Interactive Robot Chess

CS39440 Major Project Report

Author: Your Name ([ibo1@aber.ac.uk](mailto:ibo1@aber.ac.uk))

Supervisor: Dr Maxim Buzdalov ([mab168@aber.ac.uk](mailto:mab168@aber.ac.uk))

5th May 2023

Version 2.0 (Release)

This report is submitted as partial fulfilment of a BSc degree in  
Artificial Intelligence and Robotics (with integrated year in industry) (GH7P)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name Omar Ibrahim

Date 5th May 2023

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Omar Ibrahim

Date 5th May 2023

Acknowledgements

I express my gratitude to the following individuals for their invaluable support and assistance during the course of this project:

Firstly, I extend my sincere appreciation to my supervisor, Dr. Maxim Buzdalov, for his unwavering availability, technical expertise, and insightful guidance that proved invaluable in facilitating my acquisition of new technologies and understanding of unfamiliar subjects.

I am also grateful for the technical support of Dr. Patricia Shaw, who provided expert guidance on ROS-related topics that were not covered in the modules, and saved me valuable time that would have been wasted on extensive research.

My family deserves my heartfelt appreciation for their unwavering support throughout my academic journey, and for their unwavering encouragement to always strive for excellence and doing my best, even during challenging times.

I am indebted to my close friends, Leon, Soheb, Frohar, Fernando, and Craig, for their attentive listening, unwavering support, and for providing an outlet to voice my concerns and frustrations.

I also express my gratitude to my dear friends Martin and Julia for their steadfast belief in my ability to complete this project.

Finally, I would like to extend my gratitude to Professor Neil Taylor for his availability to address my concerns and for supporting my decision to undertake this project independently.

Abstract

The Interactive Robot Chess project is a system that allows human chess players to play against a robotic arm. The project employs a two-component approach consisting of a user interface that runs on devices supporting the JAVA language and a robotic system developed in ROS utilizing C++, which runs on Linux. A networking protocol using TCP was established to ensure effective communication between the two system components.

Although the project does not address all challenges that may arise from a robotic chess system, it simplifies the implementation of chess piece detection and game logic by utilizing a chess board object that handles legal and illegal move checking, castling, promotion, and check. The current system does not support check mate, and the project does not offer a solution for addressing possible security risks. The robotic system is designed modularly to be easily expandable. The user interface is built upon a core library that can be utilized in conjunctions with different frameworks such as JavaFX or Material 3 compose. The JavaFX component of this project serves as a prototype to demonstrate the practical application of a user interface framework, namely JavaFX, in implementing the functionality offered by the core library.

The project's final state allows human players to play a full game of chess directly against any chess engine supporting the Universal Chess Interface (UCI) with the exception of check mate. However, the simulation does not include promotion, check mate, or resetting the simulated world. The system is currently not deployable in the real world.

Contents

[1. Background 8](#_Toc134171740)

[1.1. Introduction 8](#_Toc134171741)

[1.2. Robotic Chess systems 8](#_Toc134171742)

[1.3. Game of Chess 9](#_Toc134171743)

[2. Analysis 10](#_Toc134171744)

[2.1. Introduction 10](#_Toc134171745)

[2.2. Interface 10](#_Toc134171746)

[2.2.1. Bluetooth 10](#_Toc134171747)

[2.2.2. Wired connections 11](#_Toc134171748)

[2.2.3. Internet 11](#_Toc134171749)

[2.2.4. Conclusion 11](#_Toc134171750)

[2.3. Robotic System 12](#_Toc134171751)

[2.3.1. Which robot arm to use? 12](#_Toc134171752)

[2.3.2. How to detect the chess pieces? 12](#_Toc134171753)

[2.3.3. Creating a chess engine or using existing once? 13](#_Toc134171754)

[2.3.4. How to deal with possible security issues? 13](#_Toc134171755)

[2.3.5. Which programming language to use? 13](#_Toc134171756)

[2.4. Game of Chess 13](#_Toc134171757)

[2.5. No upfront prototyping 13](#_Toc134171758)

[2.6. Project Objectives 14](#_Toc134171759)

[2.6.1. User Stories 14](#_Toc134171760)

[2.6.2. Functional Requirements 16](#_Toc134171761)

[2.6.3. Technical Objectives 17](#_Toc134171762)

[3. Process 18](#_Toc134171763)

[3.1. What is expected of the development process 18](#_Toc134171764)

[3.2. Chosen development process 18](#_Toc134171765)

[3.3. Implementation of Scrum 19](#_Toc134171766)

[3.3.1. Product Backlog and Sprint Backlog 19](#_Toc134171767)

[3.4. Version Control System 19](#_Toc134171768)

[4. Design & Implementation 20](#_Toc134171769)

[4.1. Introduction 20](#_Toc134171770)

[4.2. Overall Architecture 20](#_Toc134171771)

[4.2.1. Communication Protocol 20](#_Toc134171772)

[4.3. ROS Robotic System Overall Architecture 22](#_Toc134171773)

[4.3.1. Introduction 22](#_Toc134171774)

[4.3.2. Preliminary concept to detect chess pieces 22](#_Toc134171775)

[4.3.3. ROS Robotic System Initial Architecture 23](#_Toc134171776)

[4.3.4. Current ROS Robotic System Architecture 25](#_Toc134171777)

[4.4. ROS Robotic System Components 27](#_Toc134171778)

[4.4.1. Introduction 27](#_Toc134171779)

[4.4.2. Chess Engine Wrapper 27](#_Toc134171780)

[4.4.3. System State Machine 29](#_Toc134171781)

[4.4.4. Robot Arm 34](#_Toc134171782)

[4.4.5. Chess piece detection 39](#_Toc134171783)

[4.5. User Interface 48](#_Toc134171784)

[4.6. Overall architecture 48](#_Toc134171785)

[4.7. System library 49](#_Toc134171786)

[4.7.1. Introduction 49](#_Toc134171787)

[4.7.2. UML Class Diagram 49](#_Toc134171788)

[4.8. Other Implementations 52](#_Toc134171789)

[4.8.1. No Check mate implementation 52](#_Toc134171790)

[4.8.2. Playing against purely the chess engine 52](#_Toc134171791)

[4.8.3. No Promotion moves for “Full simulation” 53](#_Toc134171792)

[5. Testing 53](#_Toc134171793)

[5.1. Overall Approach to Testing 53](#_Toc134171794)

[5.2. Automated Testing 53](#_Toc134171795)

[5.2.1. Unit Tests 53](#_Toc134171796)

[5.3. Manual Testing 54](#_Toc134171797)

[5.4. Comprehensiveness of the tests 54](#_Toc134171798)

[6. Critical Evaluation 55](#_Toc134171799)

[6.1. Approach of the project 55](#_Toc134171800)

[6.1.1. Preparation and scope of the project 55](#_Toc134171801)

[6.2. Analysis 55](#_Toc134171802)

[6.2.1. Decomposition of the project 55](#_Toc134171803)

[6.2.2. Requirements 55](#_Toc134171804)

[6.3. Development Process 55](#_Toc134171805)

[6.3.1. Chosen development process 55](#_Toc134171806)

[6.3.2. User Stories 56](#_Toc134171807)

[6.4. Design and Implementation 56](#_Toc134171808)

[6.4.1. Overall architecture 56](#_Toc134171809)

[6.4.2. ROS robotic system 56](#_Toc134171810)

[6.4.3. User Interface 57](#_Toc134171811)

[6.5. Testing 57](#_Toc134171812)

[6.6. Tools 57](#_Toc134171813)

[6.7. Achievement of the aims 57](#_Toc134171814)

[6.8. Future work 58](#_Toc134171815)

[6.9. Lessons learned and how I would approach it next time 58](#_Toc134171816)

[7. Figures 59](#_Toc134171817)

[8. References 60](#_Toc134171818)

[9. Appendices 63](#_Toc134171819)

[A. Third-Party Code and Libraries 63](#_Toc134171820)

[1. Code 63](#_Toc134171821)

[a) Socket Creation on Linux 63](#_Toc134171822)

[b) bytes to integer conversion 63](#_Toc134171823)

[c) JavaFX-Online-Chess 63](#_Toc134171824)

[d) Communication with a subprocess via stdin/stdout 64](#_Toc134171825)

[e) ChessBoards for the gazebo simulation 64](#_Toc134171826)

[f) ROS kortex launch and model 64](#_Toc134171827)

[2. Libraries 65](#_Toc134171828)

[B. Code Samples 65](#_Toc134171829)

[C. Communication Protocol 66](#_Toc134171830)

[D. Internal Communication Protocol 68](#_Toc134171831)

[E. Build and Run 70](#_Toc134171832)

[F. Unit Test Table 75](#_Toc134171833)

[1. ChessBoard test table 75](#_Toc134171834)

[2. Chess engine test table 81](#_Toc134171835)

[3. DataChecker test table 82](#_Toc134171836)

[4. UCIHandler test table 83](#_Toc134171837)

[G. Manual Test Table 85](#_Toc134171838)

# Background

## Introduction

For the MMP a robotic or machine learning themed topic had to be chosen. Previous experience engaging in 6/7dof robot arm systems, and the natural interest and enjoyment of working with such robotic systems that interact with the human environment, led to the decision to theme the topic around a 6/7dof robotic arm systems.

Robotic arm systems are widely used in the industry ranging from NASA’s Curiosity Rover [1] for exoplanetary exploration to ABB’s [2] automotive robotic systems for painting vehicles in the automotive manufacturing industry. The theme of the project had a wide range of possible solutions.

## Robotic Chess systems

An initial investigation into robotic systems in the game of chess, due to an inherent interest in the game of chess, lead to a rich history dating back to the 18th century when Wolfgang von Kempelen's famous "Turk", also known as The Turk or Automaton Chess Player, chess machine wowed audiences with its ability to play chess against humans. The “Turk” was a mechanical illusion that hid a human chess master inside of it to operate it. [3]

The first true chess automation would be created in 1912 by a Spanish engineer named Leonardo Torres y Quevedo. This early fascination in creating physical world automated chess systems wouldn't last long. Due to the fact that the last substantial and ground-breaking advancement into a physical automated chess system was made in 1983. In the current day and age most automated chess systems come from DIY(Do-It-Yourself) developers. [4]

This most likely resulted from the introduction and development of chess engines, notably IBM’s Deep Blue [5] chess engine. The rise of chess engines in the 20th century marked a significant turning point in the development of automated chess. Unlike traditional chess machines, which relied on a mechanical arm to move pieces, chess engines use sophisticated software algorithms to analyse positions and make moves. This allows them to play at a level that surpasses any human. The advent of computer technology and the development of powerful algorithms, combined with the ability to easily update and improve the software, has made chess engines the dominant force in automated chess. Today, these engines are used not only to play chess but also to analyse games, teach players, and even compete against each other in specialized tournaments. This doesn’t mean that chess engines are superior just that they are preferred choice in current times.

The natural complexity and variety that can be achieved with robotic chess systems is vast, ranging from abstract chess pieces with markings in a controlled environment, such as the ROS Final Project [6] created in ROS [7], to an all-inclusive Chess-Robot [8] project that use a combination of hardware and software technologies. For instance, the Chess-Robot project is implemented using a Raspberry Pi and a user interface written in Python. However, with this complexity also come challenges, such as ensuring the safety of participants and preventing potential malfunctions that could cause harm to people or damage to property, as mentioned in a newspaper article by The Guardian that reported an incident in which a chess robot reportedly grabbed and broke the finger of a seven-year-old opponent [9] was found.

## Game of Chess

Chess is a two-player board game played on a board with 64 squares of alternating colours.

Although the origin of chess is a topic of debate, it is commonly believed to have developed from the Indian game of chaturanga. This game had two key features that later chess variants retained: varied powers of different pieces and the victory objective of protecting one piece, the king. Chess eventually reached Europe through Persia, the Byzantine Empire, and the expanding Arabian empire. The modern rules and appearance of chess pieces gradually evolved with a lot of regional variation until the 15th century, when two essential rule changes became widespread: the counsellor changed gender to become the queen, and the bishop acquired increased mobility. These changes added a dynamic new element to chess and made checkmates more frequent. Although castling and the en passant capture rules were also added, it took them longer to become widely accepted. [10]

The board is labelled vertically from a to h. Horizontally, the board is labelled from 1 to 8. Each player starts with sixteen pieces, including a king, a queen, two rooks, two knights, two bishops, and eight pawns. The objective is to checkmate the opponent's king, putting it in a position where it is under attack (in check) and there is no way to avoid capture on the next move. Each player takes turns to make a move, which involves selecting one of their pieces and moving it to a new square on the board according to its individual movement rules. In some cases, the game may end in a draw, which can occur if neither player is able to checkmate the opponent's king or if both players agree to a draw. [10]

# Analysis

## Introduction

To design a robotic chess system, it was essential to identify its fundamental parts. Two possible strategies were considered for building the system.

The first approach involved using only a robotic arm, which would make the information flow within the system problematic. It would be challenging to provide the user with accurate information about the system's state and offer viable solutions for errors. Additionally, it would be difficult to initiate a chess game, stop the game, report back on illegal moves made by the human player or change settings for the chess engine.

The alternative strategy involves creating a system with a robotic arm and a user interface for users, aka human chess players. This approach would simplify information access for users and make it easier to control specific features of the system, such as setting the AI player's difficulty level and above-mentioned examples. Moreover, such a system would enable developers to create modules that provide easy access to and control over specific functions, specifically for developers.

Based on experience working with robotic arms, both in university and industry, combined with the above-stated projects, the decision was made to develop a system with a user interface. However, to ensure that the project could be completed within the allotted time limit, the interface would be designed as a prototype rather than a fully-fledged software solution. The interface would serve as a helpful module for the entire system.

## Interface

For a robotic chess system to be effective, users must be able to communicate with it through an interface and gather information from it. The interface can be made available to human players either by running it on the same hardware as the robotic system or by using an external device. With an external device interface, users can utilise their smartphones, computers, or other hardware systems, making the interaction environment more flexible and user-friendly.

There are several possible solutions for achieving this, including technologies such as Bluetooth [10], wired connections, or the internet. Each of these technologies has its own advantages and disadvantages

### Bluetooth

Bluetooth is a low power wireless radio connection that streams data over 79 channels in the 2.4GHz unlicensed industrial, scientific and medical frequency band [11]. It is widely used in hardware and is growing market [12] in ways to link devices that need to interact with one another. Bluetooth has the main downside of still being unreliable, due its inherent radio frequency band being part of the industrial, scientific and medical zone of the RF spectrum [13].

### Wired connections

Wired connections utilizing protocols such as Recommended Standard 232(RS-232) [14], Universal Serial Bus (USB) [15] and others have a major upside of being reliable, due to their direct physical connection between two devices. The issue with wired connections is that, in general, mobile devices such as smartphones, tablets, and other portable electronics aren't designed to communicate with other systems via USB, which is the primary wired connection they support. Further a direct connection to the system would be needed constantly making it more inconvenient for the user to use.

### Internet

By using the internet for communication, it is possible to link some devices wirelessly or directly over a wired connection. Because of the environment, devices that favour wireless connections or do not support direct connections can now achieve both. The internet is widely utilised, and practically every device that may be used for communication or interfacing have built-in or easily expanded functionality for using the internet.

Further the internet provides a multitude of protocols ranging from BGP, HTTP, UDP, TCP all the way to IPsec [16].

### Conclusion

After conducting research on the possible technologies for interfacing with the robotic chess system, it was found that internet provided the best solution, offering the benefits of both direct and wireless connections for any device. However, a decision had to be made on which protocol to use, with the two main contenders being TCP and UDP.

While UDP is faster, it has the downside of potential package loss, making it less reliable for the system's requirements. As reliability is crucial for the system to function effectively, TCP was chosen as the protocol. [17]

An interest in networking and learning how to achieve this led to not exploring any 3rd party libraries as possible solutions.

Ensuring data security is paramount, and sending unencrypted data may pose a significant security risk. However, as data encryption falls outside the scope of the present project, it was deemed appropriate to defer this aspect to a future iteration of the system.

To create an interface that could be run on any kind of hardware, a software language that has high compatibility with different hardware had to be chosen.

Python [18], Java [19] and Kotlin [20] all enable the running and building of their respective compilers in all major desktop environments and android. Experience in creating user interfaces in JavaFX [21], which uses Java as a base, was the deciding factor for developing in Java.

## Robotic System

The robotic system that enables the user to play chess could be achieved in a multitude of solutions. Following problems had to be addressed:

* Which robot arm to use?
* How to detect the chess pieces?
* Creating a chess engine or using existing once?
* How to deal with possible security issues?
* Allow different system states?
* Which programming language to choose?

### Which robot arm to use?

It was of interest to develop a system that could work on a multitude of robot arms with little redevelopment of the system. To achieve this a robotic middleware would suit this interest. Robotic Operating System (ROS) [7] is such a robotic middleware. Moreover, experience in developing systems in ROS made it the only plausible solution while limiting the scope of the project.

To choose which robotic arm to use for the system the university was consulted. As there was a possibility that the developed system could be run on an actual robotic arm from the university. The university provided following two possible robot arms:

* Universal Robot UR3e [22]
* Kinova GEN3 [23]

Both robot arm specifications were investigated. As both are more than capable of achieving its goals in the system. The Kinova GEN3 was chosen due to its possibility of attaching a camera to the robotic arm for chess piece detection.

Creating a system that can function efficiently in a simulated environment is crucial, as there may be instances where running the system on an actual robotic platform may not be feasible.

In order to meet this requirement, ROS Gazebo [24] will be utilized, which enables the creation of a simulated environment for the chess board, chess pieces, and robotic arm.

### How to detect the chess pieces?

The initial investigated projects that are comparable to this project showed different ways of detecting the chess pieces. As the idea for this system is supposed to play an actual game of chess it is not feasible to create an abstraction for the chess pieces.

Detecting chess pieces on their own is quite difficult. This is a consequence of chess pieces blocking the view of others and lightning and shadows having further impact on the detection of chess pieces [25]. It would be necessary to create a module to make this challenge simpler in light of those issues and the possibility that spotting chess pieces on a board is an MMP project in and of itself.

Creating a module to allow for flexible recognition of chess pieces would be useful for future upgrades to the system.

### Creating a chess engine or using existing once?

Developing a chess engine requires a significant amount of work. To achieve success in this project, it was necessary to use a well-established chess engine. Therefore, an existing chess engine was selected to reduce the scope of the project. While one option was to create a module that uses a specific chess engine, it was an intriguing idea to explore the possibility of building a wrapper around the chess engine to play chess against a wide range of various chess engines. In order to determine the feasibility of this idea, research was conducted to investigate how chess engines interact with the outside world and whether there is a standard protocol for doing so.

During the investigation, it was discovered that the Universal Chess Interface (UCI) designed by Rudolf Huber and Stefan Meyer-Kahlen could enable such interfacing [26]. Many chess engines, such as Stockfish [27], Komodo [28] and others, utilize this protocol. This indicated that a plug and play system that could utilize different chess engines was possible.

### How to deal with possible security issues?

As previously noted in the background section, a newspaper article highlighted an incident where a 7-year-old chess player was injured by a robotic arm, underscoring the significant safety risks involved in such a system. In order to ensure the safety and well-being of any human interacting with the robotic system, object detection combined with safety shutdown features would be necessary. However, implementing such a system would considerably increase the scope of the project, and as a result, it was decided not to incorporate it in the current version. However, this critical safety feature would be taken into account during the design of the overall system, and it could be incorporated in future versions of the project.

### Which programming language to use?

Python [18] and C++ [29] development is both supported by ROS [22]. The University provided experience in developing Python on the ROS middleware. Given the prior knowledge, it was obvious to select Python as the primary programming language for the robotic system. Even though Python offered more expertise and experience, the choice to use C++ for this project was motivated by a desire to advance one's C++ skills.

## Game of Chess

To limit the scope of the project the basic variation of chess will be implemented. En pesante will not be taken into consideration for this project.

## No upfront prototyping

It was decided not to prototype any of the aforementioned modules that will constitute the system. This decision was made due to the intrinsic requirement of dealing with various new software, hardware, and coding methodologies, such as networking, among others. The development process should enable the acquisition and integration of new techniques while progressing the project simultaneously.

## Project Objectives

The preceding sections have outlined the various methods for creating the proposed system and the chosen approaches. Drawing upon these methods, user stories were generated and presented in section 2.6.1. As a result, a series of requirements have been developed to break down the high-level user stories into specific requirements.

### User Stories

The succeeding user stories have been created with the help of Figma [30]. In order to present the user stories, it is necessary to first define the different types of users involved in the system.

#### Types of Users

The system will consider 3 types of Users:

* General User
  + A General User is not defined at the point of interaction and cannot be classified as any of the following user definitions.
* User
  + A User is a user that is interacting with the system user interface and has access to the user functions of the system.
* Admin (AKA Developer)
  + An Admin is a user that has access to backend information through the user interface in addition to interacting with the system in the same way a User does.

#### User Stories for a general user

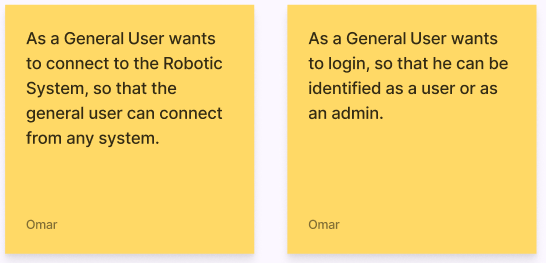


Figure 2.1 General User- User Stories

#### User Stories for User and Admin

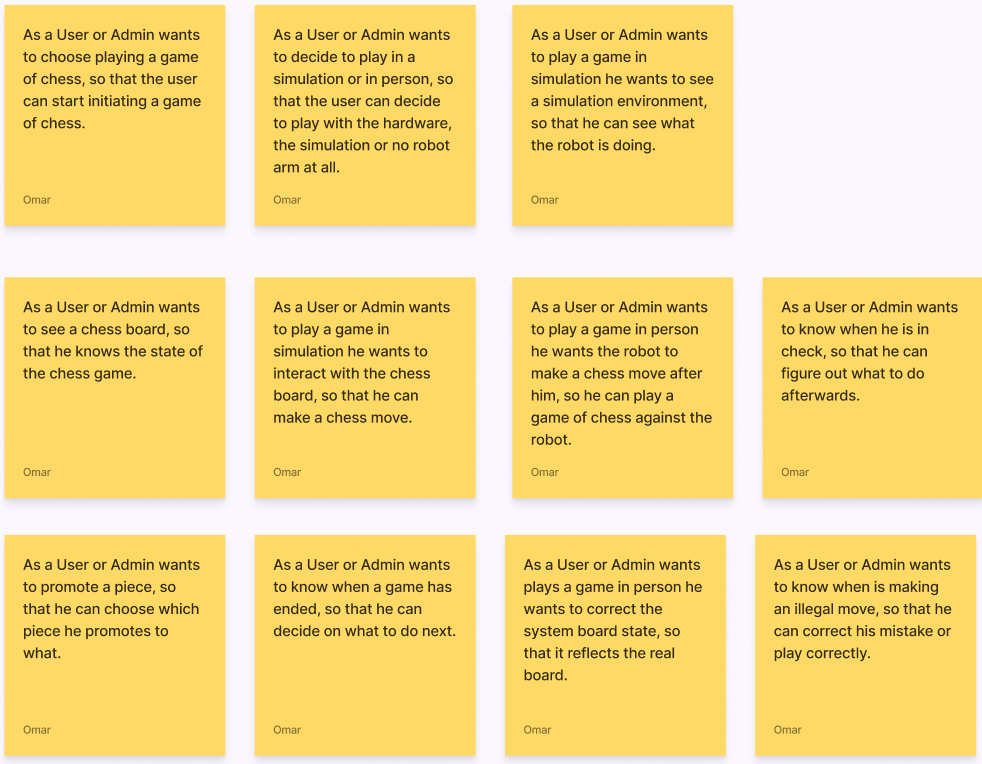


Figure 2.2 User and Admin - User Stories

#### User Stories for Admin

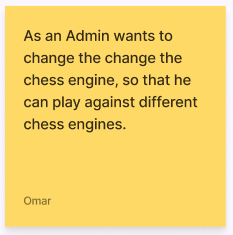


Figure 2.3 Admin - User Stories

### Functional Requirements

The functional requirements that have been identified and describe the expected interaction of a user with the system.

#### FR1 Connect to the system

A General User connects to the system via the internet, through a wired or wireless connection from different systems.

