Department of Mechanical and Mechatronics Engineering

MTE 121 Final Report

A Report Prepared For:
The University of Waterloo

Prepared By:
David Szczecina (20964967) Rohan Sharma (20947156)
Marko Vehauc (20952000) Chanuth Weeraratna (20949079)
MTE 121
150 University Ave W.
Waterloo, Ontario, N2N 2N2

November 26, 2021

Table of Contents

Table of Figures

Table of Tables

1.0 Introduction

Manually dealing cards before a card game can be tedious and subject to human error. This problem is mitigated by automating the dealing process with a robot. Furthermore, removing the need for a person to deal cards provides the players a break between games.

The card-dealing robot, shown in [FIGURE 1] deals playing cards to a variable number of players and number of cards per player, providing sufficient customization to be able to deal cards for a variety of games and situations. In addition, the robot includes three pre-programmed deal patterns which correspond to three common card games, namely Poker, President, and Go Fish, and are activated using the colour sensor. This saves the user the hassle of manually inputting all the parameters every time they want to play those games.

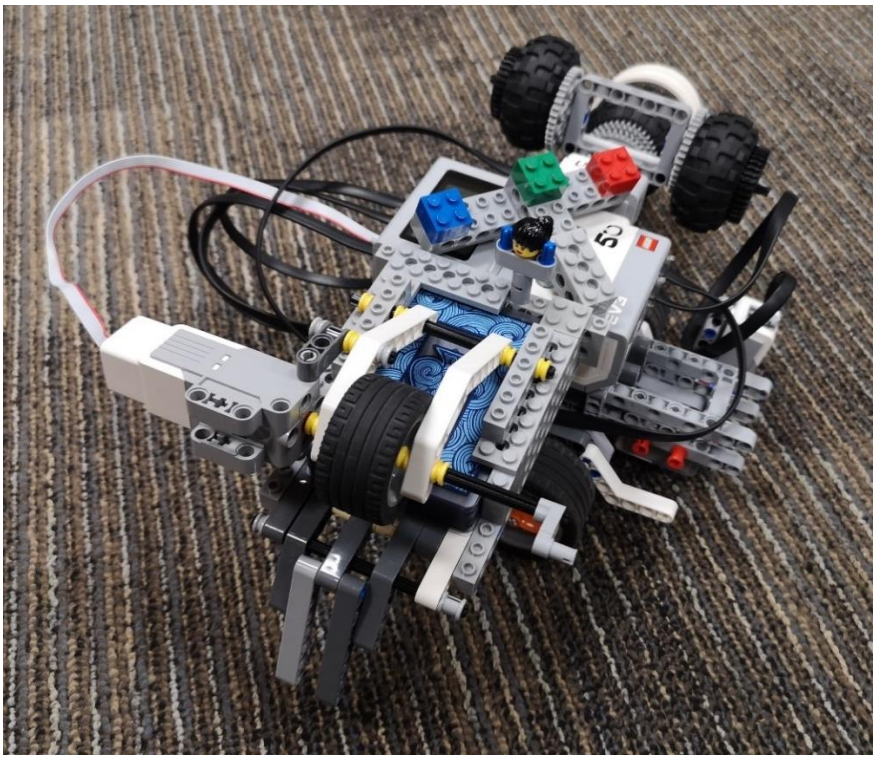


Figure 1: The Card-Dealing Robot

2.0 Software Design and Implementation

2.1 Description

Our software takes manual input through the EV3 interface and uses the information to move the robot to different positions on a table and deal cards for specific games. The colour sensor mode has simplified user inputs that makes the robot deal cards for certain games.

The main function, as shown in Figure 2, loops a main menu, and allows the user to select between Manual and Colour Mode as many times as they need, until the user presses the button to quit the program.

