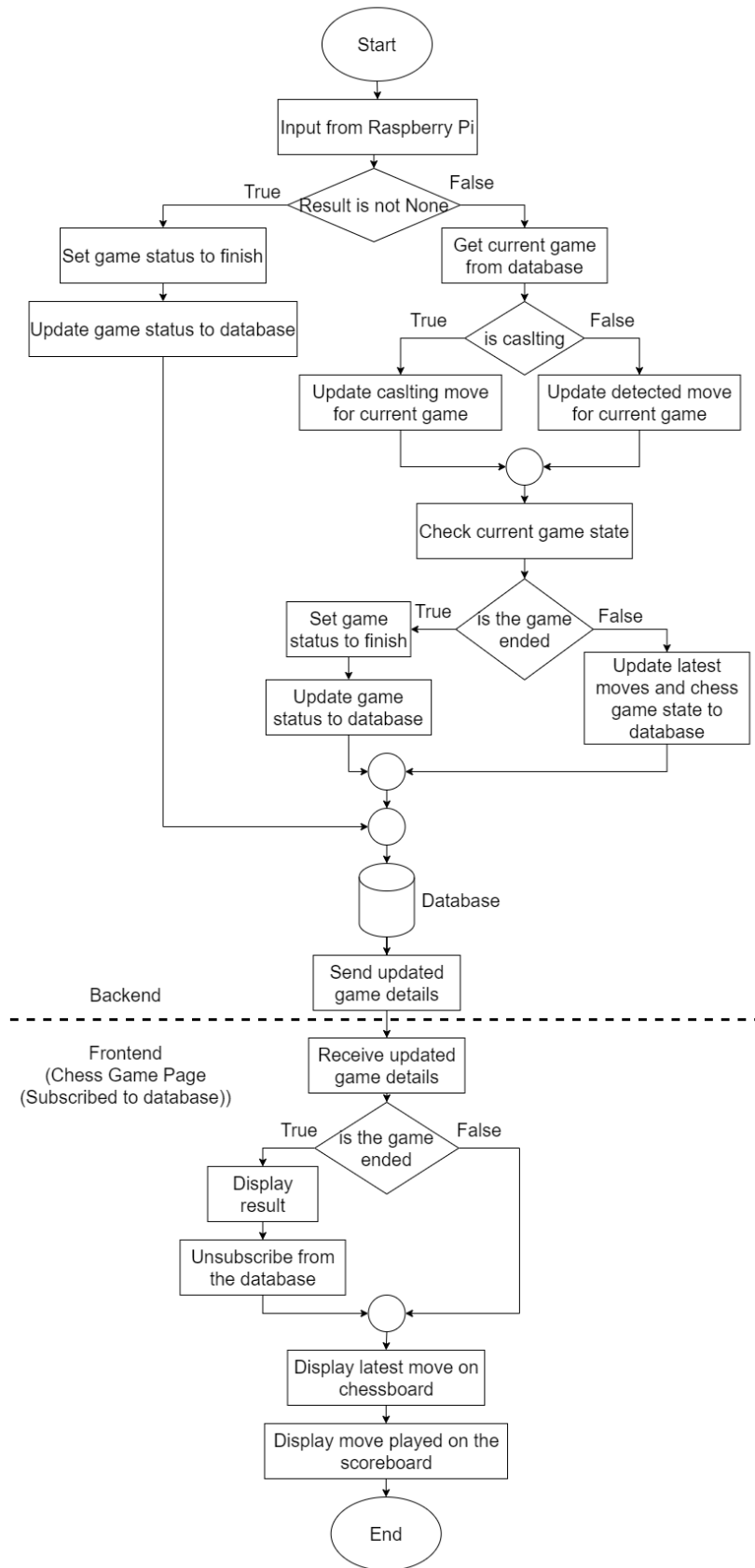


Figure 4.3.1. System design diagram for Web Application.

With reference to Figure 4.3., first of all, the web application backend will receive input from raspberry pi that comprising information such as detected move, result (resignation or draw by agreement) and castling move. The backend function will first check if it receive result information from the raspberry pi. If yes, then it will set the game status to finish and update the game status to the database. Otherwise, the function will get the current game from the database. By utilizing the library chess.js, the saved moves will be loaded into a temporary Chess object by using chess.load() function. Upon loading the moves, the function will now check if there is any castling move inputted from the raspberry pi. If there is, the castling move will be moved in the temporary Chess object and the current chess game state will be obtained in FEN notation by utilizing the function chess.fen(). Otherwise, the function will just load the detected move into the temporary Chess object and get the current chess game state in FEN notation using the function chess.fen(). After that, the function will now check if the game is ended. There are a few ways a game can be ended in normal circumstances, which are stalemate, three-fold-repetition, insufficient material and checkmate. In order to check all of this, chess.js provided a few handy functions to do so which are chess.isStalemate(), chess.isThreefoldRepetition(), chess.isInsufficientMaterial() and chess.isCheckmate() where all will return true if the conditions are met. If the game is ended, the function will set the game status to finish and update the game status to database. Otherwise, the function will just update the latest moves and chess game state to database. After updating the database, the database will send the updated game details to frontend, specifically, the chess game page that subscribed to database.

Upon receiving the updated game details from the database, the page will first check if the game is ended. If ended, the page will display the result on the screen and unsubscribe from the database. Otherwise, the page will just display the latest move on the chessboard and move played on the scoreboard. The whole process will continue

## CHAPTER 5

### SYSTEM IMPLEMENTATION

#### 5.1 Hardware Setup

Specifications	Description
Model	MSI GF63 Thin 9SC
Processor	Intel Core i7-9750H
Operating System	Windows 10
Graphic	NVIDIA GeForce GTX 1650 4GB
Memory	8GB 2333Mhz DDR4 RAM
Storage	1TB SATA SSD

Table 5.1.1 Specifications of laptop.

Specifications	Description
Model	Raspberry Pi 4 Model B
Processor	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
Operating System	Raspberry Pi OS
Memory	8GB LPDDR4-3200 SDRAM
Storage	128GB(SD Card)
Network	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet
IO ports	<ul style="list-style-type: none"> <li>• Raspberry Pi standard 40 pin GPIO header</li> <li>• 2 × micro-HDMI® ports (up to 4kp60 supported)</li> <li>• 2-lane MIPI DSI display port</li> <li>• 2-lane MIPI CSI camera port</li> <li>• 4-pole stereo audio and composite video port</li> <li>• H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)</li> </ul>



	<ul style="list-style-type: none"> <li>• OpenGL ES 3.1, Vulkan 1.0</li> <li>• Micro-SD card slot for loading operating system and data storage</li> <li>• 5V DC via USB-C connector (minimum 3A*)</li> <li>• 5V DC via GPIO header (minimum 3A*)</li> <li>• Power over Ethernet (PoE) enabled (requires separate PoE HAT)</li> </ul>
--	--

Table 5.1.2 Specifications of Raspberry Pi[22].

Specifications	Description
Model	Raspberry Pi Camera Module 3 with Auto Focus
Resolution	12-megapixel (4608 x 2592 pixels)
Image Sensor	Sony IMX708
Sensor Size	7.4mm sensor diagonal
Pixel Size	1.4 $\mu\text{m}$ $\times$ 1.4 $\mu\text{m}$
Lens	Auto Focus
Aperture	f/2.0
Field of View(FOV)	75 degrees
Video Resolution	1080p50, 720p100, 480p120
Video Output Format	RAW10
Connection	CSI-2 serial data output
Compatibility	Raspberry Pi 3, Raspberry Pi 4, Raspberry Pi Zero
Dimensions	25 $\times$ 24 $\times$ 11.5mm

Table 5.1.3 Specifications of Raspberry Pi camera[27].

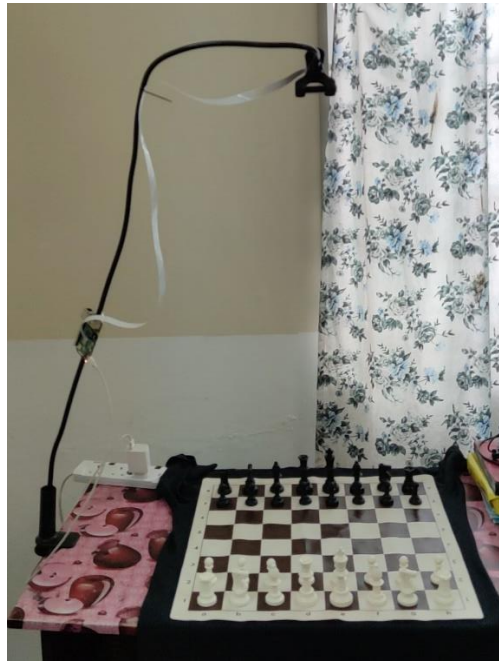


Figure 5.1.1. Hardware Setup

First of all, the chessboard is placed on top of a black cloth to ensure chessboard detection can be performed accurately. Next, the Raspberry Pi is strapped at the middle of a mobile phone holder and powered by a mobile phone charger adapter. Finally, the camera is clamped by the mobile phone clamp that is positioned on top of the chessboard perpendicularly.

## 5.2 Software Setup

Before the project begin, there are a few software are required to be downloaded to laptop and Raspberry Pi :

### 1. Visual Studio Code

Visual Studio Code is a lightweight yet powerful source-code editor developed by Microsoft. It supports various programming languages and offers features like syntax highlighting, debugging, and Git integration. Its customizable interface, extensive extension marketplace, and cross-platform compatibility make it a popular choice among developers for efficient coding workflows.

### 2. YOLOX

With a DarkNet53 backbone, YOLOX is a single-stage object detector that modifies YOLOv3 in a number of ways. In particular, a disconnected head takes the role of YOLO's. To minimize the feature channel to 256 for each level of FPN feature, a  $1 \times 1$  convolution layer is adopted. For classification and regression tasks, two parallel branches is added with two  $3 \times 3$  convolution layers each[24].

### 3. OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source software library for machine learning and computer vision. OpenCV was created to make it easier to employ machine perception in consumer products and to provide a common basis for computer vision applications. Companies can easily use and modify the source code of OpenCV thanks to its Apache 2 license. [23].

## 5.3 Settings and Configurations

### 5.3.1 Visual Studio Code Installation

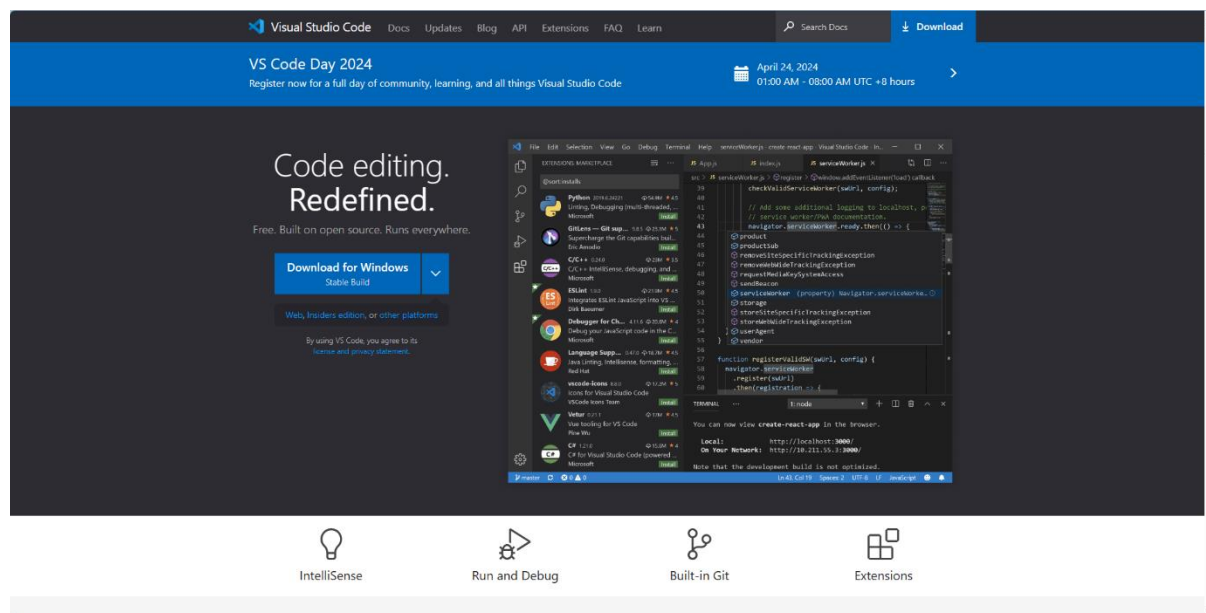


Figure 5.3.1.1. Visual Studio Code Installation.

First of all, the most important software for this project is Visual Studio Code as everything is coded in it including the web application and the chess game tracking system in Raspberry Pi. The installation file is found in the main page of Visual Studio Code after searching “Visual Studio Code” as shown in Figure 5.3.1.1. After installing Visual Studio Code, the project is ready to go.

### 5.3.2 YOLOX Installation

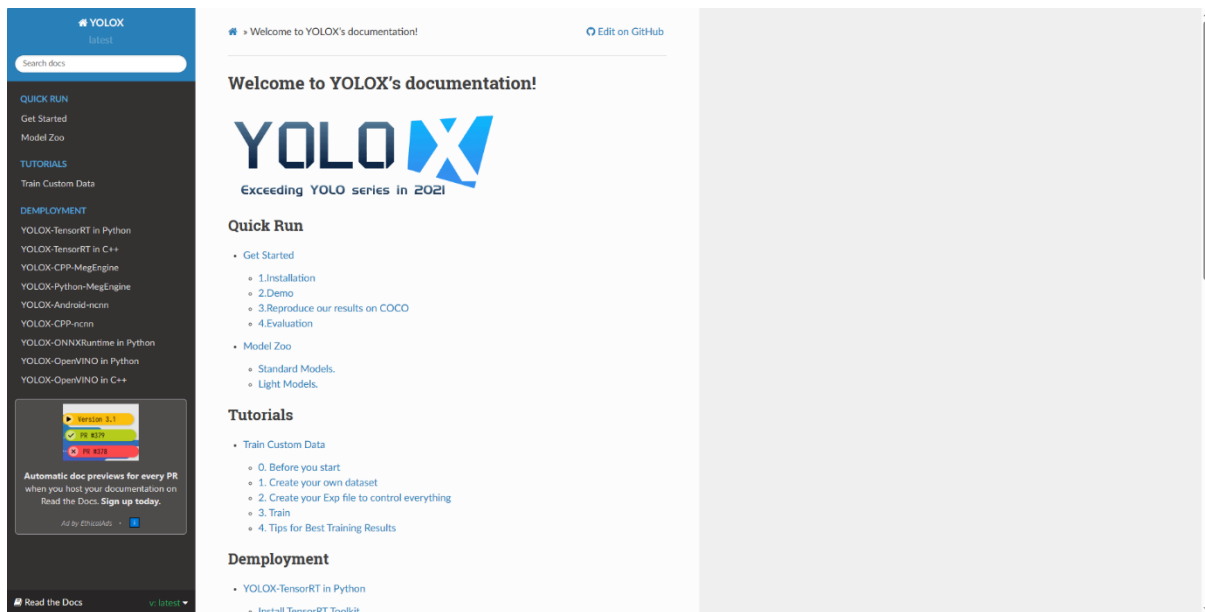


Figure 5.3.2.1. YOLOX documentation



Figure 5.3.2.2. YOLOX installation guideline.

YOLOX, one of the most component of the system, is installed by following the guidelines provided in the documentation prepared by the developer as shown in Figure 5.3.2.1. Hence, YOLOX can be installed into the laptop and Raspberry Pi easily by following the installation instructions provided as shown in Figure 5.3.2.2 by following the steps such as typing or copying the commands required and executing them in the terminal of laptop and Raspberry Pi. After installing YOLOX, model is trained by using simple commands such as “python tools/train.py -f exps/chess/yolox\_s.py -d 1 -b 4 -o --fp16”. Posterior to training the model, the trained model is then used in detecting the chess pieces by loading the model weights and setting the mode to evaluation mode by utilizing simple code as shown in Figure 5.3.2.3.

```
exp = get_exp(exp_file="/home/user/YOLOX/exps/chess/yolox_s.py")
model = exp.get_model()
model.eval()
ckpt_file = "/home/user/Desktop/FYP/best_ckpt.pth"
ckpt = torch.load(ckpt_file, map_location="cpu")
model.load_state_dict(ckpt["model"])
predictor = Predictor(model=model, exp=exp)
```

Figure 5.3.2.3. Code to load and use the model in performing inference for object detection.

### 5.3.3 OpenCV Installation

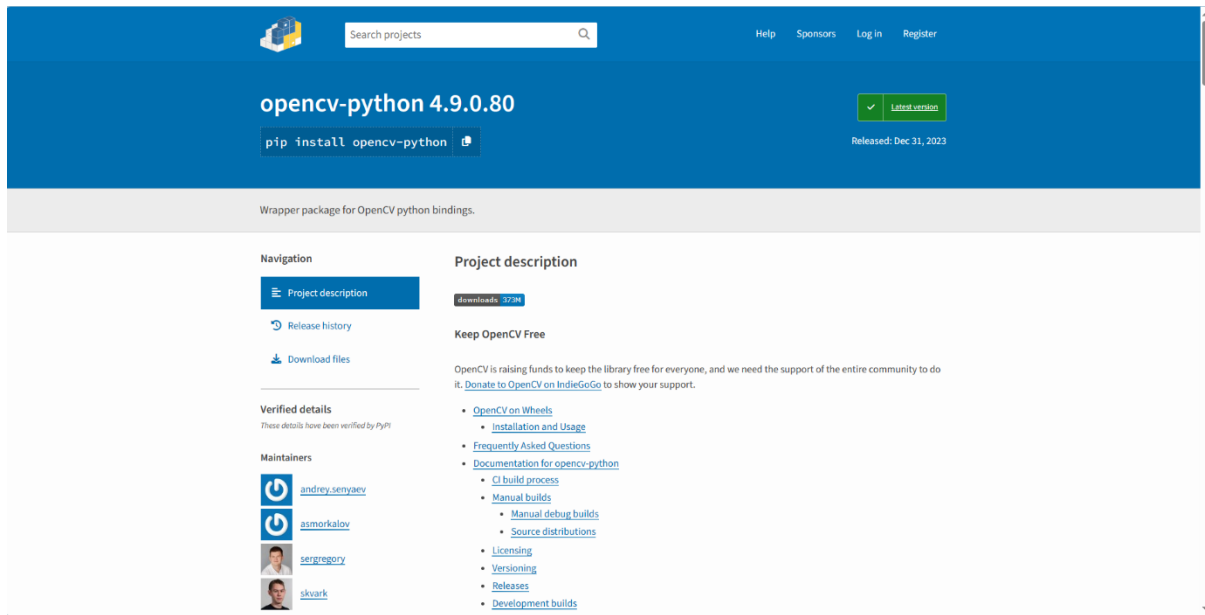


Figure 5.3.3.1 OpenCV Installation.

OpenCV, which also one of the most crucial libraries that are required in this project for processing images. It is installed in the Raspberry Pi by using the terminal commands as shown in Figure 5.3.3.1. After installing OpenCV, the libraries are now accessible for the project.

### 5.3.4 AWS Free Tier Account Creation

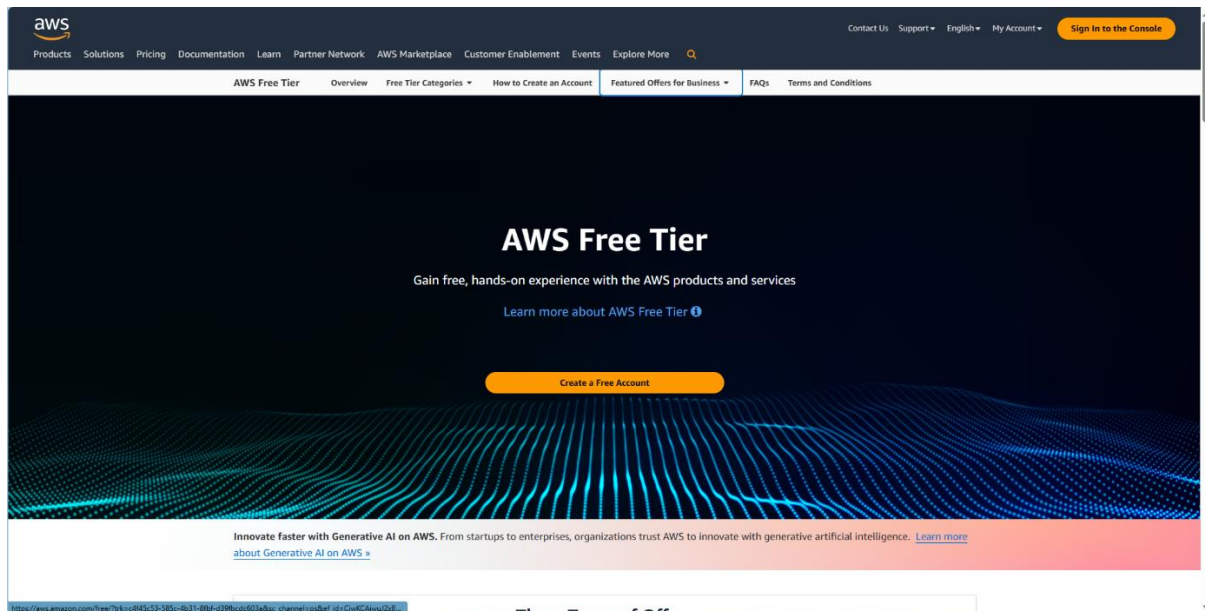


Figure 5.3.4.1. AWS main page.