#### FR2 Loging in

A General User logs in to be identified as a User or Admin.

#### FR3 Start system in simulation

A User or Admin can set the system into a simulation state. The User or Admin will be able to view the simulation world with its robot arm, chess board and chess pieces.

#### FR3.1 Initiate a game of chess in simulation

A User or Admin can initiate a game of chess in the simulation state of the system and is able to choose which chess engine to play against.

#### FR3.2 Making a player move in simulation

A User or Admin will make a player move through the UI interface when playing in simulation. The User or Admin wants to be notified if an illegal move occurred.

#### FR3.3 Feedback of the chess board in simulation

A User or Admin wants to be prompted with a visual representation of the chess board on the user interface.

#### FR3.4 Feedback of the system

A User or Admin wants to be prompted with feedback of various types, such as an illegal move, if his king is in check and more.

#### FR4 Start system in person state

A User or Admin can set the system into a in person state.

#### FR4.1 Initiate a game of chess in person

A User or Admin can initiate a game of chess in the person state of the system. The User or Admin is able to choose which chess engine to play against.

#### FR4.2 Making a player move in person

A User or Admin wants the robot arm to make a chess move after the User or Admin made a chess move. The User or Admin wants to be notified if an illegal move occurred.

#### FR4.3 Feedback of the chess board in person

A User or Admin wants to be prompted with a visual representation of the chess board on the user interface. This enables the user to identify possible problems with the state of the robotic system, if the internal representation does not match the real-world game state.

#### FR4.4 General feedback of the system

A User or Admin wants to be prompted with general feedback of various types, such as an illegal move, if his king is in check or more, on the user interface.

#### FR5 Information about game end

A User or Admin wants to be prompted the end of a chess game, if the User or Admins king or the chess engines king is in check mate.

#### FR6 Changing the types of chess engines available

As an Admin you should be able to chess the possible chess engines that are available on the system.

### Technical Objectives

#### TO1 Opening a TCP connection

Opening a TCP connection between the user interface and the system will be required.

#### TO2 Interfacing with a multitude of chess engines

It will be important to interface with a multitude of chess engines.

#### TO3 Provide a way of switching system states

It will be important to switch between simulation and in person states. This will include different ways of handling how information is processed and chess moves are executed.

#### TO4 Identification of chess pieces

As mentioned earlier on it will be important to provide a module that does not consume a lot of work to create, to identify the chess pieces and the chess board state.

#### TO5 Modular system design approach

To ensure that upgrading or modifying specific components of the system does not require a complete redevelopment, it is crucial to construct the system using a modular approach. This is particularly essential for the user interface, which serves only as a prototype and must be readily replaceable for different devices or in the event of a redesign.

#### TO4 Documentation of the code

Given the modularity of the system and the likelihood of future expansions or adaptations, it is imperative that the code is adequately documented.

# Process

## What is expected of the development process

The development process is expected to meet following requirements:

* The ability to deliver functional software on a frequent basis
* The capacity to accommodate research on specific subjects
* The flexibility to adjust to evolving requirements, even during the late stages of development
* The use of functional software as the primary measure of progress.

The mentioned requirements are rooted in the necessity to obtain technical competencies, including the ability to network between systems, master C++ programming, and work with unfamiliar hardware, such as the Kinova GEN3. This decision was touched upon in section 2.5, where it was determined that upfront prototyping would not be included.

## Chosen development process

Agile methodologies, particularly Scrum, are well-suited for projects that require adaptability and flexibility. The ability to deliver working software frequently and adapt to changing requirements, even late in development, aligns with the needs of the process outlined in the above-mentioned section. The iterative approach of Scrum provides the opportunity to test and validate new ideas and allows for revision of the implemented ideas, which ultimately results in a better product. [31]

As Scrum is a lightweight framework that helps teams generate value through adaptive solutions for complex problems. It requires a Scrum Master, Product Owner, and Scrum Team who work together to turn a selection of work into an increment of value during a Sprint. The Scrum Team and its stakeholders then inspect the results and adjust for the next Sprint. Scrum is purposefully incomplete, only defining the parts required to implement Scrum theory, and can be adapted to work with existing practices. The rules of Scrum guide relationships and interactions, making visible the relative efficacy of current management, environment, and work techniques so that improvements can be made. [31]

As Scrum is purposefully built as an incomplete framework, it necessitated the need to define and explain each constituent part of Scrum and how it is being applied. It should be noted that Scrum is primarily designed for teams, whereas the present project is a solo endeavour, thus necessitating modifications and adaptations.

## Implementation of Scrum

Due to the solitary nature of the project, several team-oriented features of the Scrum framework, including the roles of the Product Owner and Scrum Master, have been omitted. Instead, the creation of backlog items in accordance with Scrum methodology was conducted in consultation with the project supervisor, either in advance or during the planning stage. The daily progress reviews were conducted on a less frequent basis, occurring every second to third day. Furthermore, the Sprint reviews were carried out in collaboration with the project supervisor. Sprint planning was undertaken after each Sprint and presented to the supervisor for feedback on areas that could be improved.

The sprints were flexible in terms of duration, within the constraints of a minimum of one week and a maximum of three weeks.

### Product Backlog and Sprint Backlog

In Scrum, the Product Backlog is an ordered list of all the work that needs to be done to complete a project and the Sprint Backlog is a selection of items from the Product Backlog for that Sprint [31].

The representation of the Product Backlog and Sprint Backlog offers a great deal of flexibility. As a result, Kanban [32] has been selected as the preferred approach, where the user stories are populated on the Kanban board.

For the Sprint Backlog specifically the high-level User Stories were split into Spikes and Tasks. Spikes referring to research needed to be done and Tasks referring to work that needed to be done.

The Sprint Backlog was split into 4 sections:

1. Sprint Backlog
   1. Holding the User Stories from the Product Backlog to be finished in this Sprint.
2. To Do (Tasks or Spikes)
   1. Holding the created Tasks and Spikes that still needed to be done.
3. In Process (Tasks or Spikes)
   1. Holding the Tasks and Spikes that were worked at in the moment.
4. Awaiting Review (Tasks or Spikes)
   1. Holding the Tasks and Spikes that were waiting for review and acceptance to be completed.
5. Done (Tasks or Spikes)
   1. Holding the Tasks and Spikes that were finished in this Sprint.

## Version Control System

A version control system (VCS) was utilized in the project to hold the Product and Sprint Backlog and enable the branching for Sprints to not affect the main state of the project it certain features are finished. For the VCS GitHub [33] was used. The Product and Sprint Backlog were represented as GitHub Projects [34].

# Design & Implementation

## Introduction

As elaborated in Chapter 3, the project underwent an iterative research and development process, resulting in the evolution of the design over the course of the development cycle. To provide a comprehensive account of the project's progress and current implementation, the chapter will be structured as follows:

* Firstly, an overview of the overall architecture will be presented, which remained unaltered throughout the project.
* Subsequently, the evolution of the robotic architecture will be discussed, including its initial and current state, with an explanation of the factors that led to its current configuration.
* Following that, a detailed description of the key robotic system elements will be provided.
* Next, the User Interface architecture will be examined.
* Finally, the most significant user interface modules will be inspected in detail.

## Overall Architecture

In order to ensure smooth development of the system from an architectural standpoint, it was crucial to formulate an initial concept of the overall architecture and establish how communication between the user interface and the ROS robotic system would be realized.

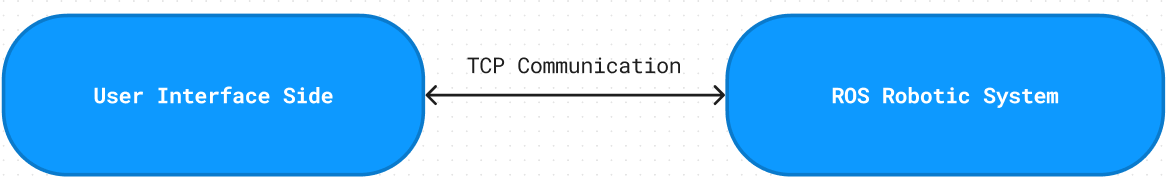


Figure 4.1 User Interface - ROS Robotic System Communication

Figure 4 illustrates the fundamental communication link between the user interface and the ROS robotic system. The implementation employs a TCP protocol agreed upon by both systems. This entailed adhering to the requirement for communication based on an Internet TCP protocol, as previously established in the Analysis chapter. The utilization of such a protocol aligns with the goal of enabling communication with various external devices, including but not limited to laptops, mobile phones, and tablets.

### Communication Protocol

In the early stages of the project, it was necessary to establish the foundation of the communication protocol, given the uncertainties regarding the amount and type of data that would be transmitted. Moreover, to guarantee the system's future-proofing, a flexible protocol was deemed necessary. As such, the following solution was developed:

#### Definition

The communication protocol adopted for the project is composed of two entities: a client and a server. While the client initiates the requests, the server is responsible for responding to these requests. It is mandatory that the server be configured on the robotic system, and the client be configured on the user interface system.

The communication protocol that has been established comprises three components:

* cmd:
  + Size: 1-byte
  + Description: Defines the intended action to be performed.
* sizeData:
  + Size: 4-byte (equivalent to int)
  + Description: Specifies the size of the data to be sent along with the command and ensures that only the required bytes are read and that the size of the data received can be accurately verified.
* Data:
  + Size: n-bytes
  + Description: Contains the actual data to be transmitted in accordance with the command.

A decision was made to utilize the cmd byte as an indicator of unsuccessful completion of the client's request. For instance, in a scenario where the user interface sends a command to execute a certain action, and the ROS system is unable to perform it, the value of the cmd byte will be utilized to indicate an error.

#### Current Protocol

The current protocol conforms to the original design of the protocol. It is attached to this document as Appendix C: A.C

## ROS Robotic System Overall Architecture

### Introduction

During the analysis phase, the components of the ROS robotic system were identified and characterized, except for the chess piece detection module. To create an initial architecture to enable smooth development a preliminary concept for simplifying the detection of chess pieces had to be developed.

### Preliminary concept to detect chess pieces

From a conceptual standpoint, it was identified that the steps of a chess game are well-defined and can be utilized to simplify the chess piece detection module. The assumption was made that a game of chess on the system would always start from the initial position. Based on these assumptions, a concept was developed to avoid the need for actual detection of the type of chess pieces.

The use of an external RGB and depth camera positioned directly above the chess board was considered as a means to detect the chess board cells and the chess pieces with their respective colours. To utilize the external camera's position data for the robot arm, it is necessary to integrate its position information into the ROS environment for the purpose of conversion. By utilizing this information, it would be possible to extract the board state of the chess board and the general position of the chess pieces in the world.

To determine the movement of a chess piece by the player, a comparison can be made between the current and previous board states of the chess board. By utilizing the colours and internal state of the chess board, the movement of a piece can be inferred. This method can be employed for legal and illegal move checking purposes. However, the issue of how to handle promotion moves arises. Since a pawn can be promoted to several pieces, this presents a challenge. To address this, an initial investigation into the frequency of promotion moves was conducted.

Due to the lack of existing research, the information used to support the decision-making process may come from sources that are not fully reliable or verified, such as Reddit. According to a Reddit post, out of 92 million examined games, only 15.5 million featured promotion moves [35]. This indicates that promotion moves occurred in only 16% of games. This was supported by another Reddit post claiming that out of 20million games examined only 4 million promotions occurred [36]. The examined games were majorly blitz games. This would equate to a promotion per 20million games of 20%.

Based on the statistics gathered, considering that promotion moves happen approximately every fifth game or even less frequently, it was determined that it is reasonable for the player to indicate the promotion move in the user interface when playing chess in the real world, compared to approaching a different design.

To pick up the chess pieces, the robotic arm would utilize the previously extracted general positions of the chess pieces to move to the designated chess piece. The arm would then use its own colour and depth sensor to grasp the piece with accuracy.

### ROS Robotic System Initial Architecture

Figure 4.2 Initial Architecture of the ROS robotic system

Figure 4.2 depicts the initial approach in designing the architecture of the robotic system. Its goal was to establish a modular, extensible, and highly adaptable system. This approach allows for targeted modifications to individual modules without necessitating a substantial overhaul of the remaining system components in the event of any future alterations.

In Figure 4.2, the grey circles symbolize the hardware and the corresponding operating environment. The arrows illustrate the flow of information.

#### System components

The main components of the system, as depicted in Figure 4.2, predominantly comprise the following components:

* **Communication API**
  + Its main responsibility is to facilitate communication with external devices by adhering to the prescribed protocol. In addition, it is tasked with the transmission of commands to the state machine.
* **System State Machine**
  + The role of the state machine is to establish a cohesive sequence of executing commands by linking all the modules together in a logical fashion.
  + This module further incorporates a safety system that facilitates the termination of system operations in the event of potential safety hazards.
* **Robot Arm State Machine**
  + The objective of the arm movement module is to grasp a chess piece and move it to a designated location.
* **Image Processing Robot Camera**
  + The responsibility of this module is to generate data that enables the robot arm to securely grasp the chess piece.
* **Chess board detection**
  + The responsibility of this module, as described in 4.2.2 is to extract the chessboard state and general chess piece position.
* **Chess Engine Wrapper**
  + The function of this module is to enable the system to engage in a game of chess by utilizing the chess engines.

Each of these system components is referred to as a ROS Node.

ROS nodes are computational processes that are interconnected through a graph structure. They communicate with each other via streaming topics, RPC services, and the Parameter Server. [37]

Furthermore, in order to maintain consistency between internal and external communication, it was determined that adherence to the external protocol would be preferable for communication among individual modules.

### Current ROS Robotic System Architecture

Figure 4.3 Current Architecture of the ROS robotic system

Figure 4.3 presents the current robotic system architecture, which differs from the initial design due to modifications in the flow of information and the absence of a module. Due to the consequence of time constraints, image processing on the robotic arm was not attempted, and hence, this module is missing in the current iteration of the architecture. Moreover, the information flow between the chess board detection module, which is responsible for processing overhead camera images, and the System state machine is absent as it is deemed unnecessary. The system state machine does not require knowledge of the chess board's detection data in its current state, and only the robot arm state machine and the target selection module need this information.

The target selection module is introduced to set and clear targets for the robot arm based on the data from the chess board detection module and the move command from the system state machine.

In the current architecture, an internal protocol has been established for communication between nodes, which differs slightly from the external protocol. This was necessary due to the differences in the internal workings of node communication from the original intent. Certain commands or executions are meant to be performed only internally, hence the need for a different protocol. The internal protocol was designed around the foundation of the external protocol and shares the similarity in certain command bytes for both internal and external commands. However, the internal protocol introduces a new component called the sender, which is used by the nodes to identify the origin of received information.

The reasoning for doing this was that in ROS there is a concept of topics that can be published and subscribed to.

A ROS topic refers to a named communication channel through which nodes can exchange messages with one another. These topics operate on a publish/subscribe model where the production and consumption of information are decoupled from one another. [38]

In order to reduce the frequency of call-backs that a single node must handle, it was determined that nodes should subscribe to their own topic and that other nodes wishing to send information to that node should publish to that topic. Therefore, a sender identifier must be included with the message to enable the recipient node to identify the source of the information.

#### Definition – Internal Protocol

* **cmd**
  + Size: 1-byte
  + Description: Defines the intended action to be performed.
* **sizeData**
  + Size: 4-byte (equivalent to int)
  + Description: Specifies the size of the data to be sent along with the command and ensures that only the required bytes are read and that the size of the data received can be accurately verified.
* **Data**
  + Size: n-bytes
  + Description: Contains the actual data to be transmitted in accordance with the command.
* **Sender**
  + Size: 1-byte
  + Description. Defines the source of the protocol.

#### Current Internal Protocol

The current protocol conforms to the original design of the protocol. It is attached to this document as Appendix D: A.D

## ROS Robotic System Components

### Introduction

This section will provide an in-depth discussion of the critical components.

### Chess Engine Wrapper

The role of the chess engine wrapper is to act as an intermediary between the robotic system and chess engines. The wrapper was designed to support a variety of chess engines, thereby enabling a plug-and-play approach. This would only be possible if a uniform interface for chess engines were available. In the analysis chapter, it was determined that the Universal Chess Interface (UCI) offers such an interface and is widely used across multiple chess engines.

#### Initial Design

In the original design, the Chess Engine Wrapper was envisioned to act as a mediator between the robotic system and a separate class responsible for managing the chess engine. The chess engine class was intended to hold the instance of the chess engine run, with the expectation that the engine would be capable of implementing illegal moves, as well as detecting check and checkmate conditions. The goal of this design was to create a modular system that could easily accommodate a variety of different chess engines, with the Chess Engine Wrapper providing a standardized interface to facilitate communication between the system and any selected engine.

By decoupling the chess engine from the rest of the system, the wrapper aimed to allow for more flexibility in selecting and integrating different engines with the robotic system. Additionally, this approach allowed for the possibility of swapping out chess engines with minimal impact on the overall system, as long as they adhered to the common interfacing provided by the Universal Chess Interface (UCI) [26].

Despite the initial design intentions, the actual implementation of the Chess Engine Wrapper underwent drastic changes as the project progressed. Insufficient research was conducted regarding the utilization of chess engines, leading to this outcome. Specifically, the way chess engines are run and that they do not implement examine legal, illegal, check and check mate conditions.

#### Current Design

In order to address this issue, the design was revised. The following class diagram only illustrates the most crucial design:

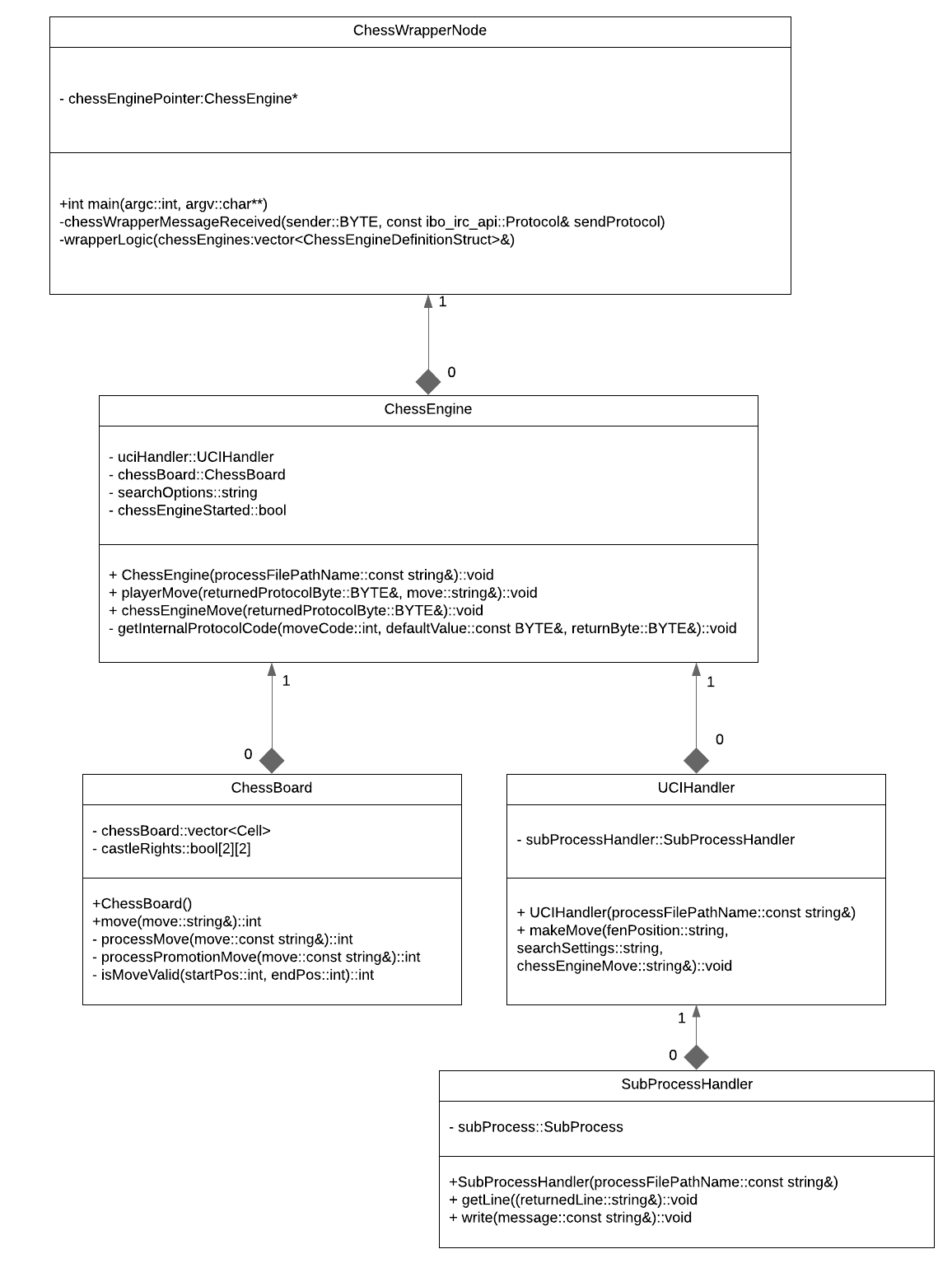


Figure 4.4 UML Class Diagram ChessWrapperNode

The present design, illustrated in Figure 4.4, was developed to address the challenges encountered during the implementation of the Chess Engine Wrapper. The ChessWrapperNode is the main component of this design, which serves as a communication interface with the internal system. It interacts with other components such as the ChessEngine, ChessBoard, UCIHandler, and SubProcessHandler. The ChessEngine is responsible for holding the instance of the ChessBoard, which represents the internal state of the chess board. The ChessBoard class encompasses the necessary functionality for checking the legality of a move, detecting check and checkmate. The ChessBoard class uses 3rd party code, this is mentioned in the ChessBoard.h file and in Appendix A.1.c). The UCIHandler is attached to the Chess Engine and used as an interface to the various chess engines that may be started.

Starting a child process was necessary to initiate a chess engine process, which does not typically build from source code. However, the lack of experience in developing with C++ and the absence of knowledge of starting processes programmatically made the task very challenging. The SubProcessHandler component was developed to handle the communication with the child process, using third-party code as mentioned in the SubProcessHandler.h file and Appendix A.1.d).

Typically, the communication between a process and an interactor is done via their stdin and stdout. Child processes created inherit the stdin and stdout file descriptor of its parent. But to ensure that the child process does not get any information from the parents stdin, and doesn’t publish information to the parents stdout a different solution was needed. In Linux, piping was utilized to redirect the standard input and output of the child process to a reader and writer to interact and communicate with the child process. The use of this design facilitates the interface between the chess engine and the rest of the system.

The present design iteration provides the capability for a user to engage in a game of chess, with the exception of checkmate and changing the difficulty of the ai player which, at present, remains unimplemented.

### System State Machine

As previously discussed in the section on the overall architecture, the system's state machine plays a vital role in coordinating the execution of commands by linking all the modules together in a logical and cohesive manner. Essentially, the state machine functions as a state machine.

