Department of Computer Science
The University of Hong Kong

COMP4801
Final Year Project

[MakerLab Project] Pick and Place Game App for 3D Printed Robotic Arm

Final Report

Team member
Wong Ka Ngai, Benny
UID: 3035568881

Wan Tsun Wai, Alan
UID: 3035569017

Supervisors
Dr. T. W. Chim
Mr. David Lee

Date of Submission: 18/4/2022

Abstract

AI versus human players in games has been an increasingly popular topic, especially after the victory of AlphaGo. This project aims at developing a Connect Four AI for STEM education that utilizes a mobile application and a robotic arm to play with human opponents. In order to accomplish the objective, multiple software and hardware tools and methods were employed to design the system workflow of the Android application. Furthermore, experiments were conducted to select the best approaches. Based on the results, computer vision with OpenCV circle and color detection was used to recognize the board, and an optimized minimax algorithm with Alpha-Beta pruning was implemented to calculate the next best move. With the completed product, players have an advanced Connect Four gaming experience by playing against the perfect AI, which can be utilized in STEM education by demonstrating the strength of AI in making decisions and recognizing objects. However, with the limitations in the application and robotic arm, the product can further be improved to enhance usability and gaming experience in the future.

Acknowledgment

Our team would like to express our sincere gratitude to Dr. T.W. Chim and Mr. David Lee for spending their time guiding us and giving us comments throughout the development of the project. Without their support, our team would struggle a lot in making progress. Additionally, our team would like to thank the Department of Computer Science, Tam Wing Fan Innovation Wing and HKU CS MakerLab for offering this project's necessary resources and equipment. The project cannot be finished without their assistance.

Table of Contents

List of Figures

List of Tables

Abbreviations

3D	Three-Dimensional
AI	Artificial Intelligence
App	Application
DOF	Degree of Freedom
G-code	Geometric Code
GUI	Graphical User Interface
HSV	Hue, Saturation, Value
IDE	Integrated Development Environment
IOS	iPhone OS
RGB	Red, Green, Blue
STEM	Science, Technology, Engineering, and Mathematics
UI	User Interface
UX	User Experience

1. Introduction

The following chapter gives an introduction to this report. First, a brief background on AI and the robotic arm is given. After that, the motivation for doing this project is discussed, followed by the objective and deliverable. Then, the literature review is presented to study related works. Finally, the contribution of members and an outline of the report are mentioned.

1.1 Background

Early in 1997, a chess computer Deep Blue defeated the world champion Garry Kasparov, marking a symbolic testament to the rise of AI [1]. Since then, AI versus human players in games has become widely researched. In 2017, an AI player AlphaGo defeated the best human player Ke Jie in the game GO, again demonstrating AI's potential in playing games [2]. In order to translate the AI decision into action, very often a human agent in between is needed. A robotic arm can be utilized to automate this process. The use of robotic arms can be traced back to 1954 when George Devol invented the first industrial arm [3]. Unlike human beings, robotic arms were able to perform with high precision and work for long hours. However, they could only follow a predefined moving path and operate without real-time calculations.

With AI algorithms and computer vision, robotic arms in the current decade can learn to make decisions without following the predefined rules by humans. As such, robotic arms can now be put into more complicated working environments like medical use. It is possible to utilize robotic arms in decision-making games to play against human players with modern powerful chips.

1.2 Motivation

Connect Four is a two-player connection board game published in 1974 [4]. A standard Connect four game contains a vertical board with seven-column and six-row, 21 yellow discs and 21 red discs for two players. Players alternate turns to drop one disc from the top into any of the seven slots. The player wins if four of his discs connect to a vertical, horizontal or diagonal line (see Figure 1). The drawing condition only occurs when the board is complete and no player can fulfill the winning condition mentioned above (see Figure 2).

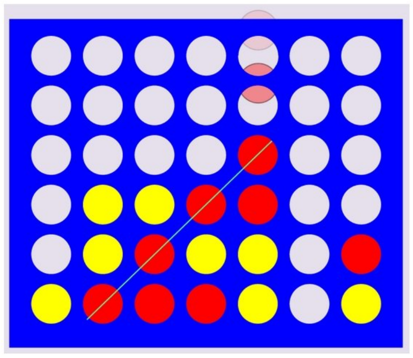


Figure 1: An example of a winning condition, where four red discs connect to a diagonal line

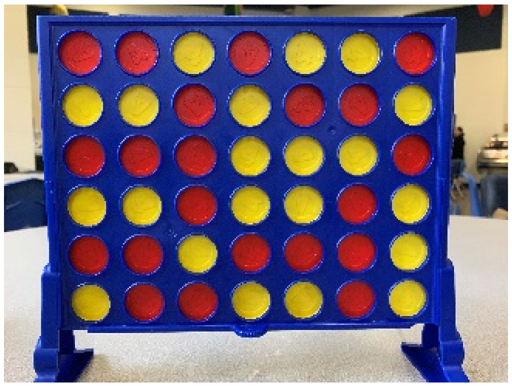


Figure 2: An example of a tie game, where the board is full and no lines can be formed

Connect Four is simple to play but hard to master. It is no easy task to make an AI Connect Four robot as the game requires the robot to be precise in picking and placing and have strategic thinking at the same time. The motivations for doing this project are illustrated below.

Connect Four is unique in itself that unlike most board games, its board is vertically placed. In order to successfully place a disc inside the board, precision control of the robotic arm is crucial, attributed to the minimal gap of the column slot. Also, it is a common practice for horizontal board games to keep the mobile phone at a fixed distance to capture the board. However, as Connect Four has a vertical board, the distance between the phone and the board may vary to calibrate correctly. It adds extra difficulties to our project.

In addition, Connect Four is a solved strategy game where the first player has an unbeatable winning strategy. This game was independently solved by James Dow Allen and Victor Allis in 1988 [5]. In theory, it is possible to build an unbeatable AI for this project. Therefore, it is believed that the AI player will be a worthy opponent for human players. Unfortunately, in comparison with unsolved games like chess, solved games like Connect Four are less popular to study. As a result, there is no existing library to use. Development in a library specializing in playing Connect Four is needed for this project, which should fill up the gap of missing libraries. It is also hoped that this project can bring some extra insights and discussions to the solved game community.

Last but not least, after finishing this project, the robot can be displayed in public spaces to play with people for entertaining and educational purposes. Unlike other complicated games, connect four is easy to pick up and everyone can enjoy it even if they are just children. Furthermore, students' interest in computer science and AI can be aroused by playing with the unbeatable AI. The rationale behind the perfect AI can be introduced to players after the game. Despite the fact that they may not fully understand the theories, they can still be impressed by the power of AI and acknowledge AI's contribution to everyday life. It is believed that this project can be helpful for STEM education.

1.3 Objective and Deliverable

This project proposes to develop a mobile application for a 3D printed pick-and-place robotic arm that can play Connect Four with human opponents. The project can be broken into two parts: a mobile application for the AI player and a fine-tuned robotic arm. The mobile application should be able to capture and analyze the board. Then, it calls the AI to calculate the next move. If possible, the AI constructed should be perfect for computing optimal moves. Next, the command and position data are sent to the robotic arm. The application should also have an easy-to-use graphical user interface that minimizes the user's learning time. Furthermore, the robotic arm should be able to receive the command from the application via Bluetooth and then pick and place discs quickly and accurately.

With the finished mobile application and robotic arm, this project should be able to deliver a mobile application that plays Connect Four with human opponents by utilizing the robotic arm and AI. A ready-to-use Connect Four AI library should also be developed. In addition, the final product should demonstrate the strength of AI algorithms in decision making and object recognition so that the product can be utilized for STEM education.

1.4 Literature Review

There are a plethora of Connect Four game applications in the app market. In particular, "4 in a row King" and "4 in a row" are studied, as they are the most and second most downloaded Connect Four games in Google Play Store, with 10M+ and 5M+ downloads respectively [6][7]. They offer a similar gaming experience, in which single-player, local two-player and online multiplayer modes are provided. In single-player mode, different levels of AI can be chosen. However, none of the AIs can perfectly solve Connect Four, and they can be defeated if the human plays optimally (see Figure 3). Their developers did not reveal the principles behind the AIs and it is impossible to know what AI techniques or algorithms are utilized in their apps.

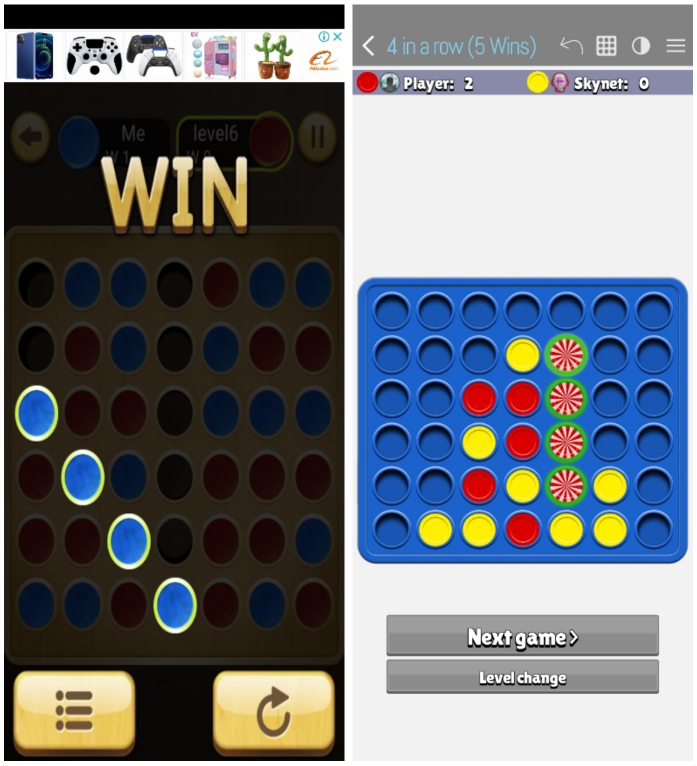


Figure 3: Two Connect Four apps with sub-optimal AI

Solving Connect Four is widely researched in the AI field. Pascal Pons's approach utilizes a minimax algorithm with Alpha-Beta pruning to exhaustively search for all possible states [8]. He implemented the perfect AI into a web-based online Connect Four solver [9]. Besides the standard gameplay, extra information like the predicted number of moves remaining and minimax scores of board states are provided in the online solver. In addition, steps to build the perfect AI are also revealed in his blogs for educational purposes. After reviewing the tutorials, it is believed that the perfect AI built can be adopted for this project and utilized for STEM education.

1.5 Contribution of Members

Table 1 illustrates how work is distributed in the project team.

Software	
• Mobile Application	
o Frontend	
▪ UX design	Benny
▪ UI design and implementation	Benny
o Backend	
▪ Bluetooth connection	Benny
▪ Cloud database	Alan
▪ Camera	Alan
▪ Player VS AI Mode	Alan
▪ Arm Controller	Benny
▪ Building Mode	Benny
▪ Score Board	Alan
• AI	
o Board detection	Alan
o Disc detection	Alan
o Connect Four AI	Benny
Hardware	
• Robotic arm enhancement	Benny
• 3D modeling	Alan
• 3D printing	Benny
• Laser cutting	Alan

Table 1: Contribution of members

1.6 Report Outline

The remaining paper is organized as follows. Chapter 2 describes the methodology involved in this project so that the robotic arm is able to pick and place discs automatically. Chapter 3 then discusses the experiments conducted in this project and their respective results. Chapter 4 follows by stating the limitations of deliverables and possible future undertakings. A short conclusion of this project is provided in Chapter 5.

2. Methodology

To achieve the aforementioned objectives, the proposed product of this project is divided into two major components, namely the software component and the hardware component. Chapter 2.1 demonstrates the overall design of this project, including the justifications for utilizing various tools for the software and hardware component, the UI and UX design of the application, and the system workflow.

Chapter 2.2 elaborates on the algorithms and methods implemented in the mobile application for the software component, detailing the object detection algorithm, the Connect Four AI algorithm, and the scoring system. The three game modes provided by the application, the Player VS AI Mode, Arm Controller and Building mode, are introduced in Chapter 2.3.

For the hardware component, in order to adapt to this project, modifications to the robotic arm are necessary. The enhancements made to the robotic arm are introduced in Chapter 2.4. Chapter 2.5 illustrates the disc stacker used in this project to facilitate picking and placing discs.

2.1 Design

The design choices made in this project are introduced in this section. Chapter 2.1.1 explains the rationale for utilizing Android Studio, Firebase and Git. Chapter 2.1.2 details the tools used in hardware development, including Arduino, Shapr3D, SolidWorks and CorelDraw. Chapter 2.1.3 describes the UI and UX design of the application. The system workflow design is presented in Chapter 2.1.4.

2.1.1 Software

Various tools are utilized during the development of the mobile application. They are illustrated below.

An integrated development environment (IDE) is a software application that provides comprehensive facilities, allowing computer programmers to develop software for a particular platform [10]. Android Studio is the IDE chosen to develop the mobile application in this project. Android Studio has provided numerous support to its developers since its first release in 2014 [11]. It offers a plethora of packages for developers, of which the OpenCV package applies to this project. Compared to other mobile operating systems, more than 70% of mobile phone users use Android [12]. It is believed that development in Android matches most users.

The implementation of the scoring system makes use of the Firebase Database. It is a cloud database solution provided by Google, which is fast and easy to set up [13]. Firebase is also compatible with Android Studio, and the provided packages foster the development of client-server interaction. Furthermore, compared to using a local database like Room or SQLite, Firebase ensures data consistency across different Android devices. Thus, it is believed that using Firebase is a better solution for this project.

Github and Git extension are used for version control, continuous integration, and deployment to facilitate collaboration. Attributed to the scale of this project, branching workflow is deployed instead of trunk-based development. The team members develop in the development branches and are merged to the main branch only when both a feature is finished and a consensus is reached by team members. Github provides the online code collaboration platform, while Git Extension provides a visualization of branches, fostering merging and version control. Collaborations can be done remotely with them.

2.1.2 Hardware

The objective of the hardware component, which is the robotic arm, is to execute the command received precisely. The Arduino program installed on the microcontroller board is responsible for receiving the application's commands and instructing the robotic arm to perform the corresponding actions. The official Arduino IDE and Visual Studio Code are used together to do Arduino programming. The Arduino IDE provides a simple interface for compiling and uploading programs, while Visual Studio Code provides advanced functions like autocomplete that enrich the coding experience. Both IDEs are utilized in the development of the Arduino program.

The claw gripper cannot pick up laying down discs and place them inside the vertical board. To allow the robotic arm to grip a disc, a disc stacker is needed for storing the discs so the robotic arm can pick up discs vertically at the same position. Since it is a tailor-made disc stacker, the 3D model is designed using Shapr3D. It is a free 3D modeling software that allows users to design intuitively without much experience. Prototype models are printed and tested using Shapr3D, Ultimaker Cura and 3D printers. After testing the prototypes and identifying their flaws, laser cutting is chosen to produce the disc stacker. To utilize the laser cutting machine, related software SolidWorks and CorelDraw were used to prepare sketches for cutting.

2.1.3 UI and UX Design

The UI is developed based on the UX hand-drawn mockup. The objective is to design a simple, responsive, task-oriented application that puts users in control. The design principle of this application is to create a modern, minimalistic user interface that is intuitive to use. In this chapter, the design aesthetics will first be introduced with UI examples. An explanation of UX design with a user flow diagram will also be given.

The app's color theme followed suggestions from Material Design [14], which is designed to be harmonious, ensure accessible text, and distinguish UI elements and surfaces from one another. Teal and pink were chosen as the primary and secondary colors as they are the colors of the Connect Four discs (see Figure 4).

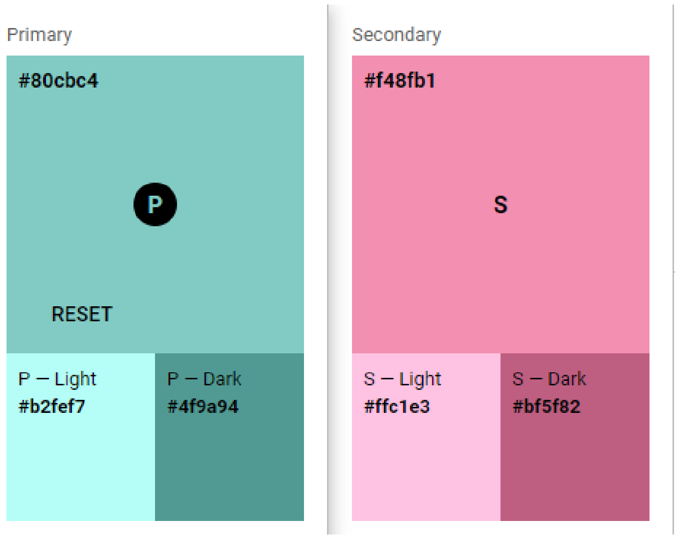


Figure 4: The color palette

It is assumed that in default, players use teal discs to play against the robot, although players can choose their color in practice. Thus, teal is chosen as the primary color. Variants of primary colors are utilized to distinguish UI elements. For example, the system bar used the dark variant to create contrast with the top app bar. The secondary color pink, which is also the representative color of the robot, is used in selection controls. For example, pink is used to differentiate the robot button from the player button when selecting the first player. Figure 5 demonstrates the use of the primary color variant in the system bar and secondary color in the robot button.

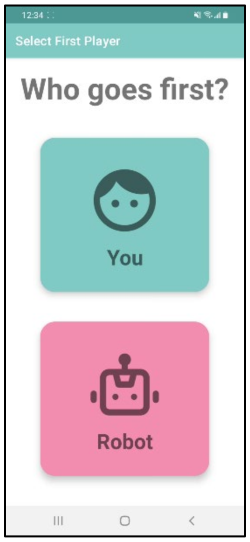


Figure 5: Example of color usage

Additional colors are also chosen to convey different categories. For example, blue is chosen as the tertiary color as it is the color of the game board. Different levels of grey are also utilized for backgrounds. In order to keep the color theme harmonious and consistent across the application, all colors used are chosen from the material design color palettes and share similar tint levels. As the design aesthetic uses light colors, the base of teal, pink and blue are chosen to be at level 200. When darker variations are needed, the level varies from 300 to 700.

Gradient colors are also employed to create the modern minimalistic feel of the application. Horizontal linear gradients are utilized in two ways in the app. The first way is a gradient of color from light to dark, and the other way is the gradient of primary and secondary colors. UI elements are more natural and attractive by employing gradients [15].

Modern UI designs often feature rounded corners as psychological studies showed that rounded corners are more appealing than sharp corners, which are considered harmful [16]. As a result, all UI elements in the app have rounded corners with different radii. In addition, some elements like CardViews also make use of shadows to be more eye-catching. Figure 6 illustrates the use of gradients, rounded corners and shadows.

The font used in this app is the default Roboto font, which applies to the modern minimalistic design aesthetic as it is sans-serif. In an attempt to match the light color theme of the app, the default text color is lighter than black. Black text is only used with capitalization to emphasize buttons. Depending on the background color, white text is also used for readability. Figure 6 also demonstrates the different use of text colors.