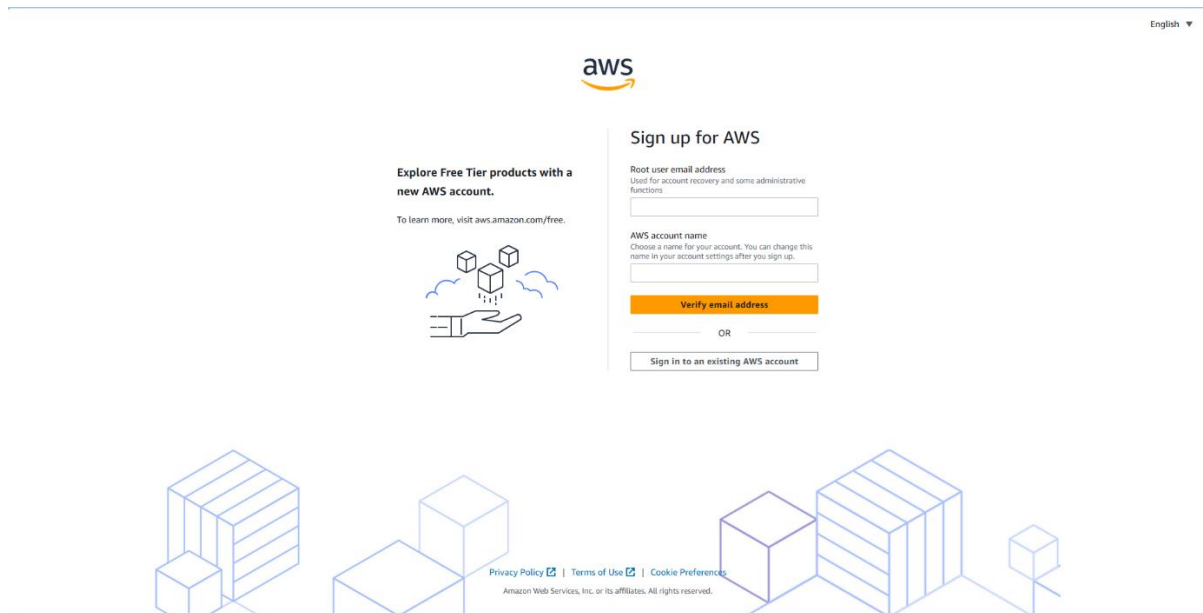


Figure 5.3.4.2. Account sign up page.

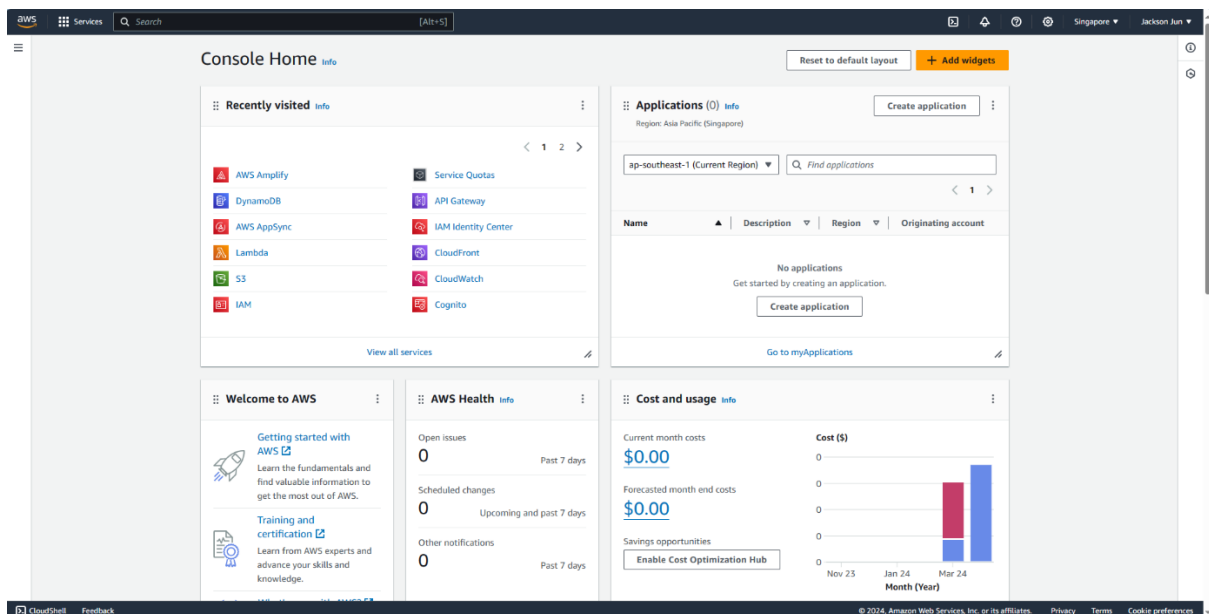


Figure 5.3.4.3. AWS console management page.

An AWS free tier account is created for this project by accessing the main page of AWS. By clicking the “Create a free account” button in the main page as shown in Figure 5.3.4.1, it will redirect to the sign-up page as shown in Figure 5.3.4.2. After filling all the information required by it such as root Gmail account, username, credit card information, address and many more, an AWS free tier account had successfully created and will redirect to AWS console management page as shown in Figure 5.3.4.3. With the free tier account, services such as AWS Amplify, AWS Lambda and AWS DynamoDB that are required in this project are now accessible.

### 5.3.5 Frontend Creation Using Command Line

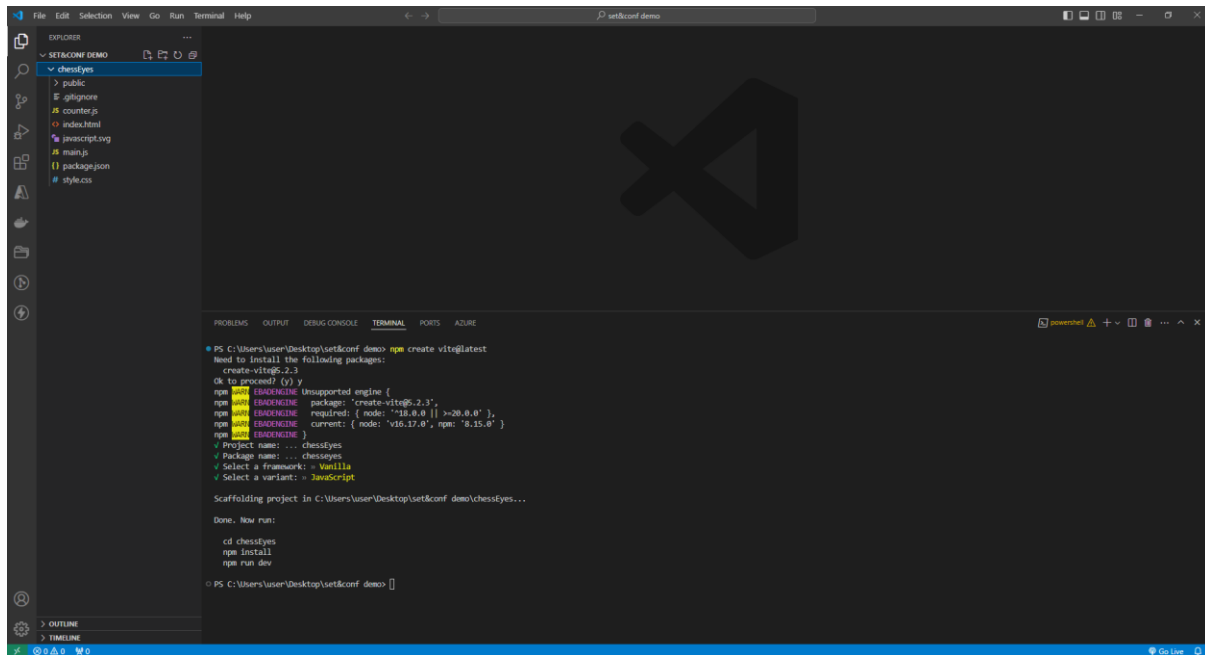


Figure 5.3.5.1. Creating a frontend using command line.

As shown in Figure 5.3.5.1, the command “npm, create vite@latest” is executed in Visual Studio Code terminal to create the frontend for the web application. After executing the command, required information such as project name, package name, framework and variant were prompted and upon entering them, a frontend application is created as shown in the project directory in the navigation bar on the left.

### 5.3.6 Backend Creation Using Amplify CLI

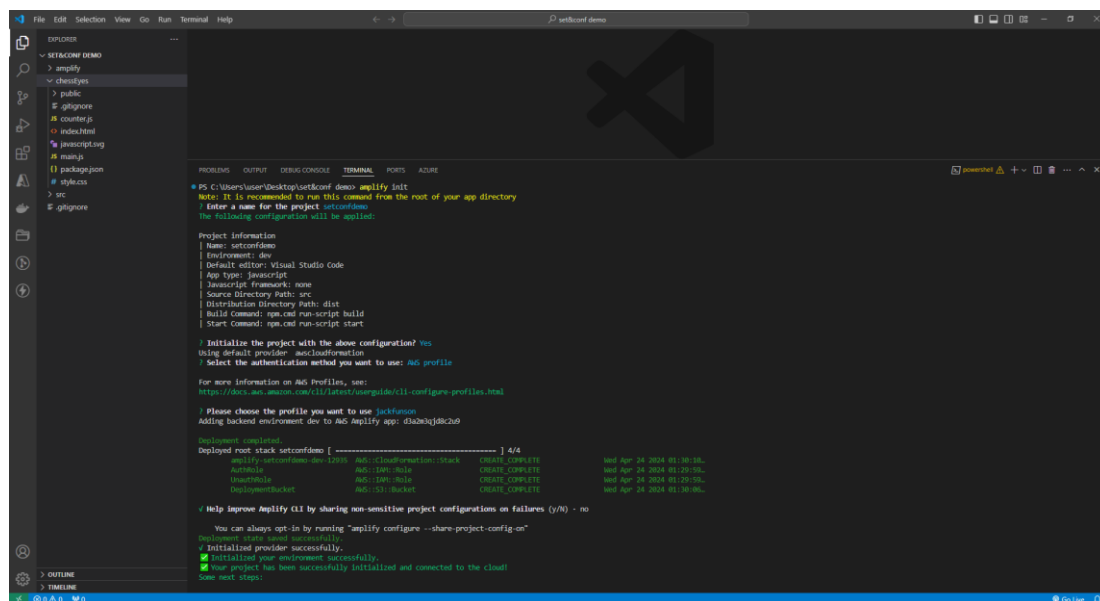


Figure 5.3.6.1. Creating backend using Amplify CLI.

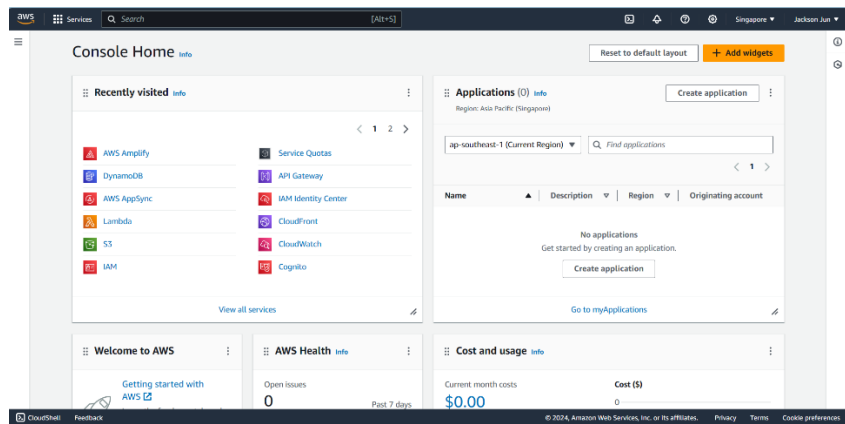


Figure 5.3.6.2. AWS console management page.

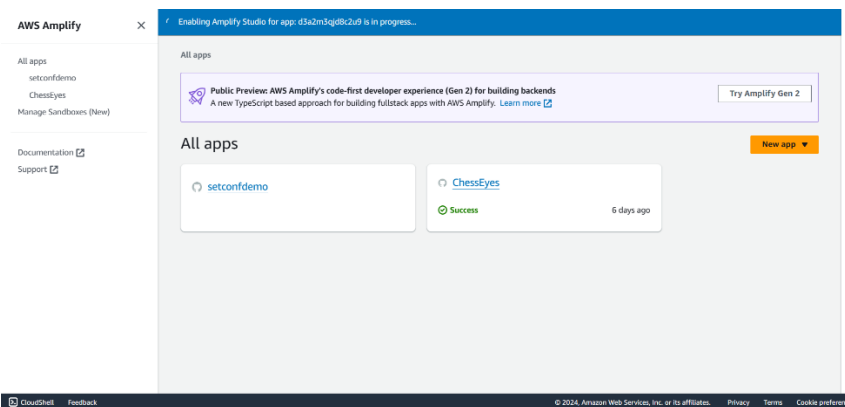


Figure 5.3.6.3. AWS Amplify main page.

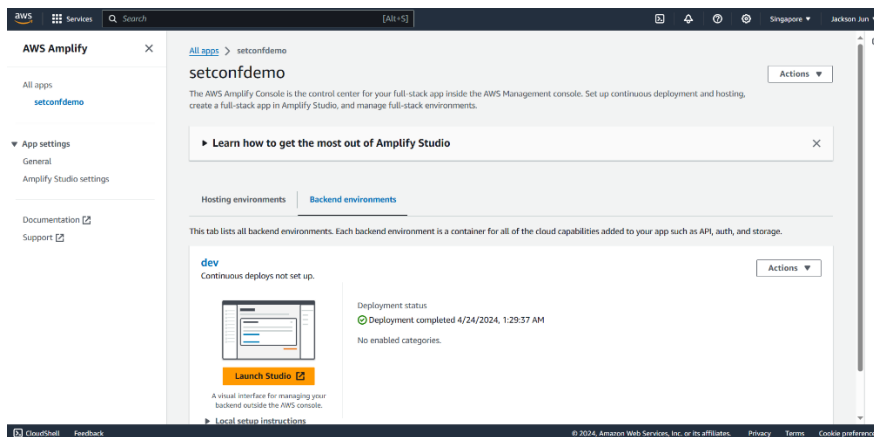


Figure 5.3.6.4. Setconfdemo app backend environment page.

In Visual Studio Code terminal, Amplify CLI command which is “amplify init” is executed to create the backend environment for the web application. After completing the configuration by setting AWS profile that links to the free tier account, the backend environment is now created and deployed in Amplify. It can be confirmed by first going to the AWS console management page as shown in Figure 5.3.6.2 and click on “AWS Amplify” and it will redirect to the main page of it. In the main page as shown in Figure 5.3.6.3, there is an



app called setconfdemo and after clicking on it and go to the backend environment tab, it is evident that the backend environment is successfully deployed.

### 5.3.7 Database Creation Using Amplify Studio

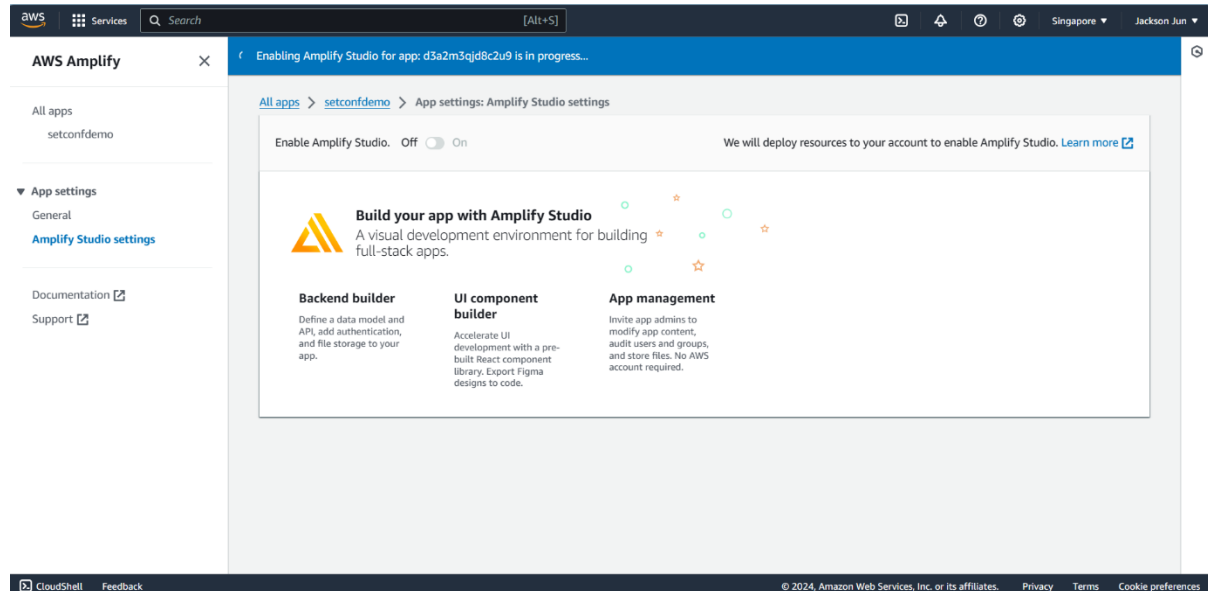


Figure 5.3.7.1. Enabling Amplify Studio

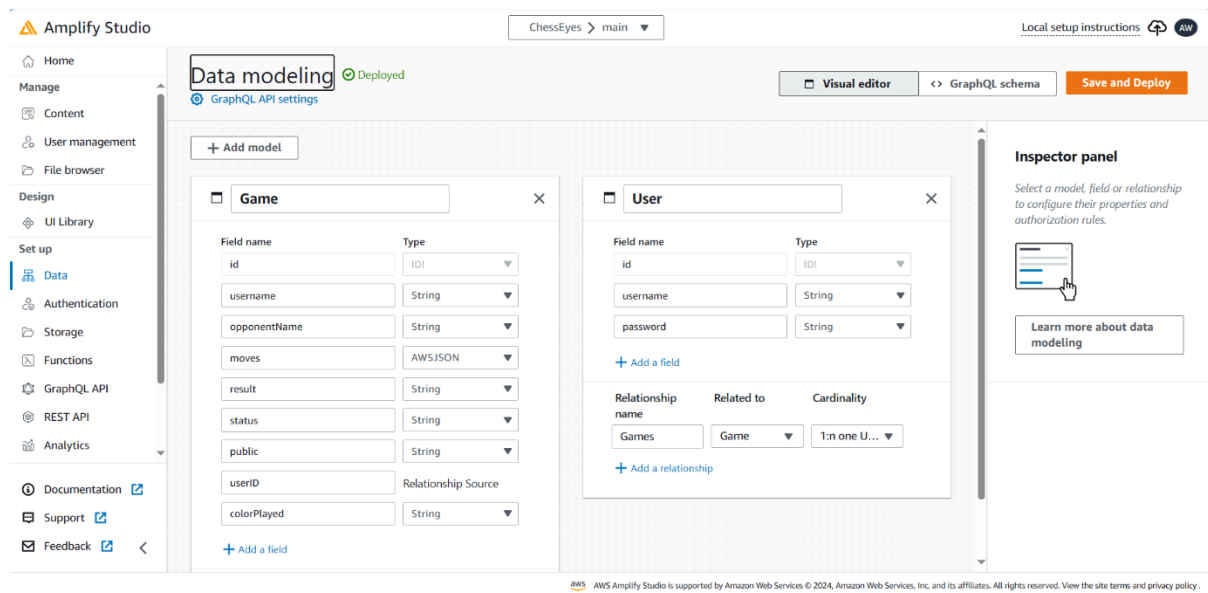


Figure 5.3.7.2. Creating database in Amplify Studio.

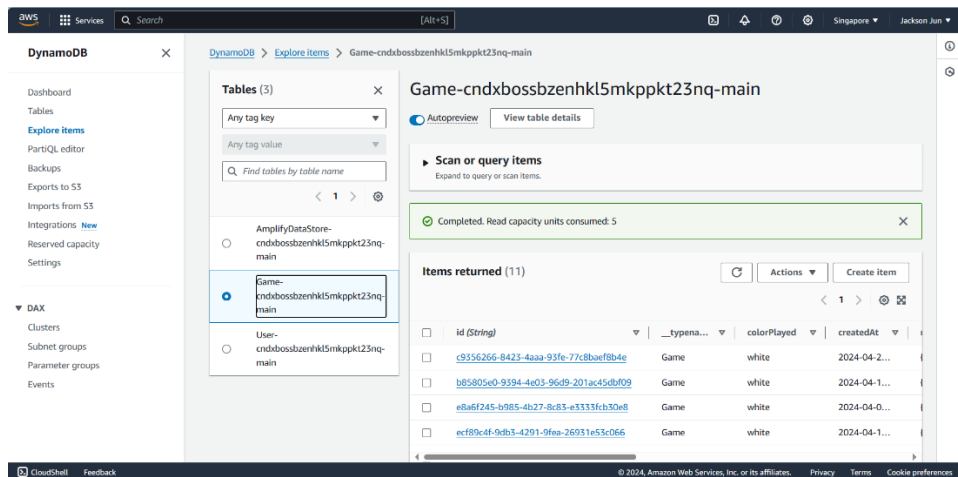


Figure 5.3.7.3. Tables successfully created in DynamoDB.