#### State Machine

Due to time constraints, the development of the system state machine was short and only one iteration was completed in a relatively brief period. This is the current version of the system state machine:

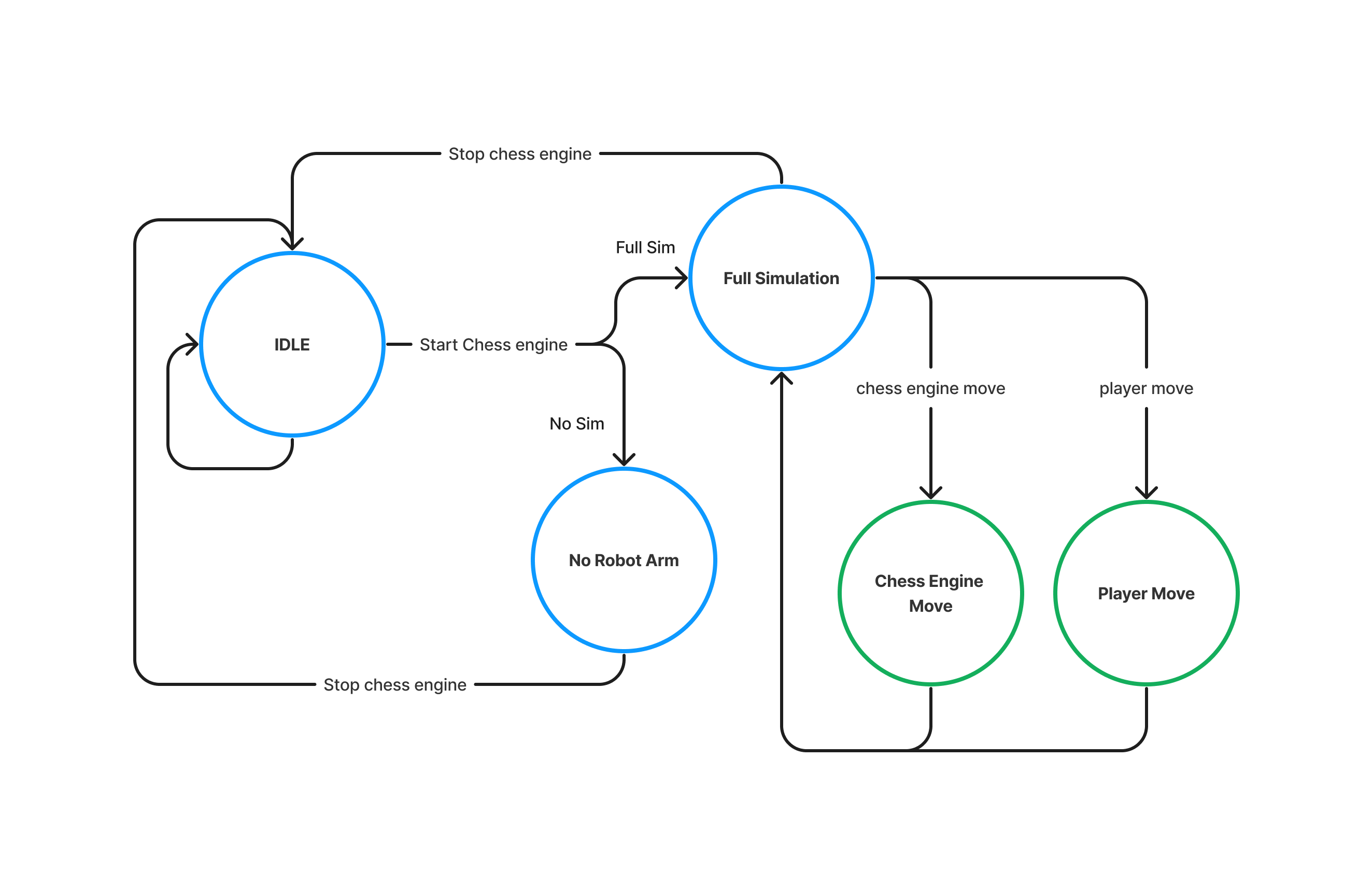


Figure 4.5 Current System State Machine

The System can currently enter 5 different states:

##### IDLE State

In the System State Machine Node, the IDLE state is responsible for receiving and forwarding messages related to changing settings and obtaining information. The current iteration of the IDLE supports following commands:

* Get chess engine names
* Start chess engine
* Set system without sim
* Set system full sim

In order to transition from the IDLE state, a "start chess engine" command must be received by the system.

Upon receiving the command, the chess engine will begin running and the system will determine which parameter the system state is currently set to. Presently, the two available options for the system state parameter are "full sim" and "no sim". It is possible to expand upon this design and include additional pathways in the state machine to accommodate for in-person use. Outside the IDLE state no other state allows the changing of parameters or obtaining this information.

##### No Robot Arm

In the "No Robot Arm" state, users are permitted to play directly against the chess engine by sending commands, such as "player move" and "chess engine move," directly to the ChessWrapperNode, which subsequently executes them. To return to the IDLE state, a "stop chess engine" command must be executed.

##### Full Simulation

In the "Full Simulation" state, users engage in a simulated game against a robot arm. This state triggers a specific simulation state machine that handles commands such as "player move," "chess engine move," and "chess engine stop" for a simulated world. However, no other command will be executed in this state. To exit the "Full Simulation" state, the chess engine stop command is invoked, which sets the system state machine back into the IDLE state.

When either the "chess engine move" or "player move" command is executed, a subsequent flow of tasks is initiated by the system. To provide a visual representation of this flow, a flowchart will be presented for both commands.

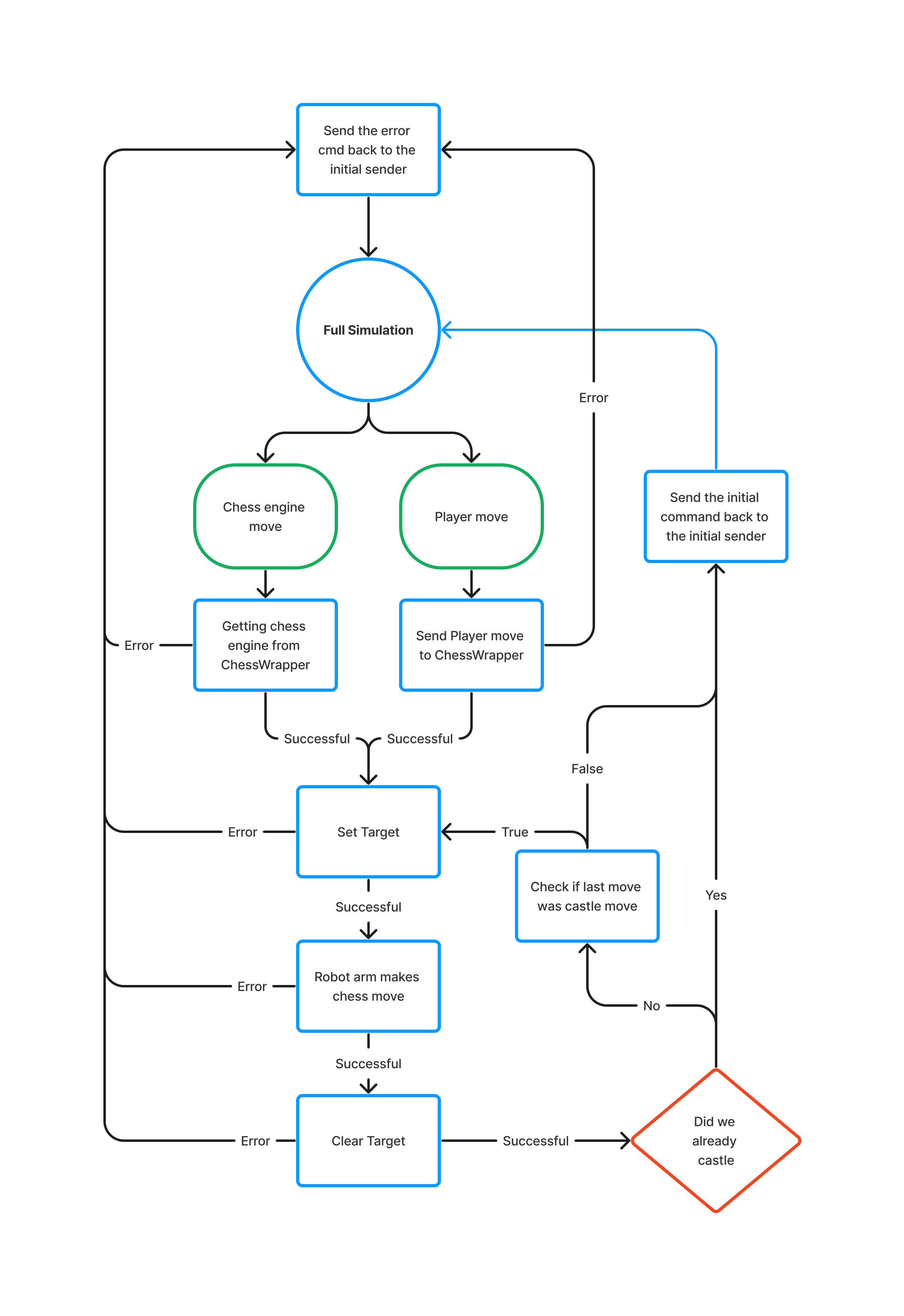
Chess engine and player move flow chart:

Figure 4.6 Chess Engine and Player move in Full Simulation Flow Chart

The provided flowchart demonstrates the sequential steps involved in executing a "chess engine" or "player" move within the context of the "Full simulation" state. The process starts by determining whether a "chess engine" or "player" move has been requested and subsequently establishes the necessary communication with the ChessWrapper. After verifying that no errors have occurred, the system sets the target for the desired move and commands the robot arm to execute it. Once the move is completed, the system clears the target and checks if a castle move was made. If a castle move was not performed, the system then checks if the last move was a castle move. The system then loops back and repeats the previous steps for the second move of a castle move. If no errors are encountered, the system sends the requested command's information back to the initial sender of the internal execution request. However, if any errors arise during the process, the system terminates the execution and returns the error message to the initial sender of the internal execution request.

#### Example frames when setting a target

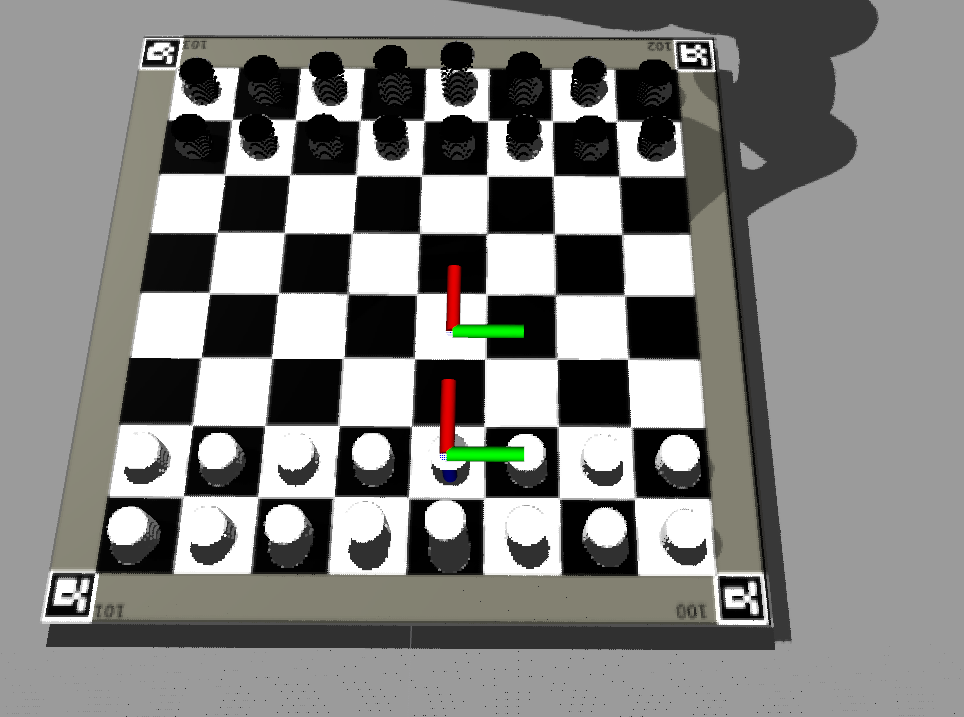


Figure 4.7 Set Target Example 1

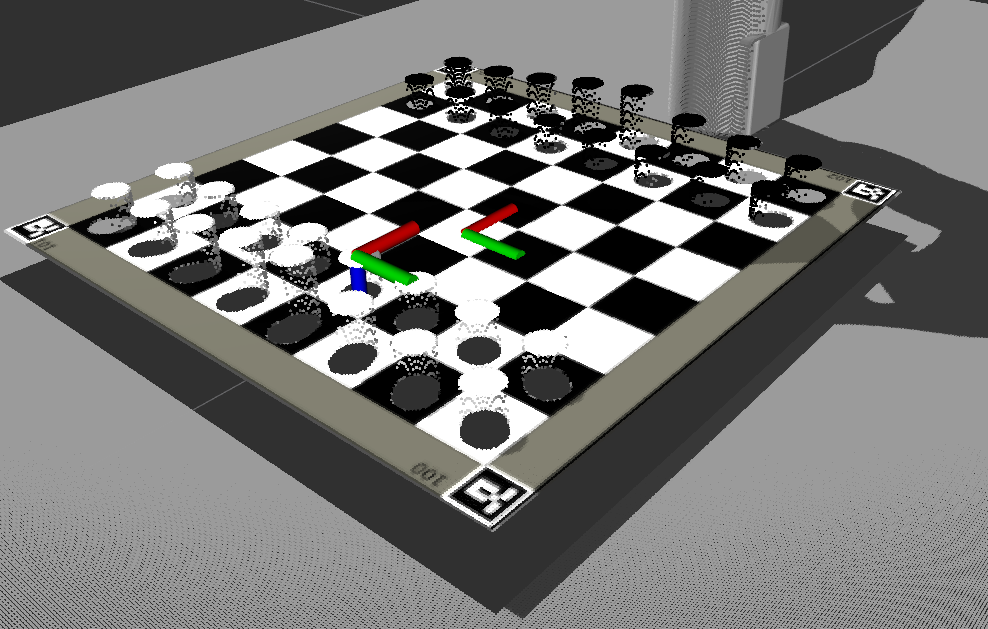


Figure 4.8 Set Target Example1 different point of view

It can be observed from Figures 4.4 and 4.5 that the target selector node generates two distinct entities: the pickup\_frame and drop\_frame. These entities are subsequently employed by the robot arm system to determine the spatial coordinates of the chess piece being retrieved and the target location to which the chess piece will be transported, respectively.

### Robot Arm

In a real-world setting, the Robot arm is tasked with executing chess movements and picking up chess pieces for the chess engine. In a "Full simulation" context, the robot arm would be responsible for both the player's and the chess engine's movements. To accomplish this, the arm relies on targets established by the TargetSelecterNode. The robot arm operates according to a primary state machine, which identifies the type of chess movement required. It then enters a pick and drop state machine specifically designed for picking up and dropping chess pieces.

Two types of chess movements must be handled: basic chess movements, in which a single chess piece moves from its original cell to its destination cell, and take movements, in which the destination cell must be cleared before the chess piece can move from its original cell to its destination cell.

The current iteration and only iteration of the overall state machine does not handle errors by the robot arm.

#### Overall state machine

Figure 4.9 Current Overall Robot Arm State Machine

##### IDLE State

Until a "robot arm move" command is received, the state machine remains in an idle state. When such a command is received, its validity is checked, along with the published information from the chess board cell detection regarding the state of the chess board. In cases where an error occurs, such as the chess board cell detection not publishing any information, or the move command being invalid, the system remains in the idle state and returns an error message to the command's initial sender.

However, if both sets of data are accurate, the destination cell's occupancy status is checked. If the destination cell is occupied, the chess piece occupying it must be removed before the chess piece at the origin cell can be moved to its intended destination. If the destination cell is unoccupied, the chess piece at the origin cell can be moved directly to its destination.

##### Clearing Destination Cell State

To clear the destination cell, the pick and drop state machine is provided with distinct origin and destination positions for picking up and dropping the chess piece. In a clearing pick and drop scenario, the pick and drop state machine is supplied with the destination cell as the pick position, and the "graveyard" as the drop position, which is a designated location in the world where captured chess pieces are moved.

Once the pick and drop state for clearing is completed, the system transitions to the moving the chess piece state.

##### Moving the chess piece State

In the moving the chess piece state, the pick and drop state machine is provided with the origin cell of the chess piece to be moved as the pick position, and the destination cell as the drop position. Upon successfully moving the chess piece, the system transitions to the sending finished back to the initial sender state.

##### Sending finished back to initial sender

During the sending finished back to the initial sender state, a message indicating successful completion of the command is sent back to the command's initial sender. Following this, the system returns to the idle state.

#### Pick and Drop state machine

The responsibility of picking up and dropping chess pieces lies with the pick and drop state machine, which receives the pick and drop positions as inputs. By providing flexibility in determining where to pick up and drop chess pieces, the pick and drop logic can be adapted to various scenarios.

##### The state machine:

Figure 4.10 Current Pick and Drop State Machine

###### Description

The pick and drop state machine remains in an idle state until a pick and drop execution is triggered. Once initiated, it initiates a sequence of executions that proceeds to the next execution when the robotic arm reaches the previous execution's position. In the event that the position is not reached, the state machine retries the same execution. Upon the completion of all executions, the pick and drop state machine returns to its idle state, and the overall state machine state is incremented.

The flow of execution is as follows:

1. Move to intermediate position for picking up a chess piece
2. Close Grippers slightly.
   1. This is to ensure that the robot arm fits in between chess pieces
3. Move to final position for picking up a chess piece
4. Close grippers to grip the chess piece
5. Move to intermediate position for picking up
6. Move to intermediate position for dropping
7. Move to final position for dropping
8. Open grippers slightly.
   1. Grippers are only slightly opened, as in the case of dropping it on the chess board there are possible other chess pieces in the vicinity. For the graveyard it would be possible to open the grippers fully. But to future proof and simplify this process the decision was made to only open the grippers slightly.
9. Move to intermediate position for dropping
10. Move to rest position and open grippers
11. Increment the overall state machine

##### Current iteration and why it is the way it is

The present version of the pick and drop state machine lacks the ability to handle potential issues that may arise during movement or gripping, such as unreachable positions, and fails to report them as errors. Instead, the system retries until the desired position is reached. This is due to a lack of sufficient time during the implementation process to address such issues. Although the current iteration does not present any issues with picking and dropping, it will require further development and refinement The decision to implement a state machine instead of a flow of executions was based on the rationale that it would enable the system to incorporate additional conditions for proceeding to the next execution in the future. Additionally, the pick and drop mechanism currently does not incorporate feedback from the robot arm camera, since the existing simulation only utilizes abstract chess pieces represented as cylinders that conform to the standard height of a chess piece.

The standard height definition of a chess piece height has been extracted from the International Chess Federation (FIDE) Handbook for standards of chess equipment [39].

### Chess piece detection

As portrayed in the overall architecture for the ROS robotic system, the detection of chess pieces necessitated a simplified approach. To accomplish this, an external camera was employed to obtain a generalized location for the chess pieces and their respective colours, enabling the robot arm to determine their precise positions and move accordingly for grasping. The robot arm camera was intended for grasping the chess pieces, as previously discussed in the robot arm section. However, due to time constraints, this feature was not implemented. Moreover, this extracted information would also be used for extracting the chess board state.

In order to overcome the challenge of extracting information regarding the chessboard cells without regard to the type of chess piece located therein, image processing techniques were utilized.

The OpenCV [40] library provides a multitude of tools for image processing. Within the ROS framework, the opencv\_apps [41] package offers a wide range of OpenCV functionalities.

#### Disclaimer

The solutions presented below are predicated upon an ideal environment and a static position for the chessboard, as time constraints prevented the exploration or implementation of a generalized approach applicable to real-world scenarios.

#### Initial Design

The initial approach involved utilized hough\_lines and contour\_moments.

hough\_lines utilize the Hough lines transform which is a technique used in computer vision for detecting straight lines in an image [41].

contour\_moments utilize the moments function of OpenCV. The moments function calculates the moments of a contour or image, which can be used to extract various properties such as the centroid, area, and orientations [41].

The hough\_lines function, using the straight lines typically present in a chessboard, can be utilized to identify straight lines in an image. By combining the resulting lines with contour moments, it becomes feasible to extract the precise positions of the individual chessboard cells.

##### Empty chess board:

Figure 4.111 hough\_lines image empty chess board initial design

Figure 4.122 contour\_moments of the hough\_lines from Figure 4.11

As illustrated in Figure 4.11, the image captured by the external camera, when processed with hough\_lines, generates lines for both the chessboard cells and the board itself. The integration of this information with contour moments permits the creation of a cell representation, as depicted in Figure 4.12.

Given the fixed position of the camera above the chessboard, and the constant dimensions of the board, it is feasible to extract the cell dimensions at the centre of the captured image. This provides insight into the expected area of a chess cell. Utilizing this expected area with a margin allows the extraction of all chess board cells.

##### Chess board with chess pieces

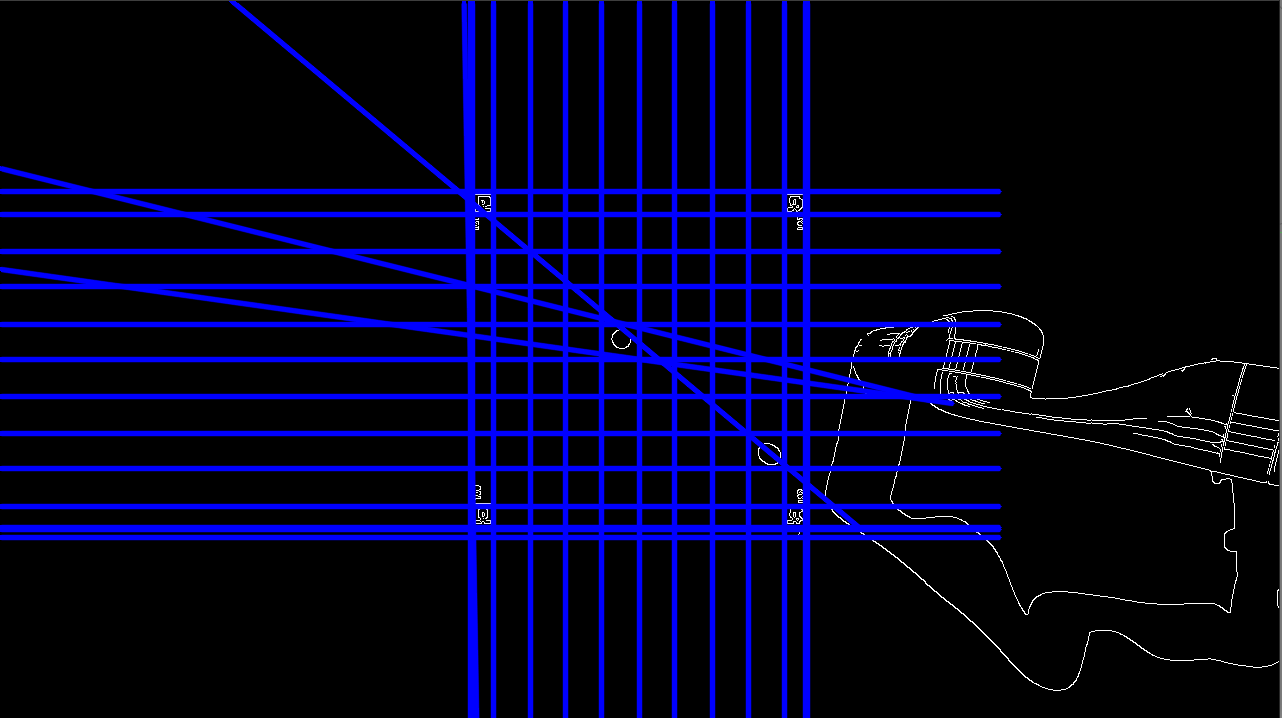
Investigating the changes of the approach with chess pieces in the cell.

Figure 4.13 hough\_lines with a chess piece initial design

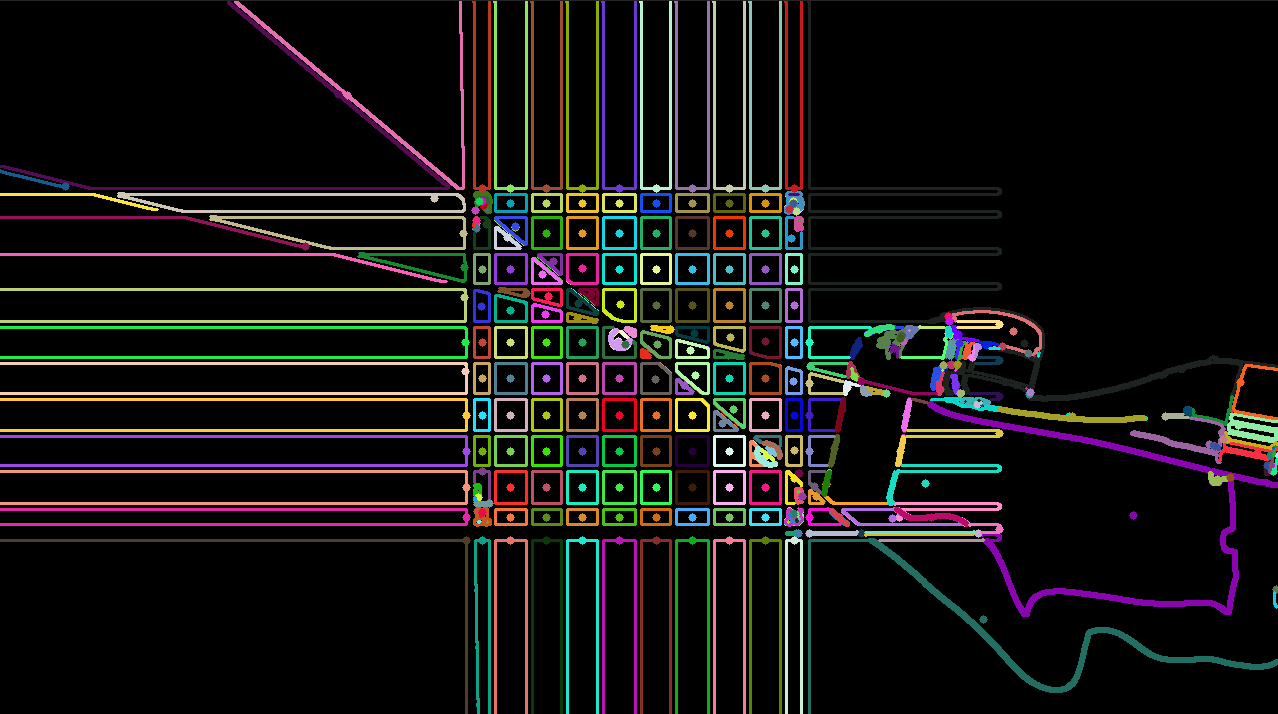
As depicted in Figures 4.13 and 4.14, the introduction of chess pieces and subsequent adjustments made by the robot arm generate additional lines that pose challenges for this design. Consequently, the initial design was abandoned, and an alternative approach was pursued.