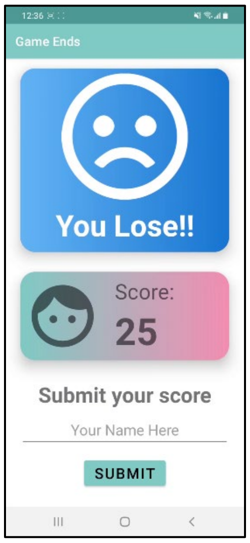


Figure 6: Example of UI elements

To maintain the minimalist app theme, no image is used across the whole application. Instead, all graphics are vector assets constructed with simple lines and circles, as shown in Figures 5 and 6. The only image asset is the app icon, which is an adaptive icon designed using Figma. The foreground and background of the icon are designed according to the rules mentioned above. In addition, gradients, shadows and rounded corners are utilized in the icon (see Figure 7).

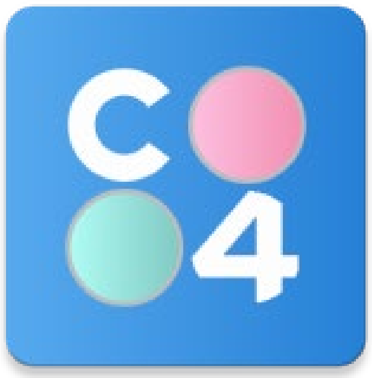


Figure 7: The app icon

UX design aims to identify and solve user problems. Users should find the app enjoyable and straightforward, enabling users to effectively and efficiently achieve their objectives. In this project, it is predictable that players' objective is to access the three game modes and play with the robotic arm. However, it is a prerequisite for the app to connect with the robotic arm via Bluetooth to use the game modes. This prerequisite is accomplished by disabling all game mode buttons except for the Bluetooth button at the home activity if no connection is detected. Their text color is dimmed to grey to show that the three game modes are disabled (see Figure 8). During development, the project team identified the user's desire to check the rankings of Player VS AI Mode. Thus, the final product added an extra scoreboard button on the homepage that can be accessed without a Bluetooth connection.

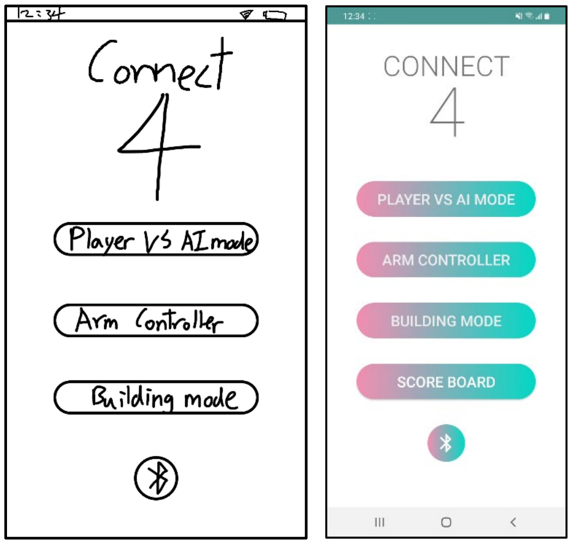


Figure 8: Mockup and UI of home activity

When the Bluetooth button is clicked, the Bluetooth activity is launched. Users press the scan button to scan nearby devices. When the robotic arm is discovered, users click the item to establish the connection (see Figure 9). The app returns to the home page, and toast messages "Connecting" are shown. The toast message "Connected" is shown once the connection is constructed and all buttons are enabled. The Bluetooth button icon changes to disable Bluetooth so that users can click it again to disconnect (see Figure 10). The toast message "Connecting fail!" will be shown if it fails to connect. If the mobile device has been paired with the robotic arm before, a Bluetooth connection will be automatically established once the app is launched. This feature provides a better, more satisfying user experience by preventing rework.

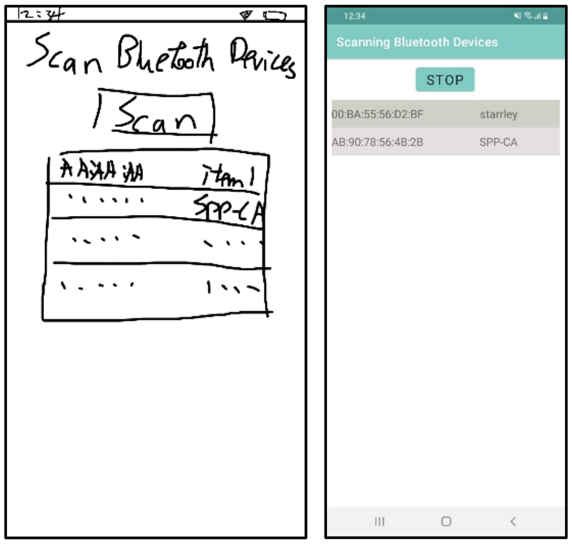


Figure 9: Mockup and UI of Bluetooth activity

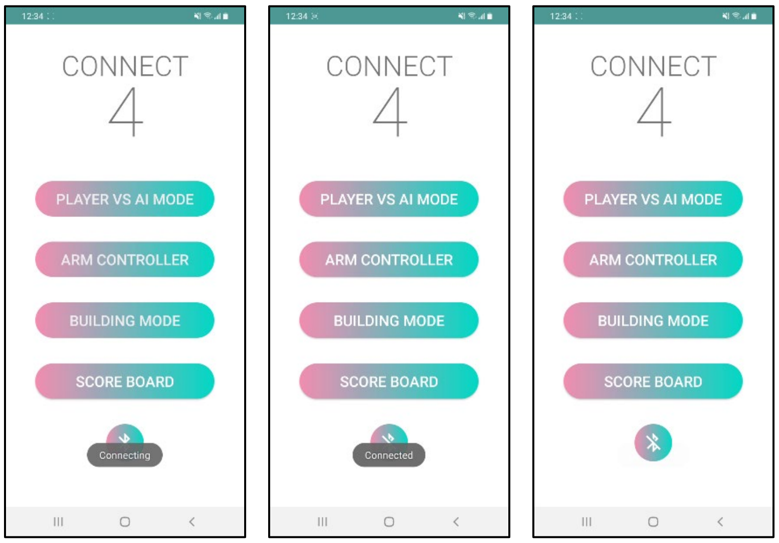


Figure 10: Process of Bluetooth connection

Figure 11 denotes the user flow of the application. The designed UX matches the usefulness, findability and usability in the UX honeycomb proposed by Peter Morville [17]. This simple structure provides high findability and usefulness, as the home page provides everything needed. The design is also usable. In an attempt to make the app learnable, the flow is designed to be straightforward and easy to follow, such that users can understand how to use the app for the first time. It is forgiving that users have a careful understanding of task flows with the instruction page in Player VS AI Mode, which minimizes user errors. Even if the user fails to capture the board after the instruction, the refresh button allows the user to reposition the phone and recapture the board without redoing everything. The app is satisfying in that the mentioned Bluetooth connection conveys connection status through toast messages and prevents rework by auto-connection. The error of an unexpected Bluetooth disconnection is also handled. If Bluetooth suddenly disconnects, all games cannot proceed. These activities will receive a broadcast message of disconnection and they will end themselves. The home page will resume to the foreground such that users can establish a connection again. Details of the three game modes will be discussed in Chapter 2.3.

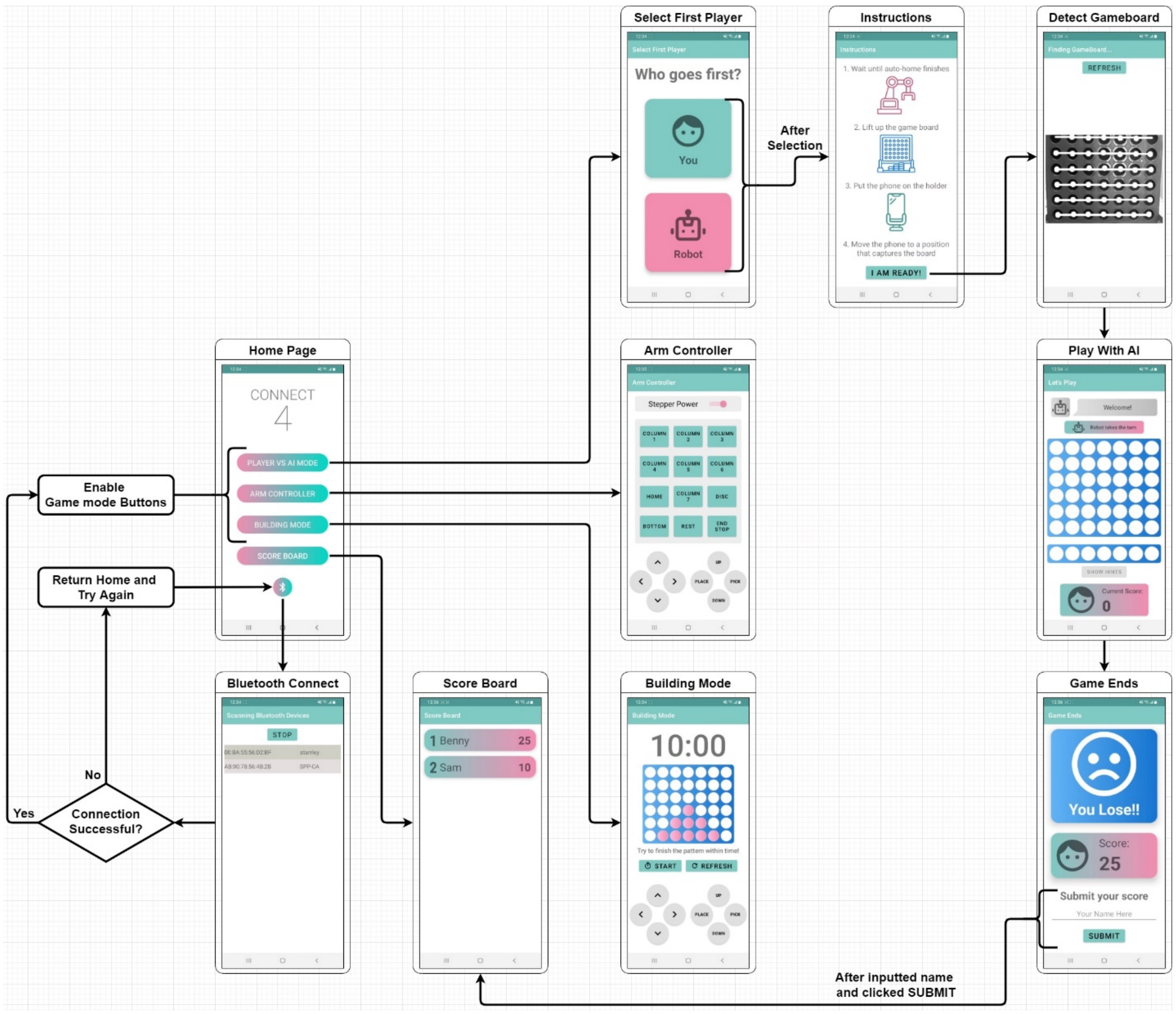


Figure 11: User flow diagram of the app

2.1.4 System Workflow

The Connect Four game system workflow of this project is demonstrated in Figure 12. Developed in Android Studio, the game app first accesses the mobile phone camera and captures the Connect Four game board images using the OpenCV library. Next, they are sent to the image processing functions. After preprocessing, the OpenCV circle and color detection algorithms perform calculations to identify the board state. Finally, the board state is temporarily stored in the application.

Android Studio utilizes Java for application development, and the board state cannot be sent directly to the Connect Four AI algorithm written in C++. Instead, Java Native Interface (JNI) is utilized to interact with native C++ code. It translates Java data types into C++ data types. By configuring with CMake, the C++ AI algorithm can be compiled for the application. The algorithm also utilizes an external book loaded into the transposition table to reduce computation time. The returned result is translated back into Java format through JNI and the application gets the optimal moves for the board state.

The application sends the corresponding G-codes to the Arduino microcontroller board via Bluetooth with the optimal moves obtained. The board then instructs the stepper motors and servo motor, controlling the robotic arm to pick and place a disc into the optimal column. The above process is repeated until the game ends.

When the game is finished, the player sends their name and game score to Firebase via the Internet. Firebase adds this new record to the cloud database and ranks all records in descending score order. The ranking is returned to the application and the player knows how well he/she performed compared to other players.

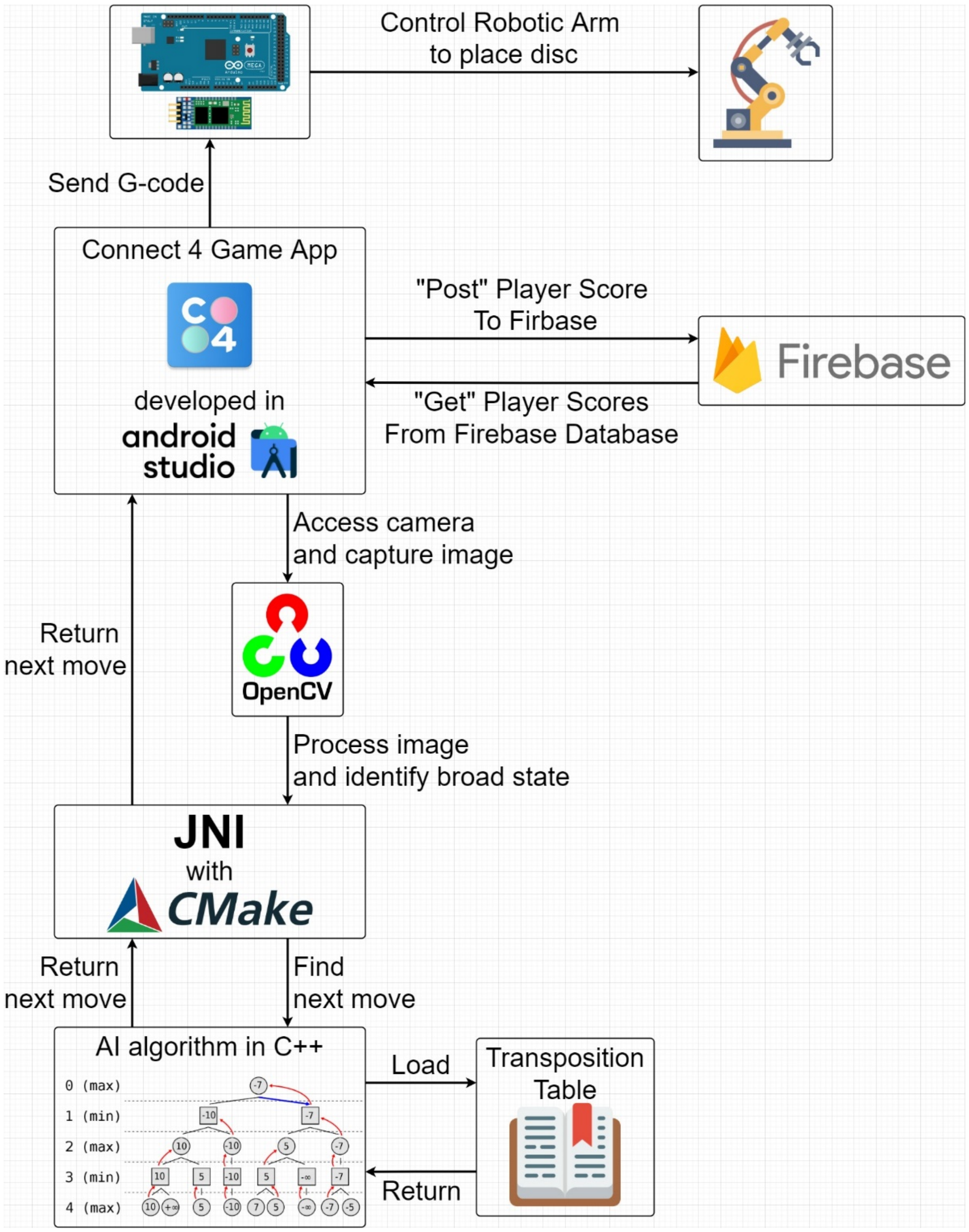


Figure 12: System Workflow diagram

2.2 Software – Mobile Application

This chapter introduces the algorithms and methods to develop the Connect Four game app. Chapter 2.2.1 details the methodology of board recognition using the OpenCV library. Chapter 2.2.2 briefly describes how the perfect AI constructed continuously determines the optimal move to play using the minimax algorithm with Alpha-Beta pruning. Finally, Chapter 2.2.3 explains the rationale behind the scoring system to evaluate players' performance.

2.2.1 Board Detection

Computer vision, a field of AI, derives meaningful information from visual inputs like images. It enables computers to observe and understand. The mobile application will adapt computer vision to recognize the board. To accomplish this, the OpenCV library will be used. OpenCV is a powerful open-source computer vision library that provides optimized algorithms to accomplish various computer vision tasks, including but not limited to object identification, face recognition and model extraction.

The procedure of Connect Four board detection works as follows. First, the board image is captured by the phone camera and stored in OpenCV Mat format, which is a matrix form commonly used for images. Then, trimming and blurring are applied to the image for noise reduction. After preprocessing, background noises are removed, leaving only the board. It then passes to the OpenCV HoughCircles() function to detect the 42 circular slots of the board [18]. The function finds not only the circle but also the center. Coordinates found in an image snapshot are stored in an array to increase accuracy. It is considered a valid circular slot if a circle with the same coordinate is detected in multiple snapshots. A valid row is identified with seven valid centers with similar y coordinates (see Figure 13). When the algorithm gets six valid rows, it successfully identifies the board.

With the 42 coordinates, the remaining work is to identify the change of color in these coordinates for disc detection. Instead of RGB, HSV is used to identify discs as hue better recognizes colors at different brightness levels. Hue values at the coordinates are obtained to check for teal or pink color. With this method, disc detection with high accuracy can be achieved.

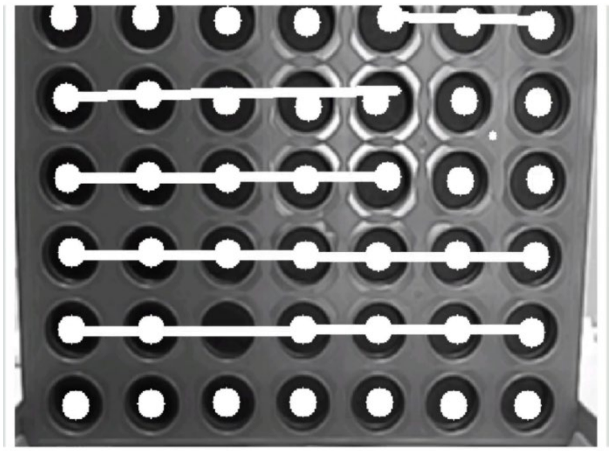


Figure 13: Process of identifying valid rows

2.2.2 Connect Four AI Algorithm

With the current game state obtained, the application calculates the next move and sends the corresponding command to the robotic arm. In this project, a minimax algorithm with Alpha-Beta pruning will be used, as minimax is widely used in two player turn-based games. In minimax, the two players are called maximizer and minimizer, where the maximizer tries to attain the highest possible score and the minimizer does the converse [19]. The algorithm is contingent upon the minimax tree, which is similar to the decision tree. Each layer of the tree consists of either maximized nodes or minimized nodes. A node represents a board state which has an associated value. The algorithm traverses the tree and chooses the path that obtains the optimal result (see Figure 14). As Connect Four is a zero-sum game, where a player's loss equals another player's gain, minimax can be further simplified to negamax. Only a single version of the recursive function is needed as the score of a position to a player is the negation of another player's score.