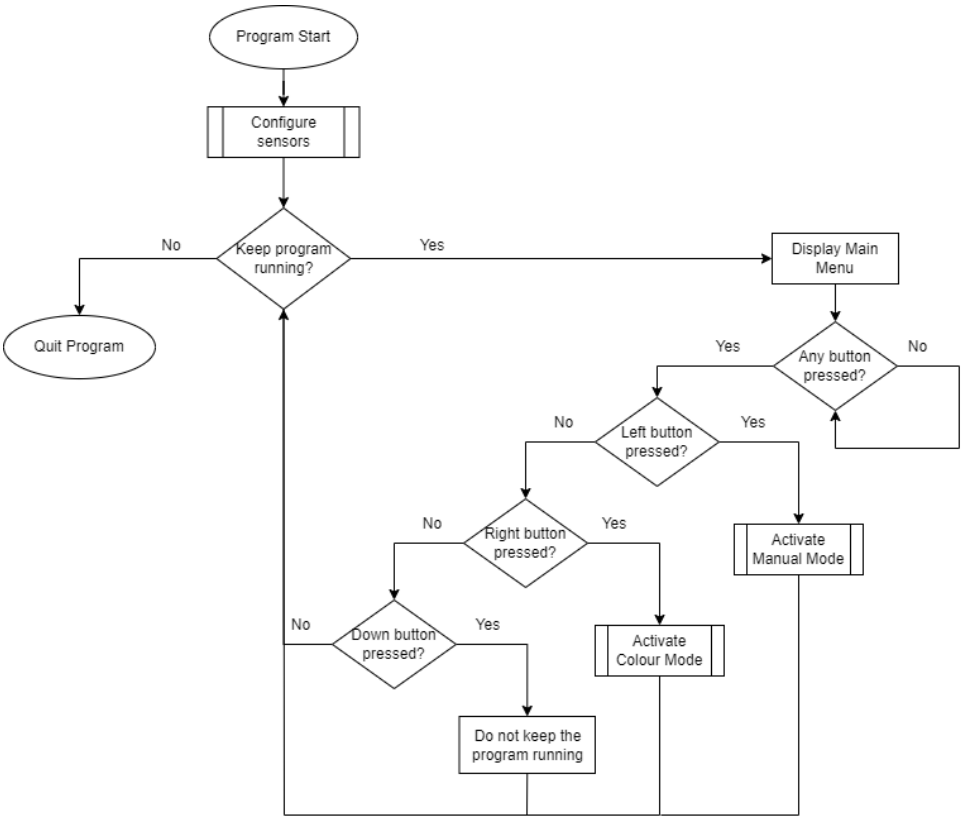


Figure 2: Flowchart for the Main Function

2.2 Task List

Manual Mode:

- User Input Phase:
    - Jokers in the deck
    - Numbers of players
    - Number of cards per player
    - Deal linearly or radially
- Execute deal according to user input

Colour Mode:

- Colour Blue (Go Fish)
    - Prompt for the number of players
    - Deal in a line
    - Ask user to “go fish”—user presses enter button to deal a card
- Colour Green (Poker)
    - Prompt for the number of players
    - Deal in a circle
    - Deal community cards (Flop, Turn, River)
    - Burns cards between rounds

- Colour Red (President)
    - Prompts for the number of players
    - Determine the number of cards per player
    - Deal in a circle

In Manual Mode, the user interacts with the buttons and display of the robot to input specific settings for dealing. The four tasks in the user input phase of manual mode are to prompt the user for whether the jokers are in the deck, the number of players, the number of cards per player, and whether to deal linearly or radially.

After the user input phase of Manual Mode, the robot enters the dealing phase where its task is to deal cards according to the user input.

Next is Colour Mode, which prompts the user to scan one of three colours, each which corresponds to a different pre-programmed card deal to accommodate for different games. Namely, the robot should deal for Go Fish when it detects blue, Texas Hold 'Em Poker when it detects green, and President when it detects red.

The Go Fish function has 3 main tasks: accepting user input for number of players, dealing a pre-programmed number of cards in a line, and afterwards dealing one card from the draw pile every time the enter button is pressed.

The Poker function has 6 main tasks: accepting user input for number of players, dealing 2 cards to each player radially, dealing the three rounds of community cards (the flop, turn, and river), and adding one card to the discard pile between each round of dealing.

The President function has 2 main tasks: accepting user input for number of players, and radially dealing the entire deck, split evenly to each player. This second task involves calculating how many cards to deal to each player given the number of players and total number of cards.

In the early design stage, a fourth colour mode function "Quick Deal" was proposed. However, it was removed during the coding process because it was deemed to be trivial.