Figure 4.14 contour\_moments of the hough\_lines from Figure 4.13

#### Current Design

Although the fundamental objective of utilizing OpenCV functions to streamline chess piece detection remained unchanged, alternative functions had to be explored. The GoodFeaturesToTrack function is capable of identifying the prominent corners within an image or a designated region of an image [42]. This library is made available in the opencv\_apps package of ROS under the name of goodfeatures\_track [41].

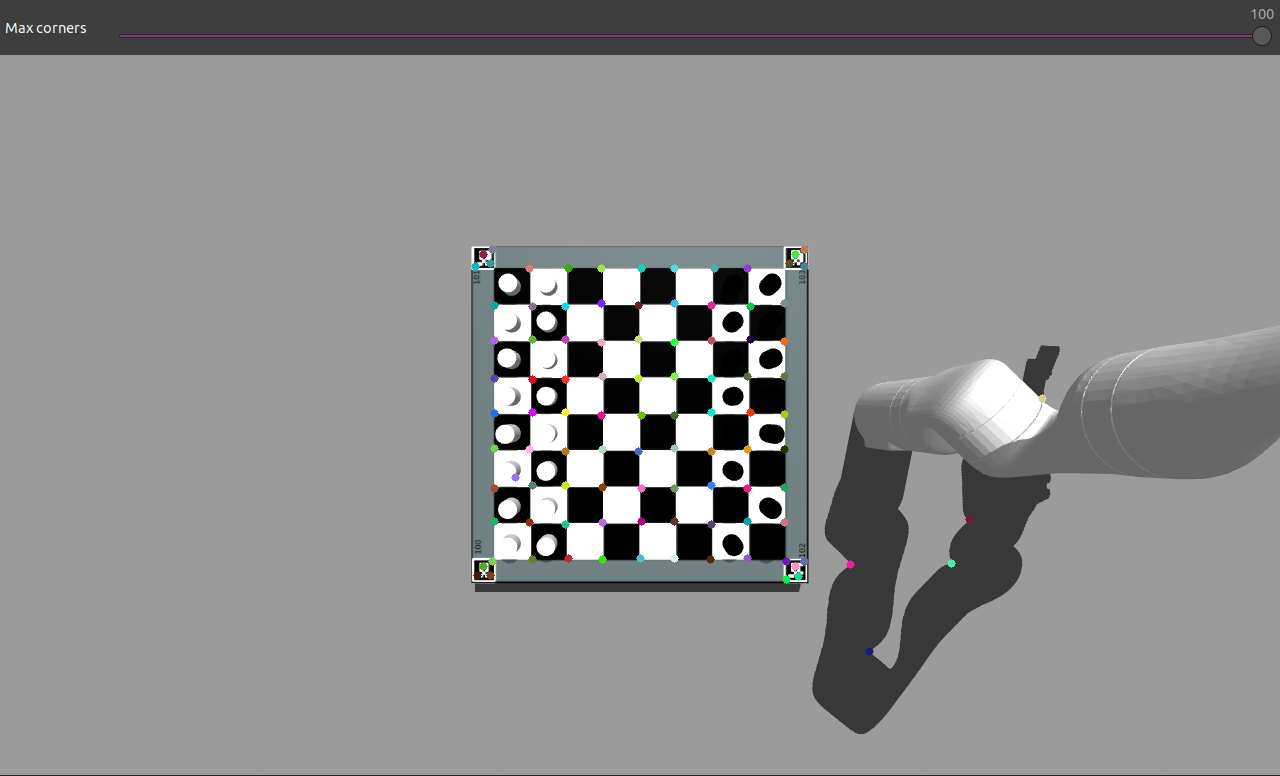
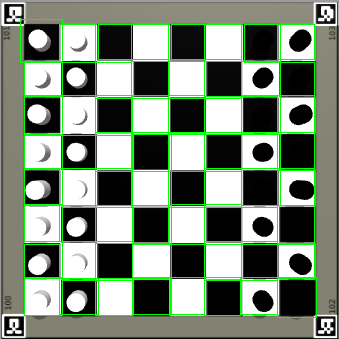


Figure 4.15 goodfeature\_track with chess pieces

Figure 4.16 extracted information from Figure 4.15

The functionality of the goodfeature\_track algorithm is demonstrated in Figures 4.15. Extracting the dimensions of the cells in the center of the captured image, given the camera's fixed position above the chessboard and the board's constant dimensions, enables us to understand how big a chess cell is supposed to be. This information enables the algorithm to estimate the expected distance between two points to form squares or rectangles for the cells. By comparing each point in the goodfeature\_track algorithm with other points based on their distances, the algorithm can determine the position of the chessboard cells and create the appropriate squares or rectangles. To optimize the algorithm's performance and avoid unnecessary redundancy, the points are sorted in a set by their x and y coordinates, thus allowing the algorithm to avoid checking points that have already been processed. These squares/rectangles are illustrated in Figure 4.15.

By utilizing the depth camera to analyze the depth of each pixel in the cell areas and having knowledge of the depth of an unoccupied cell, it is possible to determine whether a chess piece occupies a given cell and, if so, the distance of the piece from the camera. By utilizing the known depth of an empty cell and the measured distance, we can estimate the size of the chess piece. This information, combined with the RGB sensor of the camera, enables us to generate a representation of the chessboard state, indicating the location and color of each chess piece. This is illustrated in Figure 4.17. This information is then used by the target selector and robot arm.

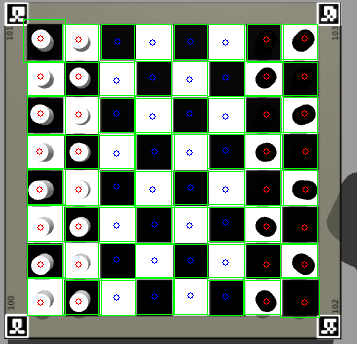


Figure 4.17 Chessboard state based on the information from depth sensor and rgb sensor

Figure 4.18 below shows the flow of data and its data type for the chessboard detection.

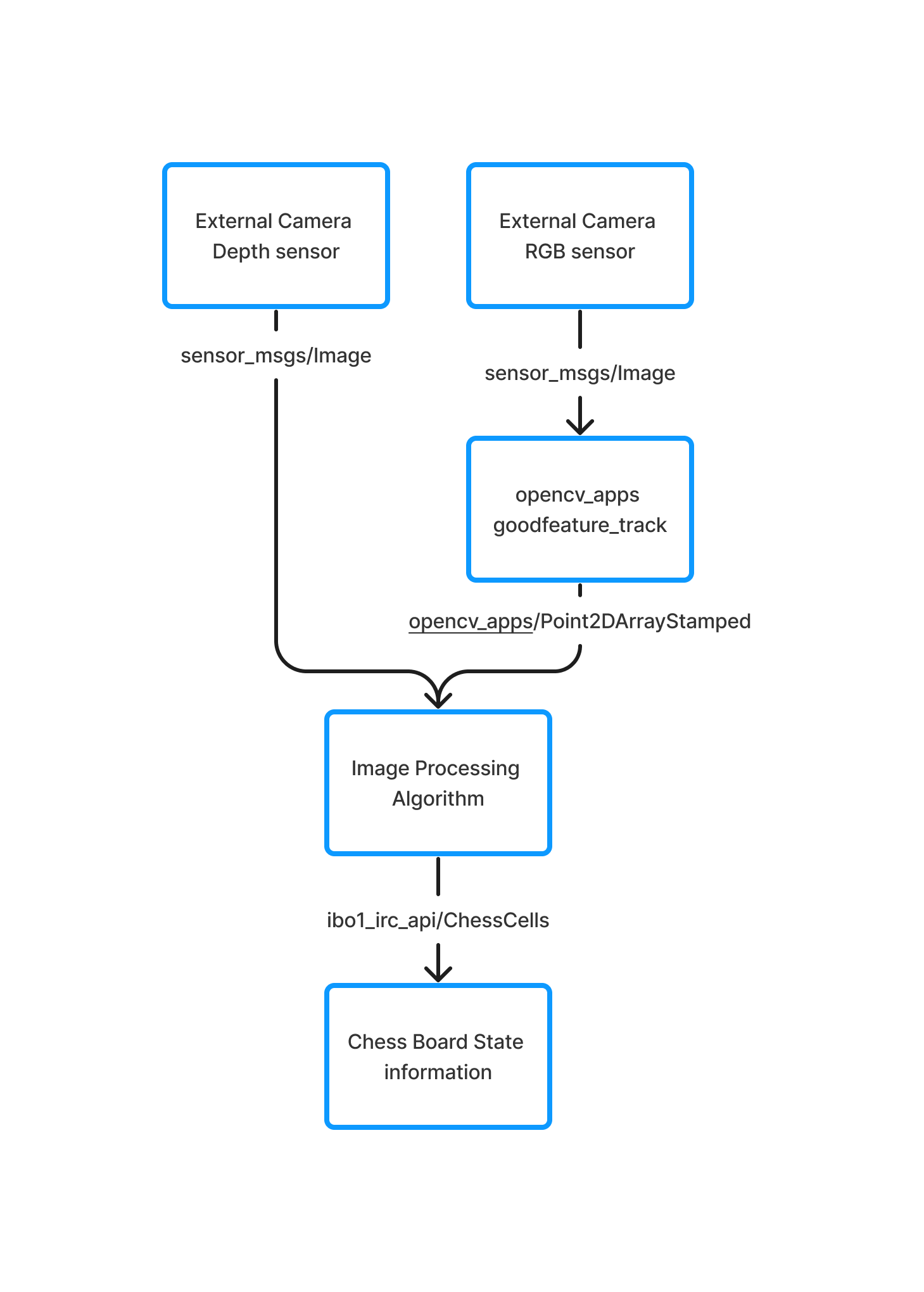


Figure 4.18 Flow of data and its data types for chess board cell detection

#### Robot arm restrictions

During the process of the robot arm moving to pick up a chess piece, the camera's view will be obstructed. Therefore, the algorithm's findings are only made available if all 64 chess cells are detected. Figure 4.19 and 4.20 provides an illustration of this obstruction.

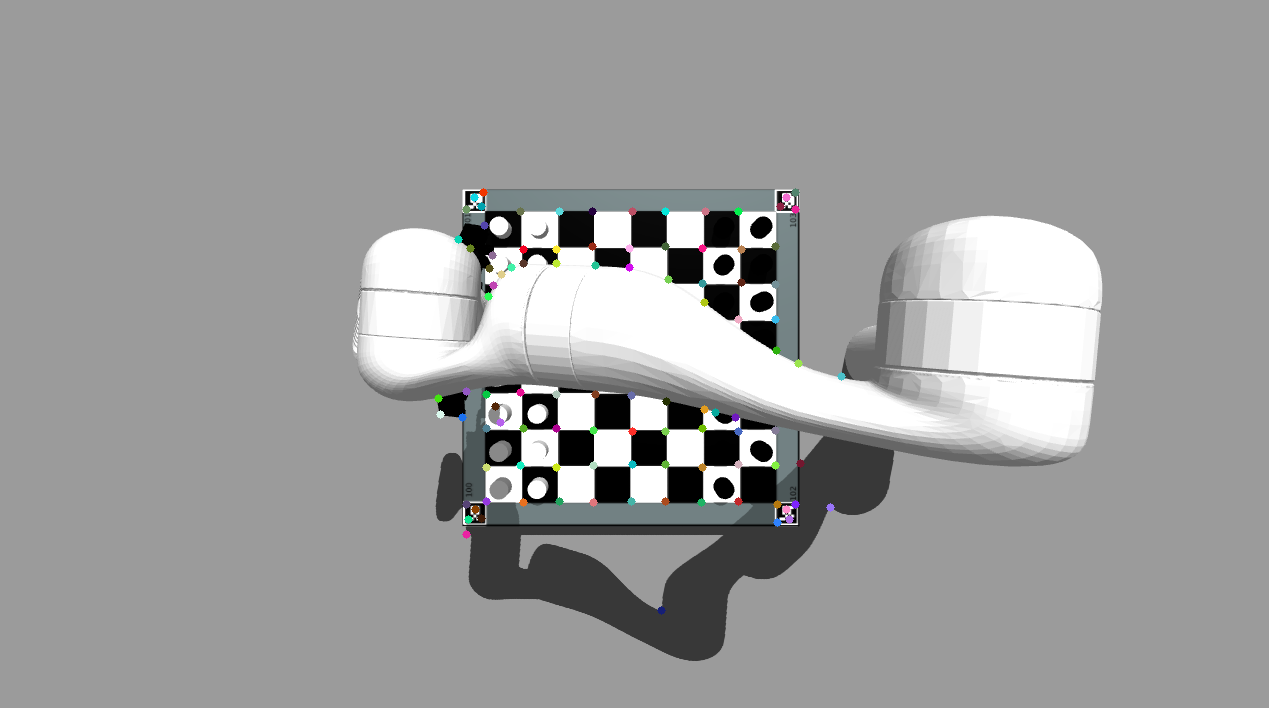


Figure 4.19 goodfeature\_track blocked by arm

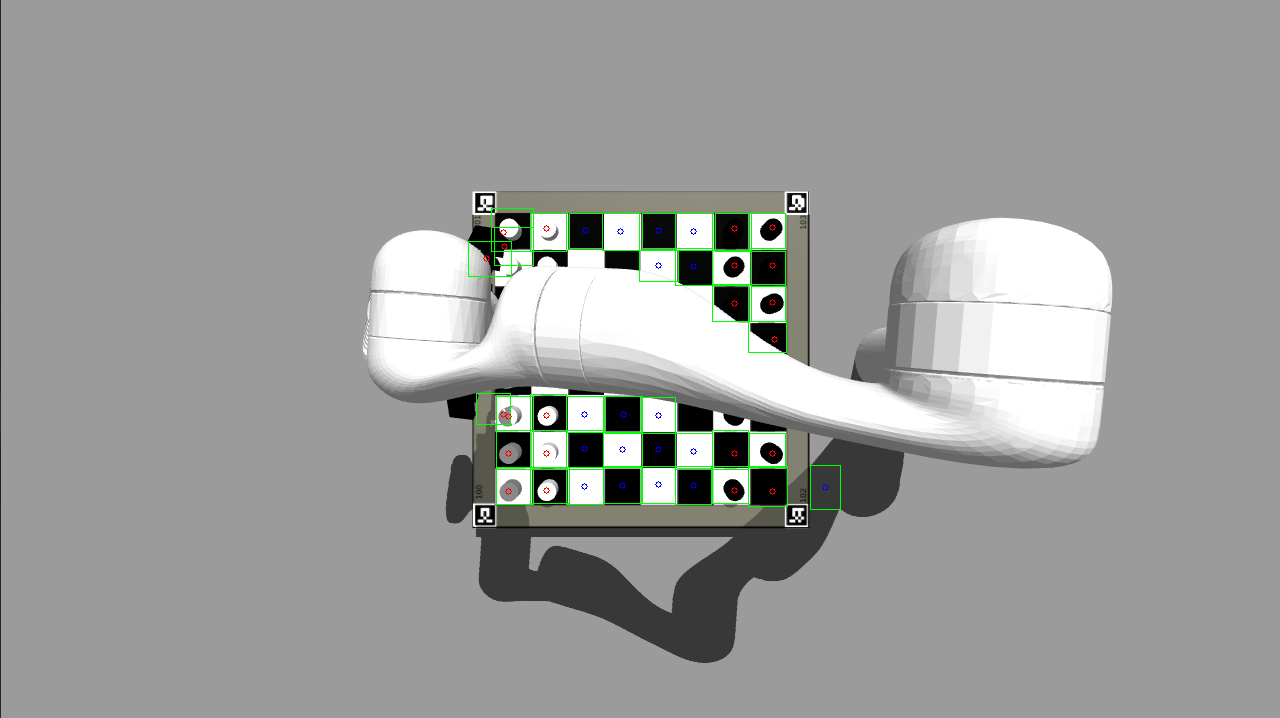


Figure 4.20 chess board information extraction blocked by robot arm

## User Interface

## Overall architecture

The project's user interface system comprises two primary components. The first is a library that facilitates the implementation of all system functions within any user interface framework. The second component is the user interface prototype, which serves as a demonstration of the library's functionality and interaction.

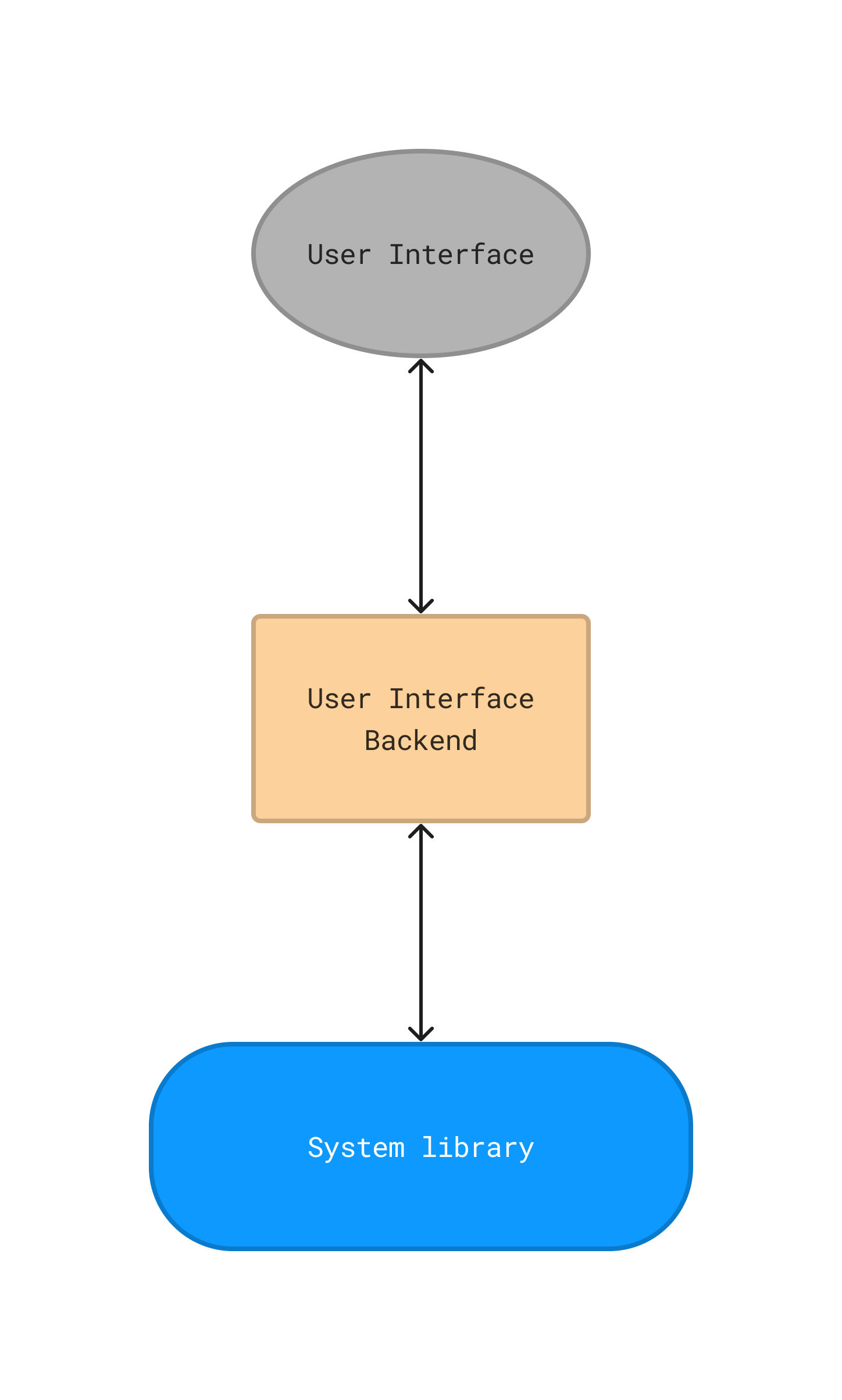


Figure 4.21 User Interface overall architecture

Figure 4.21 depicts the system library for the user interface, including the user interface backend, which specifies the visual design of the user interface. However, the backend utilized for this project is only a prototype, implemented solely to demonstrate the functionality of the system library. The user interface presented to the end-users is represented in Gray.

The System library is expected to facilitate the execution of the protocol as specified in section 4.2.1, and establish the requisite entities for usage in the user interface, thereby enabling users to engage with the functionality of the protocol through said interface.

As mentioned in the analysis chapter the system library will be written in Java [43] and the prototype of the user interface will be written in JavaFX [21].

As the user interface is a prototype to illustrate the implementation of the system library it is not deemed important enough to be mentioned in detail in this report.

## System library

### Introduction

As previously noted, the system library functions as the underlying mechanism that enables the implementation of the protocol for any user interface that is compatible with or utilizes the Java programming language. To achieve this, the system library incorporates a Java-based client that interacts with the ROS robotic system server. The library encapsulates the protocol's various commands as objects, with each object corresponding to their specific commands.

Moreover, the system library provides additional objects, such as the chess board and chess pieces, which simplify the implementation of a chess board on the user interface. The UI can utilize these objects by representing them visually, thus enabling users to employ them as necessary.

Each protocol object throws exceptions for protocol errors. These exceptions are represented as ProtocolErroExceptions.

### UML Class Diagram

Due to the size of the UML class diagram, it will be split up into multiple categories.

#### Cell and Chess pieces

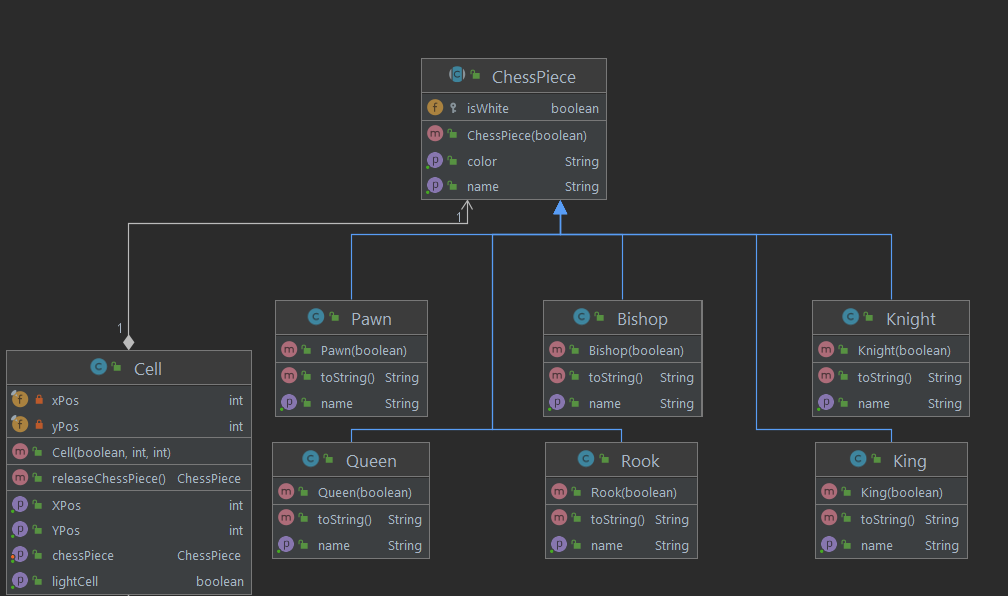


Figure 4.22 ChessPieces and Cell UML Class Diagram

#### ChessEngine, ChessBoard and IRCClient

Figure 4.23 ChessEngine, CHessBoard and IRCClient UML Class diagram

#### User, ROSSystem and ProtocolObject

Figure 4.24 User, ROSSystem and ProtcolObject UML CLass DIagram

The three UML class diagrams depicted in Figures 4.21, 4.23 and 4.24 show the types of classes that were created to implement the execution of commands for a user interface.