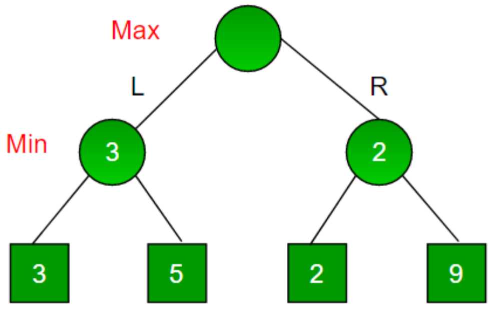


Figure 14: An example of a minimax tree, where the root chooses left with optimal value 3

However, it is still time-consuming to traverse when the tree is substantial. Alpha-Beta pruning optimizes the algorithm by cutting off branches that need not be searched as a better move is available (see Figure 15) [20]. It significantly reduces computation time.

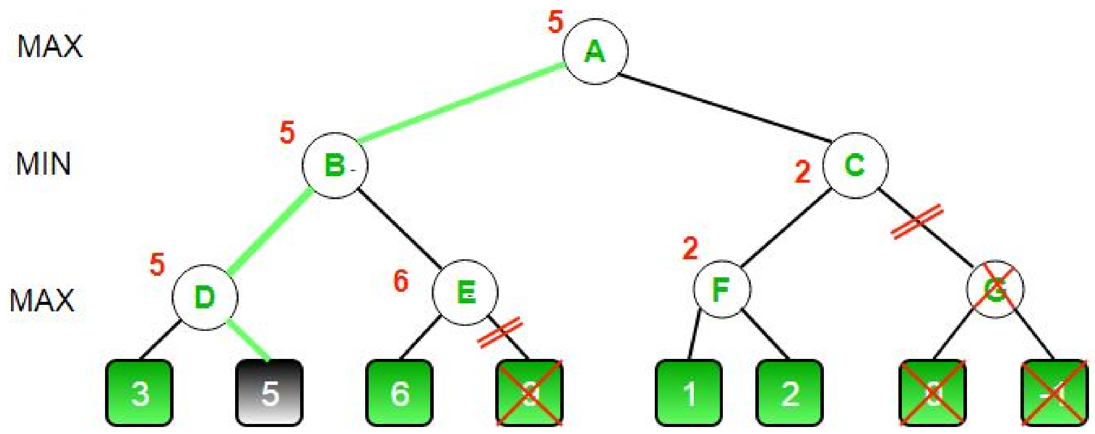


Figure 15: An example of Alpha-Beta pruning, the rightmost branch is cut as 5>C is guaranteed

It has been proven that the time complexity for the minimax algorithm is O(b^d), where b is the branching factor and d is the depth of the tree. In Connect Four, the branching factor is seven because a node can have seven children by placing a new disc into the seven columns. The search depth is 42 piles because 42 is the maximum number of turns taken for a Connect Four game. The minimax algorithm is inefficient with the exponential growth in time complexity, so Alpha-Beta pruning is implemented. Given that the best moves are searched first, employing Alpha-Beta pruning narrows time complexity to O($b^{d/2}$), reducing half of the exponent.

However, it is still inefficient and may not be able to solve in computation time. Studies have shown that the precise number of reachable states from the initial empty board state is 4,531,985,219,092, with 2,626,652,048,471 non-terminal states and 1,905,333,170,621 terminal states [21][22]. Table 2 lists all the legal Connect Four positions after each move [23]. The numbers are significant, but they are still manageable compared to other complex board games like chess. With further optimization applied to the algorithm, it is possible to build the perfect Connect Four AI with an exhaustive search.

Move	Possible States	Move	Possible States
0	1	22	22010823988
1	7	23	38263228189
2	49	24	60830813459
3	238	25	97266114959
4	1120	26	140728569039
5	4263	27	205289508055
6	16422	28	268057611944
7	54859	29	352626845666
8	184275	30	410378505447
9	558186	31	479206477733
10	1662623	32	488906447183
11	4568683	33	496636890702
12	12236101	34	433471730336
13	30929111	35	370947887723
14	75437595	36	266313901222
15	176541259	37	183615682381
16	394591391	38	104004465349
17	858218743	39	55156010773
18	1763883894	40	22695896495
19	3568259802	41	7811825938
20	6746155945	42	1459332899
21	12673345045	Total	4531985219092

Table 2: Possible states at each move

The AI is optimized by following Pascal Pons's tutorials [8]. In short, the general idea is to employ several optimization techniques such that the perfect AI can solve positions in computation time. First, positions are encoded using bitmaps instead of arrays to reduce running time significantly. Then, to prioritize the exploration of the best nodes, a move exploration order is implemented, allowing maximal reduction of the Alpha-Beta exploration window for the subsequent nodes. Next, a transposition table is utilized to cache calculated outcomes to avoid analyzing the same positions multiple times. Finally, a pre-generated book storing some common positions with respective values is also loaded to the transposition table to enhance the transposition table further. Thus, the algorithm can avoid some time-consuming calculations. Details of the steps to build the perfect Connect Four AI will be discussed in Chapter 3.3.

2.2.3 Scoring System

Besides the basic rules from the Connect Four game, new rules are added to the game to improve the learning and gaming experience of the player. A scoring system is implemented to evaluate players' performance during and after the game ends. By awarding scores to the player, it is hoped that players use it as feedback to review their mistakes, advance their skills and strive for a better score.

At the beginning of the player's turn, the app captures the current board state with the aforementioned board detection algorithm. The board state is analyzed with the Connect Four AI algorithm and the app gets an array of optimal moves. The board detection algorithm records a change in board state when the player places a disc, and by comparing it with the previous board state, the player's move can be obtained. If it is one of the optimal moves, 5 points will be awarded to the player. By observing an increase of 5 points, players understand that the move is optimal.

As the Connect Four AI is perfect, players can never defect the robot unless all moves are optimal. As a result, players may feel frustrated after many rounds of defeats and thus give up. The app motivates players to try and learn by adding a hint button to the game to assist players. When the player is clueless, he/she can press the "Show Hint" button and all optimal moves of the current board state will be displayed. However, if the player chooses to get a hint, points will not be awarded for that move. Hints are for assistance only and players should not rely on them. When players are more advanced, their strategies will be closer to optimal and the use of hints will be reduced, therefore attaining higher scores.

The highest possible score is 105, which is also the winning score. Connect Four is a solved strategy game, and the first player is guaranteed to win if all moves are optimal. If the AI goes first, the human player will undoubtedly lose. Thus, the only possible scenario to defeat the perfect AI is to play as the first player and perform optimally at all moves, like how the AI performs. When both players are optimal, the first player can only win at the last move (see Figure 16). Therefore the score without using hints will be 21 x 5 = 105. For a drawing scenario, the player needs to perform optimally for 20 moves and sub-optimally for one move. For example, placing the first disc in the third or fifth column but not the middle column is sub-optimal, giving a chance for the robot to tie. The score in a tie is 100 when no hints are used. In any other case, players will lose against the perfect AI.

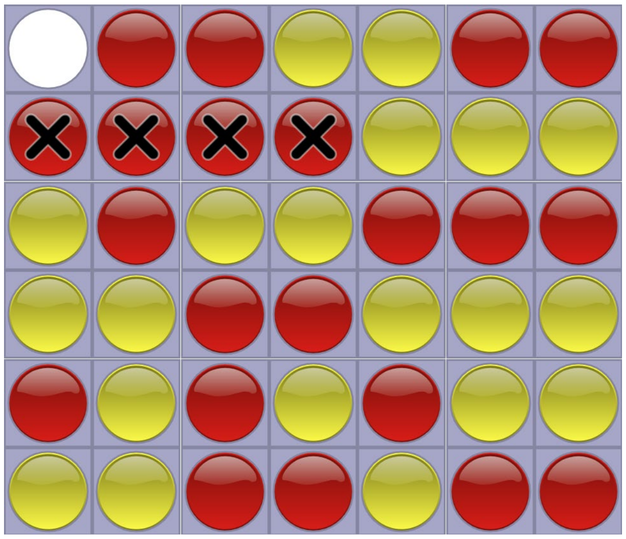


Figure 16: An example of a terminal state by two perfect players

2.3 Software – Game Modes

The proposed application is more than just an interface to play Connect Four with the AI. Three game modes are provided by utilizing the robotic arm and the game board to make the app more intriguing. The first game mode, Player VS AI Mode, is this project's main objective, where players fight against the unbeatable AI in the Connect Four game. The second game mode, Arm Controller, allows players to familiarize themselves with the robotic arm by having direct control. The third game mode, Building Mode, works like a claw machine. Players have to pick and place discs into the game board to build the specified pattern within a time limit.

By dint of the system workflow, all three game modes can only be accessed with a Bluetooth connection. Thus, the following chapters assume the Bluetooth connection between the app and the robotic arm is established.

2.3.1 Player VS AI Mode

In Player VS AI Mode, players play Connect Four on a physical game board with the AI that utilizes the robotic arm to pick and place discs. This game mode is of utmost importance as it is the objective of this project. It offers players a new, novel Connect Four gaming experience as unlike human adversaries, the opponent AI is perfect and unbeatable. By playing against the world's best player, players learn from mistakes, advance skills and strive for better scores. It is also hoped that the robustness of AI algorithms in strategic thinking and object recognition can be demonstrated to the players, raising their awareness of the AI field.

After the player enters the Player VS AI Mode, a page asking the user to choose the first player will be shown. Figure 17 shows the GUI for selecting the first player. The playing order is arranged according to the button pressed. It is noteworthy that defeat is promised if the player clicks the Robot button allowing the robot first, as the first player is guaranteed to win if all moves are optimal. It tests players' understanding of the Connect Four game and the AI.

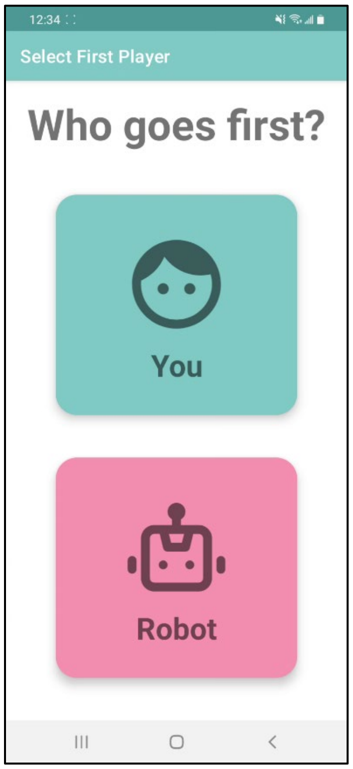


Figure 17: GUI of selecting the first player

Once the button is clicked, auto home positioning will be performed to restore the robotic arm to the home position. Next, the instruction page guides the player to set up the gaming environment (see Figure 18). As the board is vertical, it can only be lifted after auto home positioning so as not to block the process. Then, players should put the phone on a holder and place it in front of the board to prepare for capturing. With the instructions followed, players can proceed to board recognition by clicking the "I am ready!" button.

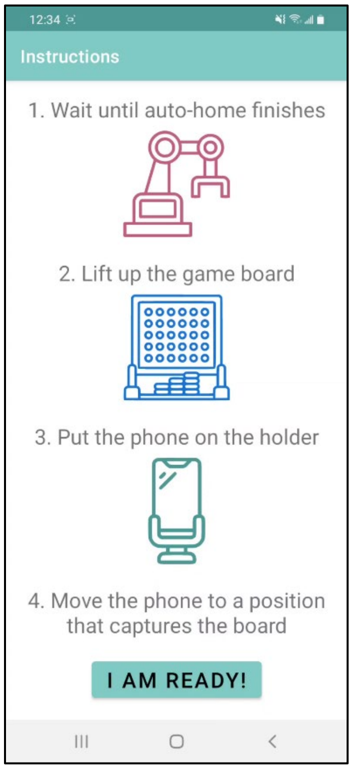

12:34
Instructions
1. Wait until auto-home finishes
2. Lift up the game board
3. Put the phone on the holder
4. Move the phone to a position that captures the board
I AM READY!

Figure 18: GUI of set up instructions

The whole board should be captured for better detection, filling the whole monochrome camera frame (see Figure 19). If the board cannot be detected after a long duration, the phone position can be nudged, and the refresh button on the top can be clicked to reanalyze the board. The detection process is finished when all 42 slots are identified, displaying six rows of linked-up dots. The camera frame becomes colored, indicating that detection is completed. The phone position should not be moved after the detection. Details of board detection will be explained in Chapter 3.1. The page then automatically proceeds to the gaming page.

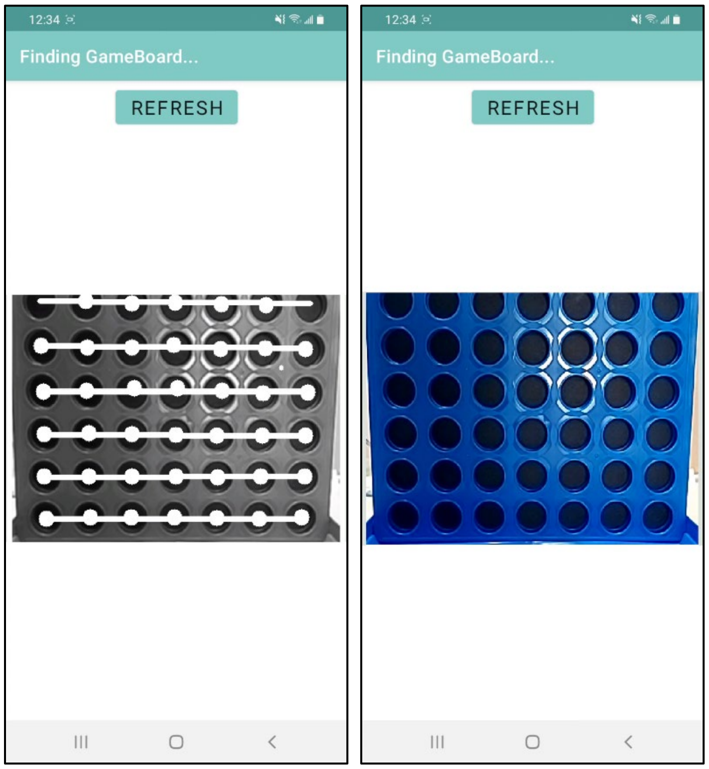


Figure 19: GUI of board detection

Figure 20 illustrates the GUI of the gaming page at the beginning. The page comprises four sections: the message, board, hint, and score section.

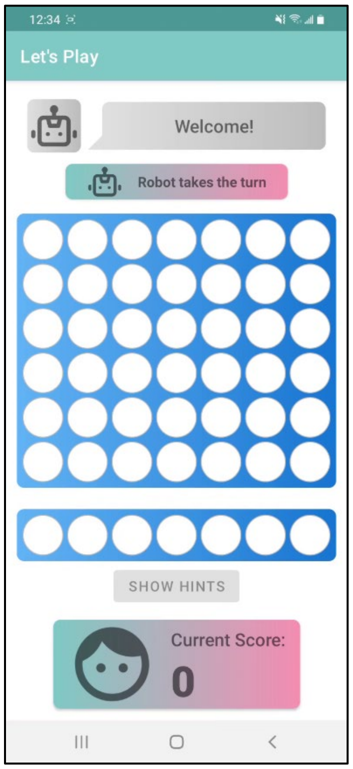


Figure 20: GUI of the initial gaming page

The message section comprises a robot icon, a chat bubble, and a turn indicator. The turn indicator varies the displayed text and icon to show the current turn, reminding the player who should be making a move. The dialog from the robot tells different information to the player depending on the game state. At the start, the robot first welcomes the player. Then, when the player makes a move, it tells the player whether it is a good move or a bad move. After the robot moves, it tells the player the expected number of moves remaining for the player to lose or win (see Figure 21). If there is a chance to draw, it also tells the player. The prediction is calculated by using the minimax score of the board state, which will be explained in Chapter 3.3.1.

The board section displays the physical board state in a virtual form, updated in real-time. The phone camera takes snapshots of the environment in the background, and board states are analyzed with disc detection. They are then retrieved and translated into the board section UI (see Figure 21).

The hint section consists of a row with seven slots and a "show hint" button. It serves as assistance to enhance the gaming experience when players fail to select a move. All optimal moves are displayed in the seven slots when the button is pressed. For example, Figure 21 shows that placing at column 6 is the only optimal move for the board state. The disc color in the hint section also changes with the color chosen to use by the player. In the case of Figure 21, the player is using teal discs. The button is disabled when it is the robot's turn.

The score section shows the accumulated score of the player during the game. Five points are added to the current score whenever the player makes an optimal move. No points will be added even if the move is optimal if the hint is used. For instance, in Figure 21, the score remains at 15 after the player's turn as the hint is used. As discussed in Chapter 2.2.3, it is hoped that players can aim for optimal moves to get a higher score.

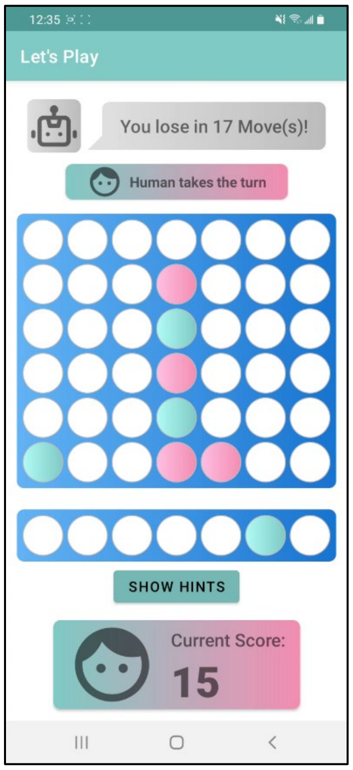
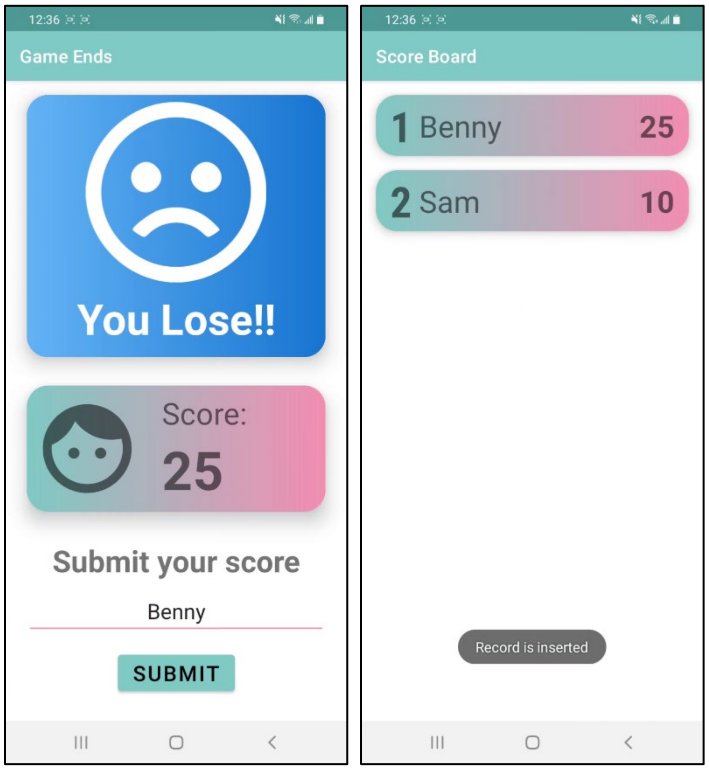


Figure 21: GUI of the gaming page with hint shown

After the game ends, players are redirected to the result page (see Figure 22). The emoji and the text change depending on winning, losing or drawing. The player's total score in the game will be displayed, and they can input their name in the text field to submit their score to the database.