The database used in this project will be GraphQL powered by DynamoDB. In order to create a GraphQL database, firstly, Amplify Studio is enabled as shown in Figure 5.3.7.1. After enabling it, Amplify Studio is launched. The database is now created by navigating to the data tab on the left in Amplify Studio as shown in Figure 5.3.7.2. In this project, 2 required tables which are Games and Users table are created by utilizing the add model function as shown in Figure 5.3.7.2. After creating the table, the database can now be deployed by clicking on “save and deploy” button. For confirmation purposes, the tables are successfully created in DynamoDB as shown in Figure 5.3.7.3

### 5.3.8 Application Protocol Interface(API) and Lambda Function Creation Using Amplify CLI

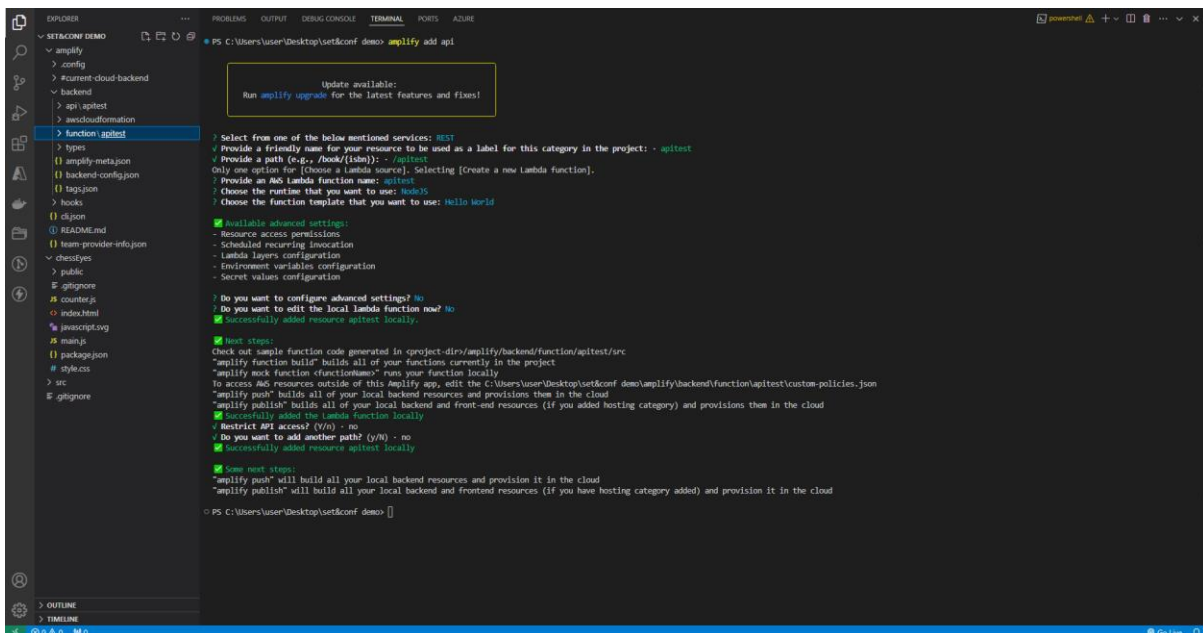


Figure 5.3.8.1. Creating API and lambda functions using Amplify CLI.

In order to create the backend function for the web application, Amplify CLI command which is “amplify add api” is used as shown in Figure 5.3.8.1. Upon executing the command, the terminal will prompt for type of API needed to be created and in this case will be REST API. After that, information such as directory name and path name is entered when prompted. Next, a Lambda function is created by providing the function name, runtime and function template when prompted. The API and Lambda function is now created as shown in the project directory in the navigation bar on the left.

### 5.3.9 Deploying the Whole Web Application to Amplify

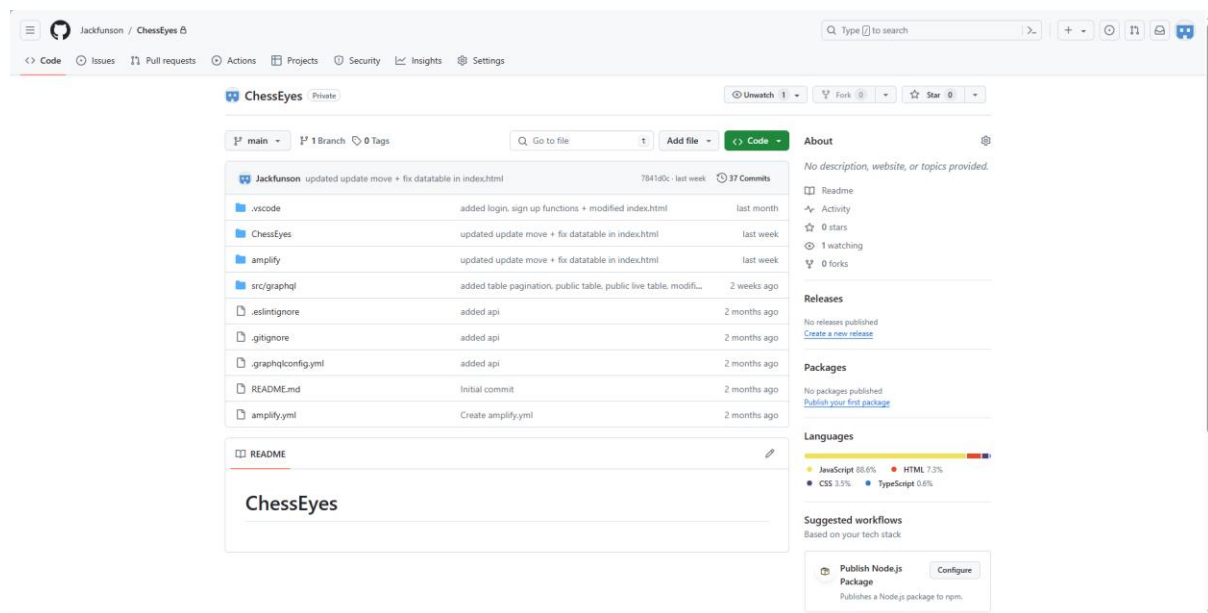


Figure 5.3.9.1. Project in GitHub repository.

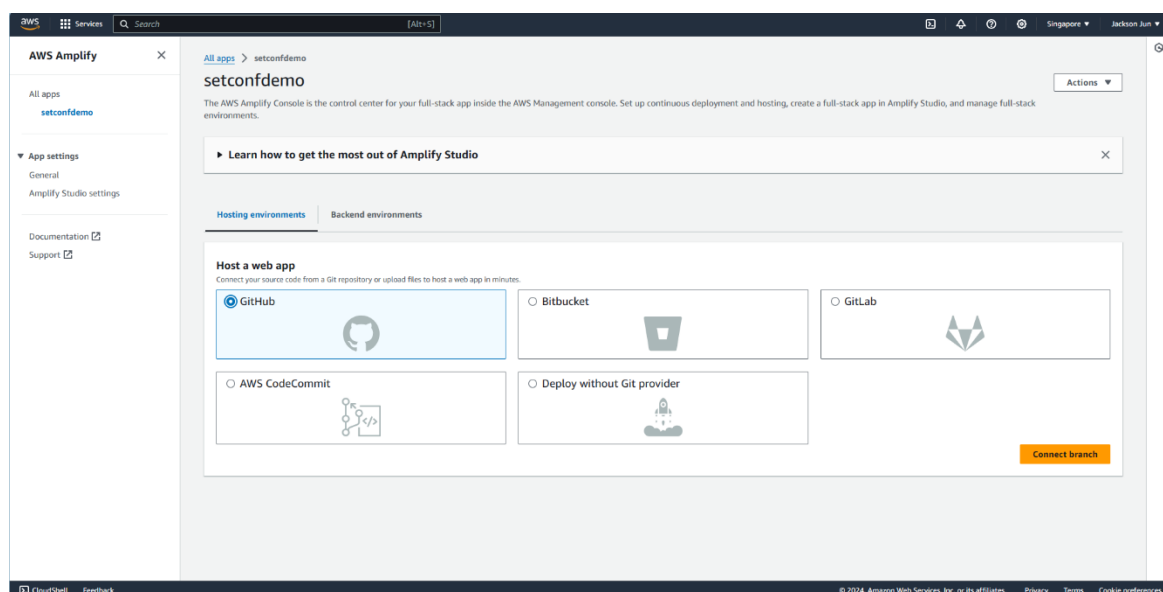


Figure 5.3.9.2. Selecting GitHub in hosting environment.

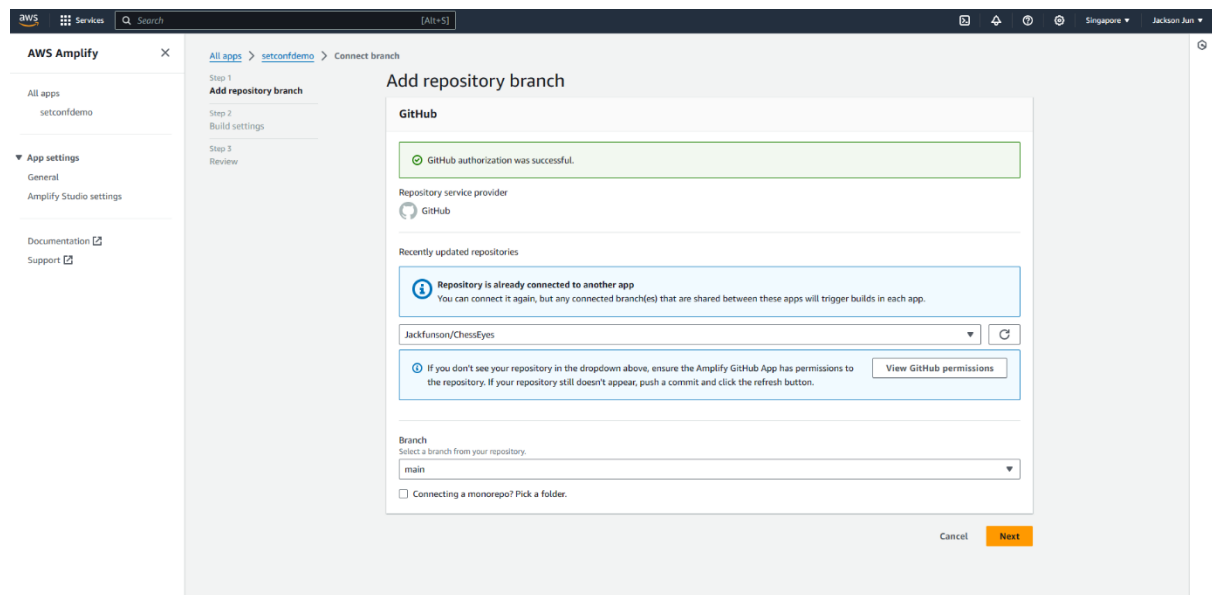


Figure 5.3.9.3. Adding the github respostory to Amplify.

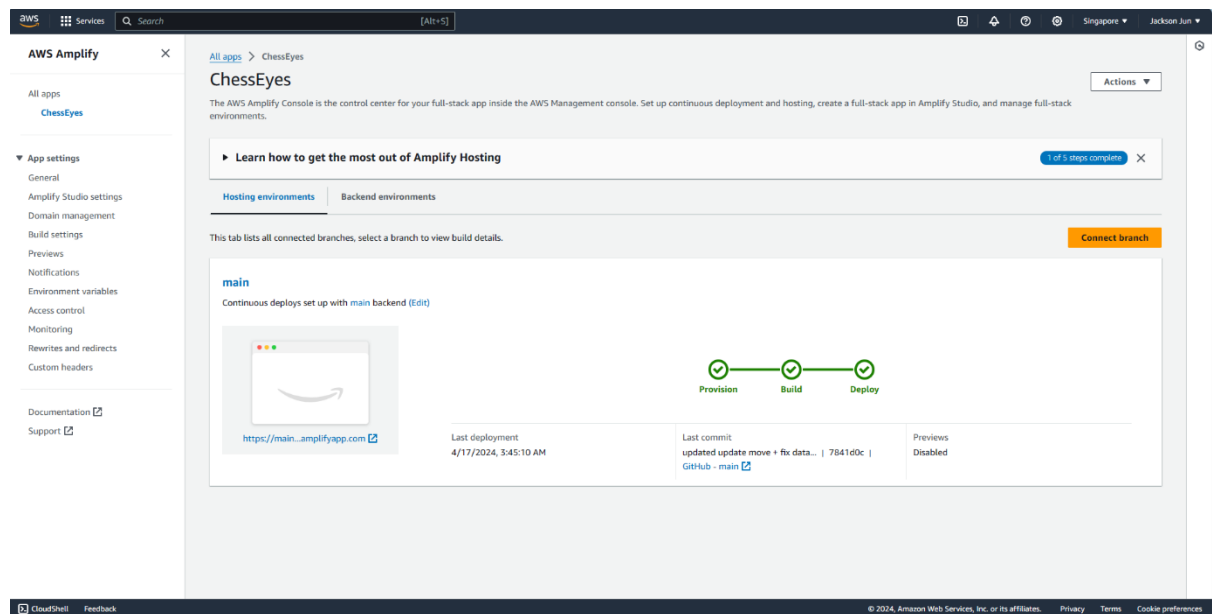


Figure 5.3.9.4. Web application successfully deployed.

Having all the previous steps done, the web application is now ready to be deployed and hosted in Amplify. Firstly, the whole project in Visual Studio Code is push or upload to a GitHub repository as shown in Figure 5.3.9.1. Next, the GitHub option is selected in Amplify app hosting environment as shown in Figure 5.3.9.2. After selecting GitHub as the deploy repository, the repository saved earlier is selected after connecting GitHub to Amplify with GitHub account as shown in Figure 5.3.9.3. Finally, the web application is successfully deployed to Amplify and can be accessed via the link generated as shown in Figure 5.3.9.4

### 5.4 System Operations

In this sub chapter, the system operations for web application and Raspberry Pi will be explained in detail. First of all, the web application can be accessed by clicking on the link generated by Amplify as shown in Figure 5.4.1 and will be redirected to the home page of the web application as shown in Figure 5.4.2. Currently, there is no signed-in user, hence, the home page only showing the public and live games to the user accessing the page.

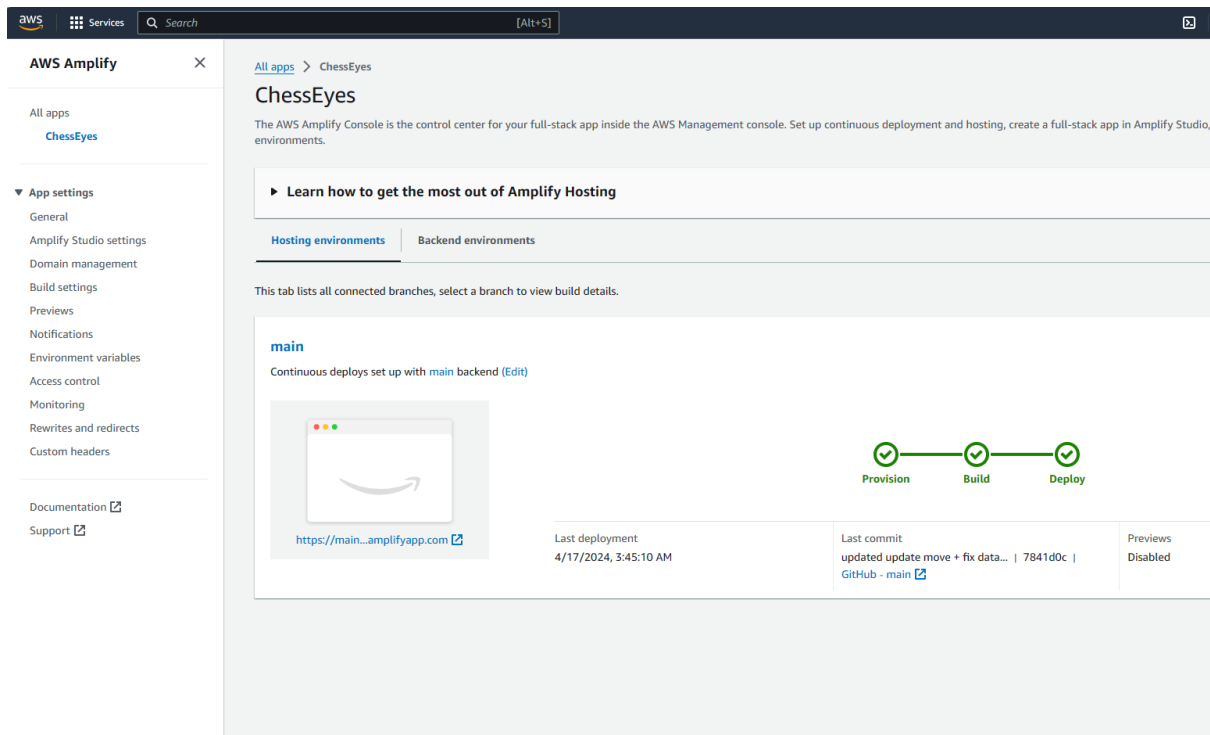


Figure 5.4.1. ChessEyes app hosting environment page.

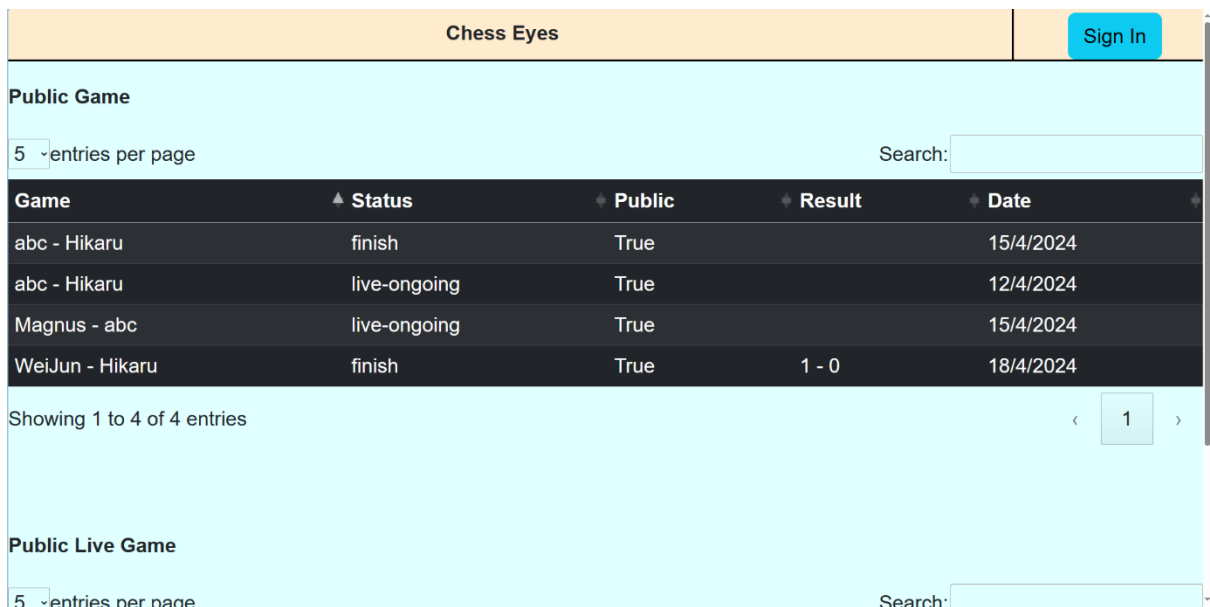


Figure 5.4.2. Web application home page.

In order to sign in to the system, the user can just click on the sign in button on the top right corner of the home page and will be redirected to the sign in page as shown in Figure 5.4.3. The user can now filling the required information such as username and password. Upon clicking on the sign in button, the system will verify the user by checking the username and password in the database. If the user is authenticated, the user will be redirected back to the home page. As shown in Figure 5.4.4, after signing in, the user's games tracked by Raspberry Pi are now shown in the home page.

Figure 5.4.3. Sign in page of the web application.

Chess Eyes				WeiJun
<b>Your Game</b>				
5 entries per page		Search: <input type="text"/>		
Game	Status	Public	Result	Date
WeiJun - BerlinDraw	live-ongoing	False		22/4/2024
WeiJun - Hikaru	finish	True	1 - 0	18/4/2024
WeiJun - ScholarMate	finish	False	1 - 0	22/4/2024
Showing 1 to 3 of 3 entries				< 1 >
<b>Public Game</b>				
5 entries per page		Search: <input type="text"/>		

Figure 5.4.4. Home page content after user signed in.

On the other hand, the user also needed to login using the username and password in Raspberry Pi in order to connect to the system as shown in Figure 5.4.5. After the user had login in into the system, the system will return all the games played by the user in a simple GUI as shown in Figure 5.4.6. The user can now create a game for track later by clicking on the “create game” button on the bottom of the GUI. After that, a new GUI will popup to allow user to enter the details of the game such as the color to be played, the public status and the opponent name as shown in Figure 5.4.7. After creating the game, a new game can be seen in the GUI and is now ready to be tracked by clicking on the “track” button on the right as shown in Figure 5.4.8.



Figure 5.4.5. Login GUI in Raspberry Pi.



Figure 5.4.6. Home page GUI in Raspberry Pi.