The classes User, ChessEngine and ROSSystem utilitze the IRCClient class to send and execute commands on the ROS robotic system. Each of them holds their own reference to the commands and errors that they deal with from the protocol.

The IRCClient holds the socket to communicate to the ROS system.

The ProtocolObject represents the definition of the protocol as seen in section 4.2.1.

The classes ChessBoard, Cell and chess pieces enable the user interface an easy inclusion of a chessboard representation and a way to utilize player moves, chess engine moves and such.

## Other Implementations

### No Check mate implementation

Due to time constraints and not being to implement check mate in time, it is currently not supported. Meaning no error messages will be send to the user even if it is set up to work. The game will not end and the user will end in an unmovable position. This was experimented but due it not producing the correct output it was removed to not impact the rest of the system.

### Playing against purely the chess engine

Due to time constraints, it was not possible to fully build this system. To still enable a multitude of features already build and a full interaction with the system it was made possible to play against the system without the robotic arm

### No Promotion moves for “Full simulation”

There was not enough time left in the project to implement promotion moves in the full simulation.

# Testing

## Overall Approach to Testing

The testing strategy entailed conducting evaluations of each significant module, algorithm, communication and execution flow utilizing Unit tests and manual tests. Unit tests were applied in instances where system communication was unnecessary, thus enabling the examination of isolated algorithms, modules and node communication. Conversely, manual tests were utilized to assess the system's standard behaviour and user interface. For example, manual tests involved evaluating the legality of chess moves made on the user interface and the systems behaviour when it is invoked. The principal evaluations conducted were integration testing against functional requirements and unit testing to verify the validity of legal and illegal moves. Additionally, testing was performed to examine data extraction and overall functionality of the chess engines.

The primary testing activities performed included integration testing to assess compliance with functional requirements, and unit testing to verify the validity of legal and illegal moves. Additionally, evaluations were conducted to examine the data extraction process and overall functionality of the chess engines.

## Automated Testing

### Unit Tests

Consistent with the testing strategy outlined earlier, Unit tests were employed to evaluate modules, algorithms, execution flows and node communications that could be isolated from the system. Specifically, the ROS robotic system utilizing the gtest [44] package of ROS constituted the primary focus of the Unit testing.

The test table created for unit testing would consist of:

* Name
* Description
* Possible initial state, as this was needed for certain tests
* Possible action, as this was needed for certain tests
* Outcome expected, what was expected
* Outcome, what was produced
* Observation, if it passed or failed or other comments

The Unit tests performed are attached as a test table in Appendix F.

These tests can also be verified and run by following the Build and Run manual attached as Appendix E.

## Manual Testing

Manual testing would suit as a way to test the integration of the overall system, user interface interactions and its capabilities.

Manual test tables would consist of following:

* Name
* Description
* Possible initial state, as this was needed for certain tests
* Possible action, as this was needed for certain tests
* Outcome expected, what was expected
* Outcome, what was produced
* Observation, if it passed or failed or other comments

The manual tests conducted involved examining the entire functionality of the system in its present state. The assessments encompassed exceptional behaviours, excluding possible complications related to the robotic arm. Not all the manual tests were recorded and are not all present in the test table that is attached as Appendix G.

## Comprehensiveness of the tests

The testing regimen suffered from a lack of comprehensiveness as sufficient time was not allocated for executing comprehensive tests. Given the time constraints and potential misinterpretation of the project's scope, the emphasis was on pushing functionality and conducting manual verification. The approach taken had certain limitations, including the inadequate testing of edge cases and a lack of comprehensive testing of algorithms. These shortcomings raise the possibility of encountering unanticipated issues in scenarios that were not previously accounted for, or when the system is subjected to unanticipated behaviour.

# Critical Evaluation

## Approach of the project

### Preparation and scope of the project

In terms of project preparation, minimal to medium efforts were undertaken. The decision to incorporate new technologies, such as C++, networking and chess engines, proved to be more challenging than initially anticipated, thereby necessitating additional research. While the decision to focus research on specific areas of implementation proved beneficial, as it prevented time and effort from understanding unnecessary aspects of the new technologies, it also resulted in a lack of prototyping. This lack of prototyping hindered the understanding of the complexity of certain aspects, and the challenges associated with their implementation.

Consequently, there was a failure to appreciate the project's detailed scope. However, based on previous experiences and a confidence in developing robotic systems, the overall project scope was well-conceived. Nevertheless, the absence of careful attention to detail for certain aspects proved to be a major challenge.

## Analysis

### Decomposition of the project

The project was effectively decomposed into its fundamental components, with due consideration given to the various subjects that the system encompasses. These subjects include, but are not limited to, chess piece detection, security risks, and the management of user information, which may be either implicitly processed or explicitly communicated to the user.

### Requirements

The fundamental objectives of the system were appropriately recognized, although the objectives themselves were somewhat broad and lacked necessary detail.

## Development Process

### Chosen development process

The elected development methodology was an adequate approach to the project, for the fact that the high adaptability of Scrum enabled its customization to the specific requirements of the project. Nonetheless, the implementation of the development process was suboptimal, as various challenges associated with the acquisition of new technologies restricted its full realization. Consequently, certain features of the methodology, including daily reviews, sprints and the requirement for module completion via testing, were not actualized. Ultimately, the development cycle mirrored that of XP more closely than that of Scrum.

### User Stories

The utilization of user stories was adequate, as they portrayed top-level interactions between users and the system. However, these stories represented large segments of work, and substantial back-end development was required to incorporate some of them. It is conceivable that a different form of representation for requirements and objectives in the development process could have been more advantageous.

## Design and Implementation

### Overall architecture

The overall system architecture and external communication protocol was well thought out, with a simple, uncomplicated layout. The communication protocol effectively reflected the intended information flow and the system's prospective capabilities despite the overall system's simplicity.

### ROS robotic system

The implementation of a modular approach in the system design was an excellent strategy that guaranteed flexibility in adapting to changes and modifications in the system's modules while information handling was generalized. The use of an internal protocol to facilitate information exchange between nodes, with the exception of the chess board cell detection node, simplified the processing of external protocol commands while allowing for the transmission of complex data and its associated meaning if needed in the future. Additionally, the decision to employ a target selector for approximating position merits recognition, as it enables autonomous gripping by the robotic arm in the future system when the chess pieces are not cylinders.

The adoption of the chess board cell and piece detection is a good approach, although its effectiveness may be limited to ideal conditions. It is plausible that this approach would still yield satisfactory results in a real environment if the chess board is initially detected, and only a masked image of the chess board is supplied to the algorithm. However, its efficacy in detecting actual chess pieces remains unclear.

The current robot arm movement strategy is simplistic in nature and only limitedly accounts for the height of the chess pieces, as it was designed to function in tandem with the robotic arm camera rather than relying solely on external camera information. However, modifying the pick and drop state machine to align with the original concept would be a feasible solution, as the robot arm state machine’s underlying approach of detecting whether a chess piece needs to be removed prior to execution is sound. Additionally, a noteworthy concept that was discussed but not included in the report was the implementation of a navigation problem to guide the robot arm's movement through the environment to emulate human-like behaviour, such as pushing chess pieces if the cells in front are vacant rather than lifting them unnecessarily.

The implementation of the chess logic is done really well, excluding the missing feature of check mate and promotion in full simulation. The possibility to play against a multitude of chess engines is a good and interesting design feature. The current missing feature of setting search options and changing engine options makes it even more interesting. The basic structure for this has been set out but it is not implemented system wide.

### User Interface

The idea to create a library as a framework that any kind of Java supported User Interface can implement is a great modular approach to the problem. It excludes the need to create different user interfaces from scratch but builds upon the framework that only needs to be expanded upon. The current prototype of the user interface is somewhat a good representation of what a properly designed user interface could include. The user interface should have been more discussed in the report. But the necessity to focus on the ROS robotic system was also there as there was a lot to cover.

## Testing

Testing was not done well for this project, there should have been more comprehensive tests. Although every function and connection that currently exists works to its intend, not everything was tested for failure or things that could go wrong. Example would be that supplying the chess engine with a wrong file path would either result in the child process not starting or starting and then the system might go into a full halt not knowing what to do and not having the possibility to report this back to the user.

## Tools

Generally, the tools used for this project were thoroughly thought through. There should have been more acceptance in using 3rd party libraries or solutions to dealing with networking.

## Achievement of the aims

Given the scope of the project as a bachelor thesis, the realization of a real-world implementation of the system proved to be beyond its capacity. However, the current version of the system constitutes a valuable achievement. Although the system presents certain deficiencies in terms of error handling, as no such mechanism has been implemented, it can be considered a solid product overall. Nonetheless, the system may experience certain issues, such as infinite loops, which should be addressed to improve its reliability.

## Future work

In the future, the system should prioritize addressing the unhandled errors by implementing appropriate error handling mechanisms. Additionally, certain parts of the system should be rewritten to support multi-threading and the inclusion of timeouts for clients. Furthermore, the next phase of the project should involve deploying the system in a real-world scenario with the inclusion of human player movement detection, to ensure its effectiveness in such an environment.

## Lessons learned and how I would approach it next time

My C++ programming skills underwent significant growth, progressing from non-existent to a level that could be considered amateurish. Acquiring knowledge about networking and its basics was an engrossing and stimulating experience. Additionally, integrating a chess engine into a system presented a challenging but intriguing task. Working with the robot within the ROS environment, however, posed another set of challenges, as the robot lacked built-in sensors for the gazebo model. Consequently, the robot model had to be expanded upon, necessitating the learning of xacro and urdf along the way. Reflecting upon my experience, in a hypothetical second iteration of the project, I would scale down its scope or forgo learning C++ from scratch, while aiming to incorporate more development processes and testing into the project.

# Figures

[Figure 2.1 General User- User Stories 14](#_Toc134154269)

[Figure 2.2 User and Admin - User Stories 15](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154270)

[Figure 2.3 Admin - User Stories 15](#_Toc134154271)

[Figure 4.1 User Interface - ROS Robotic System Communication 20](#_Toc134154272)

[Figure 4.2 Initial Architecture of the ROS robotic system 23](#_Toc134154273)

[Figure 4.3 Current Architecture of the ROS robotic system 25](#_Toc134154274)

[Figure 4.4 UML Class Diagram ChessWrapperNode 28](#_Toc134154275)

[Figure 4.5 Current System State Machine 30](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154276)

[Figure 4.6 Chess Engine and Player move in Full Simulation Flow Chart 32](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154277)

[Figure 4.7 Set Target Example 1 33](#_Toc134154278)

[Figure 4.8 Set Target Example1 different point of view 34](#_Toc134154279)

[Figure 4.9 Current Overall Robot Arm State Machine 35](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154280)

[Figure 4.10 Current Pick and Drop State Machine 37](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154281)

[Figure 4.11 hough\_lines image empty chess board initial design 40](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154282)

[Figure 4.12 contour\_moments of the hough\_lines from Figure 4.11 40](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154283)

[Figure 4.13 hough\_lines with a chess piece initial design 41](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154284)

[Figure 4.14 contour\_moments of the hough\_lines from Figure 4.13 42](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154285)

[Figure 4.15 goodfeature\_track with chess pieces 43](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154286)

[Figure 4.16 extracted information from Figure 4.15 44](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154287)

[Figure 4.17 Chessboard state based on the information from depth sensor and rgb sensor 45](#_Toc134154288)

[Figure 4.18 Flow of data and its data types for chess board cell detection 46](#_Toc134154289)

[Figure 4.19 goodfeature\_track blocked by arm 47](#_Toc134154290)

[Figure 4.20 chess board information extraction blocked by robot arm 47](#_Toc134154291)

[Figure 4.21 User Interface overall architecture 48](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154292)

[Figure 4.22 ChessPieces and Cell UML Class Diagram 50](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154293)

[Figure 4.23 ChessEngine, CHessBoard and IRCClient UML Class diagram 51](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154294)

[Figure 4.24 User, ROSSystem and ProtcolObject UML CLass DIagram 52](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134154295)

# References

|  |  |
| --- | --- |
| [1] | NASA, "NASA Science MARS EXPLORATION," NASA, 2019. [Online]. Available: https://mars.nasa.gov/. [Accessed 1 May 2023]. |
| [2] | ABB, "ABB Robotics solutions for the Automotive industry," ABB, 2023. [Online]. Available: https://new.abb.com/products/robotics/industries/automotive?gad=1&gclid=Cj0KCQjw6cKiBhD5ARIsAKXUdyYJZJKC300oQEn37bAUx4mqF-km8smXIauCIvN8SO7X-YBFLBVNT\_gaAl1cEALw\_wcB. [Accessed 1 May 2023]. |
| [3] | "Mechanical Turk," Wikipedia, 21 July 2021. [Online]. Available: https://en.wikipedia.org/wiki/Mechanical\_Turk. |
| [4] | A. Silver, "Chess News," 7 July 2022. [Online]. Available: https://en.chessbase.com/post/robots-and-chess. [Accessed 1 May 2023]. |
| [5] | "Deep Blue (chess computer)," Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Deep\_Blue\_(chess\_computer). [Accessed 1 May 2023]. |
| [6] | L. P.-A. Jan Rosell, "ROS Final project," Universitat POLITECHNICA DE CATALUNYA BARCELONA TECH, 2023. [Online]. Available: https://sir.upc.edu/projects/rostutorials/final\_work/index.html. [Accessed 1 May 2023]. |
| [7] | ROS, "ROS.org | Powering the world's robots," ROS, 2020. [Online]. Available: https://www.ros.org/. [Accessed 1 May 2023]. |
| [8] | EEDGE-tronics, "Chess-Robot Github," GitHub, 2023. [Online]. Available: https://github.com/EDGE-tronics/Chess-Robot. [Accessed 1 May 2023]. |
| [9] | J. Henley, "Chess robot grabs and breaks finger of seven-year-old opponent," The Guardian, 24 July 2022. [Online]. Available: https://www.theguardian.com/sport/2022/jul/24/chess-robot-grabs-and-breaks-finger-of-seven-year-old-opponent-moscow. [Accessed 1 May 2023]. |
| [10] | A. E. Soltis, "Chess," Britannica, 3 Apri 2023. [Online]. Available: https://www.britannica.com/topic/chess. [Accessed 1 May 2023]. |
| [11] | B. SIG, "Bluetooth Technology Overview," Bluetooth SIG, 2023. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. [Accessed 1 May 2023]. |
| [12] | B. SIG, "2022 Market Update," Bluetooth SIG, [Online]. Available: https://www.bluetooth.com/2022-market-update/. [Accessed 1 May 2023]. |
| [13] | A. Wykes, "Why is Bluetooth still so unreliable?," SOUNDGUYS, 12 September 2021. [Online]. [Accessed 1 May 2023]. |
| [14] | "RS-232," Wikipedia, 2020. [Online]. Available: https://en.wikipedia.org/wiki/RS-232. [Accessed 1 May 2023]. |
| [15] | "USB," Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/USB. [Accessed 1 May 2023]. |
| [16] | "Internet Protocol," Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Internet\_Protocol. [Accessed 1 May 2023]. |
| [17] | K. Koishigawa, "TCP vs. UDP Wath's the difference and which protocol is faster?," freeCodeCamp, 28 June 2021. [Online]. Available: https://www.freecodecamp.org/news/tcp-vs-udp/. [Accessed 1 May 2023]. |
| [18] | Python, "What is Python? Executive Summary," Python.org, 2019. [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 1 May 2023]. |
| [19] | Java, "What is Java technology and why do I need it?," Java.com, 2022. [Online]. Available: https://www.java.com/en/download/help/whatis\_java.html. [Accessed 1 May 2023]. |
| [20] | Kotlin, "Kotlin Homepage," Kotlin Foundation, [Online]. Available: https://kotlinlang.org/. [Accessed 1 May 2023]. |
| [21] | Gluon, "JavaFX," Gluon OpenJFX, [Online]. Available: https://openjfx.io/. [Accessed 1 May 2023]. |
| [22] | U. Robots, "Collaborative robots from universal robots," Universal Robots 2023, 2023. [Online]. Available: https://www.universal-robots.com/products/#:~:text=UR3e,6.6%20lbs%20(3%20kg).. [Accessed 1 May 2023]. |
| [23] | Kinova, "Imagine the possibilities," Kinova inc., [Online]. Available: https://www.kinovarobotics.com/product/gen3-robots. [Accessed 1 May 2023]. |
| [24] | O. S. R. Foundation, "Gazebo," Open Source Robotics Foundation, 2014. [Online]. Available: https://classic.gazebosim.org/. [Accessed 1 May 2023]. |
| [25] | C. Belshe, "Chess Piece Detection," California Polytechnic State university, 2021. [Online]. Available: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1617&context=eesp. [Accessed 1 May 2023]. |
| [26] | Shredder, "UCI," Shredder, 2023. [Online]. Available: https://www.shredderchess.com/chess-features/uci-universal-chess-interface.html. [Accessed 1 May 2023]. |
| [27] | official-stockfish, "Stockfish," GitHub, 2020. [Online]. Available: https://github.com/official-stockfish/Stockfish. [Accessed 1 May 2023]. |
| [28] | komodochess.com, "README.TXT," 2020, [Online]. Available: https://komodochess.com/store/pages.php?cmsid=14. [Accessed 1 May 2023]. |
| [29] | S. C. Foundation, "Standard C++," 2019. [Online]. Available: https://isocpp.org/. [Accessed 1 May 2023]. |
| [30] | Figma, "The Collaborative interface design tool," [Online]. Available: https://www.figma.com/. [Accessed 1 May 2023]. |
| [31] | J. S. Ken Schwaber, "The 2020 Scrum Guide," 2020. [Online]. Available: The iterative approach of Scrum provides the opportunity to test and validate new ideas. [Accessed 1 May 2023]. |
| [32] | kanbanize, "Kanban explained in 10 minutes," 2021. [Online]. Available: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban. [Accessed 1 May 2023]. |
| [33] | GitHub, "GitHub Homepage," GitHub, [Online]. Available: https://github.com/. [Accessed May 1 2023]. |
| [34] | G. Docs, "About Projects," GitHub, [Online]. Available: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects. [Accessed 1 May 2023]. |
| [35] | sandlube2, "Pawn Promotion Popularity," Reddit, March 2023. [Online]. Available: https://old.reddit.com/r/chess/comments/10ue8gq/pawn\_promotion\_popularity/. [Accessed 1 May 2023]. |
| [36] | wintermute93, "The pawn with the highest percentage of promotion, statistically," Reddit, February 2023. [Online]. Available: https://www.reddit.com/r/chess/comments/10vinkz/the\_pawn\_with\_the\_highest\_percentage\_of\_promotion/. [Accessed 1 May 2023]. |
| [37] | HabibOladepo, "Nodes," ROS.org, 4 December 2018. [Online]. Available: http://wiki.ros.org/Nodes. [Accessed 1 May 2023]. |
| [38] | TullyFoote, "Topics," ROS.org, 20 February 2019. [Online]. Available: http://wiki.ros.org/Topics. [Accessed 1 May 2023]. |
| [39] | FIDE, "FIDE Handbook," FIDE, 04 August 2022. [Online]. Available: https://handbook.fide.com/chapter/StandardsOfChessEquipment2022. [Accessed 1 May 2023]. |
| [40] | OpenCV, "OpenCV Homepage," OpenCV, 2019. [Online]. Available: https://opencv.org/. [Accessed 1 May 2023]. |
| [41] | iory, "opencv\_apps," ROS Wiki, 15 February 2022. [Online]. Available: http://wiki.ros.org/opencv\_apps. [Accessed 1 May 2023]. |
| [42] | OpenCV, "Feature Detection," OpenCV, [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/feature\_detection.html#goodfeaturestotrack. [Accessed 1 May 2023]. |
| [43] | Oracle, "OpenJDK," Oracle, 2023. [Online]. Available: https://openjdk.org/. [Accessed 1 May 2023]. |
| [44] | MaxGittelman, "gtest," ROS.org, 8 November 2018. [Online]. Available: http://wiki.ros.org/gtest. [Accessed 1 May 2023]. |

# Appendices

* 1. Third-Party Code and Libraries
     1. Code
        1. Socket Creation on Linux

**By:** CodeHoose bbtcpserver.cpp was utilized for the creation of C++ socket on ROS robotic system.

**It is accessible through this link:**

<https://gist.github.com/codehoose/020c6213f481aee76ea9b096acaddfaf>

**License:** I was not able to find out under what license this run. This is currently still being checked against. It is assumed it is an open license as he uses this code in a Youtube tutorial.

**Was it modified:** Yes, was partially modified and slightly rewritten for the needed usage.

**Where in the project can it be found:** IRCServer.h

* + - 1. bytes to integer conversion

**By:** Mindriot conversion of bytes to integer:

**It is accessible through this link:**

<https://stackoverflow.com/questions/34943835/convert-four-bytes-to-integer-using-c>

**License:** cc-wiki (aka cc-by-sa)

**Was it modified:** Original was not modified

**Where in the project can it be found:** IRCServer.h

* + - 1. JavaFX-Online-Chess

**By:** GitHub user Stevoisiak’ JavaFX-Online-Chess is an implementation of chess for java.

**It is accessible through this link:**

<https://github.com/Stevoisiak/JavaFX-Online-Chess/blob/master/license.txt>

**License:** MIT License for the code and icons are licensed under by Colin M.L. Burnett:

https://commons.wikimedia.org/wiki/Category:SVG\_chess\_pieces

Creative Commons Attribution-Share Alike 3.0 Unported

<https://en.wikipedia.org/wiki/en:Creative_Commons>

**Was it modified:** Yes, heavily modified for Java and it was converted into C++ with heavy modifications. The icons were not edited.

**Where in the project can it be found:** For C++ in the Chess folder. For Java in the ChessSpecific folder for the IRC\_API and in the resource and UIElements folder for the general src of the project. Where it was used it was documented in the code.

* + - 1. Communication with a subprocess via stdin/stdout

**By:** GitHub user auxiliary

**It is accessible through this link:**

<https://gist.github.com/auxiliary/852015a8b278ca49964a3eab0990115e>

**License:** MIT License

**Was it modified:** Yes, slight changes as explained in the documentation of the code

**Where in the project can it be found:** In the Utility folder of the ROS robotic system in the SubProcessHandler.h

* + - 1. ChessBoards for the gazebo simulation

**By:** GitLab users Jan Rosell @jan.rosell and Leopold Palomo-Avellaneda @leopold.palomo

**It is accessible through this link:**

<https://gitioc.upc.edu/rostutorials/gazebo_sensors_tutorial/-/tree/ab3ed9f585e19d5899b1a1fd356a4746b57ff68f/models>

**License:** Not specified, msg send to the original creator

**Was it modified:** Yes, added link to the source in the config files, to indicated where it originated from

**Where in the project can it be found:** In the models folder for the ROS robotic system, called chessboard and chessboard2

* + - 1. ROS kortex launch and model

**By:** GitHub user Kinovarobotics

**It is accessible through this link:**

<https://github.com/Kinovarobotics/ros_kortex>

**License:** <https://github.com/Kinovarobotics/ros_kortex/blob/noetic-devel/LICENSE>

**Was it modified:** Nothing only expansions and parts of it have been taken.

The robot model was expanded to include sensors.

The launch file for launching their robotics model has been taken apart into its

elemental needs for this project. And changes to launching the expanded model

**Where in the project can it be found:**

**For the model:** In the description folder of the ROS package: full\_robot.xacro.

**For the launch file:** In the launch folder for the ROS package: robotGazebo.launch

* + 1. Libraries

**OpenCV** – The OpenCV libraries for ROS under the package opencv\_apps have been used.

**License:** BSD

**OpenJFX –** JavaFX 17.0.6 for the user interface prototype.

**License:** licensed under the GPL with the class path exception

**Gtest –** The Gtest libraries for ROS under the package gtest have been used.

**License:** BSD 3-clause license

**ros\_kortex –** The ros kortex library is the robt arm used and its corresponding package for ros

noetic.