When the name is entered and the submit button is clicked, a toast message indicating successful submission will be shown, and the scoreboard page is displayed (see Figure 22). This process requires an internet connection as the Firebase cloud database is used. The scoreboard page, which is the same page as clicking the scoreboard button on the home page, is displayed to show players' rankings. Players can evaluate their performance by checking their ranks.



1 Benny 25
2 Sam 10

Record is inserted

Figure 22: GUI of the result and scoreboard page

2.3.2 Arm Controller

Players can use the Arm Controller to manually control the robotic arm and move the arm to some predefined positions. It serves as a testing platform such that players can get used to the control of the robotic arm.

After the player enters the Arm Controller, auto home positioning will restore the robotic arm to the home position. Figure 23 shows the GUI of the Arm Controller. It is divided into two components, the first half is the button pad and the second half is the gamepad.

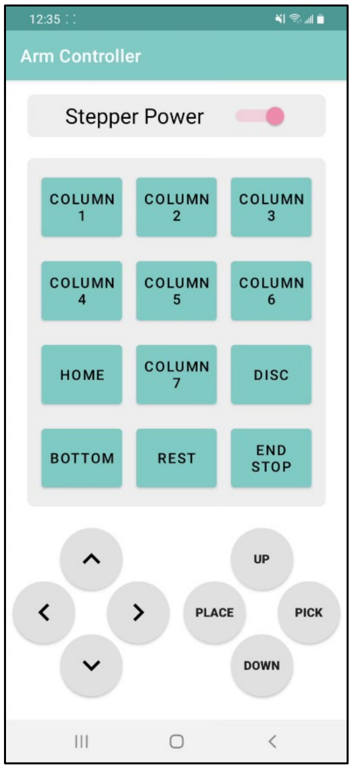


Figure 23: GUI of Arm Controller

The first half consists of a switch and 12 buttons. The switch is used to control the power of the three stepper motors. The robotic arm can be turned on or off simply by switching here. Auto home positioning will also be performed when the robotic arm is turned on with this switch to restore its starting position. The 3 x 4 keypad, which resembles the telephone keypad design, contains seven column buttons corresponding to the game board's seven column slots. The robotic arm automatically moves to these positions to place discs into column slots in the Player VS AI mode. The disc button moves the robotic arm to the position for picking discs, and the home button returns the robotic arm to the home coordinate. Three extra buttons also provide further testing and debugging: the bottom, rest, and end stop.

Players can comprehend how the robotic arm picks and places discs with the button pad. This helps provide insights for them as they need to manually control the arm to pick and place discs in the Building Mode. In addition, it also acts as a calibration tool. For example, the distance between the robotic arm, the disc stack and the game board must be fixed, as absolute positioning is required for picking and placing. Using the disc and the seven column buttons, players understand the position of picking and placing discs and are thus able to calibrate the disc stacker and game board position.

The second half of the Arm Controller is the gamepad, which has eight buttons that control the gripper and the movement of the robotic arm in the three axes. This section is forged by using fragment, which allows the same gamepad to be reused by the Building Mode. In order to imitate the actual behavior of a real-life gamepad, when the buttons are held, the robotic arm continues to move in the corresponding direction. This behavior is accomplished by repeatedly sending G-code commands that move the robotic arm along the specified axis in small increments. Players can advance their pilot skills here such that they are strengthened to play the Building Mode.

2.3.3 Building Mode

The Building Mode offers the gaming experience of playing with a claw machine. Players challenge themselves to finish the given pattern within the time limit by carefully controlling the robotic arm's claw gripper to pick and place discs into the game board.

After the player enters the Building Mode, auto home positioning will restore the robotic arm to its home position. Then, a random pattern will be drawn from the pattern list and shown to the player. Figure 24 illustrates the GUI of the Building Mode. It consists of the pattern panel and the gamepad panel.

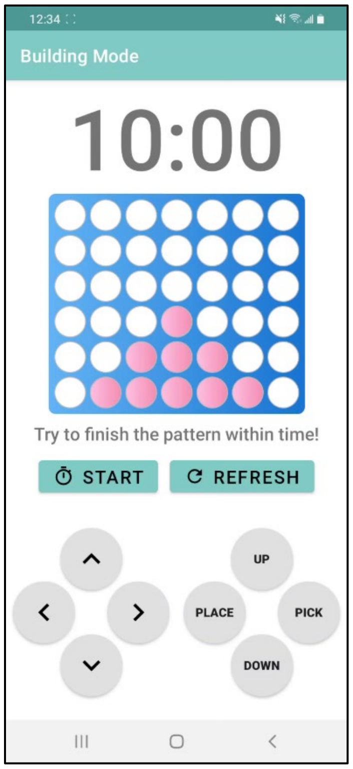


Figure 24: GUI of Building Mode

The pattern panel controls the timer and the disc pattern. First, a large countdown timer is displayed at the top to remind players of the remaining time, followed by the target pattern to build. Then, a short line of text is shown under the pattern to introduce this game mode briefly. Finally, start and refresh buttons are provided to start the timer and refresh the pattern. The gamepad panel is the same fragment used in the Arm Controller, where players utilize the eight control buttons to move the robotic arm in 3D and finish the pick and place tasks to build the pattern. It is believed that with this concise UI, new players have no problem understanding how to play with this game mode.

Before starting the game, if players are not satisfied with the pattern, or if the players have built the pattern before, the refresh button can be clicked to draw another pattern randomly. There are fifteen patterns. Each of them is built from nine to ten discs. When players are ready to play, they can press the start button to start the ten-minute countdown timer. Figure 25 demonstrates the playing stage of building mode. Once the game starts, the refresh button is disabled. The clock icon of the timer button changes to disable clock, and the text changes from “start” to “finish”, indicating that the functionality has changed to stop the timer.

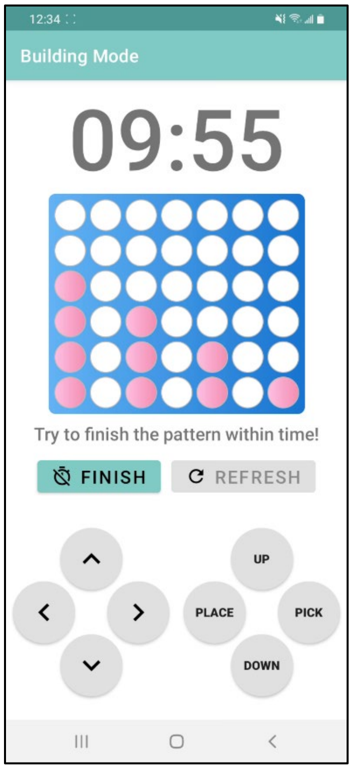
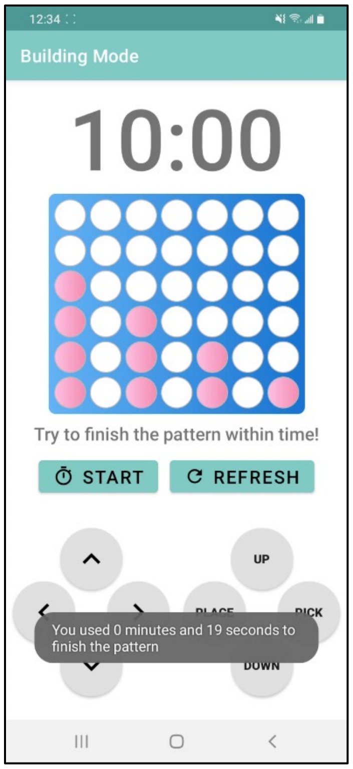


Figure 25: Playing stage of Building Mode

Players continue to build the pattern using the gamepad to pick and place discs. Players click the finish button once the pattern is finished to stop the timer. A toast message indicating the amount of time used to build this pattern is shown (see Figure 26). The countdown timer is reset to ten minutes, and the refresh button is enabled. It goes back to the state before starting the game, such that players can effortlessly start a new round without quitting this page. However, the target pattern is not refreshed, but they can always play with another pattern by clicking refresh. With the time spent feedback provided, players are encouraged to challenge the same pattern repeatedly to master the pattern and reduce the time spent.

If players fail to build the pattern within the time limit, a toast message "Time's up!!" will be shown. The page will also reset as mentioned so that players can challenge again without returning home and entering Building Mode. They are encouraged to play more rounds with the same pattern to advance their skills and finish the pattern within the time limit.



You used 0 minutes and 19 seconds to finish the pattern

Figure 26: Ending stage of Building Mode

2.4 Hardware – Robotic Arm

This chapter introduces the robotic arm utilized to play Connect Four. Chapter 2.4.1 provides the background information on the given robotic arm. Chapter 2.4.2 describes the enhancements made to the robotic arm such that it is adapted to this project.

2.4.1 Background of the Robotic Arm

The robotic arm utilized in this project is provided by HKU CS MakerLab, which is a modified version of the robotic arm created by Florin Tobler [24]. In comparison with the original design, the robotic arm has a more extended arm and a servo gripper that utilizes the Tower pro SG90s servo (see Figure 27). Although the arm is changed, it still has three degrees of freedom, allowing movements along the three rotational axes (see Figure 28).

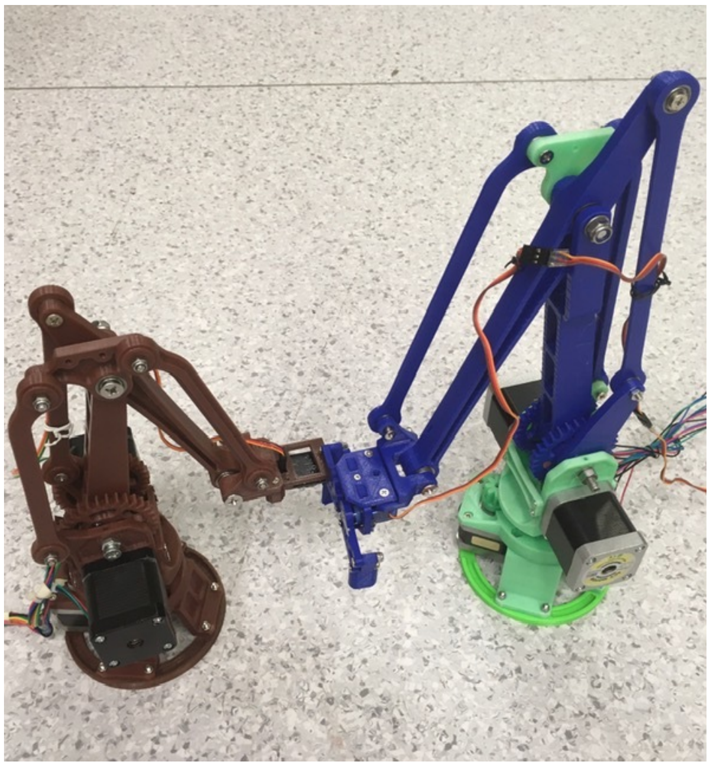


Figure 27: Comparison between the original design (left) and Makerlab design (right)

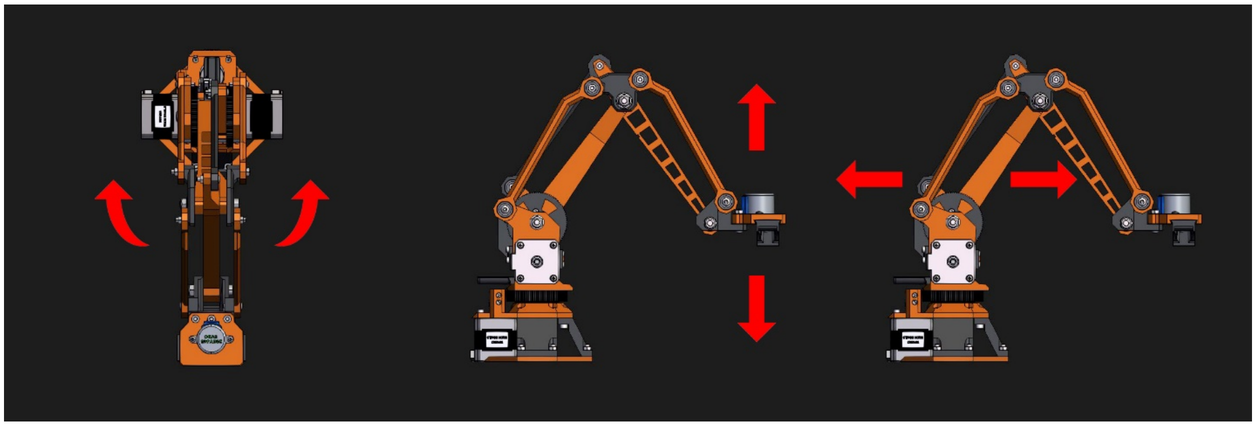


Figure 28: Three-DOF robotic arm

The original geometry algorithm designed by Florin Tobler can also be applied to the robotic arm. Figure 29 shows Florin Tobler's contribution to the geometric structure of the robotic arm and the corresponding kinematic algorithm. Based on his design, the robotic arm community developed Arduino firmware to control the robotic arm [25]. The firmware is open-source, allowing modifications based on different variations of the robotic arm. Makerlab also developed the firmware that suits the modified robotic arm [26].

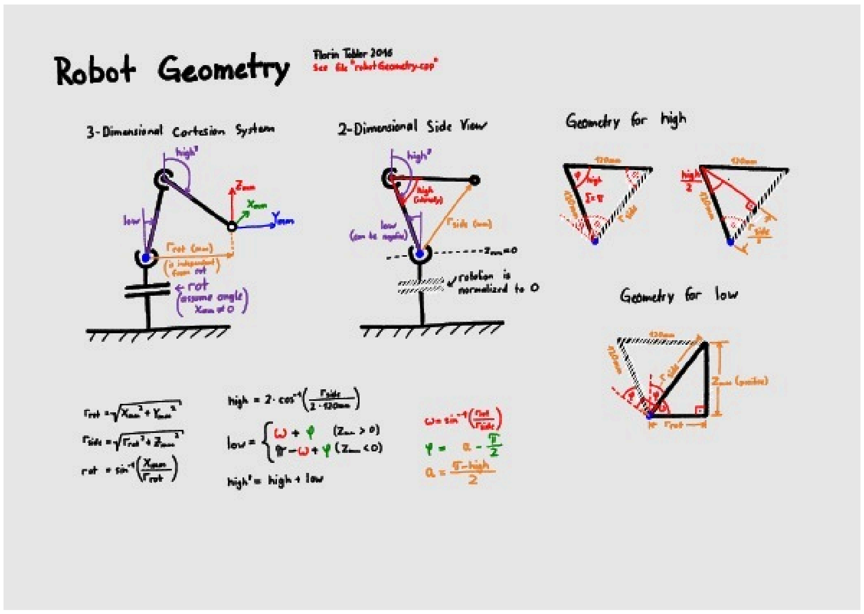


Figure 29: Robot geometry designed by Florin Tobler

The x, y, and z-axis movements on the 3D Cartesian coordinate plane are accomplished by varying the number of steps of the three stepper motors. RAMPS 1.4 expansion board is installed on top of the Arduino MEGA 2560 such that a stable current can be provided to the stepper motors. However, it is too complicated to specify the number of steps to move the arm. Therefore, G-code is applied to instruct the robotic arm. G-code stands for "Geometric Code", which instructs machines to move to a location with a specified moving speed and path [27]. In this project, when the arm receives the G-code commands, the Arduino program will automatically interpret the commands and translate them into the number of steps required for each motor. For example, G1 X0 Y225 Z180 moves the end effector to the specified x, y and z coordinates. Using G-code commands is more intuitive to control and fine-tune the robotic arm movement than using steps, which fosters the implementation of picking and placing discs.

A Bluetooth module SPP-CA with a baud rate of 9600 is also installed to provide a Bluetooth connection to the robotic arm. The connection can be established quickly, and the module's light indicates the connection state. When it is blinking, it can be discovered by the android phone and is ready for connection. Once the connection is successful, it keeps the light on. With the Bluetooth module, G-codes can be sent via Bluetooth, enabling the control of the robotic arm using the application.

2.4.2 Enhancements

In order to adapt to play Connect Four, enhancements to the robotic arm are necessary. The three stepper motors have no absolute encoder and thus cannot perform absolute-type positioning by themselves. However, the arm needs to initialize at an absolute position such that the Cartesian coordinate system can refer to. Three end-stops switches are installed to implement the auto home positioning function. Figure 30 shows the two extra components installed on the robotic arm for placing the three end-stop switches.

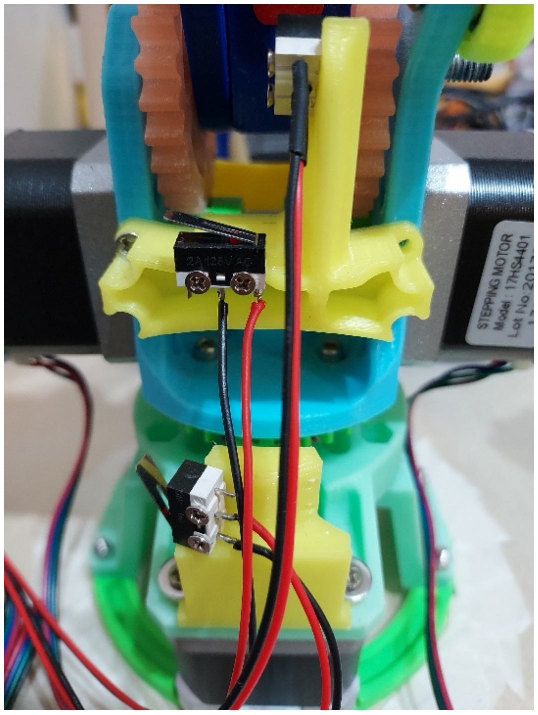


Figure 30: Two extra components for placing end-stop switches

The Arduino program developed by Makerlab is reinforced to include the auto home positioning function regarding the community firmware. Figure 31 reveals the default configuration of the community robotic arm [28]. The coordinate of the home position is defined with INITIAL_X, INITIAL_Y and INITIAL_Z (in mm), and their values are calculated with respect to shank length and end effector. The modified robotic arm utilized in this project has a longer shank length than the original one. Therefore, the value of INITIAL_Z, which is the shank length, is 180 instead of the default 120. The mini servo gripper employed has an end effector offset of 45, so the value of INITIAL_Y, which is the sum of shank length and end effector offset, is 225. INITIAL_X remains unchanged at 0.