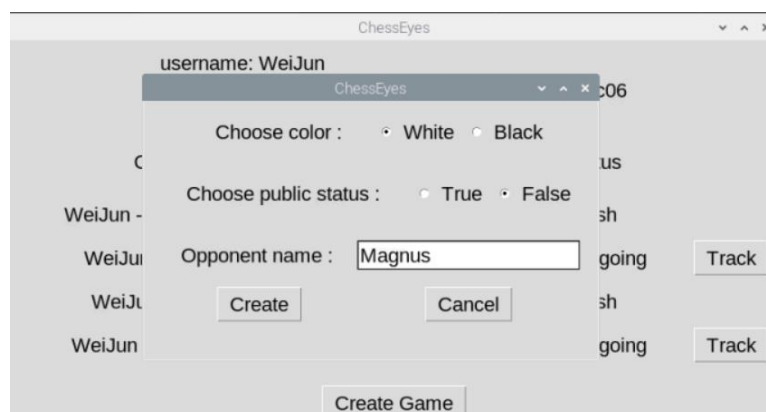


Figure 5.4.7. Create game popup GUI in Raspberry Pi.

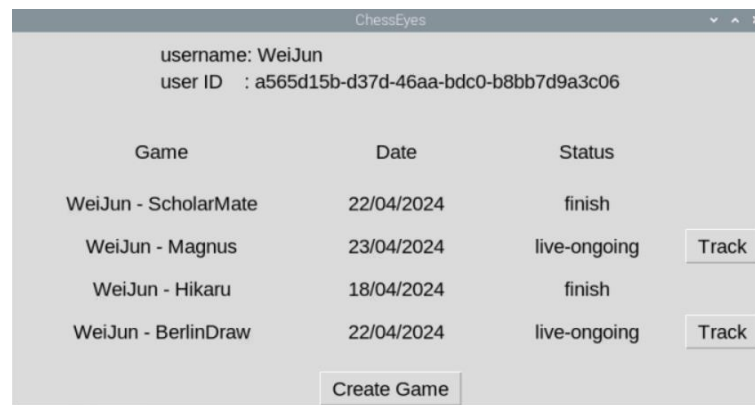


Figure 5.4.8. Updated home page GUI in Raspberry Pi.

On the side of the web application, a newly added game can be seen in the home page as shown in Figure 5.4.9. Upon clicking onto the row of the newly added game in the table, the web application will redirect user to the chess game page as shown in Figure 5.4.10. Upon loading the chess game page, the web application first retrieves the game's details from the database such as the opponent name, moves and game status. Posterior to obtaining the game details, the web application will then display the opponent name and moves in the scoreboard on the right. Additionally, the web application will then check the game status. If the game is finished, then the web application will not subscribe to the database for any updated data about the game and will do otherwise. Hence, in this case, the web application will subscribe to the database for any updated data.

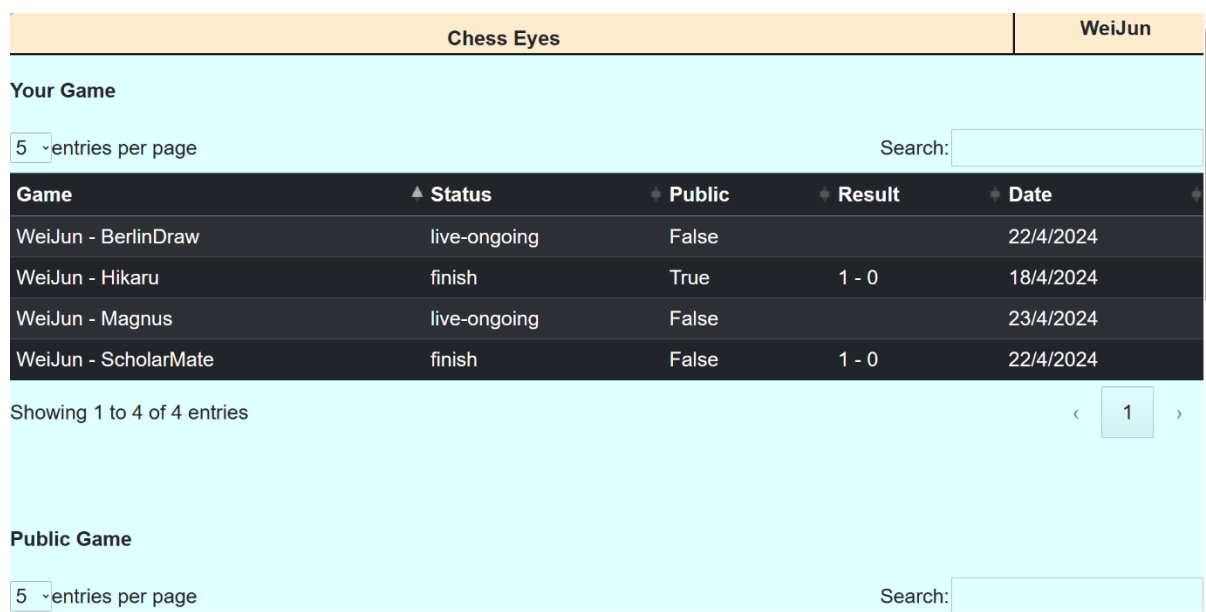


Figure 5.4.9. Updated home page in web application.



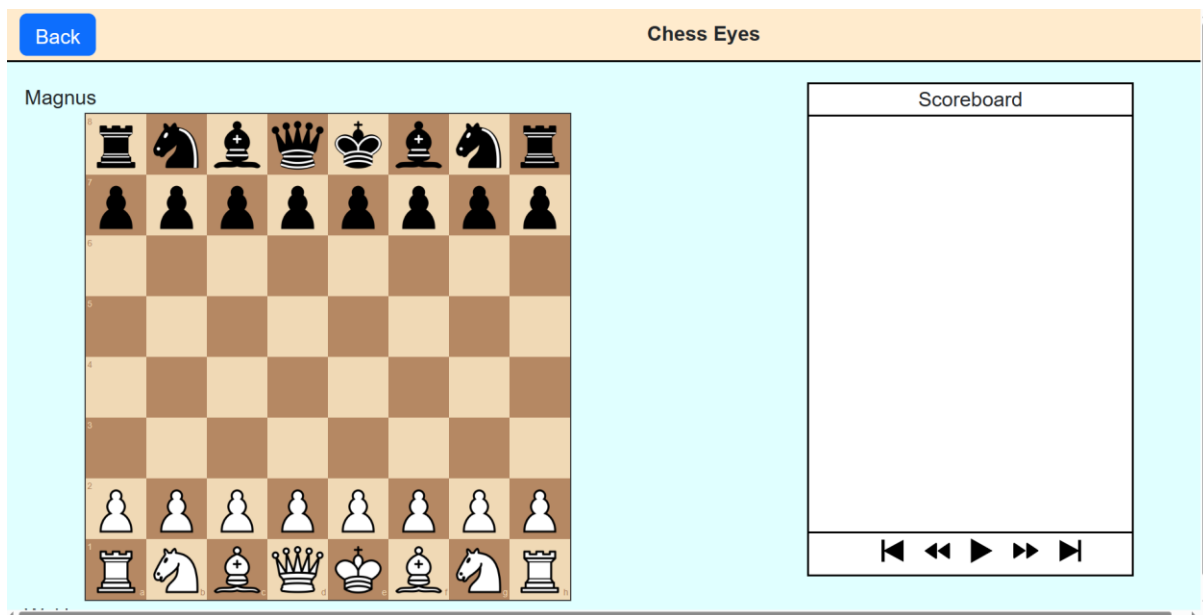


Figure 5.4.10. Chess page in web application.

Now, user can then click on the track button in the home page GUI in Raspberry Pi to start tracking the game. Upon clicking on the track button, a new GUI will be created to show all the processes the system takes to process the chess game image captured by the camera as shown in Figure 5.4.11.

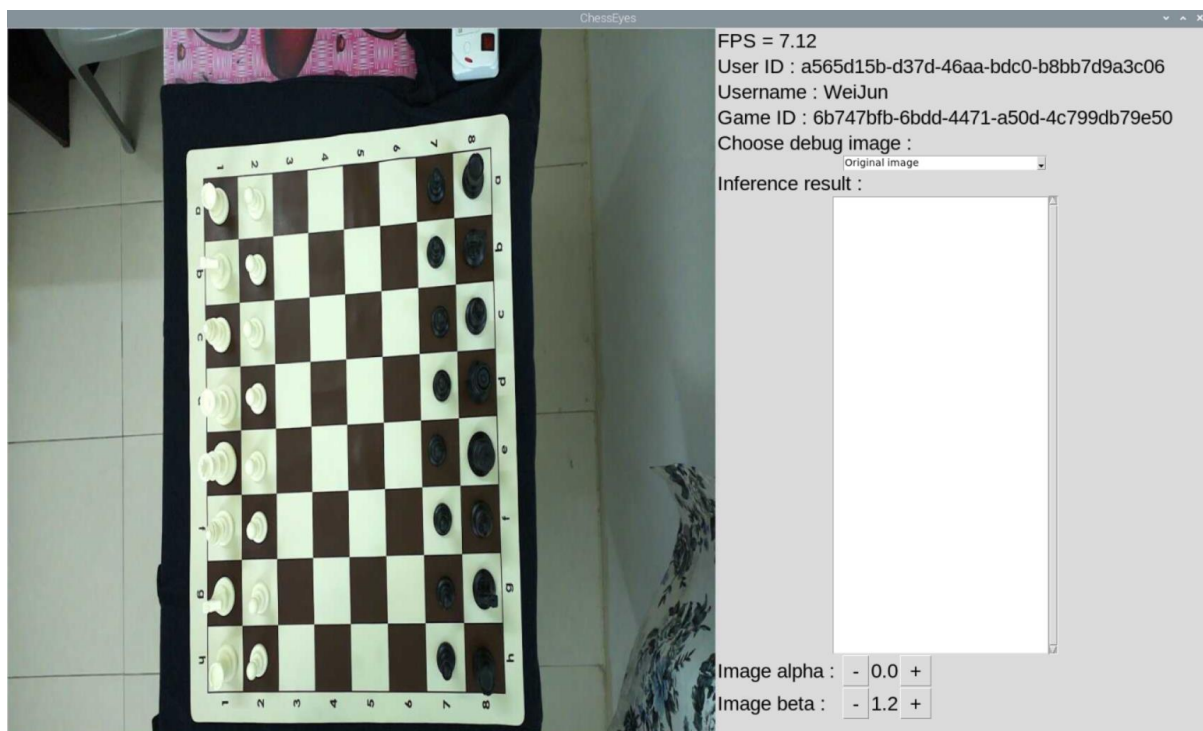


Figure 5.4.11. Chess game tracking GUI in Raspberry Pi.

After the camera captured the raw image in RGB format, the raw image will then be processed by converting to gray scale image first and apply gaussian filtering to the gray scale image to reduce noises of the image. The processed image will then be used in obtaining a threshold image using Threshold() function provided by OpenCV. As shown in Figure 5.4.12.

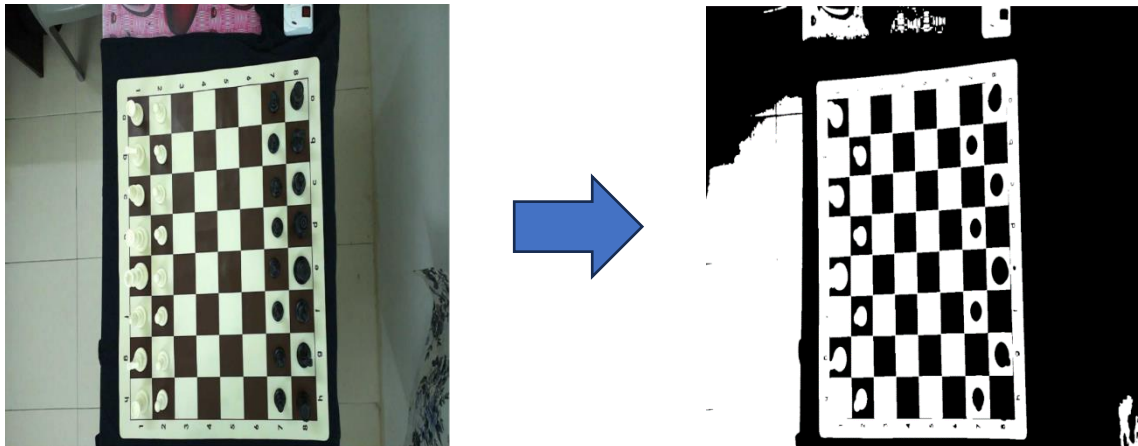


Figure 5.4.12. Threshold image obtained from processed raw image.

The threshold image obtained will then be used in finding the largest contour which also represents the contour of the chessboard in the image. The result of process can be seen in Figure 5.4.13. After obtaining the contour, the polygonal curve of the contour will be approximated using approxPolyDP() function in OpenCV library to obtain the 4 corner points of the contour of the chessboard. After successfully approximating the 4 corner points as shown in Figure 5.4.14, these points will then be used in transforming the chessboard image found in the raw image into a top-view perspective image as shown in Figure 5.4.15. The newly transformed top-view perspective image will incidentally become the first reference image used in comparison to detect the chess moves.

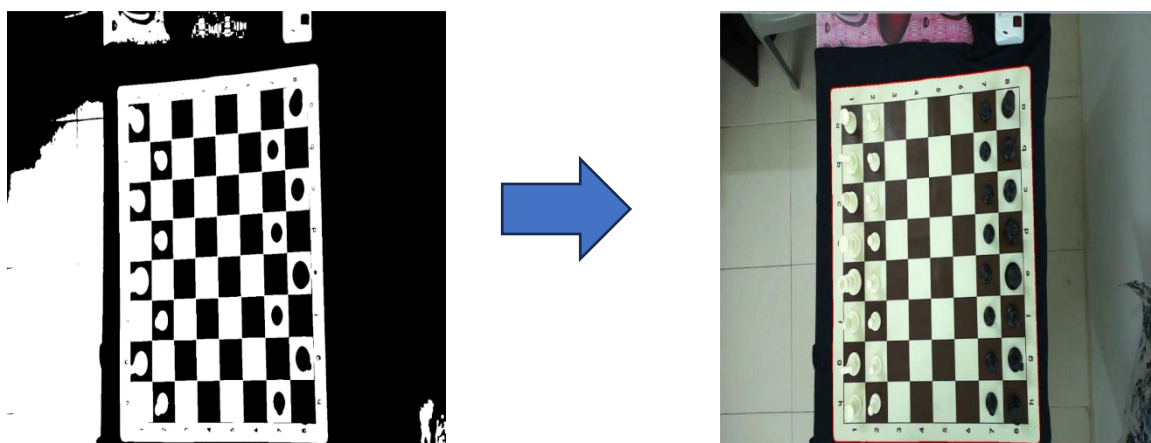


Figure 5.4.13. Contour of chessboard obtained from threshold image.

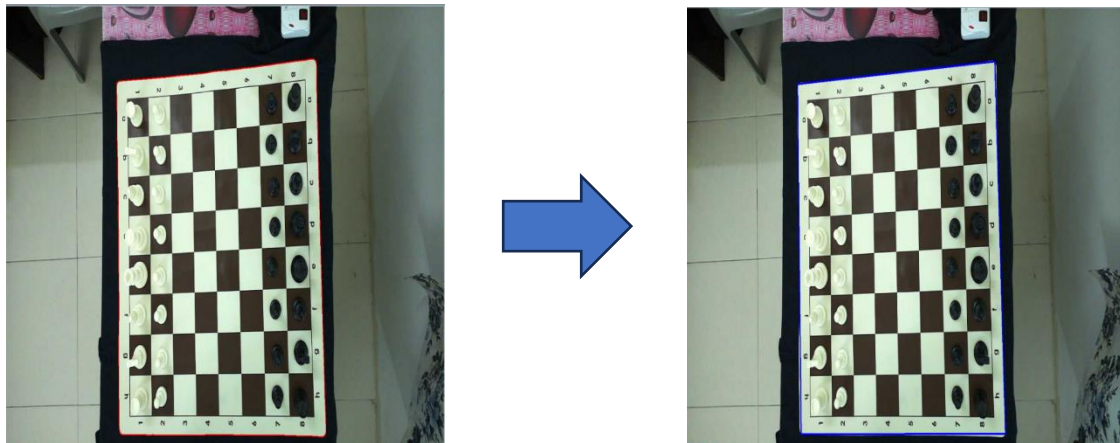


Figure 5.4.14. 4 corners points found using approxPolyDP() from contour.

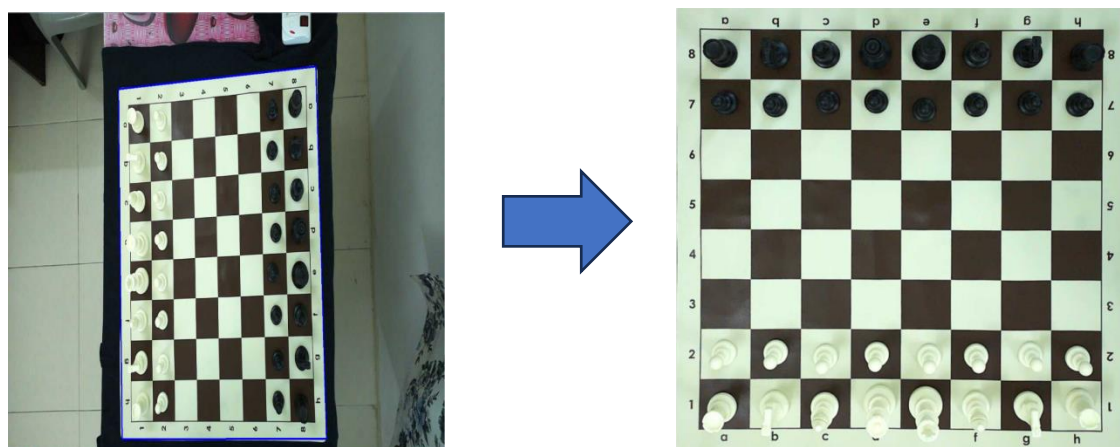


Figure 5.4.15. Top-view perspective image obtained using the 4 corner points.

Upon obtaining the top-view perspective image, the corner points of each square grid in the image will be found using findChessboardCornersSB() provided by OpenCV as shown in Figure 5.4.16 indicated by little blue dots. These points is crucial to be found because it will be more accurate and efficient when it comes to chess move detection.

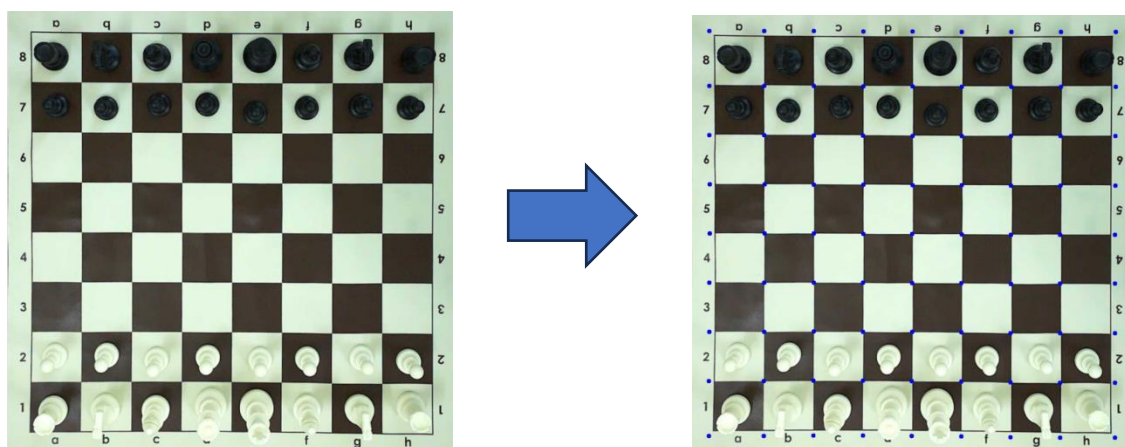


Figure 5.4.16. Square grid corner points found from the top-view perspective image.

Finally, the Raspberry Pi is now ready to track the game. When raspberry pi detected move played by player, it will first compare the reference image and the latest snapshot captured by the camera using `absdiff()` function provided by OpenCV library as shown in Figure 5.4.17 to obtain a image that contain changes between the two images in terms of black and white bits where black bits indicate no change and white bits indicate change.