2.3 Design Decisions

We chose to program the different card games to either deal cards in a line or in a circle. Both modes use 3 basic functions: dealCard() which deals cards using motorC, driveDist() which drives the robot forward, and rotateRobot() which rotates the robot.

The preprogrammed game modes playGoFish(), playPoker(), and playPresident() use the 2 card dealing modes to deal the correct number of cards into their respective places. Each game mode corresponds to a different colour, which allows the user to choose which card game to play by using the colour sensor.

Alternatively, the user can select the manual mode. In the manual mode, the number of players, number of cards per player, and whether jokers are in the deck are selected through a

series of questions on the EV3 interface. Using the left and right buttons as indicated by the text on the display, the values can be increased or decreased, and every time the value changes the screen refreshes to display the updated value. The user then selects how the cards should be dealt—in a circle or in a line—and the robot deals the cards into their piles.

Specifically, each function was picked based on our perception on how a user may interpret our robot. The first function picked was the colour mode to allow the user to start the robot and set up a game in under a minute. This function was picked for how simple and fast it is to set up a game.

The next option was manual mode, which was implemented for the scenario in which a player wanted to play a game that was not one of the preprogrammed ones. This function allowed for the user to decide on many options, from the number of players to the number of cards and how to deal, which gave the user infinite options on games to be played.

Moreover, each function was picked from the viewpoint of the user and what they would want from this robot, and what options would entice them to continue using this dealing machine.

Table 1: List of Functions

Function	Coder	What It Does
Void configSensors()	Rohan	Configures all sensors. (Gyro, Motor Encoder, Colour Sensor)
Int maxCardsPerPlayer(int numPlayers, int numCards)	David	Given the number of players and total number of cards, it determines the maximum number of cards each player can receive.
Int determineAngle(int numPlayers)	Marko	Given the number of players, it determines the angle between all players if they are sitting in a circle.
Void dealCards(int numCards)	Chanuth	Given an integer, it deals that many cards using motorC.
Void dealLinear(int numPlayers, int cardsPerPlayer)	David	Given the number of players and cards each player receives, the robot will deal the cards in a line.
Void radialDeal(int numPlayers, int numCards, int distance)	Marko	Given the number of players, distance from robot to players and cards each player receives, the robot will deal the cards in a circle.
Void manualMode()	David	Allows the user to manually select the number of cards in the deck, number of players, cards each

		player receives and how to deal the cards. Then deals the cards according to those parameters.
Void playPoker()	Marko	Asks the user to input parameters and deals cards in order to play poker.
Void playPresident()	Chanuth	Asks the user to input parameters and deals cards in order to play president.
Void playGoFish()	Rohan	Asks the user to input parameters and deals cards in order to play go fish. Then asks user to if they want to deal a card, to click enter button.
Void rotateRobot(int angle)	Marko	Given an integer, rotates the robot that many degrees.
Void colourMode()	Rohan	Uses the colour sensor to choose which game to play based on what colour it sees.
Int driveDist(float dist_cm)	Chanuth	Given a float, drives robot forward for that distance and returns the time it took to drive that far.

2.4 Data Storage:

The main variables for the program are stored at the top of each function. For deal cards the enc_limit is used alongside the card length variable to move the motor and deal one card. The Deal Linear and Poker function have the DIST_BW_PILES variable which sets the optimal length between each pile of cards. In Manual Mode, numCards, numPlayers, cardsPerPlayer, sets the user input for each function. Finally, DIST_TO_PLAYER is used in the President and Poker games along with the linear function and manual mode.

2.5 Testing

We started our testing process by compiling and running each of our functions individually. By testing our program in small parts, we were able to isolate where our problems were which shortened our debugging process.

Starting with the most basic functions, we first confirmed that our robot can deal cards, drive forward, and rotate. Next, we tested the functions that deal cards in a certain pattern, dealLinear() and radialDeal(); because we already tested the basic functions, we knew that the errors were not because of how they worked. Finally, we tested our functions that play games since these functions call all the previously mentioned smaller and medium functions.

As time progressed, this act of building on top of already confirmed working code allowed for the basic debugging to be isolated to a specific area and made the final debugging more manageable and more noticeable.