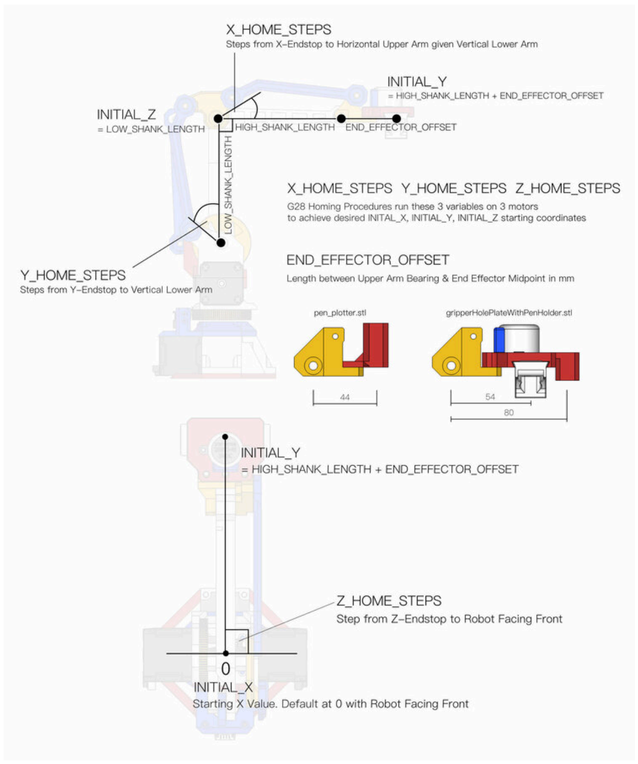
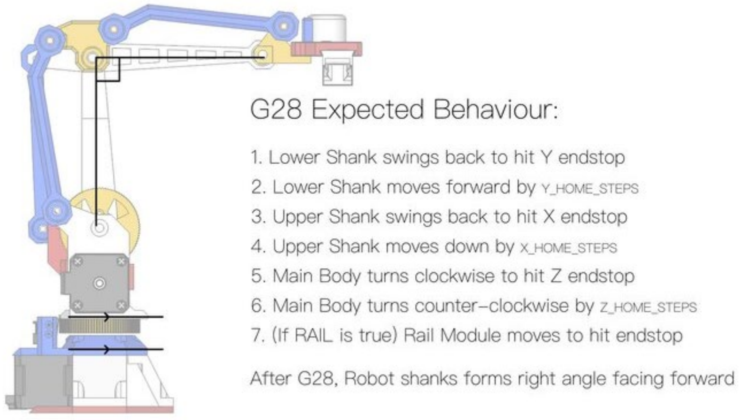

X_HOME_STEPS
Steps from X-Endstop to Horizontal Upper Arm given Vertical Lower Arm
INITIAL_Y
= HIGH_SHANK_LENGTH + END_EFFECTOR_OFFSET
INITIAL_Z
= LOW_SHANK_LENGTH
HIGH_SHANK_LENGTH
END_EFFECTOR_OFFSET
LOW_SHANK_LENGTH
X_HOME_STEPS Y_HOME_STEPS Z_HOME_STEPS
G28 Homing Procedures run these 3 variables on 3 motors to achieve desired INITIAL_X, INITIAL_Y, INITIAL_Z starting coordinates
Y_HOME_STEPS
Steps from Y-Endstop to Vertical Lower Arm
END_EFFECTOR_OFFSET
Length between Upper Arm Bearing & End Effector Midpoint in mm
pen_plotter.stl
gripperHolePlateWithPenHolder.stl
44
54
80
INITIAL_Y
= HIGH_SHANK_LENGTH + END_EFFECTOR_OFFSET
Z_HOME_STEPS
Step from Z-Endstop to Robot Facing Front
0
INITIAL_X
Starting X Value. Default at 0 with Robot Facing Front

Figure 31: Default configuration of the robotic arm

The G-code used to initialize auto home in this project is G28. The behavior of auto home positioning is illustrated in Figure 32. The stepper motors move shanks and the main body to hit their corresponding end stops. Once the end stop is hit, motors will move in the reverse direction by the home step value. Home step values of the three axes are tested to suit the project's robotic arm. A right angle is formed between the lower and upper shank once the auto home is finished.



1. Lower Shank swings back to hit Y endstop
2. Lower Shank moves forward by Y_HOME_STEPS
3. Upper Shank swings back to hit X endstop
4. Upper Shank moves down by X_HOME_STEPS
5. Main Body turns clockwise to hit Z endstop
6. Main Body turns counter-clockwise by Z_HOME_STEPS
7. (If RAIL is true) Rail Module moves to hit endstop

After G28, Robot shanks forms right angle facing forward

Figure 32: Behavior of auto home positioning

The G-code commands that control the mini servo gripper are also modified to be more intuitive. M3 and M5 are the commands controlling the on and off of the servo gripper. After modification, M3 T90 represents a wide-open of the gripper, and M3 T0 represents a complete close. The reverse applies to M5, where M5 T0 represents a wide-open of the gripper, and M5 T90 represents a complete close. Using 90 degrees as the moving range makes it easier to specify the angle to open and close the gripper.

It could be dangerous if the robotic arm moves beyond the limit. Thus, an Arduino function is added in the interpolation section to check if the movement exceeds the operation range of the robotic arm. Figure 33 demonstrates the operation range of the arm. The community firmware contains the function and formula to calculate R_MIN and R_MAX, and it is utilized to implement the operation range. Besides, the application also keeps track of the 3D coordinates to ensure that all moving commands sent are within range.

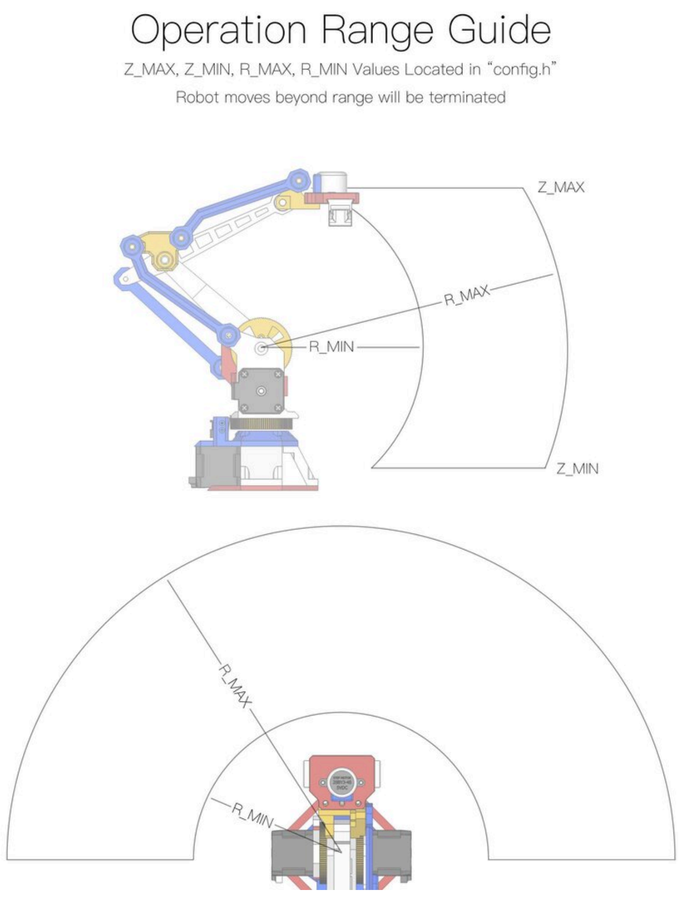

Operation Range Guide
Z_MAX, Z_MIN, R_MAX, R_MIN Values Located in "config.h"
Robot moves beyond range will be terminated
Z_MAX
R_MAX
R_MIN
Z_MIN
R_MAX
R_MIN

Figure 33: Operation range of the robotic arm

2.5 Hardware – Disc Stacker

The robotic arm needs to place discs inside the vertical game board, so a disc stacker that stores discs vertically is necessary to make gripping possible. The disc stacker should be able to roll out a new disc to refill once the gripper picks a disc. With this behavior, the claw gripper can grip discs continuously at the exit of the disc stacker until there is no disc left. To simplify the design of the disc stacker, no mechanical parts and motors are involved. Three ramps with a slope of 10 degrees are designed, utilizing gravitational forces to roll out discs.

Figure 34 shows the sectional view of the disc stacker. The ramps, front and back plates are created by laser cutting wooden and acrylic plates. Joint clamps are designed and printed with 3D printers to combine the plates as a disc stacker (see Figure 35). The clamps are trapezoidal prisms with a 1.5 cm width by 1 cm height groove to fit the components. The bottom base is 3 cm, with a height of 2 cm and a depth of 5 cm. The dimension is designed to also act as the support base of the disc stacker as the stacker has a width of 1.5 cm only, which cannot sit by itself. The stacker can stand without toppling with the clamps installed at the bottom. With this design, the stacker can be assembled and disassembled effortlessly. Attempts leading to this final design will be discussed in Chapter 3.4.

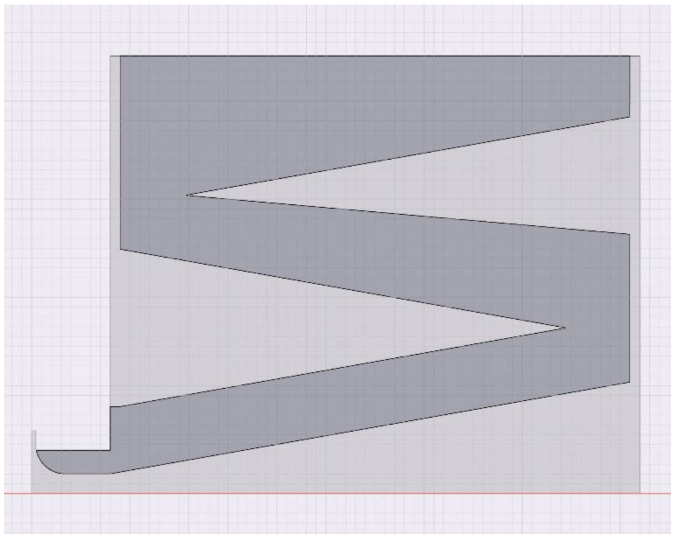


Figure 34: Sectional view of the disc stacker

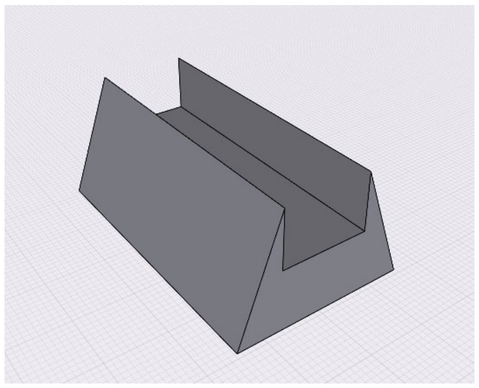


Figure 35: Joint clamp 3D model

3. Experiments, Results and Discussion

The project team has done multiple tests to acquire the best possible outcomes in different aspects of this project. The following chapter demonstrates the experiments conducted. Two approaches are undertaken to detect the Connect Four game board accurately, and they are discussed in Chapter 3.1. The attempts to determine the disc detection method are described in Chapter 3.2. Chapter 3.3 details the steps to construct the perfect Connect Four AI, and Chapter 3.4 presents the trails done in building the disc stacker.

3.1 Experiment 1: Board State Detection

Computer vision contributes a vital role in the application to retrieve the game state from the physical game board accurately. If board state detection fails to deliver correct results, subsequent Connect Four AI cannot generate optimal moves and wrong decisions will be made, causing garbage in, garbage out. Thus, two approaches are tested to select the best way to detect the board state.

3.1.1 Approach 1: TensorFlow Lite Neural Network

Neural network model training with TensorFlow Lite is the first tested approach. YOLOv5s model, which is widely used in object detection, is selected for supervised learning. Since there is no existing library for Connect Four, a dataset must be prepared. First, photos of the board and the discs were captured and labeled manually (see Figure 36). Then, supervised learning is performed to generate the custom model in the Pytorch framework. After translation, the model was applied to the app for board detection. It is hoped that the board state can be constructed by identifying all discs in the image.

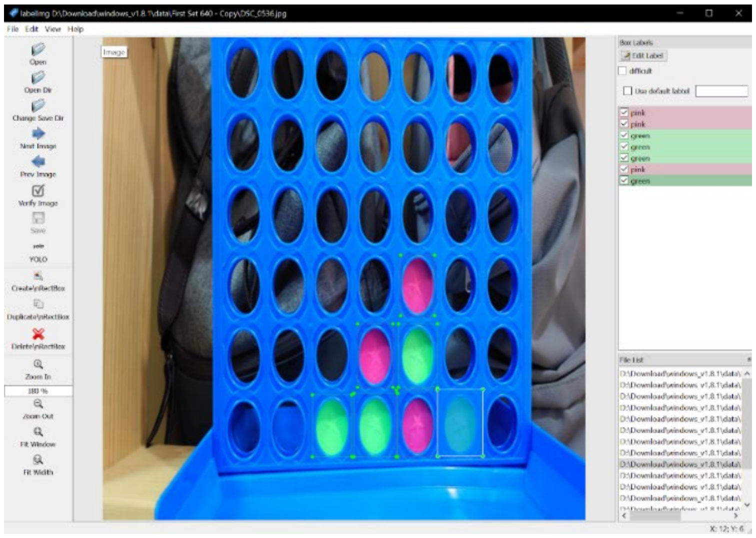


Figure 36: Preparing custom dataset for training

The performance of the trained model was not satisfactory. Figure 37 demonstrates the results of the model. The detection algorithm is too sensitive to recognize irrelevant objects, and multiple objects were detected in the same area that detection boxes overlap and interfere. Moreover, it fails to detect some discs on the board. With its unstable performance in detecting discs, an accurate board state cannot be inferred.

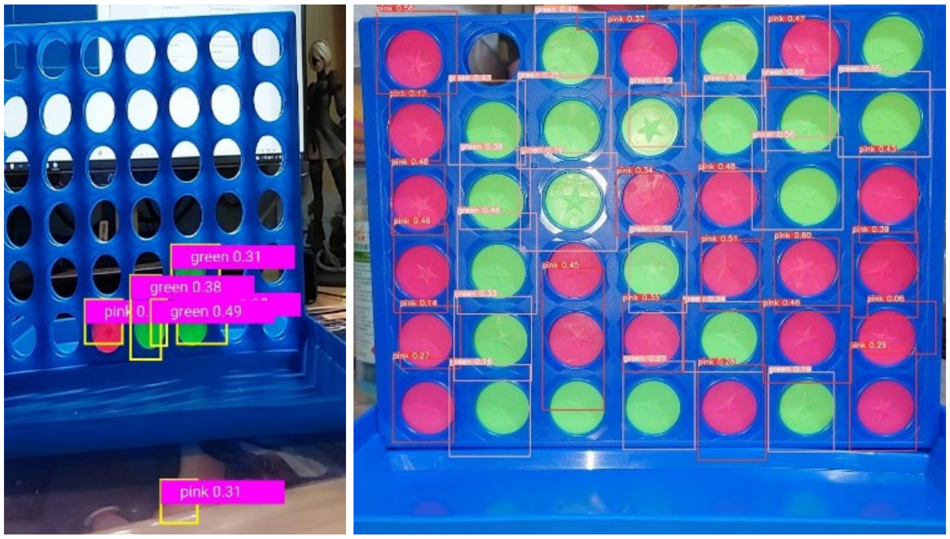


Figure 37: Detection results by the model

3.1.2 Approach 2: OpenCV Circle Detection

Instead of inferring the board state from detecting disc objects, another approach is tested to detect the whole board first and then respond to state changes. The Connect Four board consists of 42 circular slots, so detecting 42 circles in a 7x6 grid should identify the game board. In particular, OpenCV HoughCircles() function is utilized to detect circles efficiently [18].

The function is derived from Hough Circle Transform, which is based on the circle equation $r^2 = (x - h)^2 + (y - k)^2$. With h and k being the x and y coordinate of the circle center and r is the radius, a circle can be formed. Figure 38 shows how Hough Circle Transform detects circles with this equation. Circles with the same radius are plotted along the edge pixels of the circle image input. These circles overlap at a point, namely the accumulator point, which is the center of the input circle.

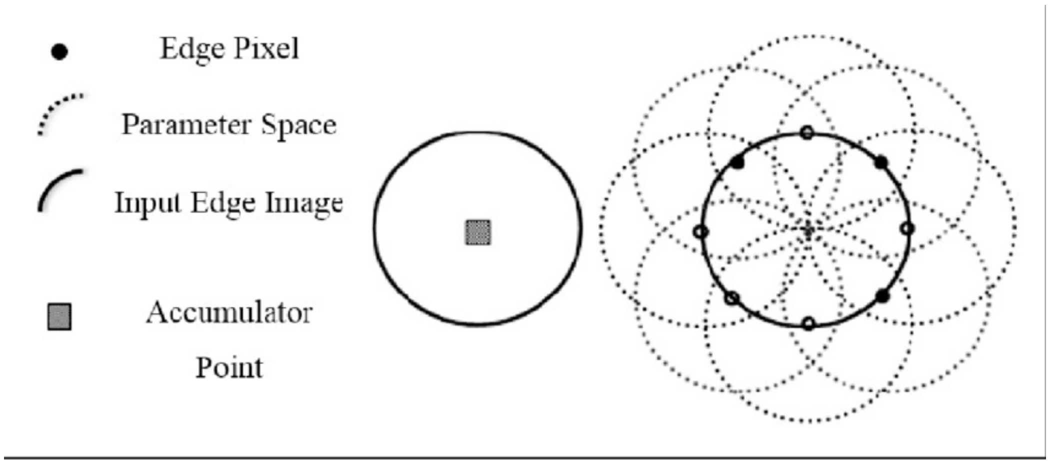


Figure 38: Hough Circle Transform

However, the radius of the input circle must be known to plot the circles along the edge. Therefore, the function utilizes a brute force approach to test a range of radii. The radius that causes the highest overlapping frequency is believed to be the actual radius of the input circle. Thus, the radius and center coordinate of the input circle can be found.

This approach successfully detects the board with the 42 circles identified, which is much more stable than the previous method. However, there is a noticeable increase in computation time for board images with complicated backgrounds. Therefore, enhancements to the algorithm are required.

3.1.3 Experiment Result: Fine-tuned OpenCV Circle Detection

The circle detection method has a reliable performance in detecting the whole board, and it is believed to be the correct approach. Enhancements are made to turn it into a robust algorithm.