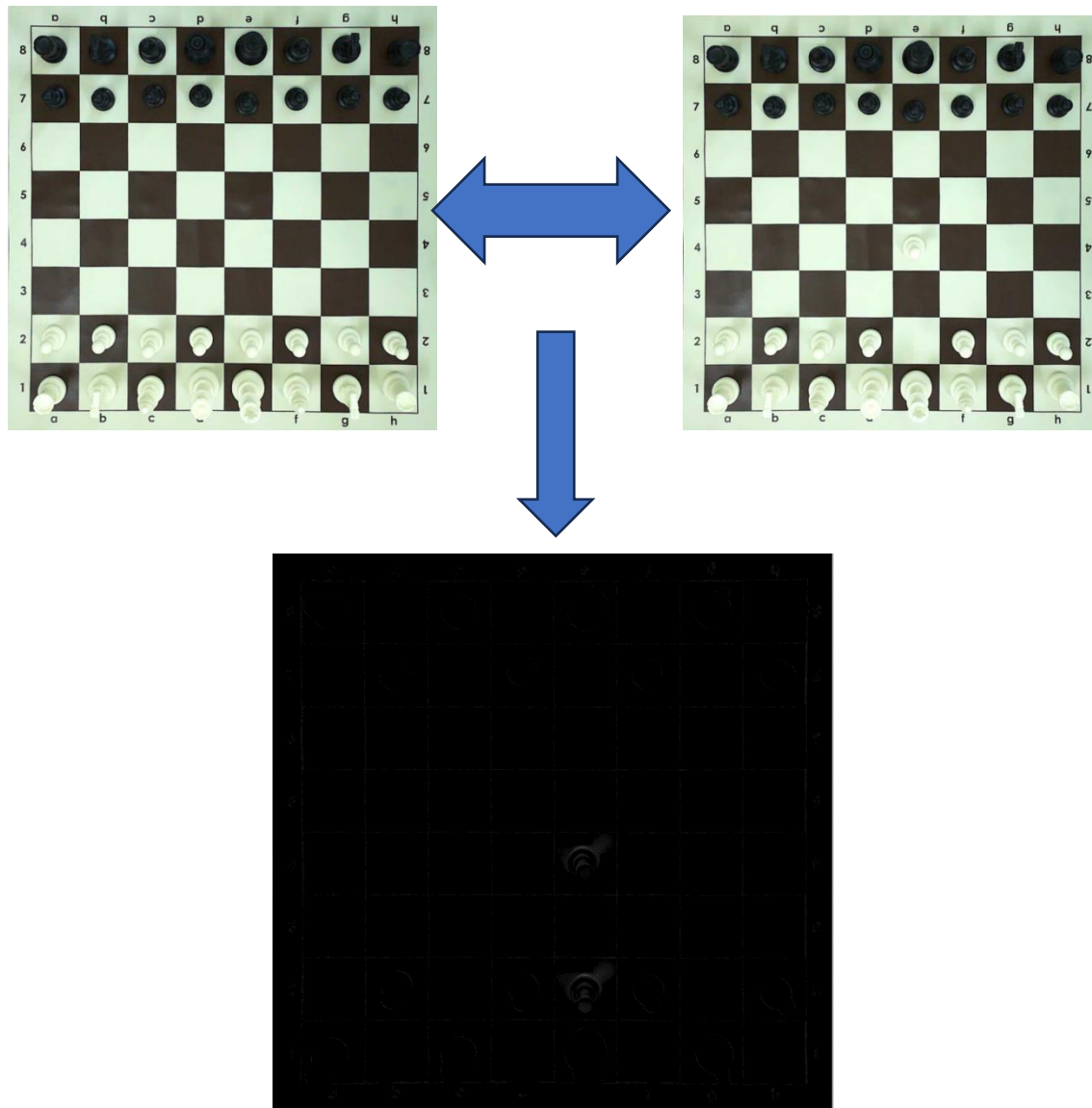


Figure 5.4.17. Comparison of reference and latest snapshot image using `absdiff()`.

The square grid corner points found earlier will then now used in constructing masks, specifically 64 masks that will segment each particular square in the image and count the non-zero bits, which is also the white bits. If the count exceed a threshold, where the threshold in this case is the size of the mask for a particular square, it indicates that there is change in that particular square and add into a list. After scanning through all 64 squares, the expected number of masks in the list is 2 that indicates a move, 3 that indicates en passant is played and 4

indicates castling and resignation or draw by agreement. For this example, there is a move detected, hence there will be 2 masks. The 2 masks will then be combined into a single empty black image to obtain a segmentation mask. The segmentation mask is then used to segment the latest snapshot image to obtain a segmented image used in YOLOX model inference for object detection as shown in Figure 5.4.18.

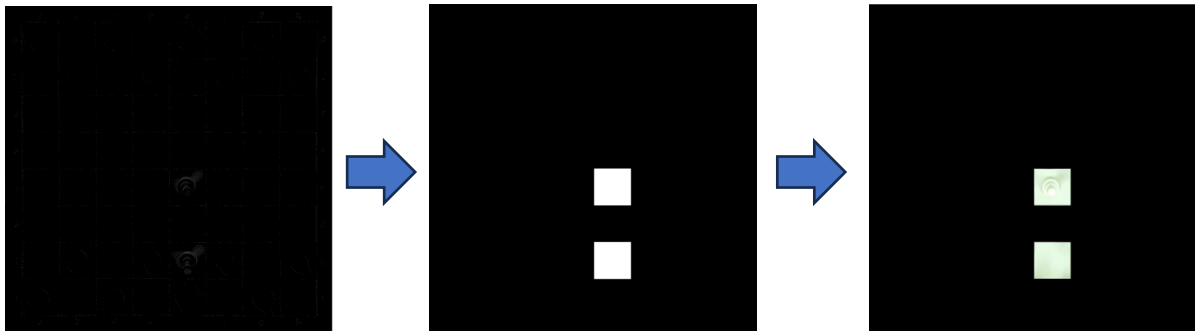


Figure 5.4.18. Segmented image obtained using segmentation mask.

After model inference and calculating the source square and destination square, the information will then be sent to the backend of the web application for further processing and update the database. Upon updating the database, the updated data will then be sent to chess game page since the page had subscribed to it. The updated data including the latest move sent by the database will then be displayed on the chessboard and in the scoreboard on the right as shown in Figure 5.4.19. Hence, the system had achieved real-time display of move played physically. The latest snapshot image will then become the reference image.

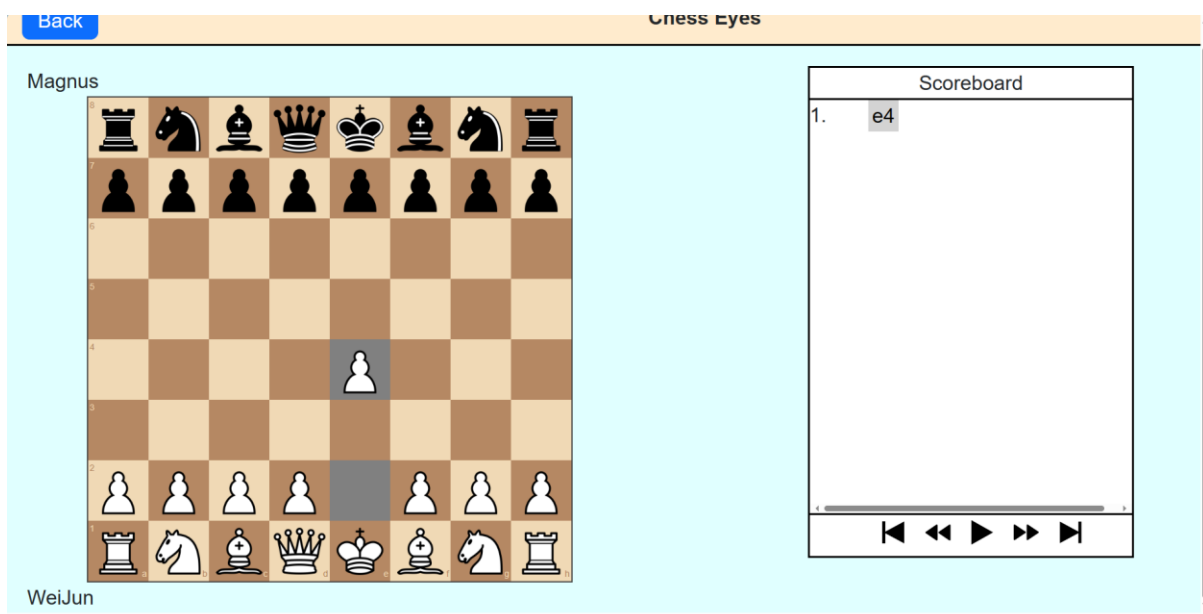


Figure 5.4.19. Chess game page displaying real-time move.

When raspberry pi detected another move, same procedure will be executed again as shown in Figure 5.4.20, Figure 5.4.21, and Figure 5.4.22. The same procedure will be applied as long as the game is ongoing.

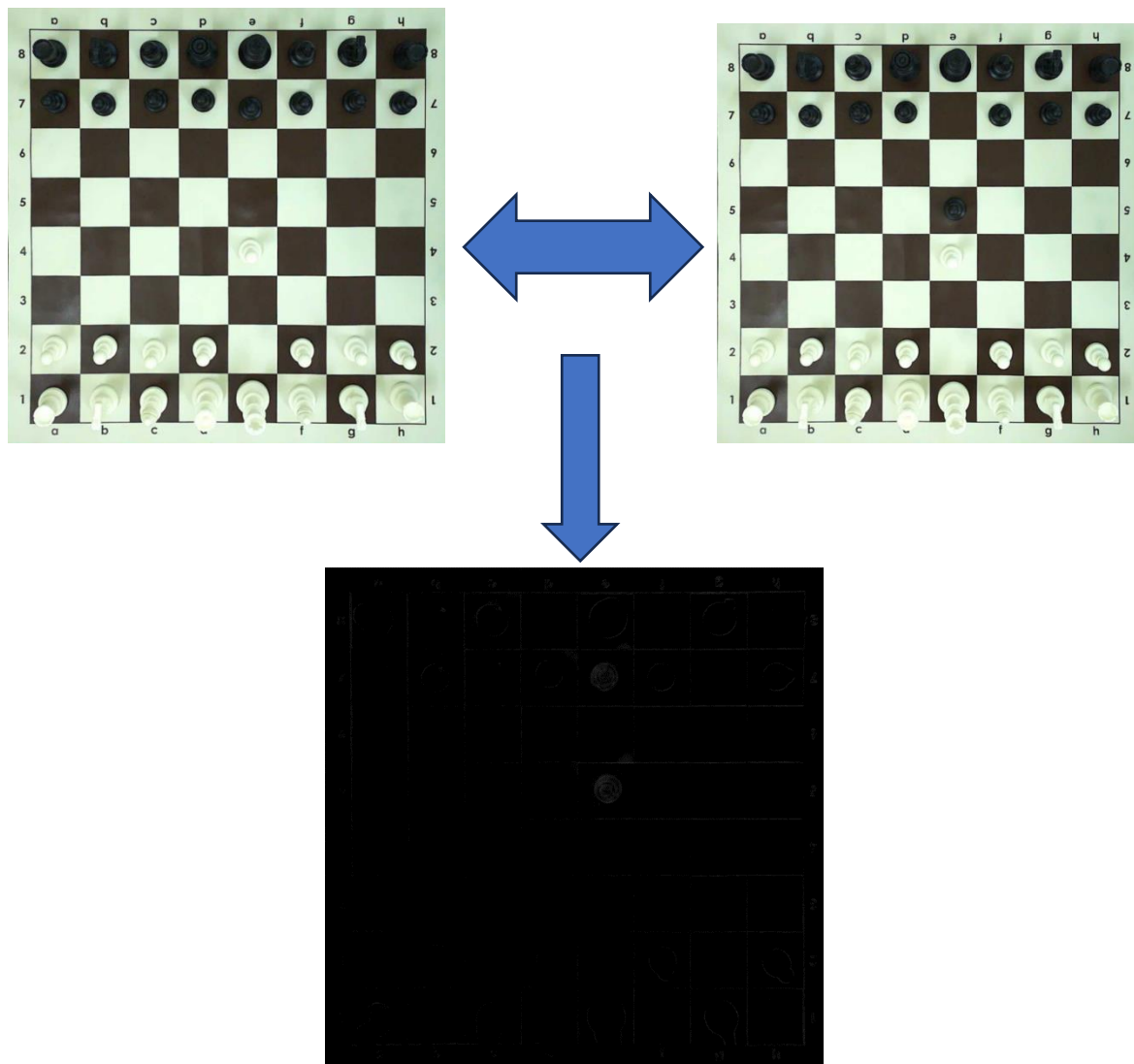


Figure 5.4.20. Comparison of reference and latest snapshot image using absdiff() for second move.

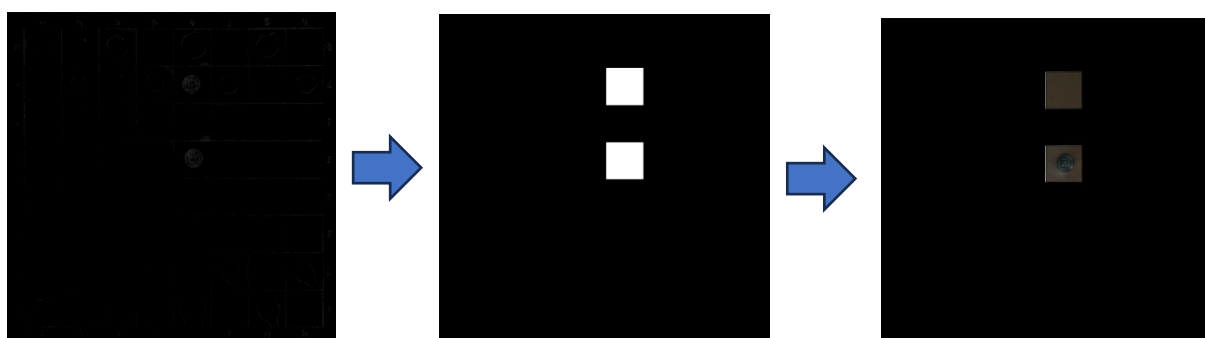


Figure 5.4.21. Segmented image obtained using segmentation mask for second move.

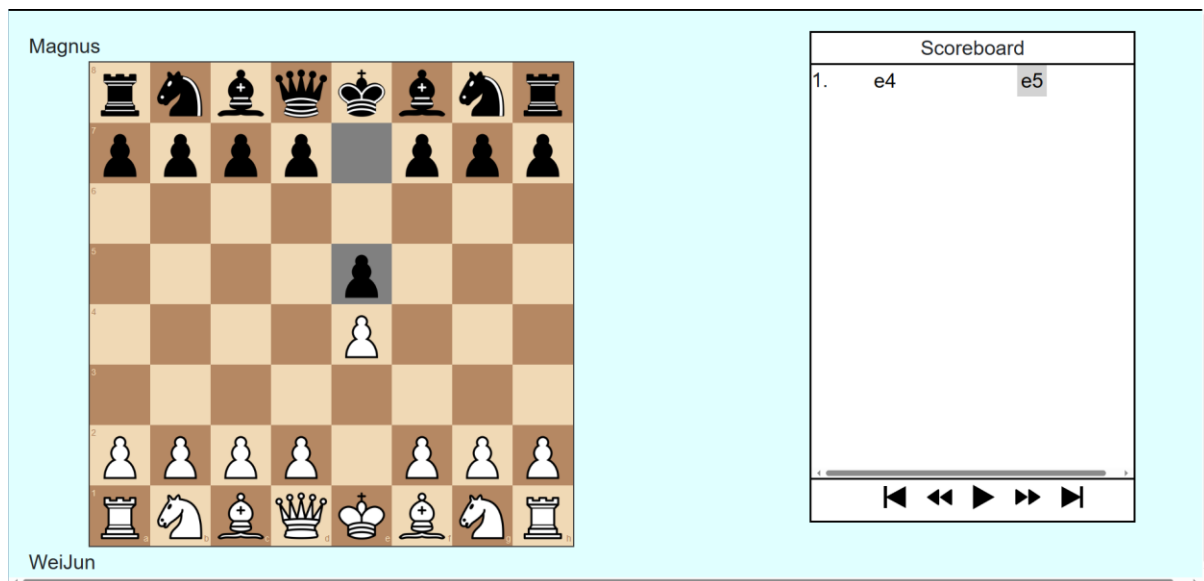


Figure 5.4.22. Chess game page displaying real-time move for second move.

### 5.5 Implementation Issue and Challenges

One of the implementation issue and challenge in this project is preparing an optimal environment. For the system to work at its best, the environment should have adequate lighting that will reduce or eliminate shadows as any shadow on the chessboard is considered a change if the change exceed a certain threshold.

Besides, the accuracy of tracking the correct move from the physically played chess game poses a challenge as it is needed to be 100%. Whenever there is any error in tracking the ongoing game, there will be error for the entire duration of the game. Hence, the chess game tracking system needed to be perfectly calibrated to handle all kind of environment which is quite a challenge as the game can be played in any location with various environment.

Moreover, the venue for the ongoing chess game should have a stable and fast network speed as the Raspberry Pi require it to send the information to the web application via internet. Inconsistency between the state of the physical chess game and the state of the 2D digital chess game displayed on the web application might occur if the previously stated requirements did not met. Thus, internet speed in the venue also poses challenge to the entire system where issue may arises.

## CHAPTER 6

### SYSTEM EVALUATION AND DISCUSSION

#### 6.1 System Testing and Performance Metrics

Below are a few inputs that can be used to test the performance of the system in terms of accuracy and consistency in tracking the physical chess game:

1. Moving piece from the previous position to the current position. For example, moving black knight from its starting position, which is g8, to f6. Thus, the system should be capable of detecting, displaying and saving the correct piece and positions.
2. Capturing piece by another piece. For instance, white pawn capturing black pawn where the white pawn's previous position is e4 and black pawn's position is d5. Hence, the system should be able to detect, display and saving the capturing move.
3. Castling move. It is a unique move where it involve two pieces which is the king and a rook to move the king to a safer square. Commonly, there are two castling move which is long castling that move the king towards its rook to the queen side and short castling that move the king towards its rook to the king side.
4. En passant. It is also a unique move that only can happen when a pawn is half pass the board and the opponent's pawn move 2 square forward to its side. Now, en passant can only be played in this move to capture the opponent's pawn in a usual way. If another move is played, en passant no longer can be played. For example, if the white pawn's current position is on e5 and black play pawn to d5, white can capture the d5 black pawn and the new position for the white pawn is d6.
5. Pawn promotion. It is also a unique move by a pawn when the pawn is moved to the opponent's first rank. A pawn can be promoted to either a knight, a bishop, a rook or a queen. Hence, the system should be able to detect the new piece begin promoted. In this project, the pawn promotion will set to be always to a queen.
6. Resignation. It is basically a declaration of surrender to the opponent by placing both king on the squares with the color of the winning side in the center of the board. For example,



If white resigns, then both king will be placed on dark square in the center of the board, specifically, d4 and e5 square. If black resigns, both king will be placed on the light square in the center of the board, specifically, e4 and d5 square.

7. Checkmate. It is a game-ending declaration by the winning side when the opponent's king does not have any more available square to move to. Hence. The system should be able to detect checkmate when move is begin played.

As for the performance metrics of the system, it is defined as how accurate the system can track the chess game without making an error which is detecting wrong squares, detecting extra square that should not be there and many more In the best-case scenario, the system should be capable of tracking the chess game from the start to the end of the game successfully, which also means the accuracy of the system in tracking the chess game is 100% without a single error and without any human intervention.

## **6.2 Testing Setup and Result**

### **6.2.1 Testing Setup**

There are 4 setup or test cases that can be used in testing the system with the inputs stated in the previous sub-chapter are as follows:

1. Play a game that includes moving piece from the previous position to the current position, capturing piece by another piece, castling move, pawn promotion and checkmate.
2. Play a game that shows the game will be played until a player resign, for example, when play until 10<sup>th</sup> move, black resigns.
3. Play a game that shows en passant move and white resigns
4. Play a game that shows draw by agreement from the start.

The results of all the four testing setup can be seen in the following sub-chapters.

### **6.2.2 Result**

#### **6.2.2.1 Result for First Test Case**

In testing the first test case, a game of 55 moves is played to test the stated scenarios in previous sub-chapter. First of all, the chess tracking system are able to detect piece moving from

the previous to the current position and display in the web application. For example, when the white pawn is moving from e2 to e4 which successfully detected by Raspberry Pi as shown in Figure 6.2.2.1.1, the detected move is sent to the web application and the web application is also successfully display the correct move played by the white player as shown in Figure 6.2.2.1.2

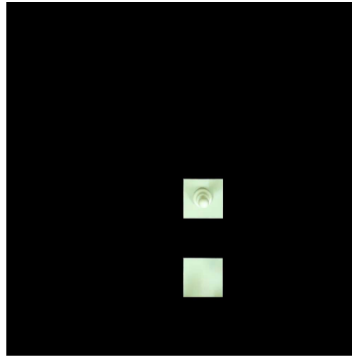


Figure 6.2.2.1.1. White pawn move from e2 to e4 detected by Raspberry Pi.

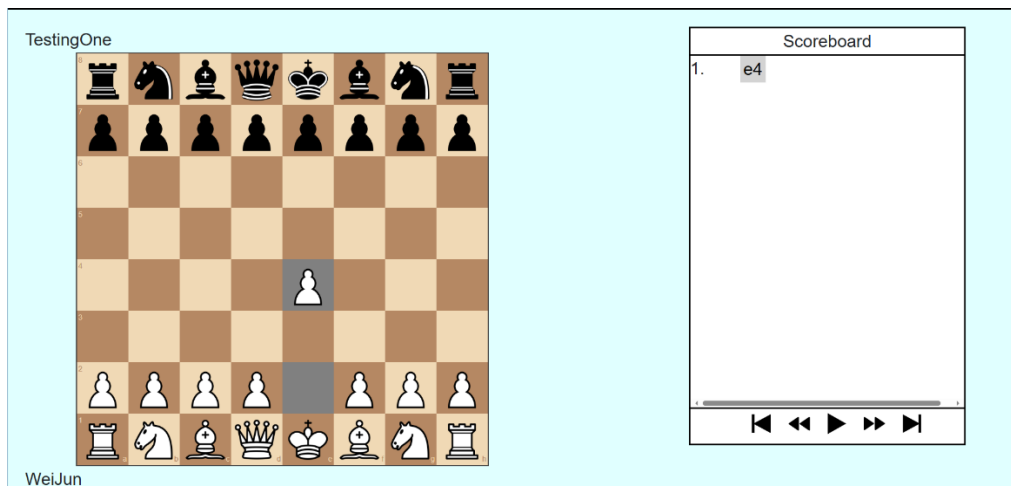


Figure 6.2.2.1.2. Detected move displayed correctly in web application.