**License:**  <https://github.com/Kinovarobotics/ros_kortex/blob/noetic-devel/LICENSE>

**OpenJDK –** The Java jdk used is 17.0.6 OpenJDK

**License:** licensed under the GPL with the class path exception

**ROS –** Robotic middle ware for the robotic system

**License:** Apache 2.0

**Gazebo –** Simulation environment used with ROS.

**License:** Apache 2.0

**Rviz –** Visualization tool for ROS

**License:** BSD, Creative Commons

**Gradle –** Build automation tool for Java

**License:** Apache 2.0

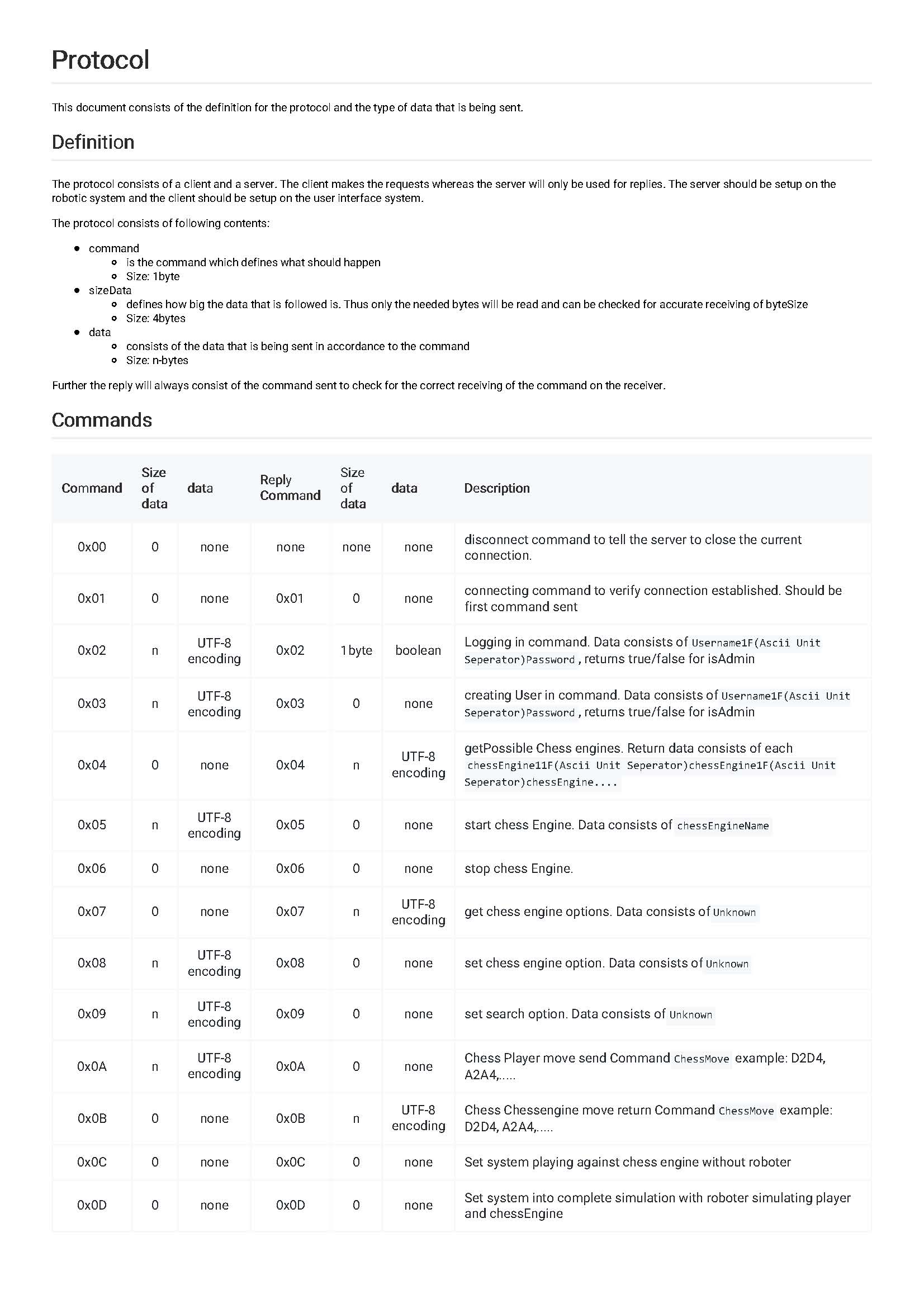
**Stockfish –** A chess engine used and part of the project files.

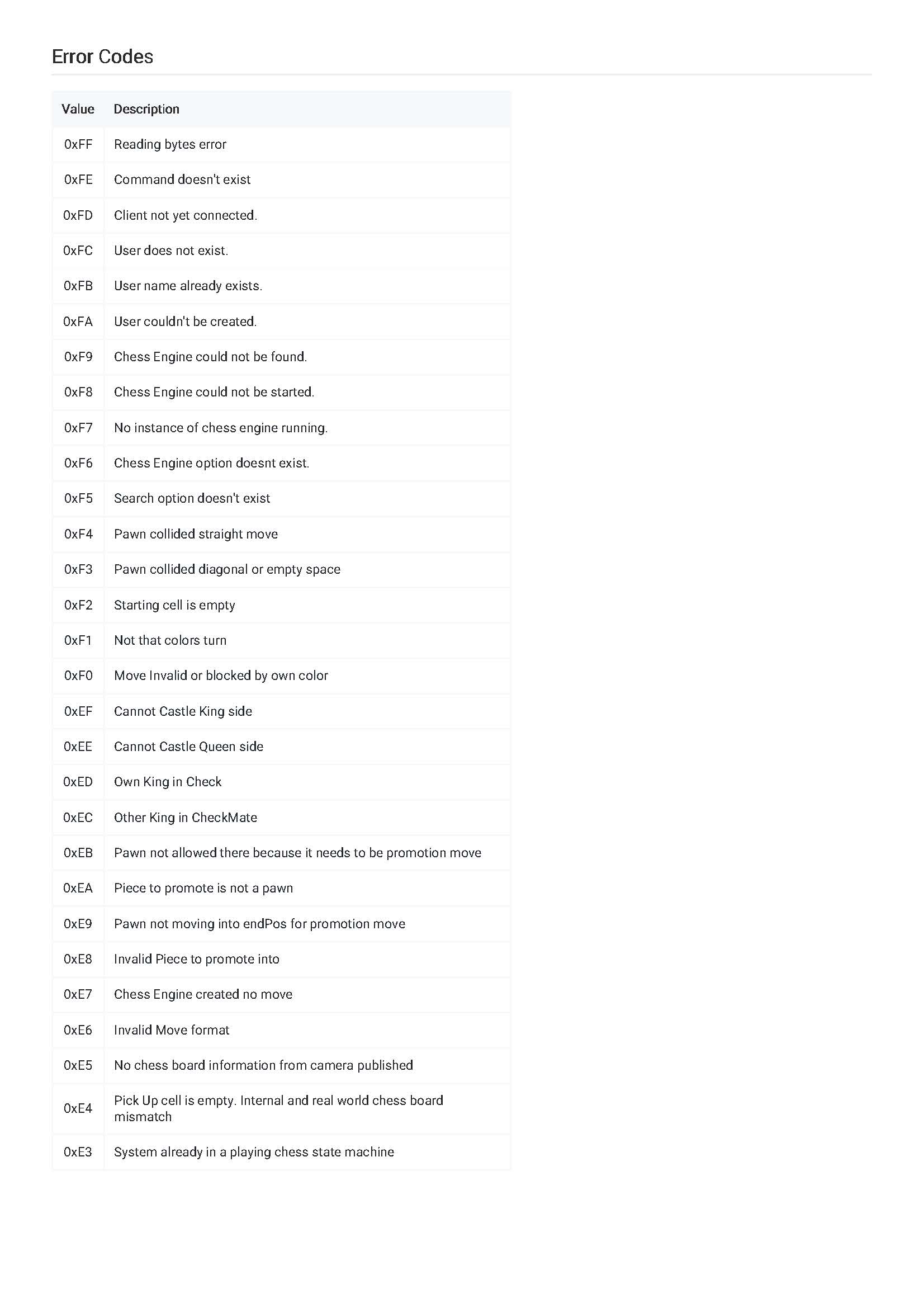
**License:** GPLv3

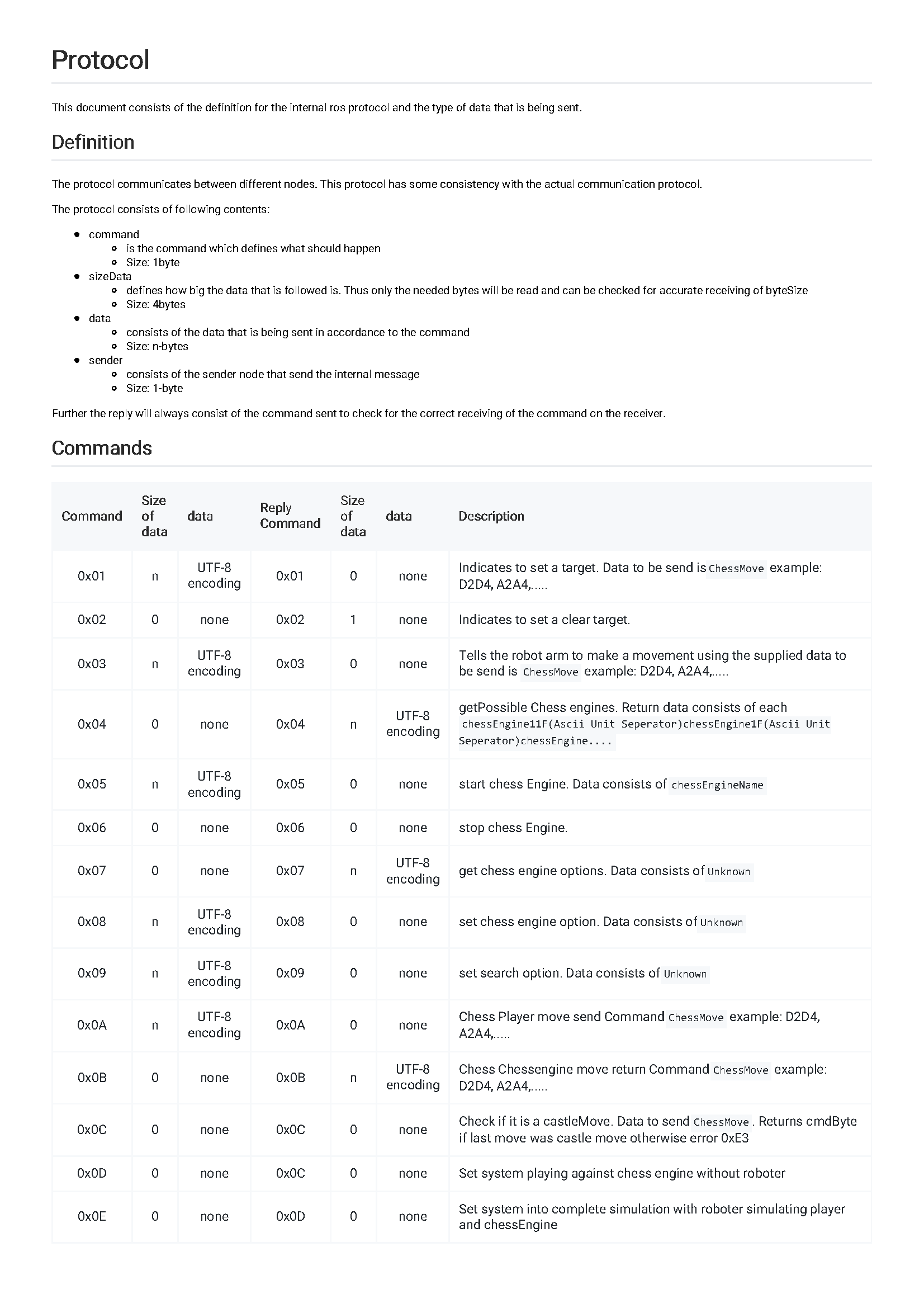
**Komodo** –A chess engine used and part of the project files. The version used is

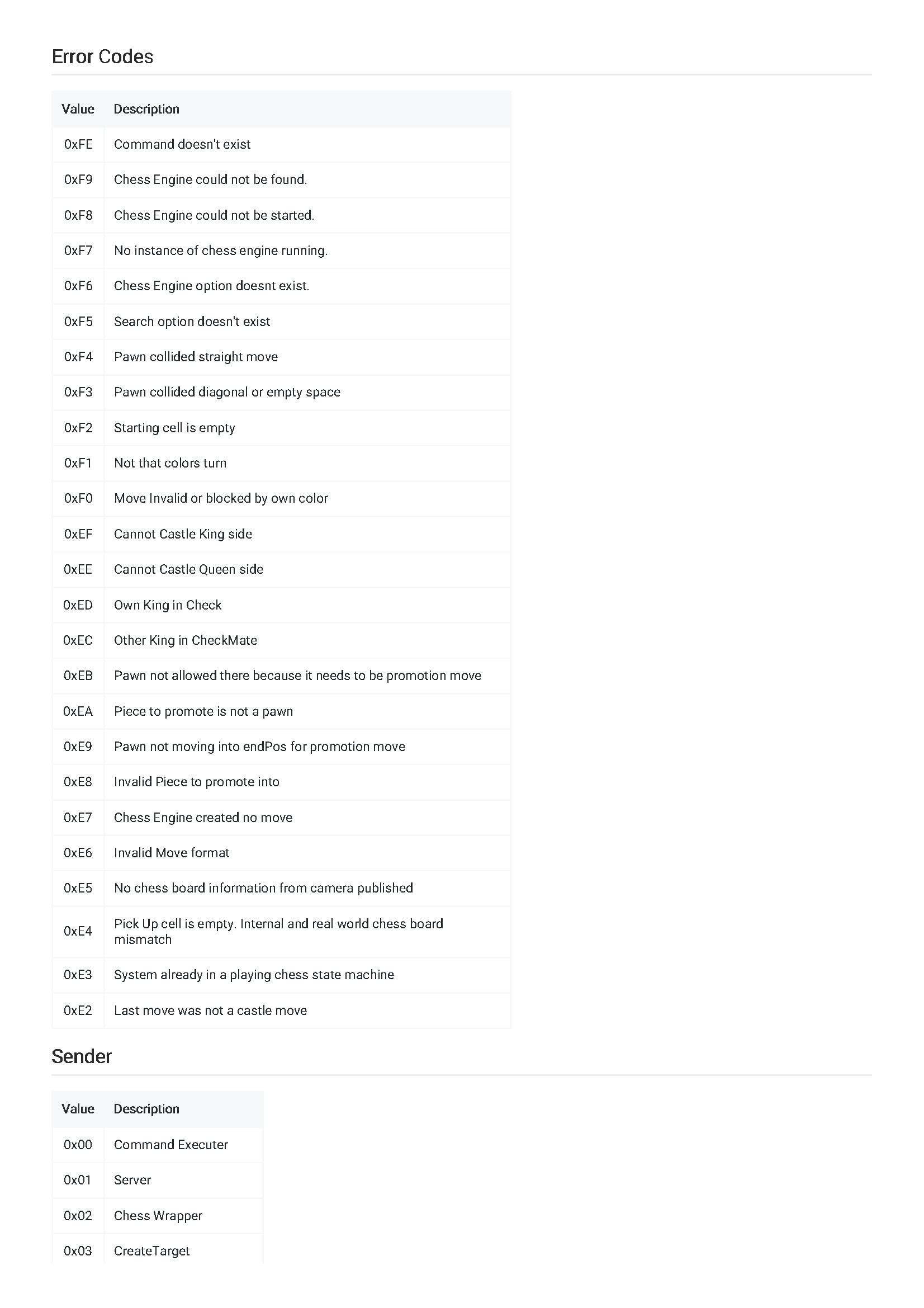
Komodo 14 free.

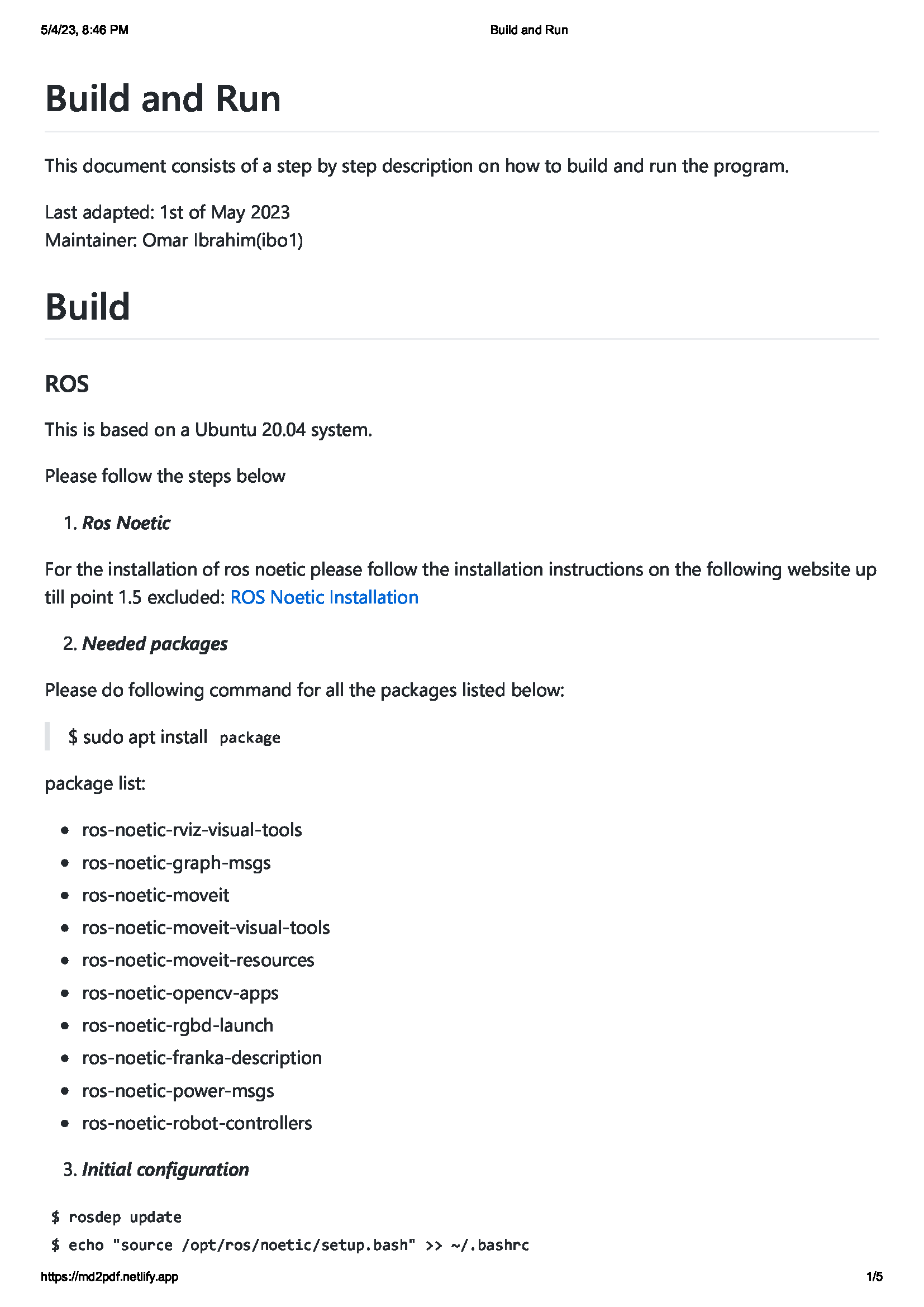
**License:** Proprietary, but free version used for non-commercial use.

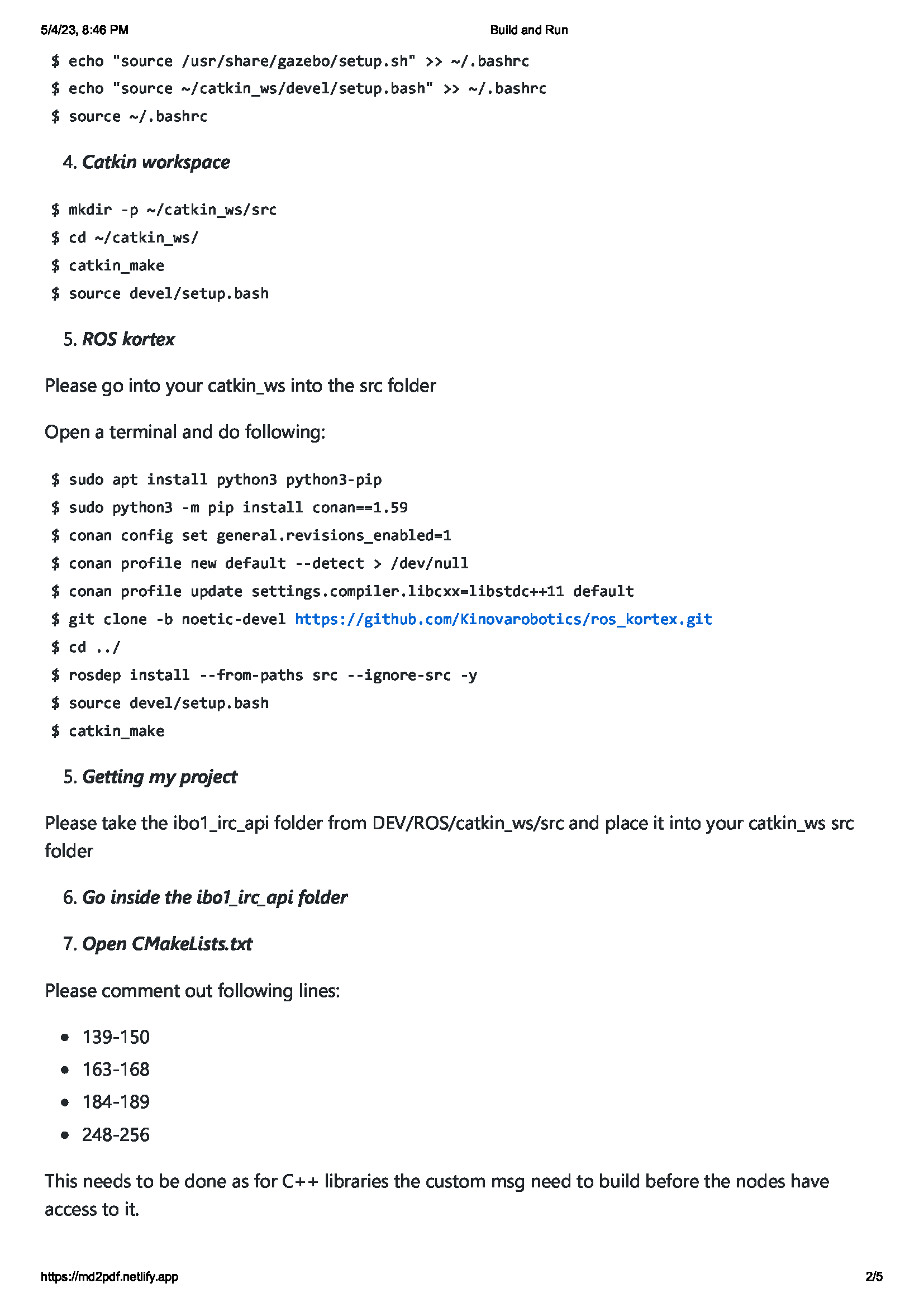
* 1. Code Samples
  2. Communication Protocol