Trimming is applied to the input images to delete unnecessary, complicated backgrounds. The image is also blurred to reduce noise, sharpness and details, facilitating edge detection of circles. Tests are also conducted to fine-tune the function's input parameters, making it more applicable to the project scenario. For example, the range of radii is reduced to identify the game board's circular slots quickly.

3.2 Experiment 2: Disc Detection

With the circle detection approach implemented, the next step is to detect discs that fall inside the slots. In particular, color detection can be utilized to identify color changes at the 42 center coordinates. The problem is reduced from complicated disc object detection to simple color detection. RGB and HSV color spaces are tested to identify their performance accuracy.

3.2.1 Approach 1: RGB Color Space

RGB color space is based on the RGB color model, where red, green and blue lights are combined to produce different colors. White light is produced when each color is mixed together, and black is produced without mixing. Figure 39 shows the RGB color space, in which the more significant the RGB value, the lighter the color. Although it is commonly used, its performance is poor in color recognition as it is correlated to light intensity. In this project, the disc color of same-colored discs in RGB color space differs depending on the light direction and position, so it fails to classify that these discs share the same color.

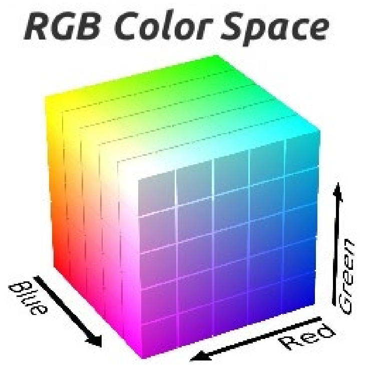


Figure 39: RGB color space

3.2.2 Approach 2: HSV Color Space

A color space that is independent of light intensity should be used. HSV color space, which stands for hue, saturation and value, can solve the problem. Hue represents the color, saturation represents the amount of grey mixed, and value represents brightness. Hue values express colors in degrees, which can be used to check if the input falls within the color range of teal and pink (see Figure 40). By separating color from saturation and brightness, robustness to the change of light can be achieved.

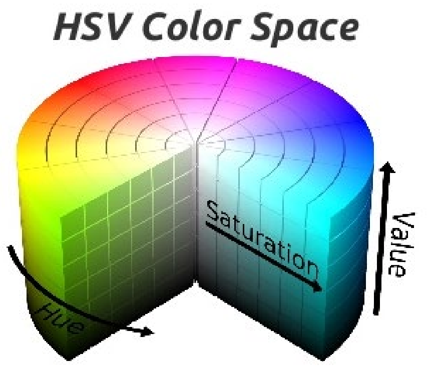


Figure 40: HSV color space

3.2.3 Experiment Result: HSV Disc Detection

After testing in different environments, HSV is more robust to the change in environment. With different conditions and light levels, HSV can still detect the colors teal and pink accurately. Disc detection employing HSV color space is outstanding with no errors identified. Thus, it is the approach applied in this project.

3.3 Experiment 3: Connect Four AI

One of the motivations for doing this project is to build an unbeatable Connect Four AI, but it is not the main objective of this project. Many existing Connect Four AIs use depth limit approaches. Some even apply machine learning. These AIs are workable and adequate for this project, but it is still hoped to reach the extra goal of building a perfect AI. Thanks to Pascal Pons, his approach to solving Connect Four is exhaustive, meaning that the AI is always optimal [8]. This chapter details the steps to build the perfect AI by following his tutorials.

3.3.1 Approach 1: Minimax with Alpha-Beta Pruning

The minimax algorithm is a typical solver for many board games. For example, in Connect Four, each node is a board position and the algorithm searches for the best path in the tree. At each layer of the tree, two players, namely the maximizer and minimizer, choose the best move to maximize the score, which is also minimizing the opponent's score.

The score of a position can be positive, negative or null, and it is calculated as follows. If the current player can win, the score is positive and is calculated by 22 – the total number of discs needed to win. If the player can win with the last disc, the score will be 22 – 21 = 1. If the player can only lose, the score is negative and is calculated by the total number of discs needed to defeat the player by the opponent – 22. If the opponent can win with the last disc, the score for the player will be -1. A null score only occurs if the game can end with a tie. Figure 41 demonstrates a position with a score of 18. The current red player can win in two moves with the fourth disc, so 22 – 4 = 18. After the red player places the third disc, the yellow player becomes the current player, and the score of this position will be -18 as 4 – 22 = -18. This scoring method is not only used in the minimax algorithm but also in the calculation to predict the remaining number of moves to win or lose by the player.

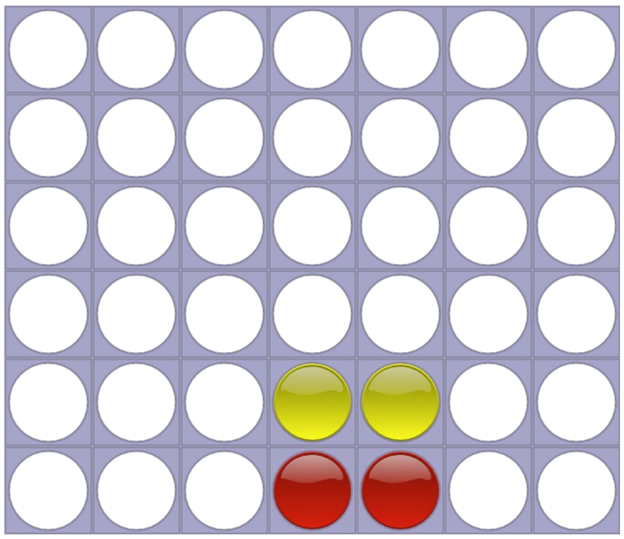
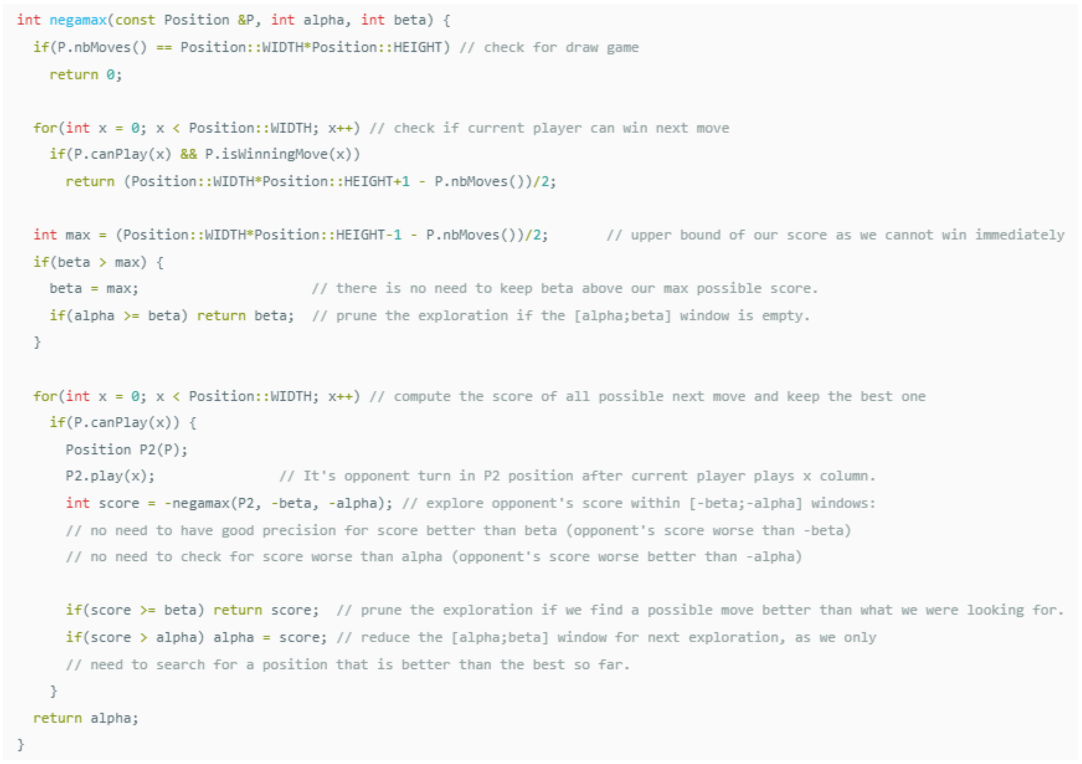


Figure 41: An example board state with a score of 18

The minimax recursive algorithm can be implemented with this scoring method. Scores are assigned to the terminal nodes by using this method. For non-terminal nodes, when it is the player's turn (maximizer), the score is the maximum of the next positions. When it is the opponent’s turn (minimizer), the score is the minimum of the next positions. As mentioned above, the player’s score of a position is the opposite of the opponent’s. The negamax variant can be applied such that one recursive function calculates for both the maximizer and minimizer.

Alpha-Beta pruning is implemented to the negamax variant for optimization. Alpha is the best option found for the maximizer, and Beta is the best option found for the minimizer. By tracking the [Alpha, Beta] window at each node, if Alpha >= Beta, pruning occurs. For a minimizer, the highest score that offers to the parent maximizer can only be the current Beta, which is useless to the parent as it can choose a value of Alpha. Therefore, the remaining unexplored children for this minimizer can be pruned. The same logic applies to the maximizer. Figure 42 shows the actual implementation of negamax with Alpha-Beta pruning.



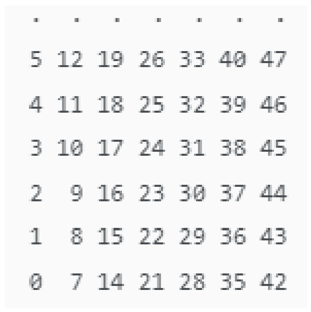
```
int negamax(const Position &P, int alpha, int beta) {
  if(P.nbMoves() == Position::WIDTH*Position::HEIGHT) // check for draw game
    return 0;

  for(int x = 0; x < Position::WIDTH; x++) // check if current player can win next move
    if(P.canPlay(x) && P.isWinningMove(x))
      return (Position::WIDTH*Position::HEIGHT+1 - P.nbMoves())/2;

  int max = (Position::WIDTH*Position::HEIGHT-1 - P.nbMoves())/2;       // upper bound of our score as we cannot win immediately
  if(beta > max) {
    beta = max;                     // there is no need to keep beta above our max possible score.
    if(alpha >= beta) return beta;  // prune the exploration if the [alpha;beta] window is empty.
  }

  for(int x = 0; x < Position::WIDTH; x++) // compute the score of all possible next move and keep the best one
    if(P.canPlay(x)) {
      Position P2(P);
      P2.play(x);               // It's opponent turn in P2 position after current player plays x column.
      int score = -negamax(P2, -beta, -alpha); // explore opponent's score within [-beta;-alpha] windows:
      // no need to have good precision for score better than beta (opponent's score worse than -beta)
      // no need to check for score worse than alpha (opponent's score worse better than -alpha)

      if(score >= beta) return score;  // prune the exploration if we find a possible move better than what we were looking for.
      if(score > alpha) alpha = score; // reduce the [alpha;beta] window for next exploration, as we only
      // need to search for a position that is better than the best so far.
    }
  return alpha;
}
```

Figure 42: Implementation of negamax with Alpha-Beta pruning

However, the algorithm can only solve positions with more than 14 moves. For early game states, the trees are still too significant after pruning, and they cannot be solved within a minute. In order to implement this AI into the application, it should be able to solve any board state instantly. More optimization steps are taken to achieve this target.

3.3.2 Approach 2: Bitboard

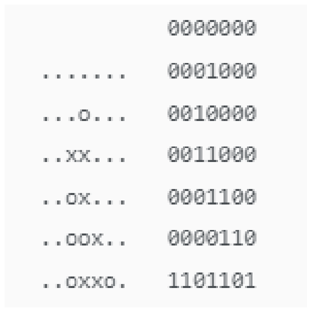
Board position can be encoded using a bitmap and stored in a compact way, which can speed up the operations with the game board. The Connect Four board has seven columns and six rows, which is 42 slots. 49 bits are used to represent the board, with an additional row added on top for convenience. Figure 43 shows the board in bit order.



```
.  .  .  .  .  .  .
5 12 19 26 33 40 47
4 11 18 25 32 39 46
3 10 17 24 31 38 45
2  9 16 23 30 37 44
1  8 15 22 29 36 43
0  7 14 21 28 35 42
```

Figure 43: The game board in bit order

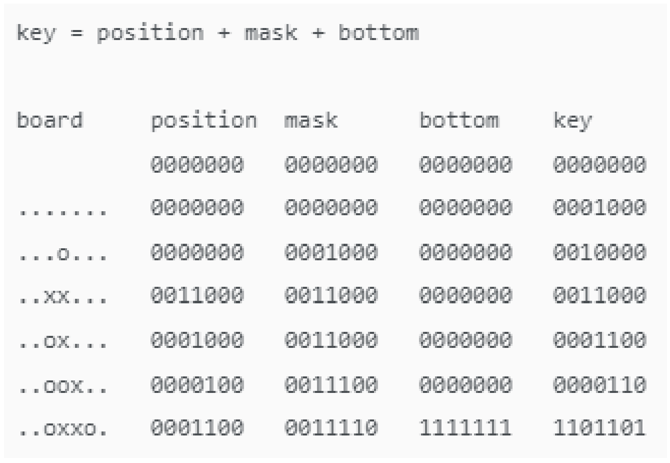
The current player's discs are encoded as 1, while the opponent's discs are 0. To distinguish between the opponent's discs and empty cells, the lowest empty cell for each column is encoded as 1. The extra row added on top can be used to encode the situation of a full column. Figure 44 demonstrates the implementation of this encoding method. The discs are encoded as 1 for the current player x and 0 for the opponent o. Extra 1s are added to each column to indicate the start of empty slots.



```
          0000000
.......   0001000
...o...   0010000
..xx...   0011000
..ox...   0001100
..oox..   0000110
..oxxo.   1101101
```

Figure 44: An example of a bitboard key

Positions are stored using two bitmaps such that bitwise operations can be computed efficiently. One bitmap contains only the discs played by the current player, and another bitmap is a mask locating all non-empty slots. Figure 45 illustrates how the previous example is stored in position bitmap and mask bitmap. The previous key can be obtained by adding the two bitmaps with a bottom mask.

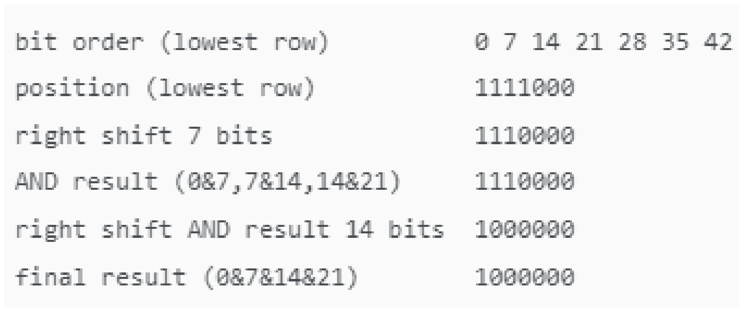


```
key = position + mask + bottom


board      position  mask      bottom     key
           0000000   0000000   0000000    0000000
.......    0000000   0000000   0000000    0001000
...o...    0000000   0001000   0000000    0010000
..xx...    0011000   0011000   0000000    0011000
..ox...    0001000   0011000   0000000    0001100
..oox..    0000100   0011100   0000000    0000110
..oxxo.    0001100   0011110   1111111    1101101
```

Figure 45: Board state stored in position and mask bitmap

With the two separated bitmaps, bitwise operations can be easily performed to play with the bitboard. For example, to know if a column is playable, the mask can be used to check the availability of the highest slot of the column. To play a column, an XOR operation is performed on the current position with the mask. Then, an extra bit that represents the played slot is added to the mask.

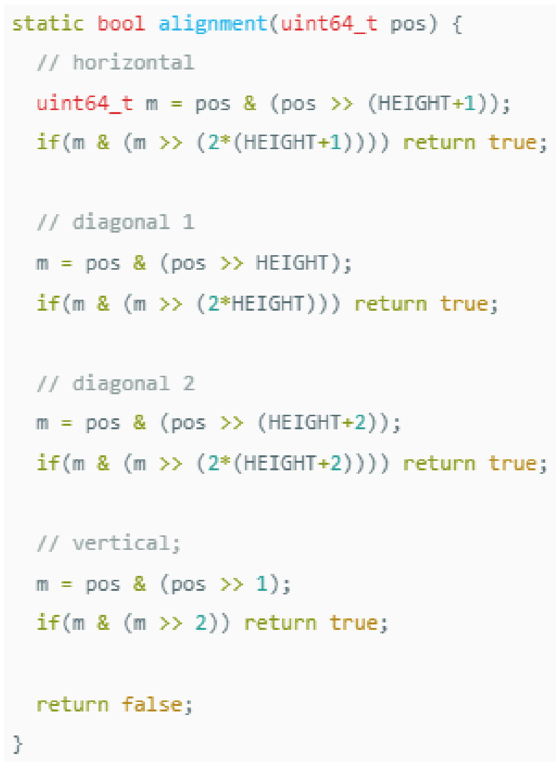
The checking of a winning condition can be done quickly using bitboards. For example, slots are checked horizontally to check for a horizontal four in a row, like slots 0, 7, 14, and 21. The difference between horizontal slots is 7. By the same token, vertical checking has a difference of 1, and the two diagonal checkings have a difference of 6 and 8. With these differences, bit shifting can be performed to check for alignment. For example, four discs played by the current player occupy slots 0, 7, 14, and 21, and they are encoded into 1s in the position bitmap (see Figure 46). Then, a copy of the position bitmap is shifted to the right by 7, bitwise AND is performed with the original bitmap. The resulting bitmap has three 1s at slots 0, 7, and 14, indicating the result of AND operation to the bit representations of disc 1&2, 2&3, 3&4. Next, a copy of this result is shifted to the right by 14 and performed AND with the original result. The bit at slot 0 of this final result indicates the result of AND operation to the bit representation of disc 1&2&3&4. If the final result is not 0, there must be a horizontal alignment.



```
bit order (lowest row)          0 7 14 21 28 35 42
position (lowest row)           1111000
right shift 7 bits              1110000
AND result (0&7,7&14,14&21)     1110000
right shift AND result 14 bits  1000000
final result (0&7&14&21)        1000000
```

Figure 46: An example of alignment checking

By shifting bits and combining bits, all horizontal alignments of a position can be checked at once, without concern for the newest disc. Figure 47 shows the actual implementation of alignment checking. The above logic applies to vertical and the two diagonals by changing the right shift to 1, 6 and 8 respectively. With the implementation of bitboard, the data structure in this AI is optimized. Bit operations replace indexing, reading and writing of arrays, enhancing efficiency.



```
static bool alignment(uint64_t pos) {
  // horizontal
  uint64_t m = pos & (pos >> (HEIGHT+1));
  if(m & (m >> (2*(HEIGHT+1)))) return true;

  // diagonal 1
  m = pos & (pos >> HEIGHT);
  if(m & (m >> (2*HEIGHT))) return true;

  // diagonal 2
  m = pos & (pos >> (HEIGHT+2));
  if(m & (m >> (2*(HEIGHT+2)))) return true;

  // vertical;
  m = pos & (pos >> 1);
  if(m & (m >> 2)) return true;

  return false;
}
```

Figure 47: Implementation of bitboard alignment checking

3.3.3 Approach 3: Move Ordering

The exploration order of child nodes has a significant impact on the efficiency of Alpha-Beta pruning. If better nodes are explored first, the [Alpha; Beta] exploration window for the sibling nodes can be reduced, thus increasing pruning. Conversely, if the worst nodes are explored first, not much pruning occurs and the performance will be similar to the original minimax. In Connect Four, the best next move is unknown, so heuristics are implemented in an attempt to order moves optimally.