On the 4<sup>th</sup> move for white, the white player performs a special move which is castling move, specifically, short-castling. Raspberry Pi also successfully detected the special move as shown in Figure 6.2.2.1.3 with 4 squares segmented from the latest snapshot. The castling move is then sent to the web application and the web application is also successfully displaying the castling move for white as shown in Figure 6.2.2.1.4. The same result can also be seen in the 5<sup>th</sup> move for black, where black player also performs castling move and the move also successfully displayed in the web application as shown in Figure 6.2.2.1.5 and Figure 6.2.2.1.6.

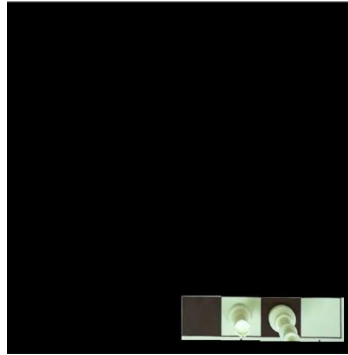


Figure 6.2.2.1.3. White castling move detected by Raspberry Pi.

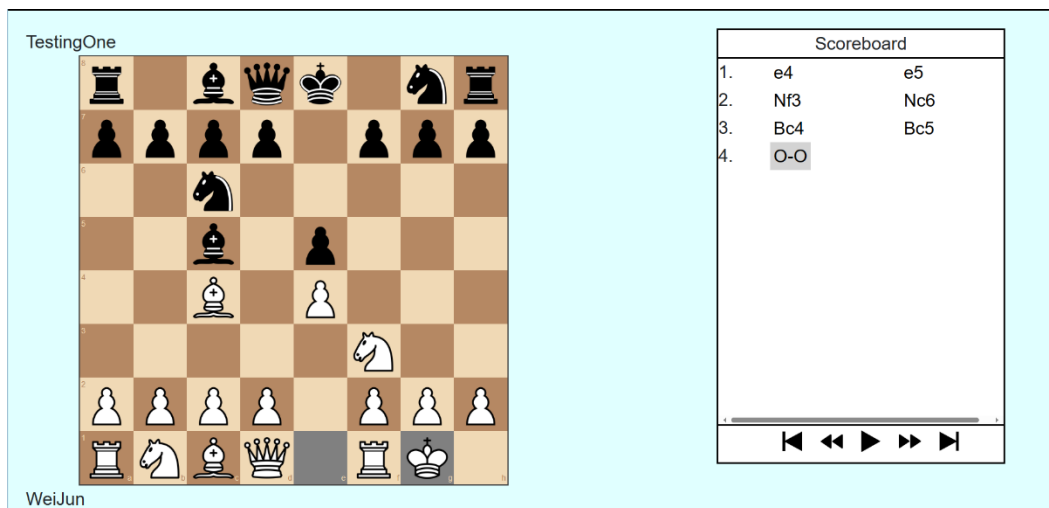


Figure 6.2.2.1.4. White castling move displayed correctly in web application.

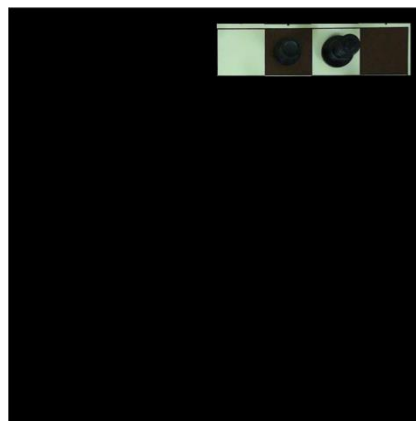


Figure 6.2.2.1.5. Black castling move detected by Raspberry Pi.

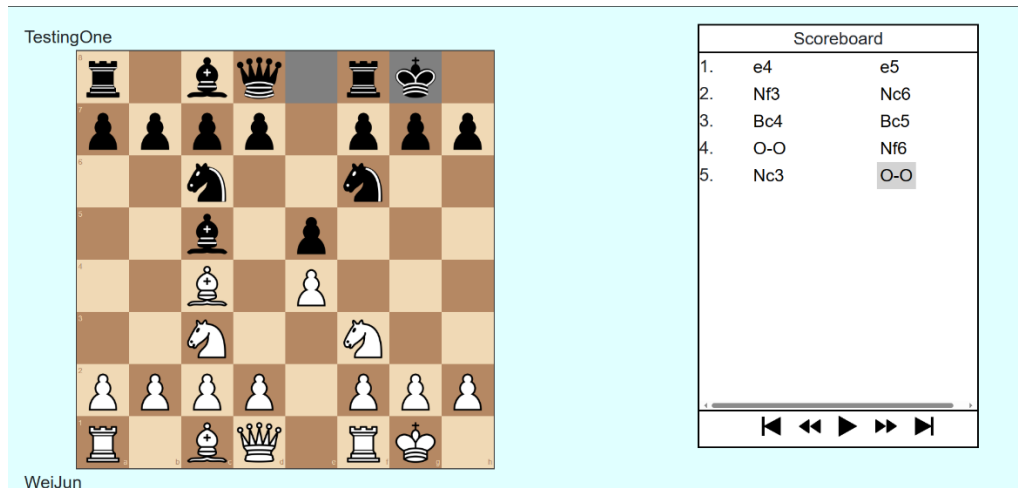


Figure 6.2.2.1.6. Black castling move displayed correctly in web application.

Furthermore, on the 8<sup>th</sup> move for white, white player moved his knight from c3 to d5, as detected by Raspberry Pi and can be seen in Figure 6.2.2.1.7, where the move is also being displayed correctly in the web application as shown in Figure 6.2.2.1.8. Now, the black player played a capturing move with his black knight on f6, moving from f6 to d5, where d5 is the square white knight is on. The Raspberry Pi detected the move as shown in Figure 6.2.2.1.9 and sent it to the web application, which also successfully displaying the capturing move played by black player as shown in Figure 6.2.2.1.10.

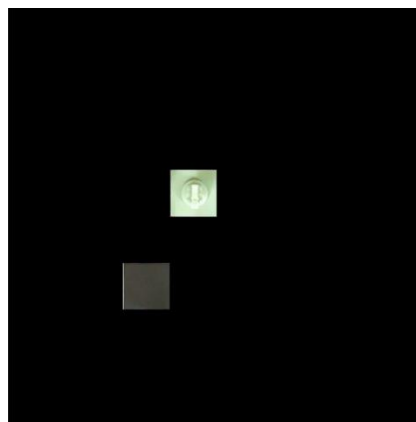


Figure 6.2.2.1.7. White knight moved from c3 to d5 detected by Raspberry Pi.

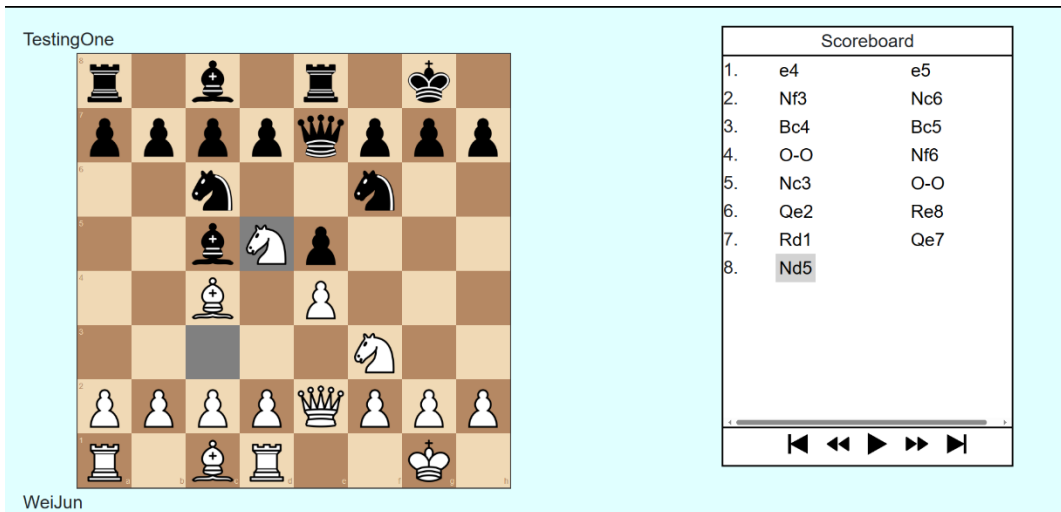


Figure 6.2.2.1.8. White knight moved from c3 to d5 displayed correctly in web application.

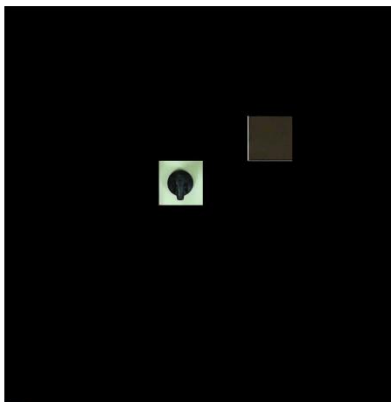


Figure 6.2.2.1.9. Black knight moved from f6 to d5 detected by Raspberry Pi.

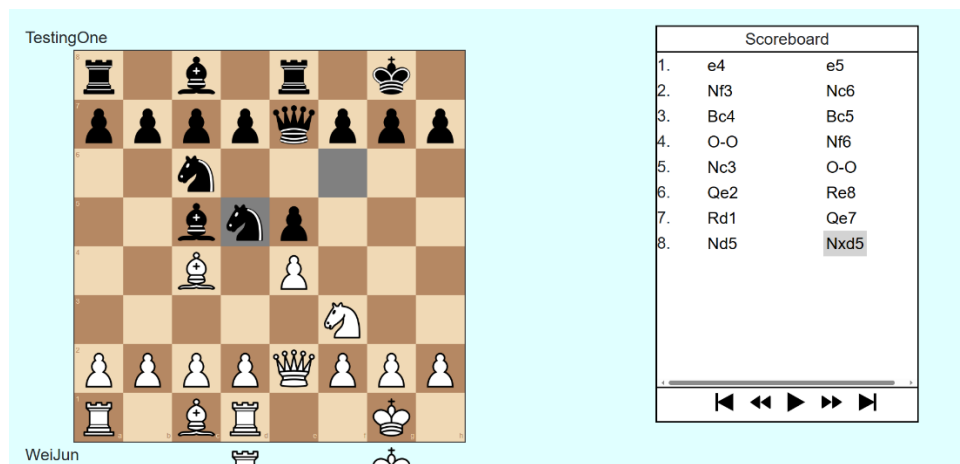


Figure 6.2.2.1.10. Black knight capturing move displayed correctly in web application.

Moreover, on black 50<sup>th</sup> move, black player moved the king from h8 to h7 as detected by Raspberry Pi and sent to web application where the move is displayed correctly as shown in Figure 6.2.2.1.11. Now, white player has the opportunity to move the pawn on b7 to b8 and promote the pawn to a queen in which the white player done so. The pawn promotion move is detected by Raspberry Pi as shown in Figure 6.2.2.1.12 where the pawn is replaced by an extra queen, and the pawn promotion move is sent to the web application where the web application is also successfully displayed the pawn promotion move correctly as shown in Figure 6.2.2.1.13.

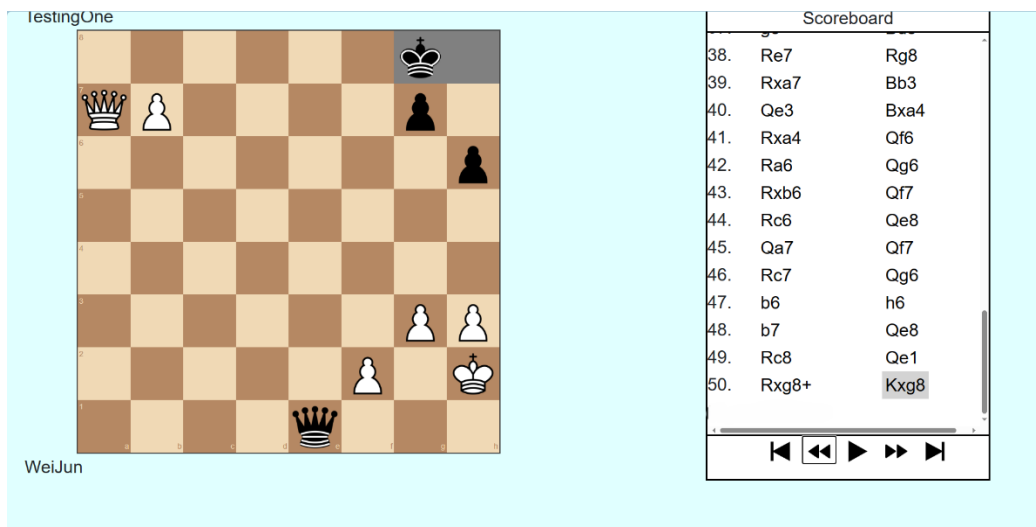


Figure 6.2.2.1.11. Black player moved king from h8 to h7 as displayed in web application.

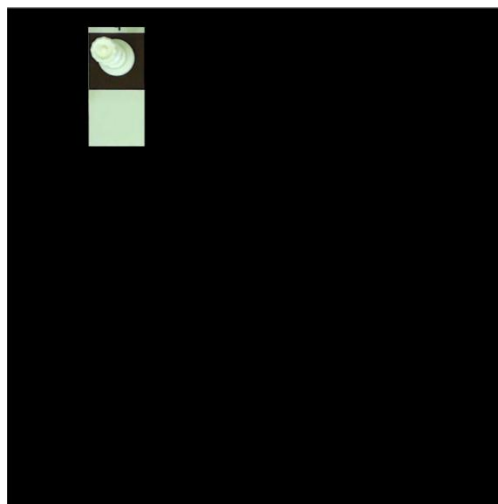


Figure 6.2.2.1.12. White player promote the pawn to queen by moving pawn in b7 to b8 as detected by Raspberry Pi.

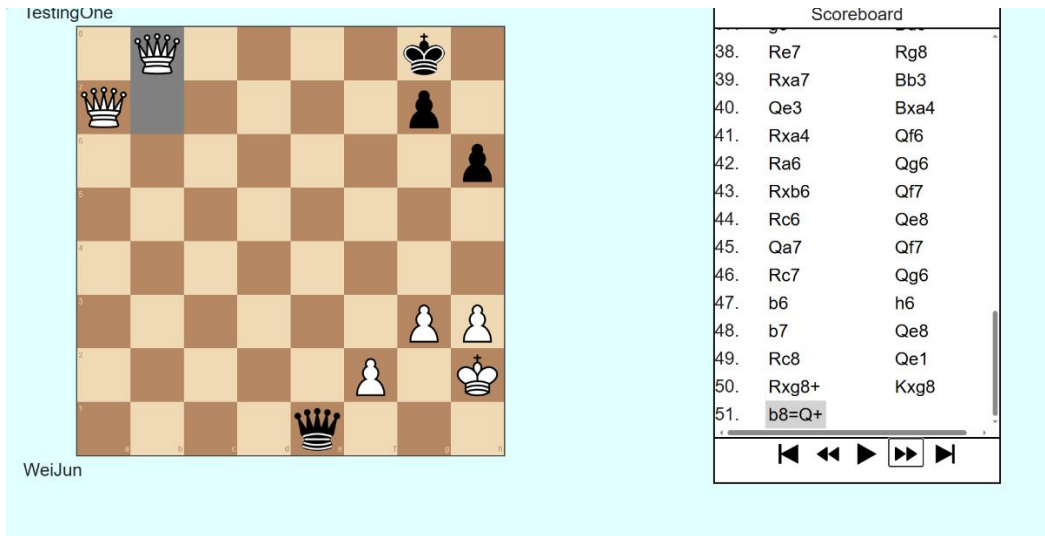


Figure 6.2.2.1.13. The pawn promotion move displayed correctly in web application.

Lastly, on black 54<sup>th</sup> move, black player moved the queen from e1 to f1 as shown in Figure 6.2.2.1.14. On the next move, white player moved his queen on the “b” file to capture the pawn g7, which resulted in checkmating the black king as detected by Raspberry Pi which can be seen in Figure 6.2.2.1.15. The capturing move is then sent to the web application for further processing. After processing the latest move played by white player, which is the capturing move, the system declared victory for white by checkmating the black king and the capturing move with the result is displayed successfully in the web application as shown in Figure 6.2.2.1.16.

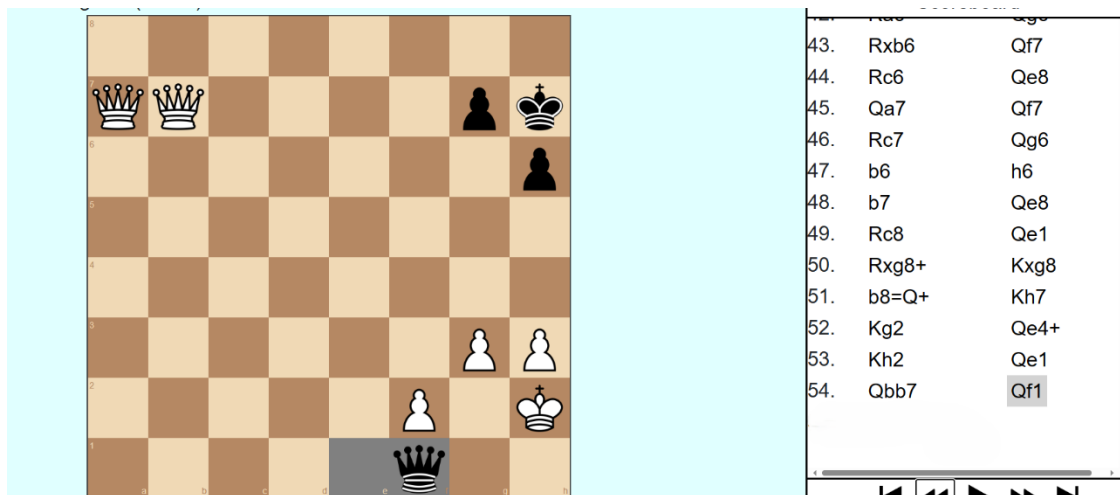


Figure 6.2.2.1.14. Black player moved the queen from e1 to f1 displayed in web application.

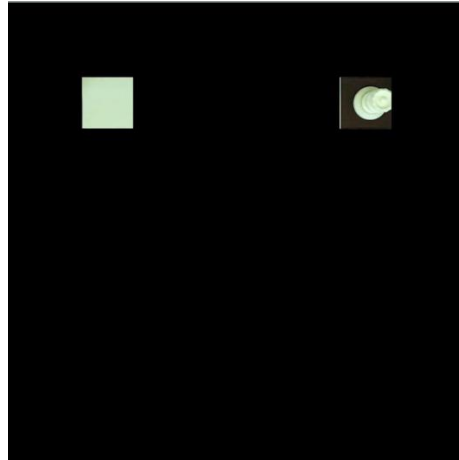


Figure 6.2.2.1.15. White player's capturing move detected by Raspberry Pi.

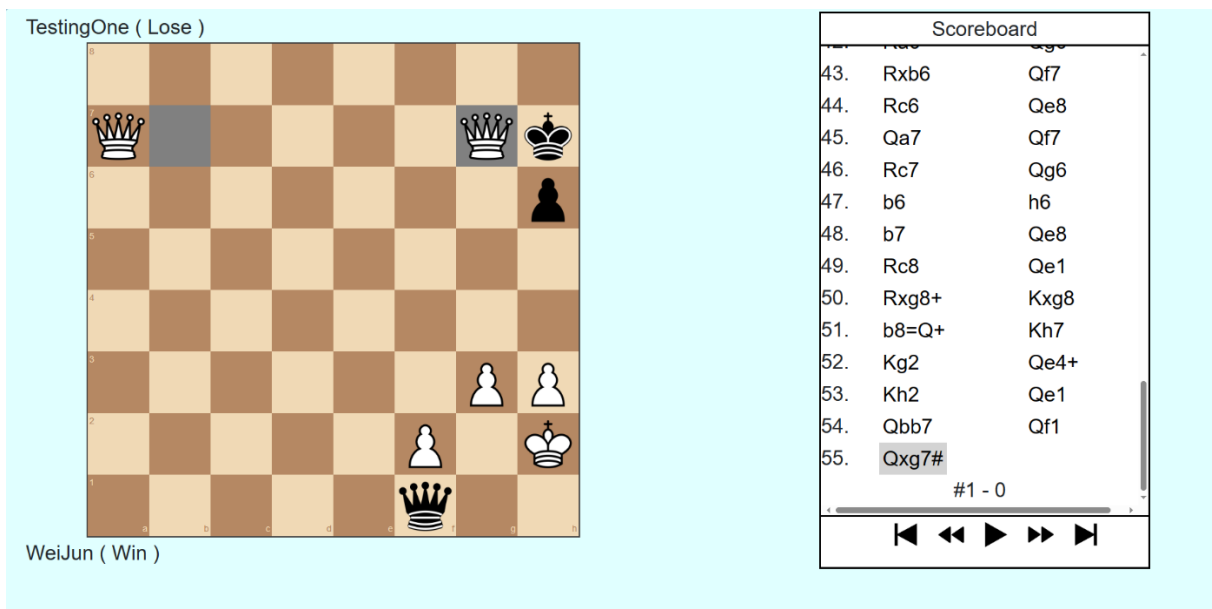


Figure 6.2.2.1.16. White player's latest move and result of the game displayed correctly in the web application.