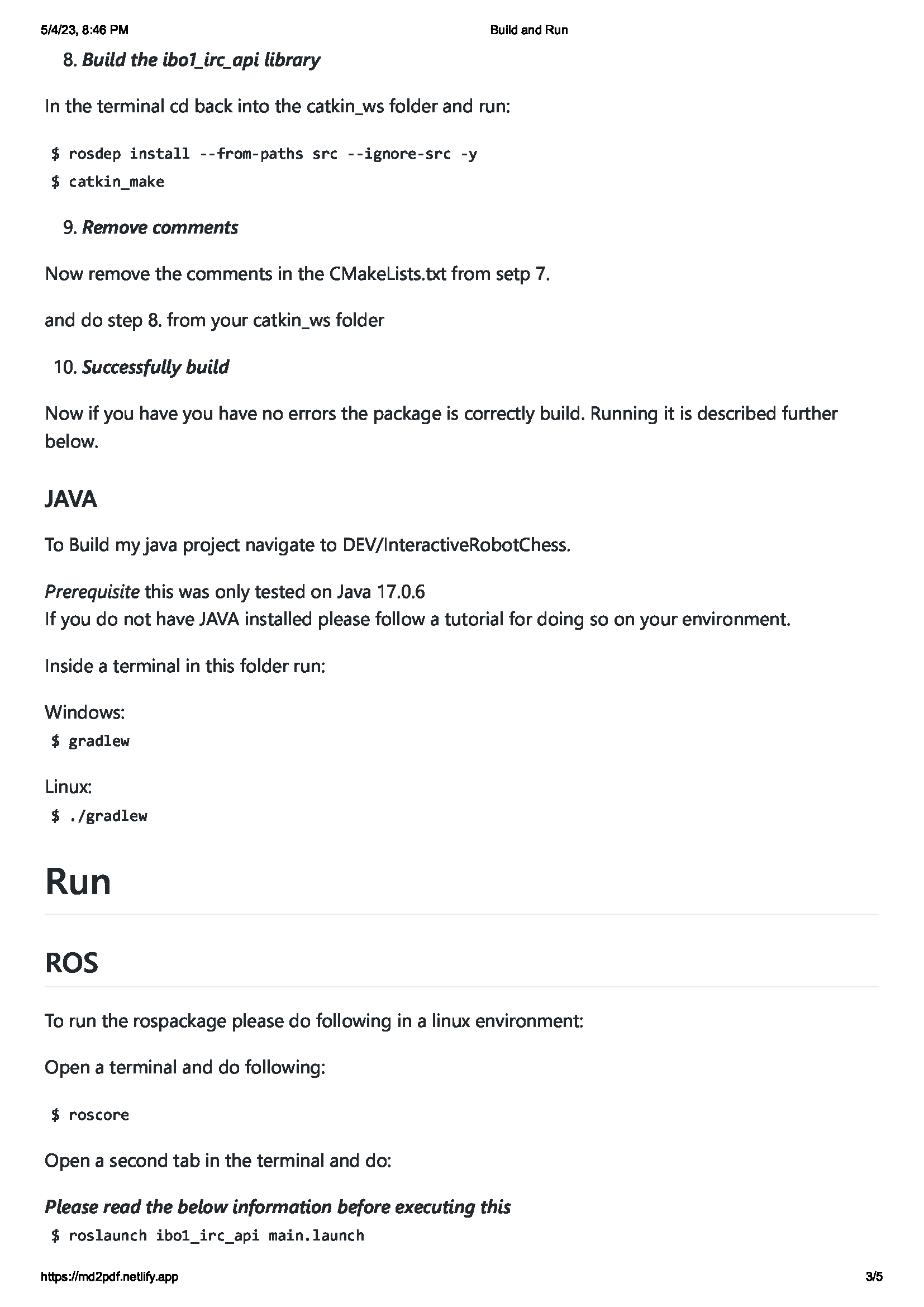


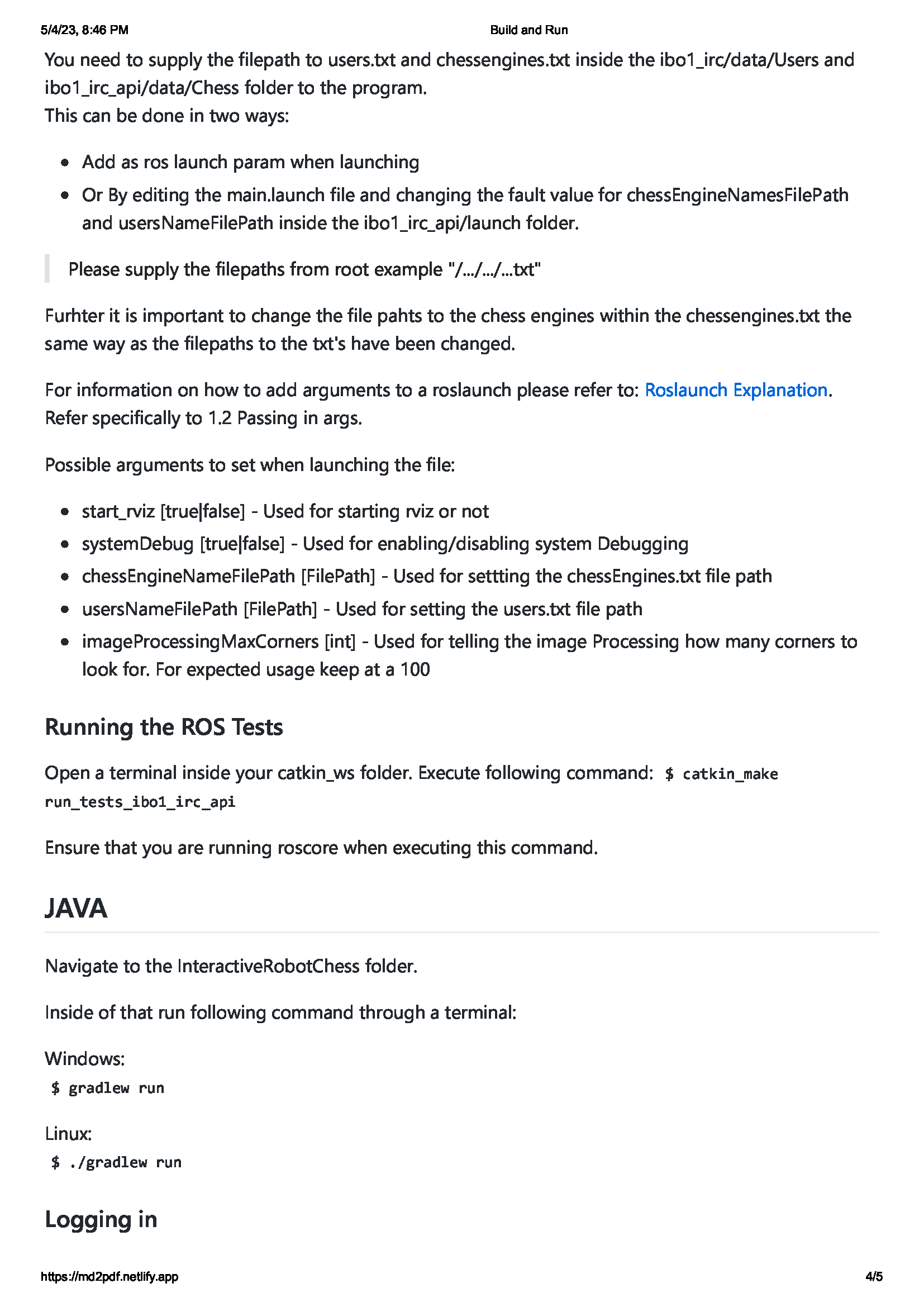
* 1. Internal Communication Protocol

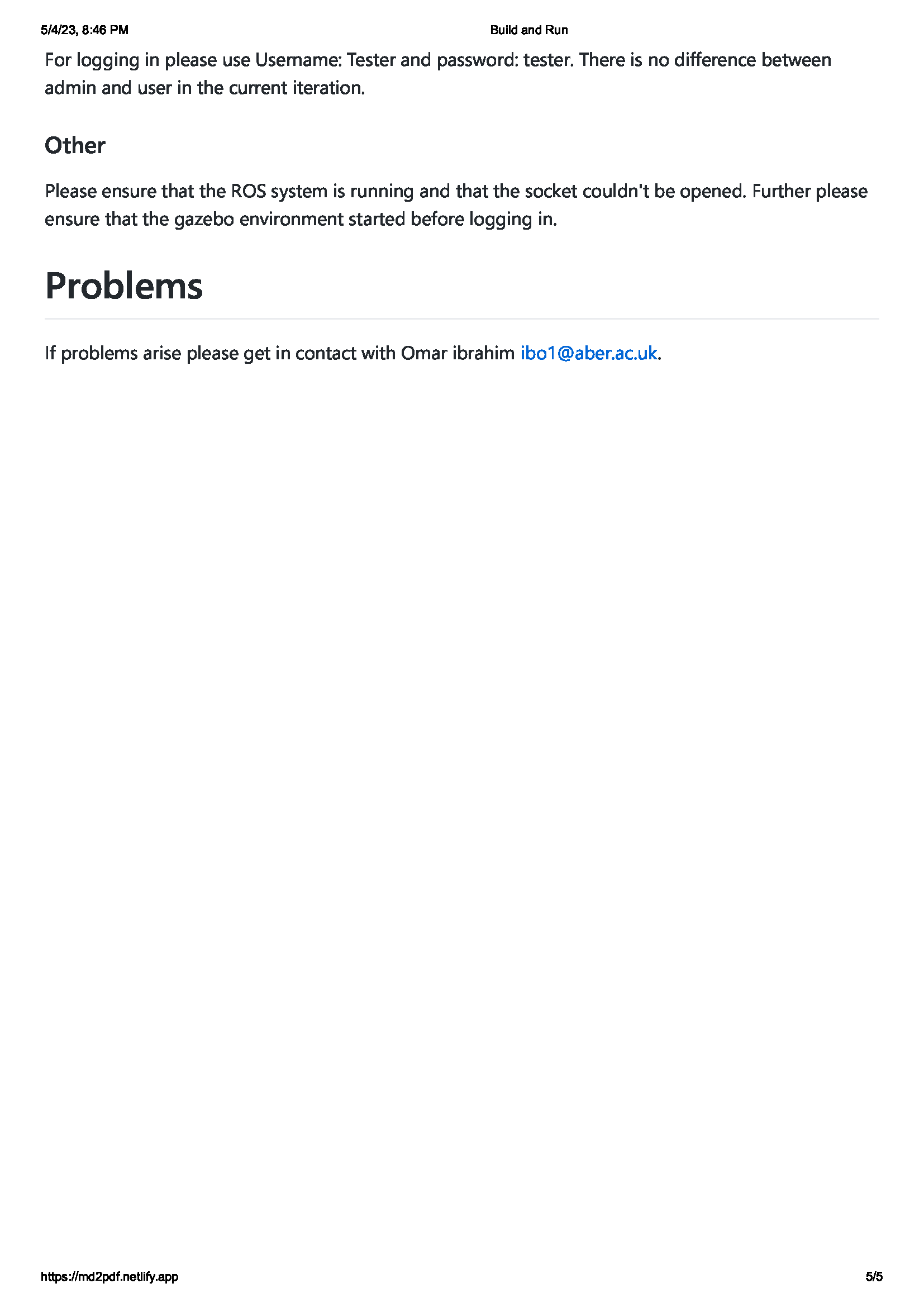


* 1. Build and Run

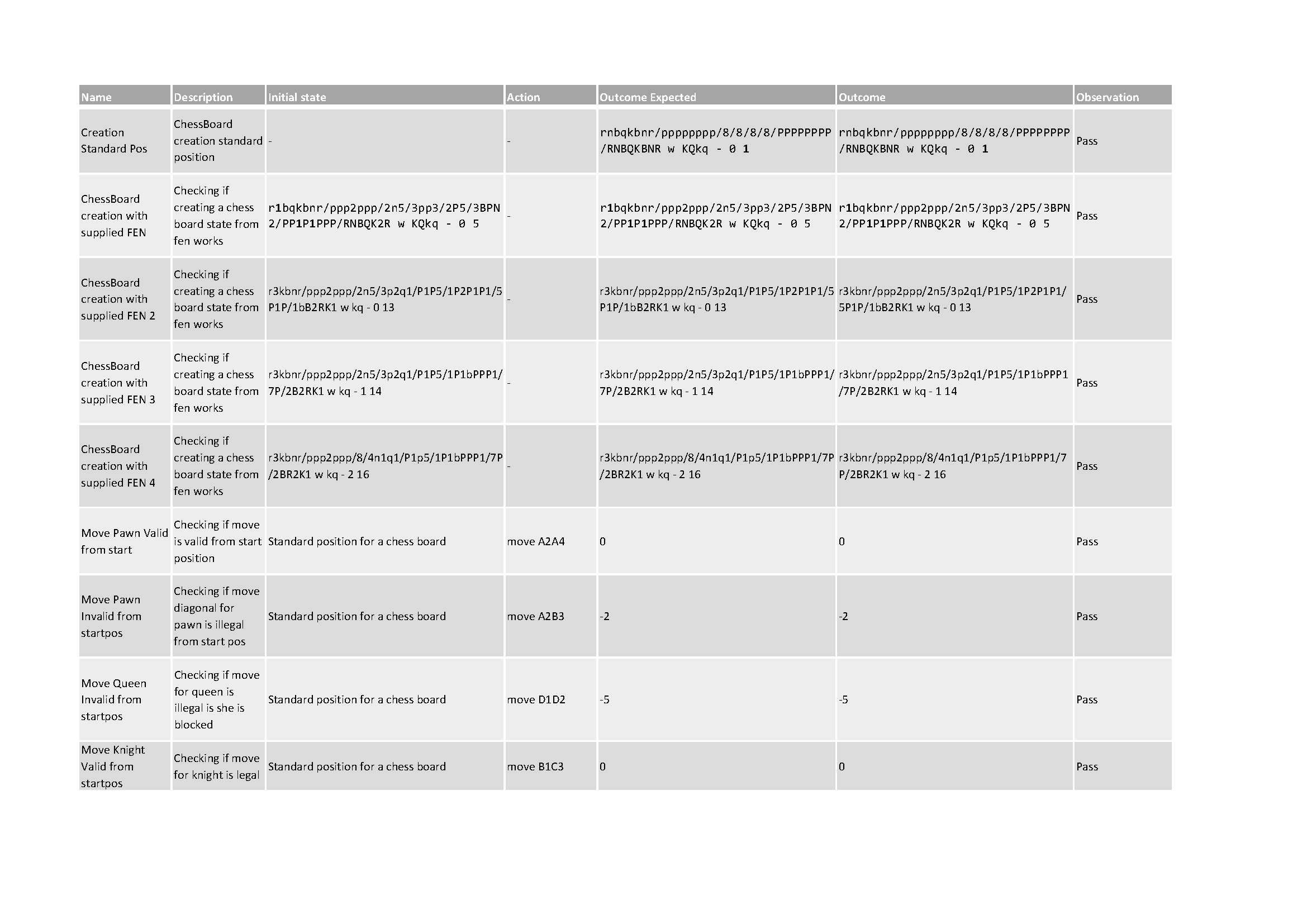


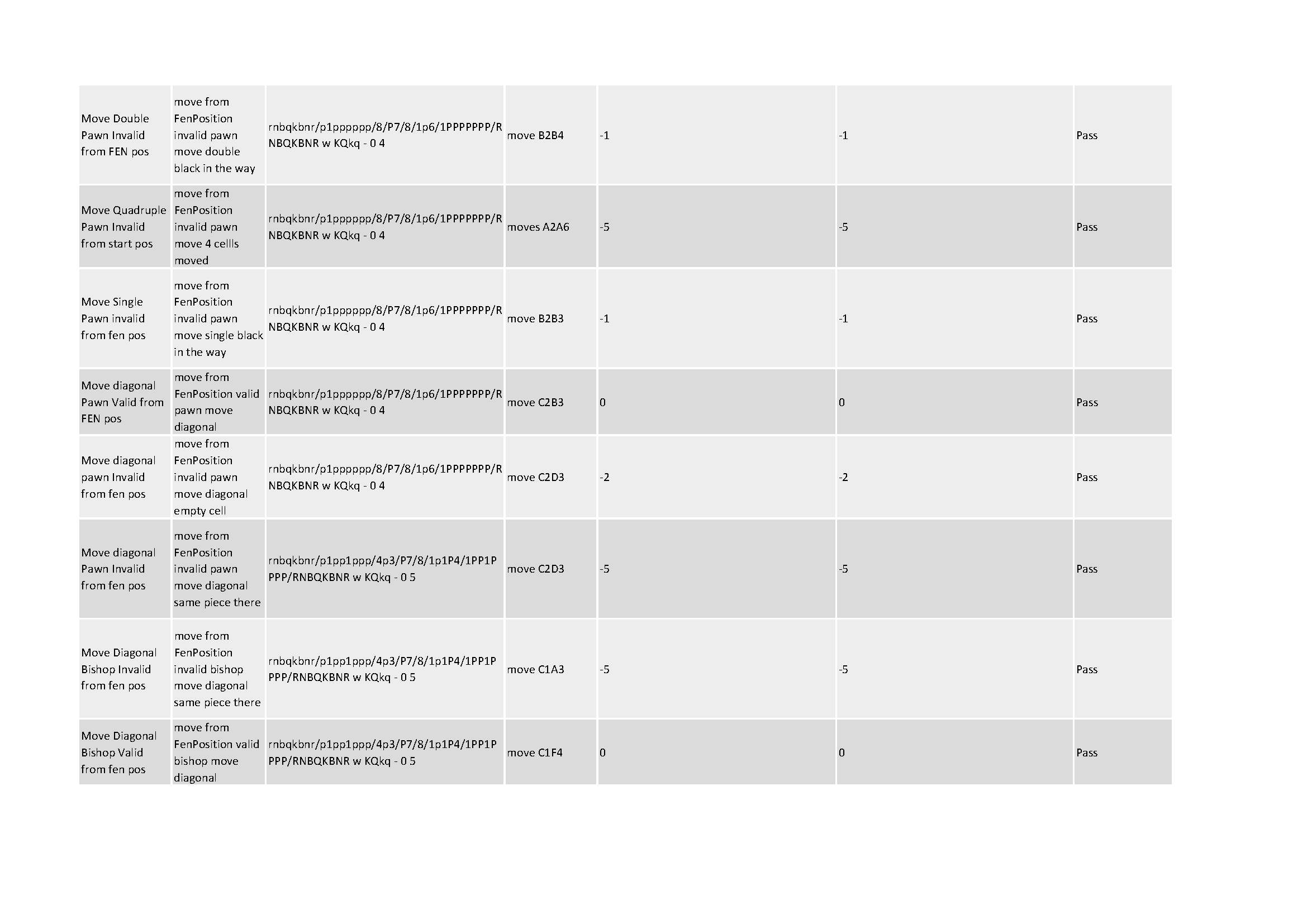


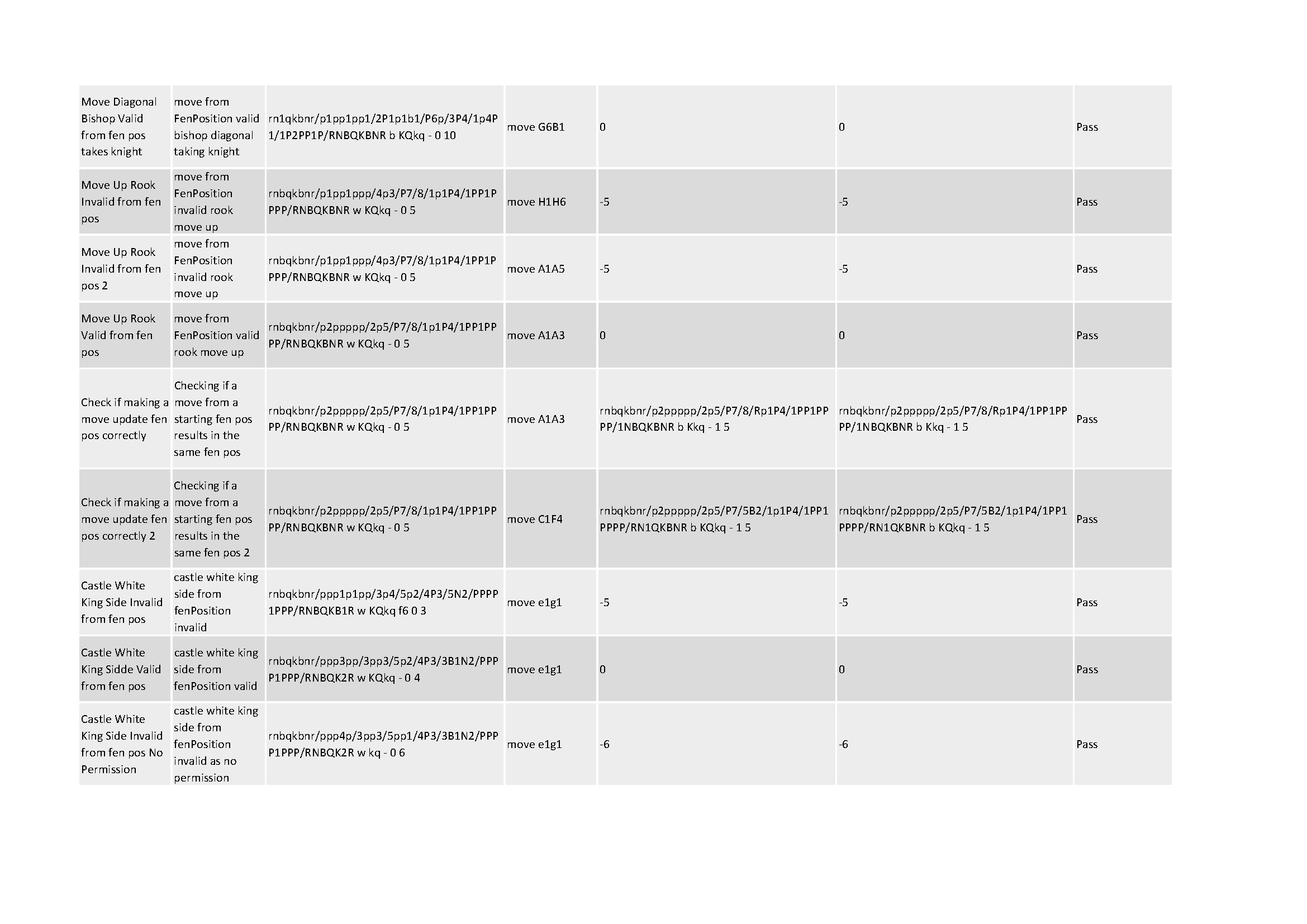


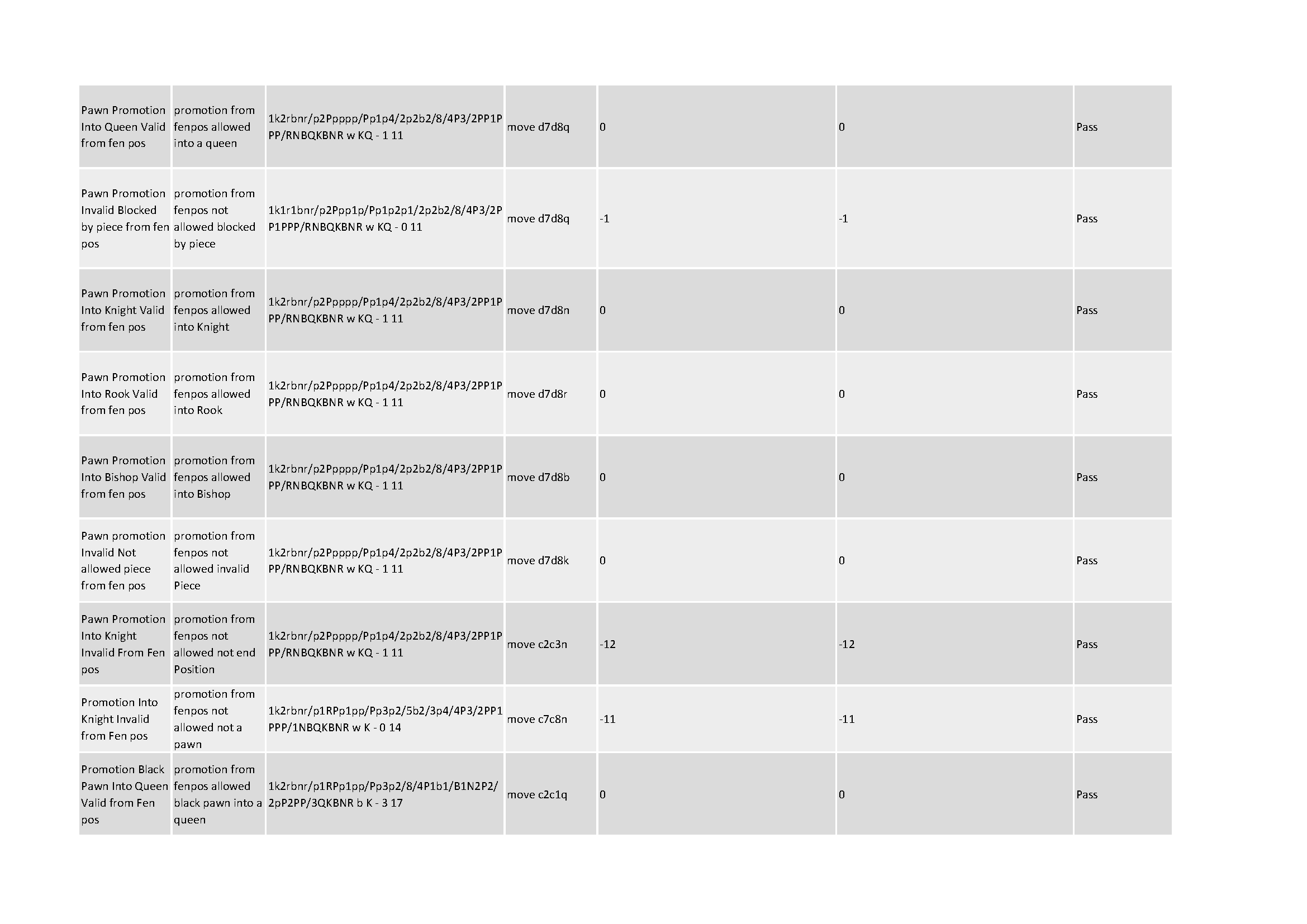


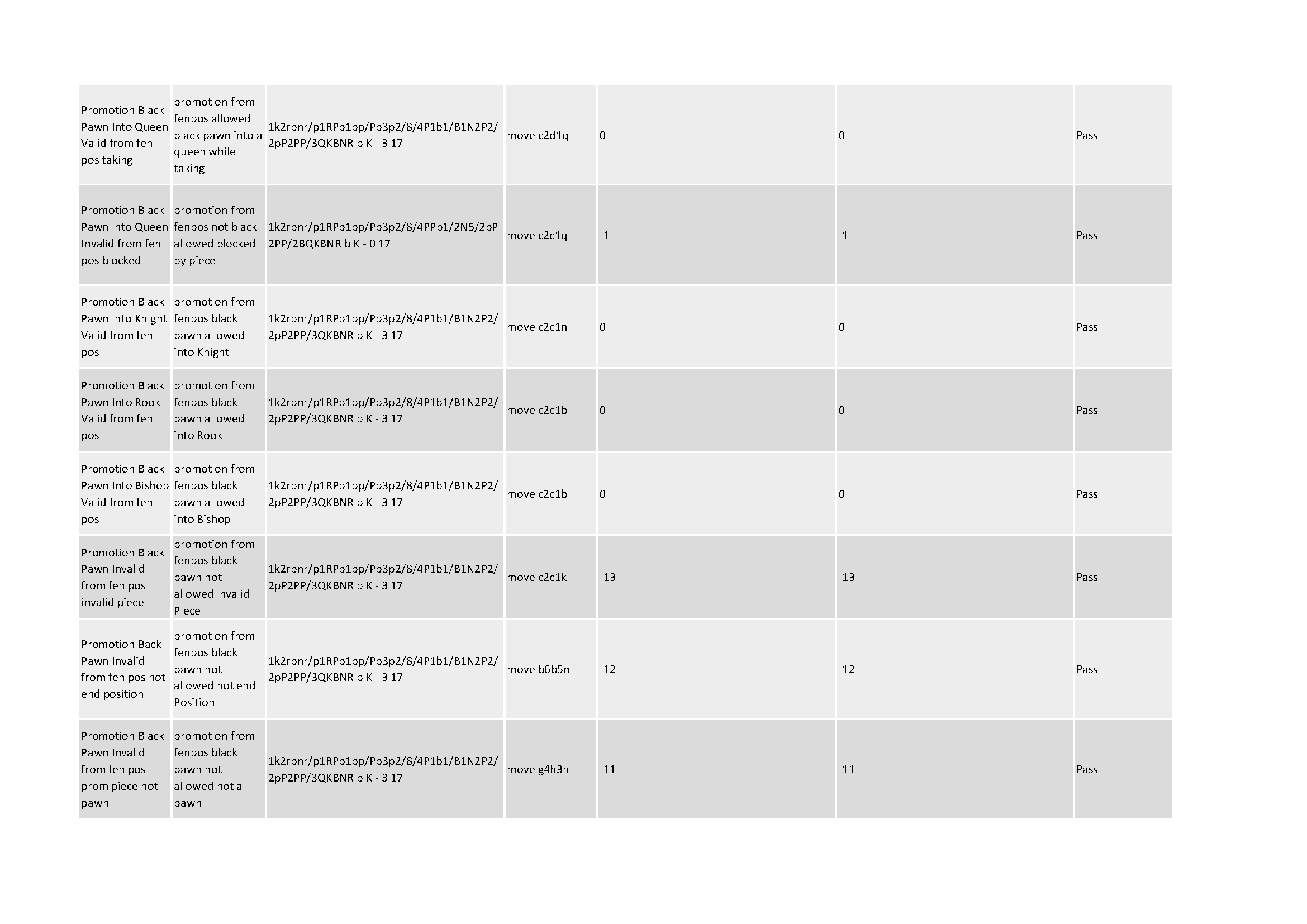