To begin with, move order can be improved by exploring moves in the middle column first. Middle column moves have higher chances to produce alignments and on average, they are better moves that should be explored first. The strategy is to explore from the middle column to the edge columns, which is in a static order of columns 4, 3, 5, 2, 6, 1, 7. This simple approach has little overhead as no complicated calculation is involved in reordering child nodes, yet it enhances efficiency by increasing pruning.

The following strategy avoids exploring bad moves leading to the opponent’s win at the next turn. There are three rules to follow. First, the disc should be placed in a column where the opponent has a winning slot to block the opponent’s winning. Second, the disc should not be placed under the opponent’s winning slot, as it facilitates the opponent’s winning. Third, if the opponent has two or more winning slots, it is guaranteed to lose and no moves need to be explored. By following these rules, child nodes of positions will be all possible positions that do not make the opponent win at the next move. This method reduces the total number of explored nodes and increases pruning.

In addition, the possible moves are evaluated by checking for 3-disc alignments to improve move ordering better. Moves with open-ended 3-disc alignments have the potential to create winning opportunities later in the game, and they are in general better moves than others. Similar to the winning condition check mentioned above, bitboard bitwise operations are performed to identify all open-ended 3-disc alignments and count the winning spots of a board position. With the number of winning slots identified for board positions, possible moves can be sorted by their winning slot count. In case of a tie, the initial middle-to-edge ordering mechanism retains.

3.3.4 Approach 4: Transposition Table

When the tree is being explored, the same positions are repeatedly analyzed as they can be reached from different sequences of moves. Re-computing the explored board positions can be avoided by caching the outcomes. Similar to dynamic programming, memory spaces are traded off to decrease computation time. A simple approach is employed to keep the newest positions and override previous entries with no collision management. This approach can increase the cache's hit rate as recently explored nodes are close to the current position and are more likely to hit. With Alpha-Beta pruning, the positions' values can be upper or lower bound. Both cases are stored in the transposition table, with a constant offset added to lower bound values to differentiate them.

As mentioned above, the position key contains 49 bits, associated with 8 bits value. 64 bits entry can store the key-value pair in the transposition table, but in an attempt to be more memory efficient, only the last 32 bits of the key will be stored. The last 32 bits can be gotten with position key modulo 2^{32}. The transposition table uses modular indexing. It contains S entries and S is an odd number, so an entry is stored at position key modulo S. With S and 2^{32} being prime with each other, by the Chinese remainder theorem, the index and key pair (key%S, key%2^{32}) can map back to the 49bits key which is smaller than Sx2^{32}. S is chosen to be the smallest prime number greater than 2^{23}. With 5 bytes per entry and 2^{23} entries available, the transposition table has a size of about 40MB.

After following all the tutorials by Pascal Pon and finishing all the optimizations, the AI can now solve all positions. Table 3 illustrates the benchmark conducted by Pascal Pons with 6000 positions. These positions are divided into 6 test sets according to the number of moves and remaining moves. In early game states where at least 28 moves remain, the AI needs around 5s to solve a position. The time taken is much longer than other test sets as it has a substantial average number of explored nodes per test case.

Test Set name	nb moves	nb remaining moves	Mean time	mean nb of pos	K pos/s
End-Easy	28 < moves	remaining < 14	4.722 µs	54.93	11,630
Middle-Easy	14 < moves <= 28	remaining < 14	39.90 µs	517.4	12,960
Middle-Medium	14 < moves <= 28	14 <= remaining < 28	3.736 ms	48,450	12,970

Begin-Easy	moves <= 14	remaining < 14	275.5 µs	3,693	13,400
Begin-Medium	moves <= 14	14 <= remaining < 28	113.4 ms	1,459,000	12,870
Begin-Hard	moves <= 14	28 <= remaining	5.667 s	72,490,000	12,790

Table 3: Benchmark result of the AI

Considering the robotic arm requires time to pick and place discs, it is not acceptable to wait for an extra 5s to calculate early game states as the added idling time would be too long. Pascal Pons's online Connect Four solver can solve early game positions much faster than the AI built from his tutorial [9]. After research, it is discovered that an extra book is added to the original AI to stimulate the calculation of early game states. The book is a pre-generated 32MB byte sequence file that stores key-value pairs [29]. By adding an extra function to load the book into the transposition table, it has 32MB of ready-to-use entries in the cache right after initialization. As a result, the table is no longer empty at the start, saving time for lengthy calculations at early game states.

3.3.5 Approach 5: Transforming into an Android Library

The Connect Four AI is written in C++, and it cannot be implemented directly into the Android application, which utilizes Java. However, it is possible to add C++ code to the Android project with the help of the Java Native Interface (JNI) [30]. First, add the native source files to the Android project under the cpp directory. Next, build script CMake is configured to build the code into a library. Then, the path to CMake is provided to Gradle such that it can package the library with the app.

Although the procedures sound simple, the actual implementation is not because the C++ codes are not intended to use as a native library. JNI acts as an intermediate layer for communication between Java and C++ codes. The board state is translated from Java string jstring into C++ string std::string via JNI for AI computations. The scores returned by the AI are also translated from C++ back to Java via JNI. The AI codes are also renovated to be an Android native library. A function is added for JNI to call, replacing the original main function. The original std::ifstream for book loading cannot be used to load the book under the same directory. In Android, assets should be put in the assets folder, which will be handled by Android Asset Packaging Tool (AAPT). To avoid compression of the book by AAPT, the file extension is changed to .jpg, which AAPT ignores. Changing the file extension has no impact on the book, as it is just a byte sequence file. AssetManager is used instead to read the book, and functions that load the book into the transposition table are modified to adapt to the change.

3.3.6 Experiment Result: Connect Four AI Android Library

The Connect Four AI has successfully transformed into an Android library with the above modifications. It can optimally solve any Connect Four board states. Any Android project can easily import this library to solve Connect Four perfectly. The library is available to download at https://github.com/bennywong3/c4solver-android-library.

3.4 Experiment 4: Disc Stacker

It is not simple to design a workable disc stacker. The task of the disc stacker is to roll out all 21 discs and auto-refill once the robotic arm picks a disc. In addition, all discs should be gripped at the same position and the stacker should store at least 21 discs. Several versions of stackers are developed to incorporate these features, and different tools and methods are utilized.

3.4.1 Approach 1: 3D Printing

Before building the model, a draft of the disc stacker is drawn (see Figure 48). The posited disc stacker should have three ramps utilizing gravity to refill discs at the exit. Discs are put inside the stacker at the top opening and rolls out at the bottom exit, where the robotic arm grasps discs.

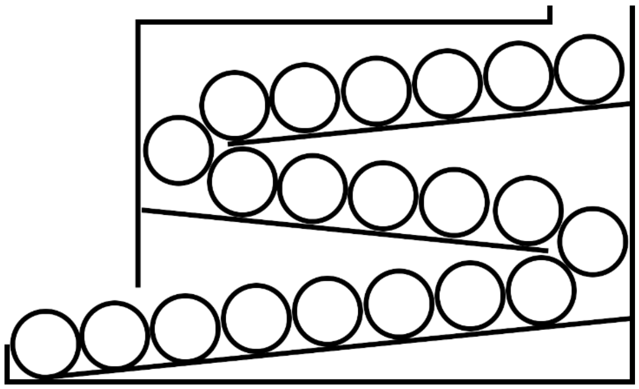


Figure 48: The draft of the proposed disc stacker

The first version is designed based on the draft (see Figure 49). Three tracks with a 5-degree slope have a height of at least 3 cm as the disc diameter is 3 cm. Unfortunately, the 3D model is too large to fit in the build plate of Ultimaker 3 extended, the exit has to be printed separately, and the model has to be printed vertically. As a result, support structures are generated for overhangs and bridges. However, support removal caused damage to the ramps, and discs cannot slide down smoothly (see Figure 50). Moreover, the track height provides no tolerance that discs could get stuck and fail to travel through.

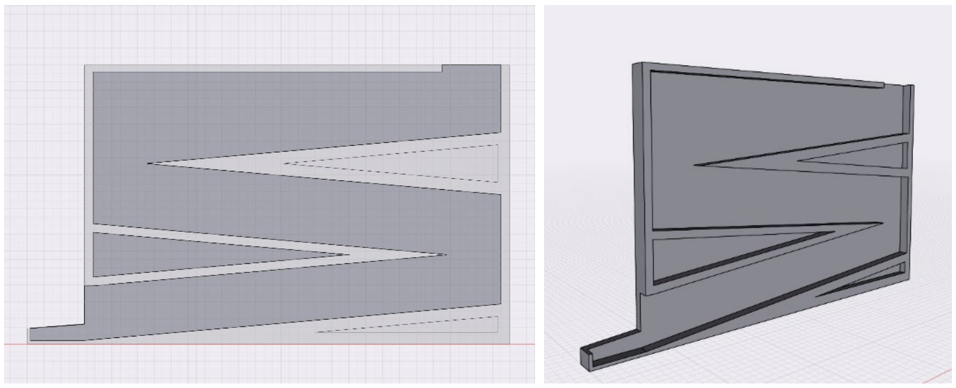


Figure 49: 3D model of disc stacker version 1

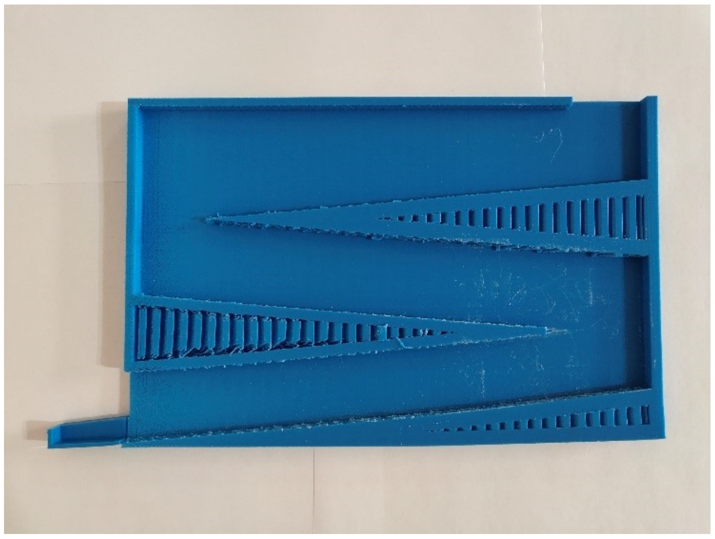


Figure 50: The printed disc stacker version 1

With the experience from version 1, version 2 is designed to increase the track height, providing more tolerance for discs to traverse (see Figure 51). In addition, Ultimaker S5, with a larger build volume, is chosen to print the model horizontally.

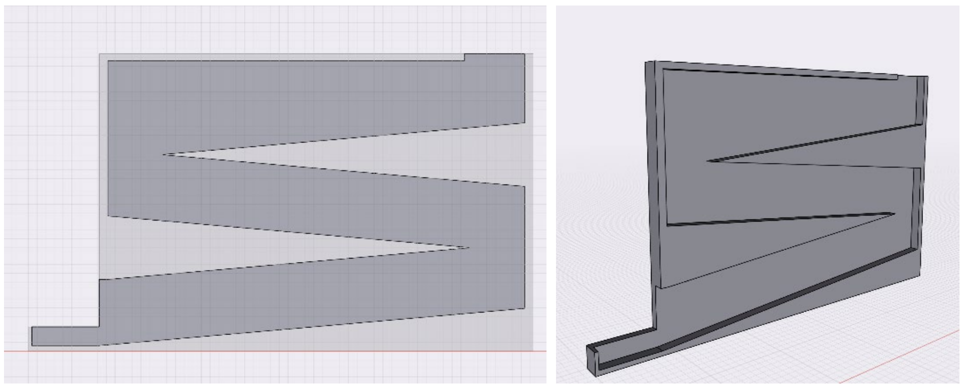


Figure 51: 3D model of disc stacker version 2

However, problems are identified in this version. Warping occurs even if a raft is used (see Figure 52). The exit is too shallow that sometimes discs roll to the exit and then pop out. The 5-degree ramps also fail to provide enough gravitational force for every disc to roll.

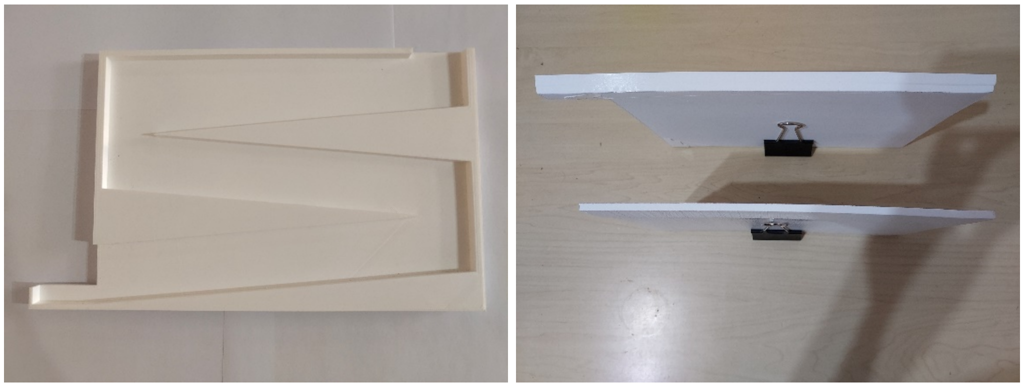


Figure 52: The printed disc stacker version 2

3.4.2 Approach 2: Laser Cutting

Shortcomings of using 3D printers are identified with the above experience. Print failures cannot be fixed easily and reprinting is a must. Models are subject to the build volume of 3D printers and print-in-place is not possible for large designs. It is also time-consuming to print large models and many PLAs are wasted if the print fails.

As the disc stacker design consists of only simple shapes, laser cutting can be utilized instead of 3D printing. The ramps, front and back plates can be created by cutting wooden and acrylic plates. Models can be sizable as the physical space inside 3D printers no longer bounds the designs. A new version is created to have 10-degree slopes, with a deeper exit that fits the rounded disc shape to prevent discs from popping out (see Figure 53).

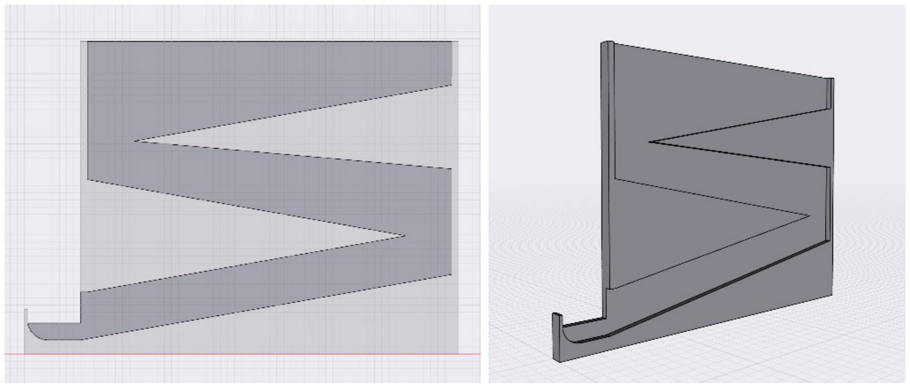


Figure 53: 3D model of disc stacker version 3

The model is separated into layers for cutting (see Figure 54), and they are translated into dxf format to use the GCC LaserPro Spirit for laser cutting. Discs can smoothly roll out with this design.

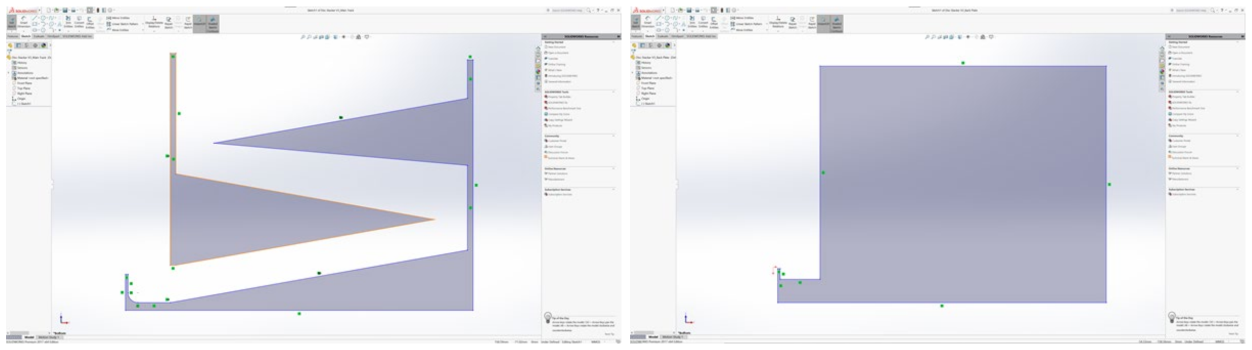


Figure 54: Layers for laser cutting

3.4.3 Experiment Result: Hybrid Approach

There are various methods that can combine the layers. However, the model fails to balance after integration as the base width is only 1.5cm. To address this issue, joint clamps are designed and 3D printed to act as the support base and hold layers together (see Figure 55).

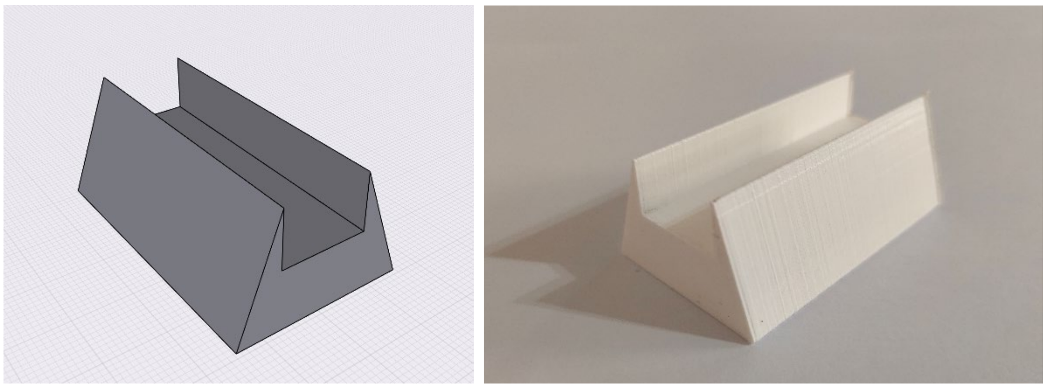


Figure 55: 3D model and the printed joint clamp

The trapezoidal prism shape provides rigid support to the stacker, and the groove tightly joints layers together. Figure 56 shows the final version of the disc stacker. Layers are assembled by the joint clamps and can be taken apart effortlessly when needed. Multiple tests are conducted to prove that it can smoothly roll out all 21 discs to the exit with no disc popping out. It is assured that the robotic arm can grasp all discs at the exit position.