As a summary for the first test case, the physical chess game tracking system are able to successfully detect piece movements, capturing move, castling move, pawn promotion move, and declaring checkmate for the player that played the latest move and displaying the moves correctly in the web application.



### 6.2.2.2 Result for Second Test Case

In testing the second test case, a game of 10 moves for both white and black player is played to test the system is capable of detecting resignation declaration by player, specifically in this test case, black resignation, resulting in white winning the game. In order to test the second test case, both king has to be placed on the light squares, e4 and d5 squares. As shown in Figure 6.2.2.2.1, the chess game state is currently in the 10<sup>th</sup> move for both players.

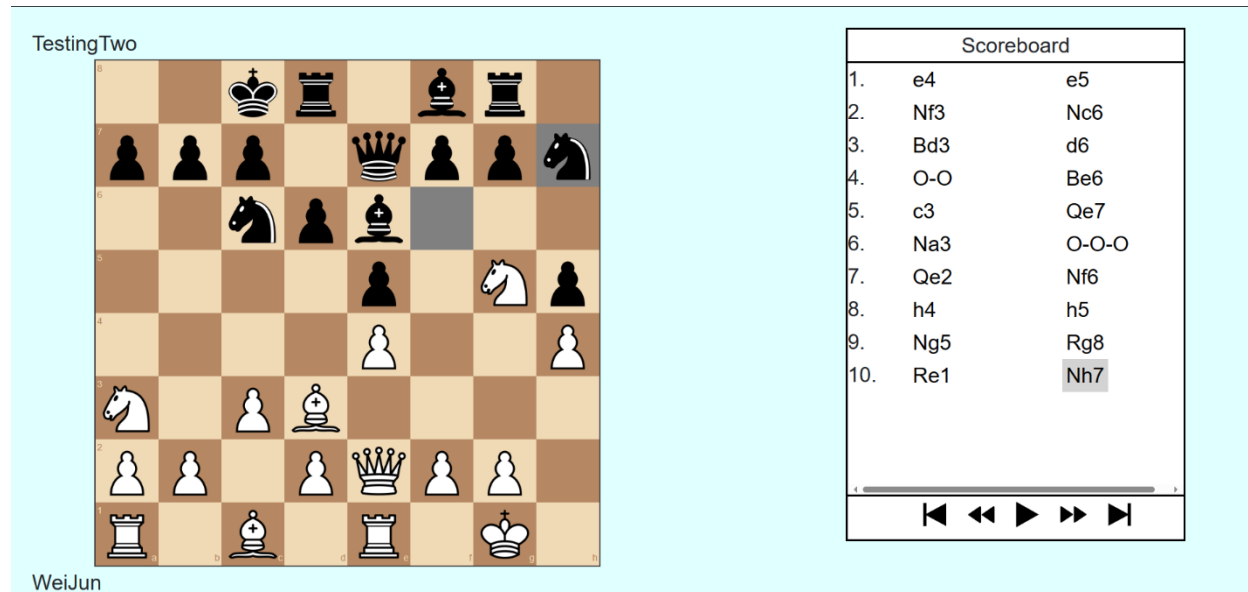


Figure 6.2.2.2.1. Chess game state for test case two shown in web application.

Now, black player decided to resign by declaring resignation and placing both the king to the e4 and d5 squares. Upon placing them on e4 and d5 squares, Raspberry Pi now detected 4 squares that have changes in them as shown in Figure 6.2.2.2.2. After model inference and calculating the source square and destination square for both the pieces, if the destination squares is in e4 and d5, which is true in second test case, the Raspberry Pi will then sent the result to the backend of the web application to declare white had won by black resignation. If the backend of the web application received result from Raspberry Pi, it will not processes any move and immediately update the result in the database for that game. Hence, the web application now will just display the result in the chess game page as shown in Figure 6.2.2.2.3 that white is victorious.

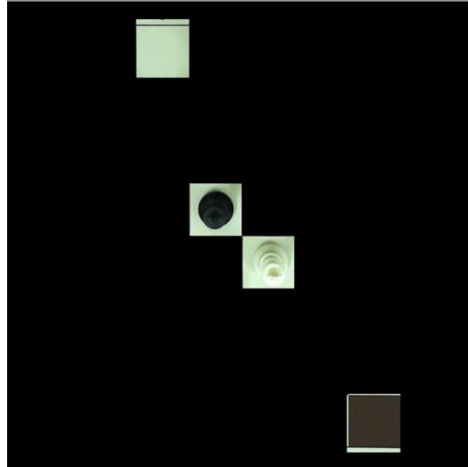


Figure 6.2.2.2.2. Both kings is detected to be moved from their source square to the center of the board on light squares by Raspberry Pi.

TestingTwo ( Lose )

WeiJun ( Win )

Scoreboard		
1.	e4	e5
2.	Nf3	Nc6
3.	Bd3	d6
4.	O-O	Be6
5.	c3	Qe7
6.	Na3	O-O-O
7.	Qe2	Nf6
8.	h4	h5
9.	Ng5	Rg8
10.	Re1	Nh7
#1 - 0		

Figure 6.2.2.2.3. Result displayed in the web application that white is victorious.

In conclusion for the second test case, the physical chess game tracking system is able to detect black resignation that results in white being victorious in any state of the game. Additionally, the web application is also able to display the result correctly in the chess game page by adding the result to the side of the player names and in the scoreboard on the right.

### 6.2.2.3 Result for Third Test Case

In testing the third test case, a game with just 2 moves for both players is played to create a condition for the player to play the special move, en passant. Additionally, after the move being played, white resigns. As shown in Figure 6.2.2.3.1, the chess game state enables the white player to play en passant on the next move.

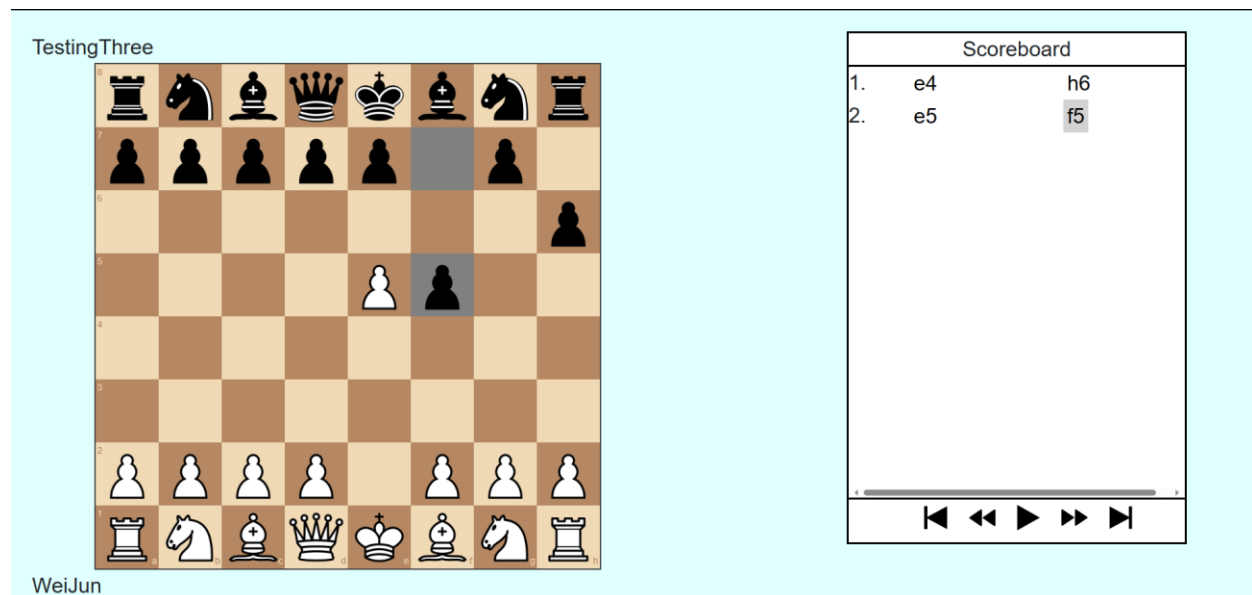


Figure 6.2.2.3.1. Chess game state that enable white to plays en passant shown in web application.

When the white player played en passant on the next move, Raspberry Pi now detected 3 squares that have changes in them as shown in Figure 6.2.2.3.2. The reason the Raspberry Pi detected 3 squares is because one square is for the source square of the white pawn, another one square is for the destination square and the final square is for the square black pawn being captured. After Raspberry Pi sent the detected move to the web application, the web application successfully displayed en passant move played by white player as shown in Figure 6.2.2.3.3.

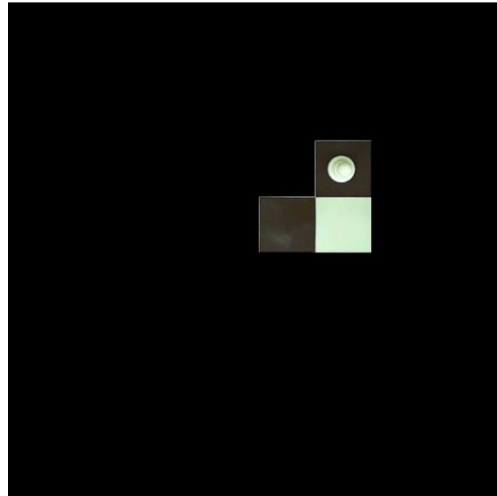


Figure 6.2.2.3.2. En passant move detected by Raspberry Pi.

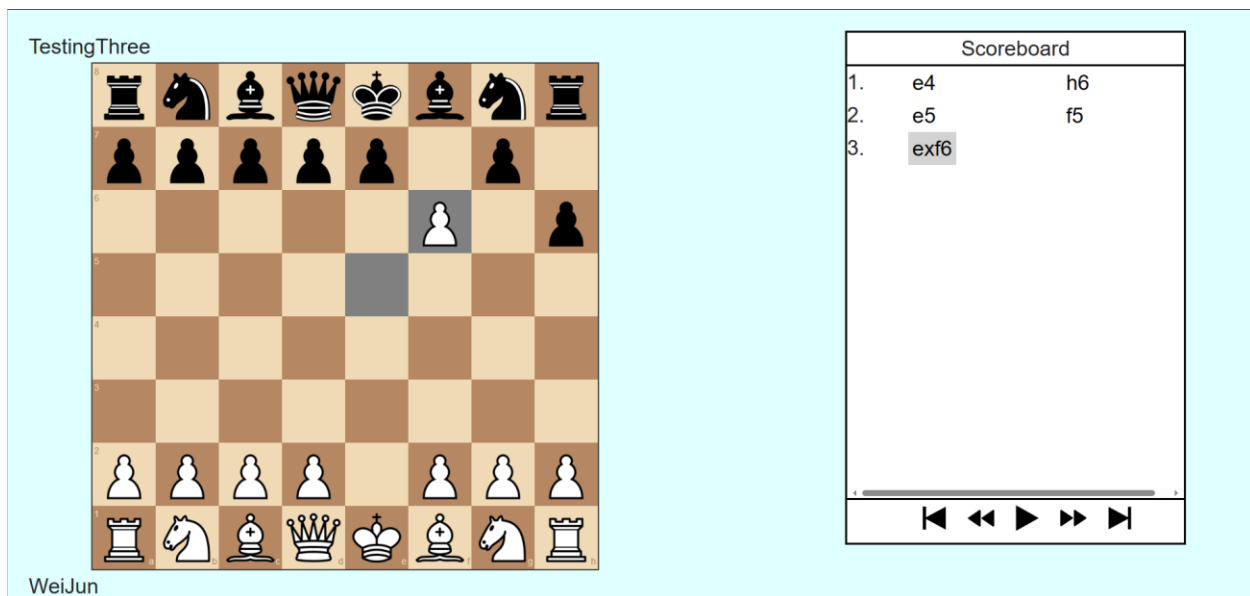


Figure 6.2.2.3.3. En passant move by white displayed correctly in the web application.

After the en passant move played by white, white decided to resign by declaring resignation and placing both kings in the center of the board, specifically, the dark squares d4 and e5. After placing both kings to the squares, Raspberry Pi now detected the kings movement to the center of the board in d4 and e5 squares as shown in Figure 6.2.2.3.4 as 4 squares with changes in them. After model inference and calculating the source square and destination square for both the pieces, if the destination squares is in d4 and e5, which is true in third test case, the Raspberry Pi will then

sent the result to the backend of the web application to declare black had won by white resignation. Upon receiving the result from Raspberry Pi, the backend of the web application will immediately update the database, and the updated data will send to the frontend of the web application. Hence, the web application will now just display the result in the chess game page as shown in Figure 6.2.2.3.5 that black is victorious.

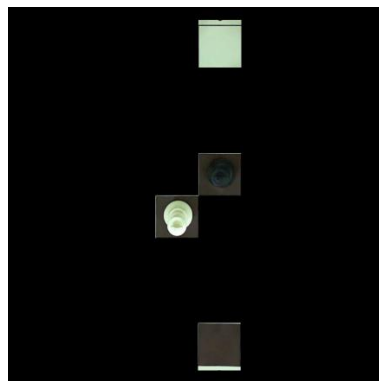


Figure 6.2.2.3.4 Both kings is detected to be moved from their source square to the center of the board on dark squares by Raspberry Pi.

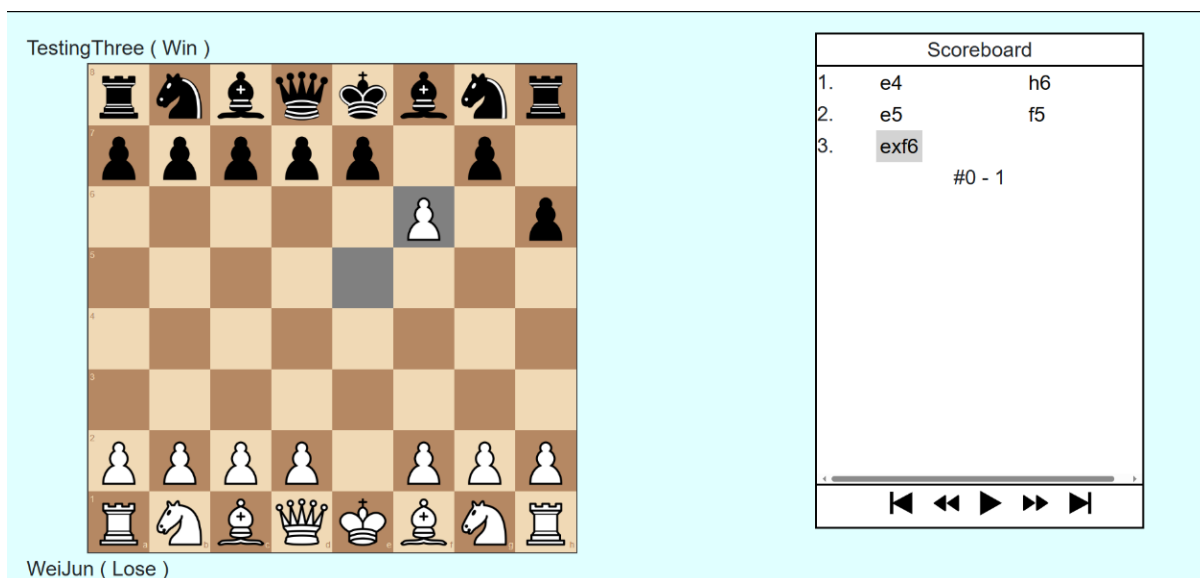


Figure 6.2.2.3.5. Result displayed in the web application that black is victorious.

In summary for the third test case, the physical chess game tracking system is able to detect en passant move played by players and white resignation at any given state of the game successfully, and the web application is also able to display the move and result correctly.

### 6.2.2.4 Result for Fourth Test Case

In testing the fourth test case, there is no move being played for the game. In other words, the game is in its starting position as shown in Figure 6.2.2.4.1.

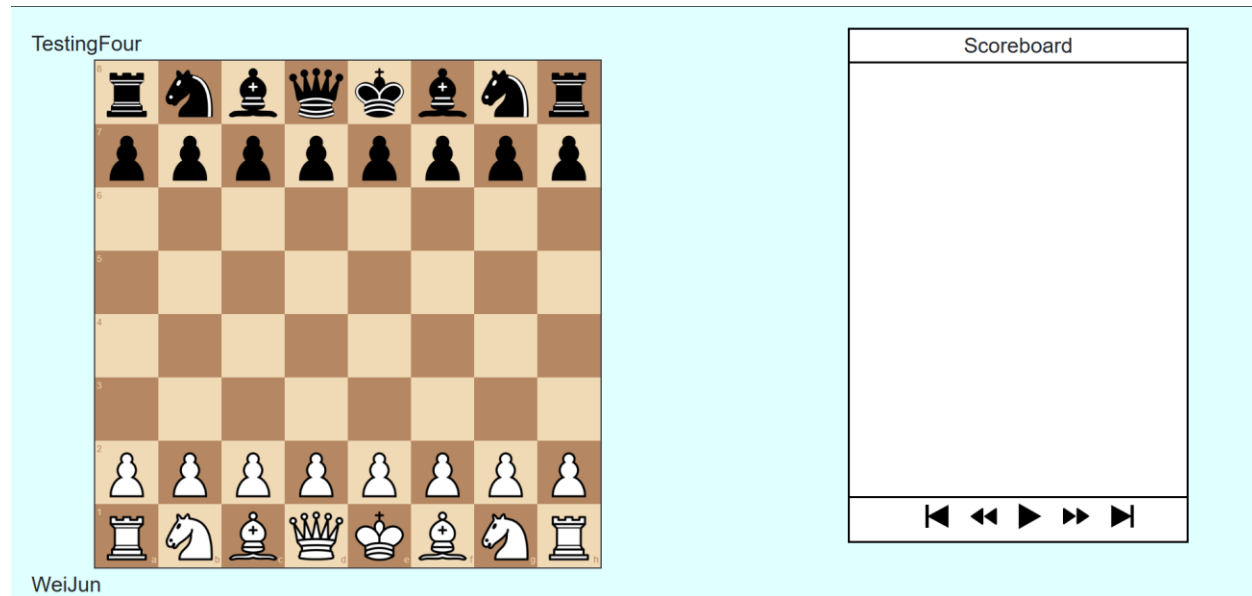


Figure 6.2.2.4.1 Starting position of the game in fourth test case.

When both the white and black players decided to draw by agreement, they placed their kings to the center of the board, specifically on the e4 and d5 squares. After placing their kings in the center of the board, Raspberry Pi now detected 4 squares that had changes in them as shown in Figure 6.2.2.4.2. After model inference and calculating the source square and destination square for both the pieces, if the destination squares is in e4 and e5, which is true in fourth test case, the Raspberry Pi will then sent the result to the backend of the web application to declare both players had drew by agreement. Upon receiving the result from Raspberry Pi, the backend of the web application will immediately update the database, and the updated data will send to the frontend of the web application. Hence, the web application will now just display the result in the chess game page as shown in Figure 6.2.2.4.3 that both players had drew.

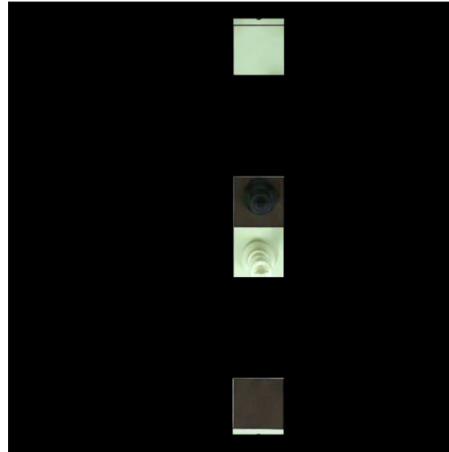


Figure 6.2.2.4.2. Both kings is detected to be moved from their source square to the center of the board on light and dark squares by Raspberry Pi.

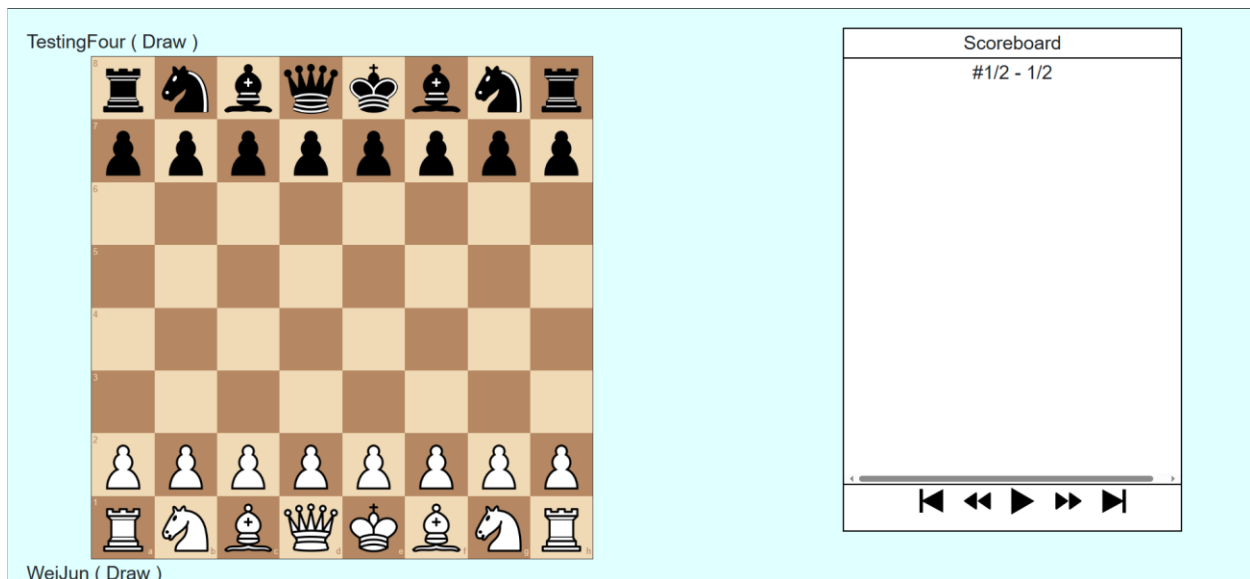


Figure 6.2.2.4.3. Result displayed in the web application that both players had drew.