* 1. Unit Test Table
     1. ChessBoard test table

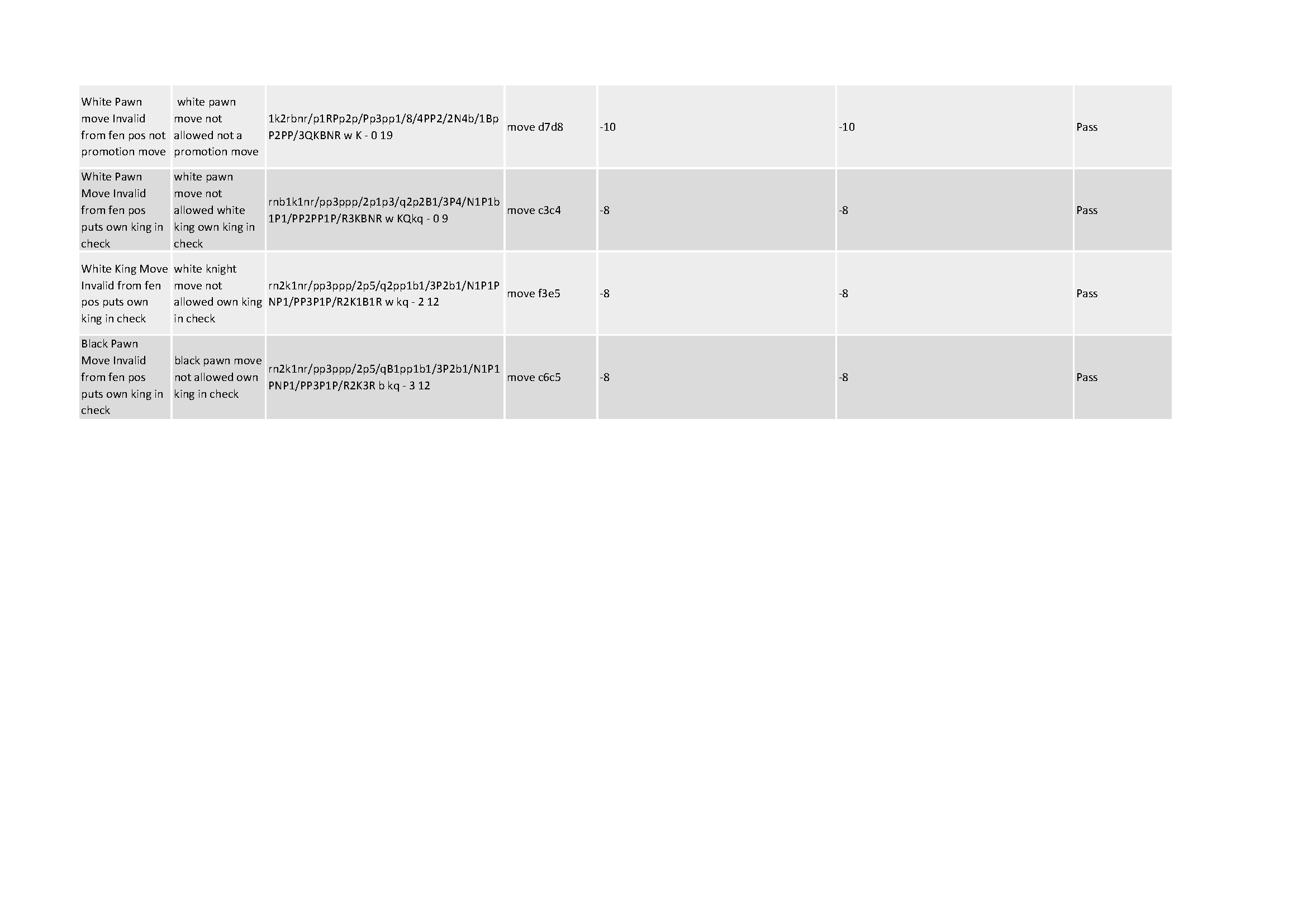




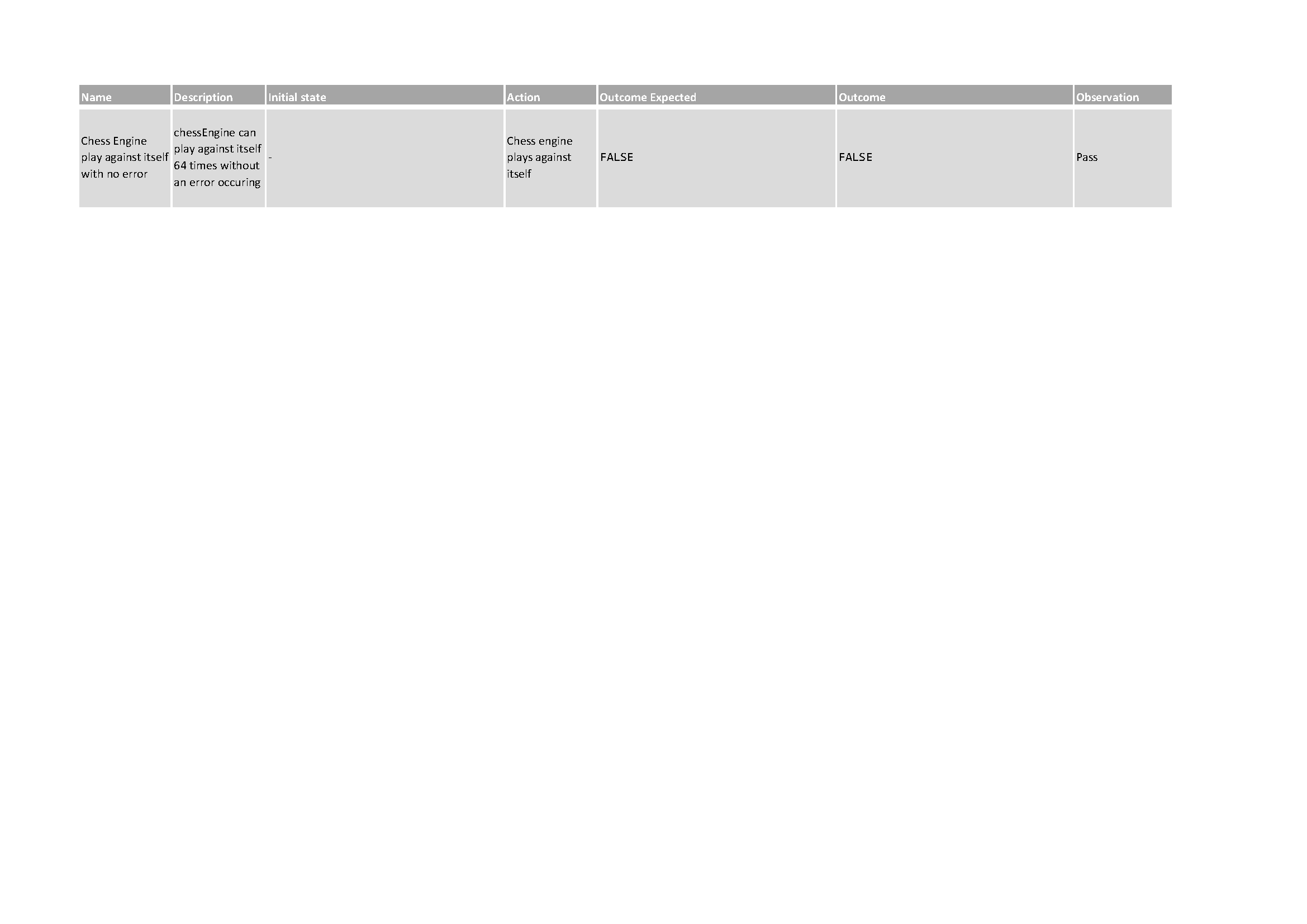




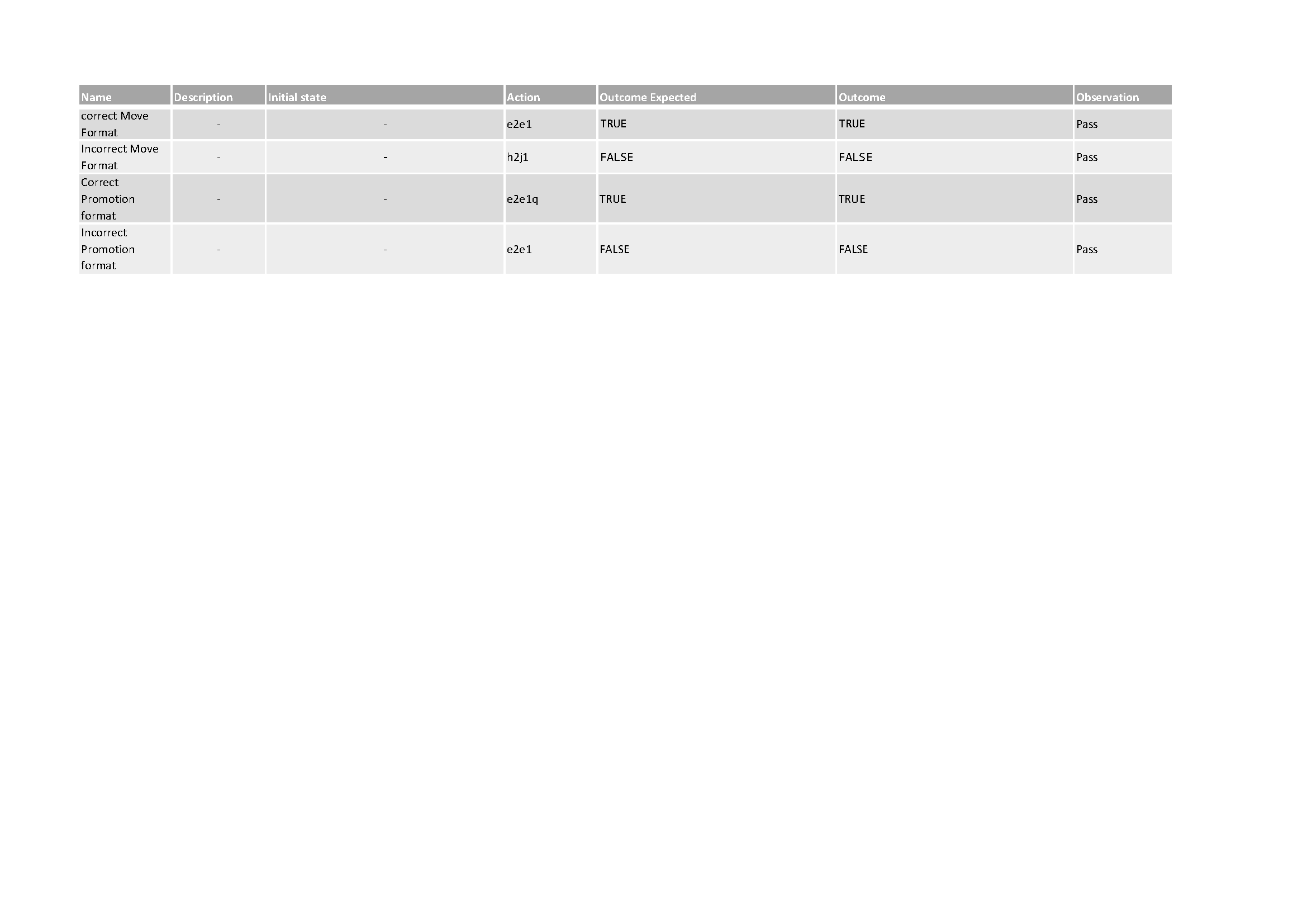




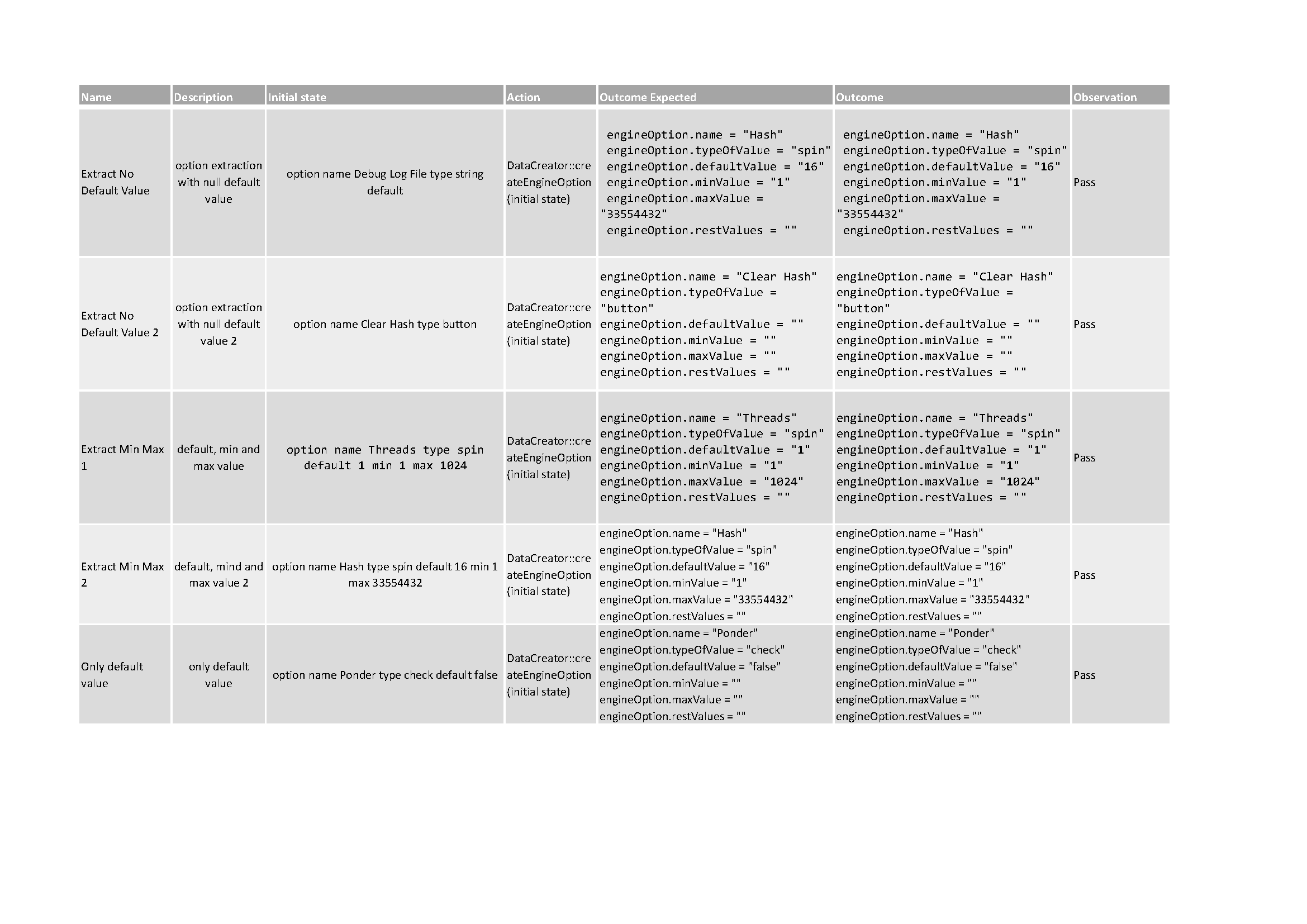
* + 1. Chess engine test table

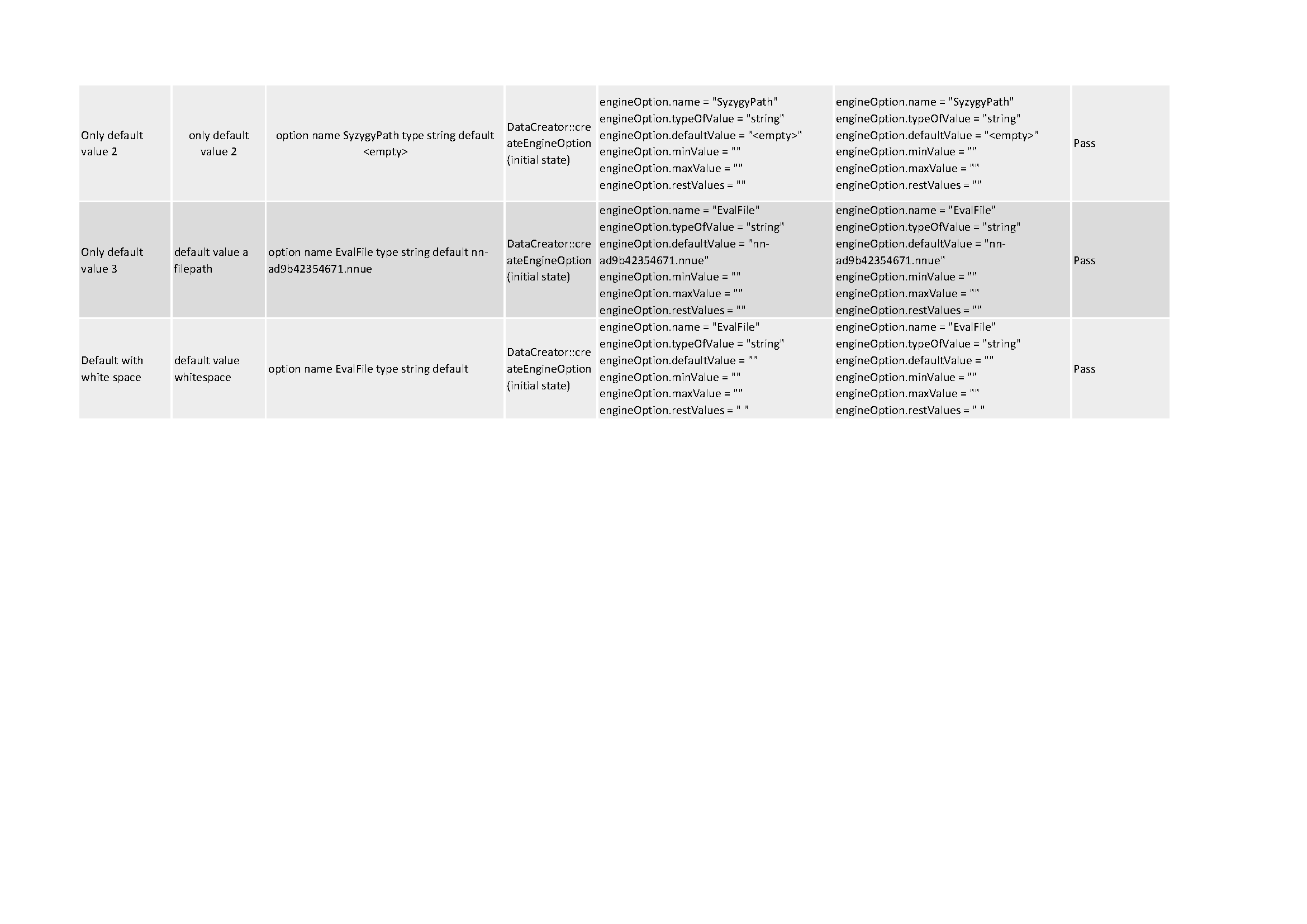


* + 1. DataChecker test table



* + 1. UCIHandler test table





* 1. Manual Test Table