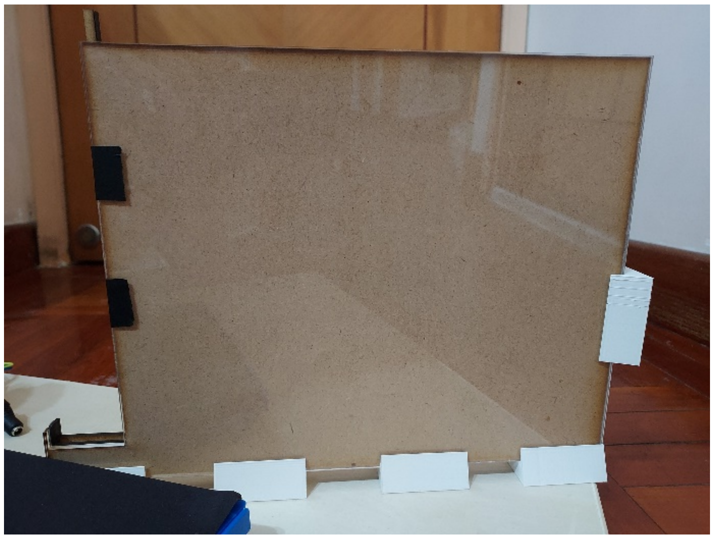


Figure 56: The final disc stacker

4. Limitation and Future Work

This chapter describes the limitations of the product delivered in this project, subdivided into software and hardware limitations. Future work can be done to undermine their effects, add extra functionalities to the product, and improve players' gaming experience.

4.1 Software Limitation

Android Studio is the chosen platform for development, so the delivered application cannot provide iOS support. This limitation is unavoidable. However, it is insignificant as well. As cross-platform support is not the primary objective of this project, this limitation is considered a minor drawback.

Another limitation is found in Connect Four game board detection. The 42 circular slots can be seen through at the original game board. However, when put into a complex environment, the detection algorithm fails to identify the board and discs effectively. For example, if the background has circular patterns, more than 42 circles will be identified and thus cannot accurately recognize the board. Furthermore, disc detection will fail if the board is placed in front of a pink or teal-colored wall as discs share the same color with the background. Therefore, the board is modified by installing black paper at the back. The above circumstances can be avoided with the black paper, but the see-through property of the game board is no longer available.

4.2 Hardware Limitation

In the project, it is assumed that the game board and the robotic arm are center-aligned. The distance between the robotic arm, game board and disc stacker is assumed to be fixed to pick and place discs into different columns with the coordinates preset. If the board is placed at another position, the arm fails to place discs inside the board. Only the distance between the board and the mobile phone can be changed during the board detection process. It is necessary to have the same setup for the robotic arm to have expected performance.

Another limitation is found in picking up discs. Ideally, the robotic arm can act like a human, picking a disc from a group of laying down discs under the vertical board and placing it inside the board. Object recognition on the laying down discs is compulsory to accomplish this. However, as the mobile phone camera is used to capture and analyze the vertical board, it is not feasible to simultaneously capture the horizontal laying down discs. Even if an extra camera is installed for disc capturing, it is necessary to build a new AI model from the ground up for recognizing and picking the laying down discs. To mitigate this, a disc stacker is employed such that the arm can pick up discs vertically at a fixed position without any recognition needed.

The project also assumes a rigid connection between components of the robotic arm and consistent behavior of the arm. However, in real-life operations, gears are not closely connected and displacement could occur, decreasing positioning accuracy. Besides, plastic gears are easier to be worn out and damaged, again aggravating the positioning performance. As a result, the robotic arm scarcely fails to place a disc into the minimal column gap.

4.3 Future Development

Addressing the limitation in board detection, a new approach can be explored such that the application can still distinguish the board from the surroundings even if the board can be seen through and noisy backgrounds are captured. For example, 3D object detection could be used to differentiate the game board from the environment. In addition, the current approach to board and disc detection can also be enhanced to be more resilient to various circumstances.

The Arm Controller can also add a manual calibration function such that the distance between the robotic arm, game board and disc stacker need not be fixed. In the final product, users can move the arm to the disc-picking and seven column positions via buttons inside the Arm Controller. Users can also manually control the arm with the gamepad. Extra functions can be included to define the board and stacker position manually. Users move the arm to the seven column and disc picking positions, and the application overrides the preset coordinates with these positions. Then, the application can locate the new board and stacker positions and play Connect Four with the players.

It is also possible to implement a difficulty system in the Player VS AI Mode. Players can only play with the perfect Connect Four AI in the delivered product, which could be too challenging and provide no sense of accomplishment for some players. Other Connect Four AIs can be developed to deliver different difficulty levels to players. For example, machine learning can be applied to build an AI that makes a similar strategy as a human player. Depth limit search can also be utilized to create a weaker version of the perfect AI.

5. Conclusion

This project aims to construct a Connect Four AI player that automatically picks and places discs using a mobile application and a 3D printed robotic arm. To achieve this objective, the employed implementations of the software and hardware components are discussed in this report. Multiple tools and techniques in software and hardware were applied in this project. In particular, several experiments were carried out to select the best approaches. Through optimizing the minimax algorithm with Alpha-Beta pruning, a perfect Connect Four AI is built to play against human players in the delivered application. By comparing the performance of different object recognition techniques, circle detection and HSV color detection are chosen to generate fast and accurate results of board states. With 3D printing and laser cutting, a disc stacker is constructed to facilitate disc picking. Despite having limitations in the application and robotic arm, the deliverables can further be improved by future undertakings, enhancing the Connect Four gaming experience and usability of the product under different scenarios.

To conclude, the finished product accomplishes the objective mentioned earlier with extra features that the AI opponent is perfect and unbeatable, and players can play games other than Connect Four with the application. The final product demonstrates the robustness of AI in strategic thinking and object recognition, which can be utilized in STEM education. Students' interest in AI and computer science can be aroused by playing with the deliverables.

References

[1] "Deep Blue (Chess Computer)," *Chess.com*. [Online]. Available: https://www.chess.com/terms/deep-blue-chess-computer. [Accessed: 27-Oct-2021].

[2] P. Mozur, "Google's AlphaGo defeats Chinese go master in win for A.I.," *The New York Times*, 23-May-2017. [Online]. Available: https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html. [Accessed: 27-Oct-2021].

[3] M. E. Moran, "Evolution of robotic arms," *Journal of robotic surgery*, 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4247431/. [Accessed: 27-Oct-2021].

[4] G. Team, "The rules of Connect 4 (according to M. Bradley & Hasbro)," *Gamesver*, 02-Jul-2021. [Online]. Available: https://www.gamesver.com/the-rules-of-connect-4-according-to-m-bradley-hasbro/. [Accessed: 27-Oct-2021].

[5] P. Pons, "Part 1 – introduction," *Solving Connect 4: how to build a perfect AI*, 01-May-2016. [Online]. Available: http://blog.gamesolver.org/solving-connect-four/01-introduction/. [Accessed: 27-Oct-2021].

[6] "4 in a row king," *Mobirix*, 20-Dec-2020. [Online]. Available: https://play.google.com/store/apps/details?id=com.mobirix.connectfour. [Accessed: 17-Apr-2022].

[7] "4 in a row," *Quarzo Apps*, 4-Jan-2022. [Online]. Available: https://play.google.com/store/apps/details?id=com.quarzo.fourinarow. [Accessed: 17-Apr-2022].

[8] P. Pons, "Solving connect 4: How to build a perfect AI," *Solving Connect 4: how to build a perfect AI*. [Online]. Available: http://blog.gamesolver.org/. [Accessed: 17-Apr-2022].

[9] P. Pons, "Connect 4 solver," *Game Solver*. [Online]. Available: https://connect4.gamesolver.org/. [Accessed: 17-Apr-2022].

[10] K. L. Busbee, “Integrated Development Environment,” *Rebus Press*, 15-Dec-2018. [Online]. Available: https://press.rebus.community/programmingfundamentals/chapter/integrated-development-environment/. [Accessed: 27-Oct-2021].

[11] E. Protalinski, “Google releases Android Studio 1.0, the first stable version of its ide,” *VentureBeat*, 08-Dec-2014. [Online]. Available: https://venturebeat.com/2014/12/08/google-releases-android-studio-1-0-the-first-stable-version-of-its-ide/. [Accessed: 27-Oct-2021].

[12] “Mobile Operating System Market Share Worldwide,” *StatCounter Global Stats*. [Online] Available: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed: 27-Oct-2021].

[13] “Firebase,” *Google*. [Online]. Available: https://firebase.google.com/. [Accessed: 17-Apr-2022].

[14] “The color system,” *Material Design*. [Online]. Available: https://material.io/design/color/the-color-system.html. [Accessed: 22-Jan-2022].

[15] R. Sparks, “Gradient-nature effortlessly achieves what takes designers so much effort to copy.,” *Medium*, 02-May-2021. [Online]. Available: https://uxplanet.org/gradient-nature-effortlessly-achieves-what-takes-designers-so-much-effort-to-copy-e08c07e82421. [Accessed: 17-Apr-2022].

[16] S. Subramaniyan, “The rounded user experience,” *Medium*, 30-Jul-2020. [Online]. Available: https://uxplanet.org/the-rounded-user-experience-ff7a1898ab33. [Accessed: 17-Apr-2022].

[17] E. Macpherson, “The UX honeycomb: Seven essential considerations for developers,” *Medium*, 08-Oct-2019. [Online]. Available: https://medium.com/mytake/the-ux-honeycomb-seven-essential-considerations-for-developers-accc372a398c. [Accessed: 17-Apr-2022].

[18] “Hough Circle Transform,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html. [Accessed: 18-Oct-2021].

[19] "Minimax Algorithm in Game Theory | Set 1 (Introduction)," *GeeksforGeeks*, 31-Mar-2021. [Online]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/. [Accessed: 27-Oct-2021].

[20] "Minimax algorithm in Game theory: Set 4 (alpha-beta pruning)," *GeeksforGeeks*, 18-Aug-2021. [Online]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/. [Accessed: 27-Oct-2021].

[21] S. Edelkamp, and P. Kissmann, "Symbolic Classification of General Two-Player Games," in *Ki 2008: Advances in artificial intelligence*, Heidelberg, Berlin: Springer, 2008, pp. 185–192.

[22] B. Haran, "Connect four - numberphile," *YouTube*, 01-Dec-2013. [Online]. Available: https://www.youtube.com/watch?v=yDWPi1pZ0Po. [Accessed: 17-Apr-2022].

[23] *The on-line encyclopedia of integer sequences® (OEIS®)*. [Online]. Available: https://oeis.org/A212693/b212693.txt. [Accessed: 17-Apr-2022].

[24] F. Tobler, "Robotarm by ftobler," *Thingiverse*, 14-Aug-2016. [Online]. Available: https://www.thingiverse.com/thing:1718984. [Accessed: 17-Apr-2022].

[25] 20sffactory, "20sffactory/COMMUNITY_ROBOT_ARM: Repository of Community Robot Arm Documents," *GitHub*, 20-Jan-2022. [Online]. Available: https://github.com/20sffactory/community_robot_arm. [Accessed: 17-Apr-2022].

[26] Hkucs-Makerlab, "Hkucs-makerlab/robotarm," *GitHub*, 26-May-2020. [Online]. Available: https://github.com/hkucs-makerlab/robotArm. [Accessed: 17-Apr-2022].

[27] Dejan, "G-code explained: List of most important G-code commands," *How To Mechatronics*, 06-May-2020. [Online]. Available: https://howtomechatronics.com/tutorials/g-code-explained-list-of-most-important-g-code-commands/. [Accessed: 17-Apr-2022].

[28] 20sffactory, "Firmware guide," *20sffactory*, 20-Jan-2022. [Online]. Available: https://www.20sffactory.com/robot/resource/firmware. [Accessed: 17-Apr-2022].

[29] P. Pons, "Release opening book · Pascalpons/Connect4," *GitHub*, 25-Jan-2019. [Online]. Available: https://github.com/PascalPons/connect4/releases/tag/book. [Accessed: 17-Apr-2022].

[30] "Add C and C++ code to your project : android developers," *Android Developers*. [Online]. Available: https://developer.android.com/studio/projects/add-native-code. [Accessed: 17-Apr-2022].

Appendix A

Installation guide of the Connect Four game application and Arduino program

Follow these steps to install the Connect Four game application.

1. Visit the following link.
   https://github.com/bennywong3/Play-Connect4-with-Robotic-Arm-via-App
2. Click on the “Code” button on the right. A dropdown should appear.
3. Click on “Download ZIP”, and the repository should be downloaded.
4. Extract the zip file. A folder named “Play-Connect4-with-Robotic-Arm-via-App-main” should appear.
5. Install Android Studio and open the project folder in Android Studio.
6. Enable USB Debugging mode on the Android device and connect it to the computer.
7. After Android Studio recognizes the device, click the “Run ‘app’” button to install the Connect Four game application to the device.

Follow these steps to install the Arduino program.

1. Visit the following link.
   https://github.com/bennywong3/robotic-arm-arduino
2. Click on the “Code” button on the right. A dropdown should appear.
3. Click on “Download ZIP”, and the repository should be downloaded.
4. Extract the zip file. A folder named “robotic-arm-arduino-main” should appear. Rename it to “robotArm”.
5. Install Arduino IDE and open the “robotArm.ino” with the IDE.
6. Connect the Arduino Mega 2560 to the computer. Under “Tools”, choose the COM port that is connecting to the board and select the board as “Arduino Mega or Mega 2560”
7. Click “Upload” to install the program on the board.

Appendix B

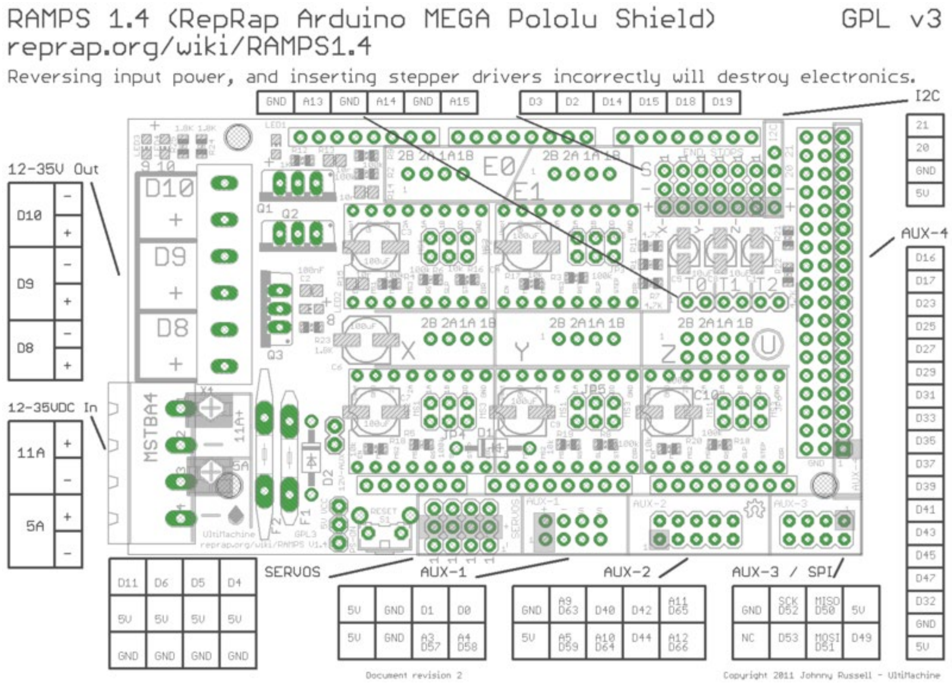
G-code command list

G-code example	**Function called**	**Description**
G1 X0 Y225 Z180	cmdMove(cmd)	Move the robotic arm to coordinate (0, 225, 180)
M3 T45	cmdGripperOn(cmd)	Open the gripper at 45 degrees (90 degrees is a wide-open and 0 degrees is a complete close)
M5 T45	cmdGripperOff(cmd)	Close the gripper at 45 degrees (90 degrees is a complete close and 0 degrees is a wide-open)
M17	cmdStepperOn()	Turn on stepper motors
M18	cmdStepperOff()	Turn off stepper motors
G28	homeSequence()	Use end-stop switches to perform auto home positioning

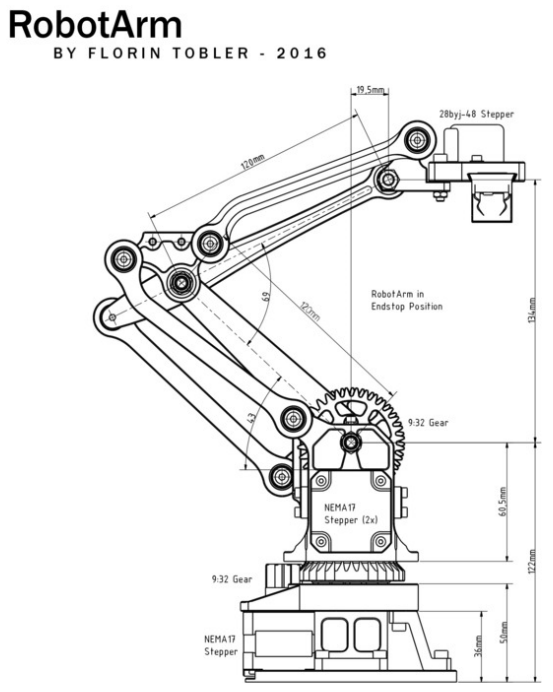
Appendix C

Related diagrams

RAMPS 1.4 pinouts



Blueprint of the original robotic arm by Florin Tobler



Ingrediences

6x	Bearing 4x13x5mm (624zz)
10x	Flanged Bearing 13x6x5mm (F686zz)
1x	Thrust Bearing 25x42x11mm (51105)
3x	NEMA17 Stepper Motor
1x	28byj-48 Stepper Motor
1x	RAMPS 1.4
1x	Arduino Mega 2560
3x	Stepper Driver (DRV8825 or A4988)
several	M4 and M3 Screws & Nuts..
	M5 Threaded Rod & Nuts..
	Washers, Wire, Cable ties..

Plastics

1x	Base	
1x	stabilizer	
1x	stabilizer_endstop	
2x	GearBig	
3x	GearSmall	
1x	Lever	
1x	LowerShank	
1x	UpperShank	
1x	Manipulator	
1x	Pleuel	
2x	Pleuel_bend	
1x	triplate	
1x	gripperBase	
2x	gripperFinger	
1x	gripperHolePlate	
1x	Socket	
1x	GearRotate	<< look for tight fit
3x	Leg	<< your preferred height
1x	baseRing	<< optional