In a nutshell, the fourth test case shows that the physical chess game system is capable of detecting draw by agreement declaration between both players and the web application is successful in displaying the result the shows both players had drew by agreement.

### 6.2.3 Error Analysis

There is no perfect system in the world, so as the system implemented in this project. During system testing, errors had occurred in term of detecting extra squares that are not supposed to be detected. The errors is caused by the shadow of the pieces with higher height such as the queen and the king in an environment where the light setting will cast the shadow of the pieces with higher height to the other squares. Hence, when comparing the difference between the reference image and the latest snapshot image using absdiff(), the shadow is considered changes between the images. For example, with reference to Figure 6.2.3.1, Raspberry Pi detected 3 squares that contain changes in term when the black queen is moved from f6 square to g6 square. Due to the height of the queen, the shadow of it will be casted to the square where the black pawn is on. Hence, Raspberry Pi mistaken it as changes in the square and segmented the square the pawn is on which results in error in detecting the correct queen move. The same error also occurred in Figure 6.2.3.2.

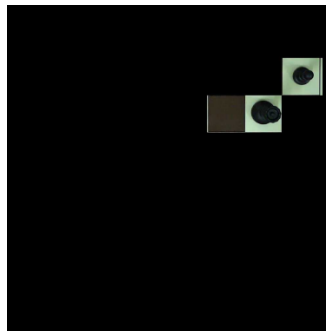


Figure 6.2.3.1. Raspberry Pi mistakenly detected 3 squares where it should detect 2 squares only.

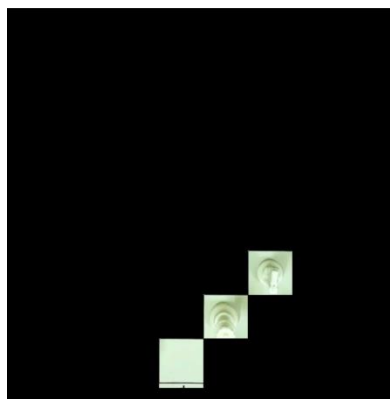


Figure 6.2.3.2. Raspberry Pi mistakenly detected 3 squares where it should detect 2 squares only.



As for the error in detecting the moves as shown in Figure 6.2.3.3, when the white queen moved from b7 square to g7 square, Raspberry Pi also detected 3 squares with changes. In this case, the error is caused by the residue shadow in the reference image. Hence, when the queen moved from b7 to g7, the shadow of white queen that casted to the top left square in the latest snapshot is now gone. So in comparing both images, the residue shadow is considered as changes in that square, resulting in Raspberry Pi considering that square being one of the square that contain significant changes.

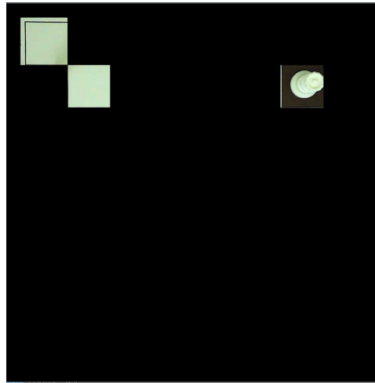


Figure 6.2.3.3. Raspberry Pi mistakenly detected 3 squares where it should detect 2 squares only.

### 6.3 Objectives Evaluation

With reference to all the results in previous sub-chapters, a novel and alternative way in tracking, saving and potentially broadcasting the physical chess game on the web application using Raspberry Pi is proven to be highly plausible. Aside from that, a YOLOX deep learning model is also trained to detect and recognize chess pieces and deployed on Raspberry Pi. Furthermore, a web application that is capable of displaying real-time data for the game being played and saving chess game that is hosted by AWS Amplify is also built. In an optimal environment, the physical chess game system is capable of tracking a physically played chess game from the start to the end without error and without intervention.

## CHAPTER 7

### CONCLUSION AND RECOMMENDATION

#### 7.1 Conclusion

Due to increasing chess players around the world, chess tournament and competition is becoming more prevalent in this modern era. Additionally, every chess tournament often participated by hundreds of players. Due to limited budget, it is quite challenging to track every chess game using the pricey current method, which is using electric chessboard with unique magnetic chess piece, may forces the organizer of the tournament and competition to reduce the size of the tournament and competition.

Hence, the physical chess game system is capable of significantly reduce the budget of organizing a tournament and competition using Raspberry Pi. The system installed in the Raspberry Pi is capable of detecting chessboard, chess move and classifying the chess piece accurately. Additionally, a web application that received information from the Raspberry Pi can also help the organizer to track every single game begin played in the tournament and competition in real time and potentially broadcasting to live streaming platforms.

#### 7.2 Recommendation

In the future, the system can be improved by handling all kinds of environment which no longer require an optimal environment for the system to work perfectly. These environments include dark environments, bright environments, shadows caused by light settings and many more. After the system is capable of handling all kinds of environments, the system may potentially replace the method used concurrently.

Besides, the deep learning model can also be trained using other datasets that includes all types of chess pieces that available in the market nowadays. For example, the dataset used in this project only contain white and black chess pieces, hence, the model can be trained with other datasets such as light brown and dark brown chess pieces.

## REFERENCES

- [1] H. M. Ahmad and A. Rahimi, "Deep learning methods for object detection in Smart Manufacturing: A Survey," *Journal of Manufacturing Systems*, vol. 64, pp. 181–196, Jun. 2022. doi:10.1016/j.jmsy.2022.06.011
- [2] Y. Xie, G. Tang and W. Hoff, "Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN," 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 2018, pp. 2001-2009, doi: 10.1109/WACV.2018.00221.
- [3] Y. Xie, G. Tang, and W. Hoff, "Geometry-based populated chessboard recognition," in Tenth International Conference on Machine Vision (ICMV 2017), International Society for Optics and Photonics, 2017.
- [4] C. Belshe, Chess piece detection - cal poly, <https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1617&context=eesp>.
- [5] Y. -A. Wei, T. -W. Huang, H. -T. Chen and J. Liu, "Chess recognition from a single depth image," 2017 IEEE International Conference on Multimedia and Expo (ICME), Hong Kong, China, 2017, pp. 931-936, doi: 10.1109/ICME.2017.8019453.
- [6] C. Matuszek et al., "Gambit: An autonomous chess-playing robotic system," 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 4291-4297, doi: 10.1109/ICRA.2011.5980528.
- [7] C. Koray and E. Sumer, "A computer vision system for chess game tracking," Semantic Scholar, <https://www.semanticscholar.org/paper/A-Computer-Vision-System-for-Chess-Game-Tracking-Koray-S%C3%BCmer/2dd2bcaee6de2c78da93153af879395ea3a8da9>.
- [8] G. Ranganathan, "An economical robotic arm for playing chess using visual servoing," Research Gate, [https://www.researchgate.net/publication/343232020\\_An\\_Economical\\_Robotic\\_Arm\\_for\\_Playing\\_Chess\\_Using\\_Visual\\_Servoing](https://www.researchgate.net/publication/343232020_An_Economical_Robotic_Arm_for_Playing_Chess_Using_Visual_Servoing).
- [9] A. Indreswaran, "Chess piece recognition using machine learning technique," Academia.edu, [https://www.academia.edu/22168310/Chess\\_Piece\\_Recognition\\_using\\_Machine\\_Learning\\_Technique](https://www.academia.edu/22168310/Chess_Piece_Recognition_using_Machine_Learning_Technique).
- [10] A. Mehta, "Augmented reality chess analyzer (archessanalyzer): In-device inference of physical chess game positions through board segmentation and piece recognition using Convolutional Neural Network," arXiv.org, <https://arxiv.org/abs/2009.01649>.

## REFERENCES

- [11] M. A. Czyzewski, A. Laskowski, and S. Wasik, "Chessboard and chess piece recognition with the support of Neural Networks," Semantic Scholar, <https://www.semanticscholar.org/paper/Chessboard-and-Chess-Piece-Recognition-With-the-of-Czyzewski-Laskowski/93ef9ae4cb111c78344a7ef0c408d47bdd1dee17>.
- [12] G. Petkov, "Tracking and annotating a chess game - University of Edinburgh," University of Edinburgh, [https://homepages.inf.ed.ac.uk/rbf/DISSERTATIONS/ug4\\_20130500.pdf](https://homepages.inf.ed.ac.uk/rbf/DISSERTATIONS/ug4_20130500.pdf).
- [13] "E. Miscellaneous / 01. laws of chess / FIDE laws of chess taking effect from 1 January 2023 / FIDE handbook," International Chess Federation (FIDE), <https://handbook.fide.com/chapter/E012023>.
- [14] "2017-10-14-TEC chapter in Fide handbook-accepted," FIDE, [https://www.fide.com/FIDE/handbook/Standards\\_of\\_Chess\\_Equipment\\_and\\_tournament\\_venue.pdf](https://www.fide.com/FIDE/handbook/Standards_of_Chess_Equipment_and_tournament_venue.pdf).
- [15] "Chess notation & algebraic notation," Chess.com, <https://www.chess.com/terms/chess-notation>.
- [16] Bochkovskiy, Alexey, et al. "YOLOv4: Optimal Speed and Accuracy of Object Detection". 2020. [https://csucalpoly.primo.exlibrisgroup.com/permalink/01CALPSU/hls1s0/cdi\\_arxiv\\_primary\\_2004\\_10934](https://csucalpoly.primo.exlibrisgroup.com/permalink/01CALPSU/hls1s0/cdi_arxiv_primary_2004_10934)
- [17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9, 2015.
- [18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.
- [19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.
- [20] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
- [21] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in BigLearn, NIPS Workshop, 2011.
- [22] Raspberry Pi, "Raspberry pi 4 model B specifications," Raspberry Pi, <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
- [23] "About," OpenCV, <https://opencv.org/about/>.

## REFERENCES

- [24] “Papers with code - yolox explained,” Explained | Papers With Code, <https://paperswithcode.com/method/yolox>.
- [25] “Top 10 open-source NoSQL databases in 2020,” GeeksforGeeks, <https://www.geeksforgeeks.org/top-10-open-source-nosql-databases-in-2020/>.
- [26] “Chess ratings,” Chessratings.top, <https://www.chessratings.top/>.
- [27]R. P. Ltd, “Raspberry Pi Camera Module 3,” Raspberry Pi. <https://www.raspberrypi.com/products/camera-module-3/>

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Semester 3</b>	<b>Study week no.: 2</b>
<b>Student Name &amp; ID: Yee Wei Jun 20ACB01647</b>	
<b>Supervisor: Dr Teoh Shen Khang</b>	
<b>Project Title: Physical Chess Game Tracking Using Raspberry Pi</b>	

## 1. WORK DONE

The work done in the last fortnight is exploring the available free web application hosting services on the internet. In the end, I chose Amazon Web Services(AWS). After that, I explored the services provided by AWS and decided on using AWS Amplify to host the web application about to be built. Experiments also done on AWS Amplify by building a simple web application first and host it on Amplify. After that, the web application is now used as an template to build the web application required in this project . After some time, the web application is built completely.

## 2. WORK TO BE DONE

For the next 2 weeks, the backend of the web application will be built and hosted in AWS.

## 3. PROBLEMS ENCOUNTERED

There is problem in configuring AWS Amplify, but the problem is resolved after referring to the documentation of it and solutions found online.

## 4. SELF EVALUATION OF THE PROGRESS

The current progress is going smoothly.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Year 3 Semester 3	Study week no.: 4
Student Name & ID: Yee Wei Jun 20ACB01647	
Supervisor: Dr Teoh Shen Khang	
Project Title: Physical Chess Game Tracking Using Raspberry Pi	

## 1. WORK DONE

The work done in the last fortnight is building the backend environment for the web application using AWS Lambda. After some days, the backend environment for the web application is now done.

## 2. WORK TO BE DONE

Build a database using AWS DynamoDB and test end-to-end of the web application

## 3. PROBLEMS ENCOUNTERED

There is some logic errors encountered in building the backend environment , but after some debugging, the problem is resolved.

## 4. SELF EVALUATION OF THE PROGRESS

The current progress is going smoothly.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Semester 3</b>	<b>Study week no.: 6</b>
<b>Student Name &amp; ID: Yee Wei Jun 20ACB01647</b>	
<b>Supervisor: Dr Teoh She Khang</b>	
<b>Project Title: Physical Chess Game Tracking Using Raspberry Pi</b>	

## 1. WORK DONE

The work done for the last fortnight is successfully built a database that will store the data for the game that tracked by Raspberry Pi. End-to-end testing also conducted to ensure there is no error in the web application.

## 2. WORK TO BE DONE

Connect Raspberry Pi to allow it to send the information such as detected move from it to the backend of the web application. Additionally, FYP 2 report will also be started.

## 3. PROBLEMS ENCOUNTERED

There is no problem encountered for the last fortnight.

## 4. SELF EVALUATION OF THE PROGRESS

The current progress is going smoothly.



Supervisor's signature



Student's signature



# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Semester 3</b>	<b>Study week no.: 8</b>
<b>Student Name &amp; ID: Yee Wei Jun 20ACB01647</b>	
<b>Supervisor: Dr Teoh Shen Khang</b>	
<b>Project Title: Physical Chess Game Tracking Using Raspberry Pi</b>	

## 1. WORK DONE

The work done for the last fortnight is successfully connect the Raspberry Pi that allow it to send data to the backend environment of the web application. The detected move sent from Raspberry Pi is successfully processed, saved to database and displayed on the frontend of the web application. Content also added to the FYP 2 report.

## 2. WORK TO BE DONE

Improves the chess game tracking system in raspberry pi and solve some unforeseen problem raised. Additionally, content will also be added to the FYP 2 report.

## 3. PROBLEMS ENCOUNTERED

There is no problem encountered for the last fortnight.

## 4. SELF EVALUATION OF THE PROGRESS

The current progress is going smoothly.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Semester 3</b>	<b>Study week no.: 10</b>
<b>Student Name &amp; ID: Yee Wei Jun 20ACB01647</b>	
<b>Supervisor: Dr Teoh Shen Khang</b>	
<b>Project Title: Physical Chess Game Tracking Using Raspberry Pi</b>	

## 1. WORK DONE

The work done for the last fortnight is successfully improve some of the component of the detection system that increased the detection accuracy for the system. Additional content also added to the FYP 2 report.

## 2. WORK TO BE DONE

Perform system testing and continue to improve the chess game tracking system in Raspberry Pi. Additionally, new content will also be added to FYP 2 report.

## 3. PROBLEMS ENCOUNTERED

There is no problem encountered for the last fortnight.

## 4. SELF EVALUATION OF THE PROGRESS

The current progress is going smoothly.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Semester 3</b>	<b>Study week no.: 12</b>
<b>Student Name &amp; ID: Yee Wei Jun 20ACB01647</b>	
<b>Supervisor: Dr Teoh Shen Khang</b>	
<b>Project Title: Physical Chess Game Tracking Using Raspberry Pi</b>	

## 1. WORK DONE

The work done for the last fortnight is successfully performed system testing from the side of Raspberry Pi to the side of web application. Additional content also added to the FYP 2 report.

## 2. WORK TO BE DONE

Record demo video that shows process from start to end of the entire system. The FYP 2 report will also be finalized.

## 3. PROBLEMS ENCOUNTERED

There is no problem encountered for the last fortnight.

## 4. SELF EVALUATION OF THE PROGRESS

The current progress is going smoothly.



Supervisor's signature



Student's signature

# POSTER



UNIVERSITI TUNKU ABDUL RAHMAN  
FACULTY OF INFORMATION AND COMMUNICATION  
TECHNOLOGY

Student : Yee Wei Jun  
Supervisor : Dr. Teoh Shen Khang

## Physical Chess Game Tracking Using Raspberry Pi

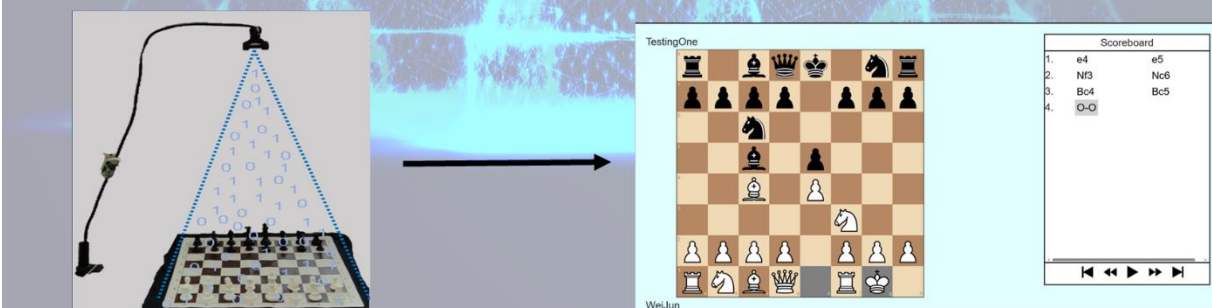
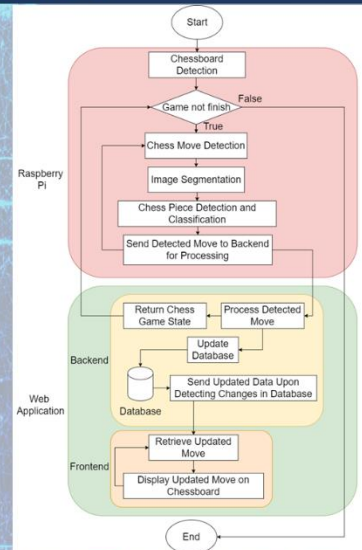
### Description

Due to increasing chess players in the world, chess tournament organizers are short of budget to keep track of every chess game with the current method, which is electrical board and uniquely manufactured chess piece. Hence a novel and alternative method is proposed, providing the organizers a better and effective way in tracking every game.

### Objectives

- Provide a novel and alternative way in tracking, saving and potentially broadcasting physical chess game on web application using Raspberry Pi.
- Train a model that is capable of recognizing chess pieces that will be deployed on Raspberry Pi.
- Tracking a physically played chess game from the start to the end without error and without human intervention
- Construct a web application that capable of displaying real-time data of the game and saving physical chess game to database.

### Method



## PLAGIARISM CHECK RESULT

20ACB01647\_FYP2.docx

### ORIGINALITY REPORT

**11**%

SIMILARITY INDEX

**9**%

INTERNET SOURCES

**4**%

PUBLICATIONS

**5**%

STUDENT PAPERS

### PRIMARY SOURCES

<b>1</b>	<a href="http://eprints.utar.edu.my">eprints.utar.edu.my</a> Internet Source	<b>1</b> %
<b>2</b>	<a href="http://docplayer.net">docplayer.net</a> Internet Source	<b>1</b> %
<b>3</b>	Submitted to Universiti Tunku Abdul Rahman Student Paper	<b>1</b> %
<b>4</b>	<a href="http://arxiv.org">arxiv.org</a> Internet Source	<b>1</b> %
<b>5</b>	<a href="http://digitalcommons.calpoly.edu">digitalcommons.calpoly.edu</a> Internet Source	<b>1</b> %
<b>6</b>	<a href="http://par.nsf.gov">par.nsf.gov</a> Internet Source	<b>1</b> %
<b>7</b>	Submitted to University of Strathclyde Student Paper	<b>&lt;1</b> %
<b>8</b>	<a href="http://myassignmenthelp.com">myassignmenthelp.com</a> Internet Source	<b>&lt;1</b> %
<b>9</b>	<a href="http://www.chess.com">www.chess.com</a> Internet Source	<b>&lt;1</b> %

<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	Yee Wei Jun
<b>ID Number(s)</b>	20ACB01647
<b>Programme / Course</b>	Bachelor of Computer Science
<b>Title of Final Year Project</b>	Physical Chess Game Tracking Using Raspberry Pi

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index: <u>11</u> %</b>  <b>Similarity by source</b> Internet Sources: <u>9</u> % Publications: <u>4</u> % Student Papers: <u>5</u> %	
<b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>	
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

*Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.*

\_\_\_\_\_  
Signature of Supervisor

Name: Dr. Teoh Shen Khang

Date: 25 April 2024

\_\_\_\_\_  
Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_



## UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

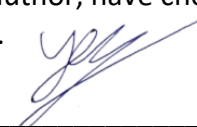
### CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB01647
Student Name	Yee Wei Jun
Supervisor Name	Dr. Teoh Shen Khang

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

  
\_\_\_\_\_  
(Signature of Student)

Date: 25<sup>th</sup> April 2024