Furthermore, whenever we made changes to the code it was saved as a new version, with a changelog in the title block summarizing the differences between the current version and the previous one. This ensured that no previous work was lost, which allowed us to roll back to a previous version of the code if a newer version created unforeseen bugs. The inclusion of a

changelog further allowed us to pin down where the errors appeared in our code, which streamlined the debugging process and kept all the group members up to date on the changes.

2.6 Problems Encountered

Most of our functions did not work the first time we tried to run them on the robot. This was originally anticipated, so we planned to have 2 days where our group would meet up to debug our code and test it on the robot.

While some functions had small errors in them that caused them to not work properly, a few functions had to be completely redesigned so that they could work as intended. Certain functions such as the deal functions, specifically the linear function began to improperly deal with time.

In day 2 of our collaborative debug session, we noticed that the linear function would begin to drive and position the robot properly, but it would either deal more than one card at a time or would never stop dealing, in a way stuck in a loop.

Another problem faced was hardware issues with the specific robot, which began to fail successful portions of our program. Specifically, the colour sensor began to read improper colours, which resulted in the selection of an incorrect game. Fortunately, many of these problems were solved by resetting of the robot.

3.0 Conclusion

In summary, the robot serves to automate the card-dealing process, providing functionality for manual input and pre-programmed deals for specific games.

The main tasks of the robot are to accept user input using the buttons or colour sensor, dealing cards according to either the manual input for the functions for Poker, President, and Go Fish.

The most important features of the code that make the robot successfully do its tasks are the looping main menu that allows multiple games to be played without quitting the program, the user input phase of Manual Mode and the robot's ability to perform the deal specified, and the three pre-programmed deals selected using the colour sensor in Colour Mode.

3.1 Recommendations

The mechanical design of our robot could be redesigned to make it function better. Since the card holder and motorB are attached to one side of the robot, it is unbalanced and leans to one side. This causes one wheel to have more traction than the other which affects the way the robot drives. A temporary solution has been to add a counterbalance to one side. However, a more elegant and permanent solution could be to mount the cardholder on top of the robot or on the back side of it.

The mechanism that deals cards could also be redesigned to deal cards more consistently, as currently it will occasionally deal multiple cards at once. To solve this, numerous tests would

have to be conducted with different types of playing cards, motor speeds, and types of wheels mounted onto motorB that would grip the top card while allowing it card to glide over the cards below. Trial and error would help to solve this issue, but that would require an immense amount of time and effort for a such a small task in the grand scheme of this project, which as a team we felt was not needed at this time.

Furthermore, additional functionalities could be implemented to allow the robot to better adapt to a variety of uses and improve user interaction. Firstly, the addition of the quick deal would allow the user to quickly deal one card without having to reach under the dispensing wheel or go through the user input in Manual Mode. Secondly, adding a back button to the menu system would increase the ease of use, as it would allow the user to correct any input mistakes without needing to restart the program or wait through an undesirable deal.

3.2 Industry Changes

The robot deal cards and follows the correct path according to the game functions and the manual mode, however there are practical issues when using our robot design in the industry.

Firstly, we would change how the card deal. The software for dealCard() function is inaccurate since the length of cards do not always match the encoder distance as it either under or over measures the distance. Additionally, the card holder for the robot, the motor pushing the cards, and the cards itself produce too much friction, sometimes cause unintended cards to deal. For example, when the robot was set for manual mode to deal one card for each player, two or more cards could be dealt and sometimes even flipped. The games in the function such as poker, go fish, and president, all require the cards to be face down, therefore, this issue would greatly impact the experience of the game.

Finally, the last improvement to make the robot more practical would be to have the card deal function and the robot itself be faster by increasing the motor speed. Currently the motors are set so that the robot can maneuver precisely however it takes longer than it would take someone to manually deal it. Therefore, increasing the motor speed along with the precision would make the robot effective to be used on a variety of games aside from the currently implemented games.

Appendix A: Final Code Used in the Robot Demo

```
/*
	Combined Code v.20
	Team 50

	Changelog:
	* colourMode: quick deal removed
	* playPoker: DIST_TO_PLAYER increased from 50 -> 100 and
	  DIST_BW_CARDS from 20 -> 10
	* main: add shut off functionality
*/


// configures the sensors
void configSensors()
{
	SensorType[S3] = sensorEV3_Color;
	wait1Msec(50);
	SensorMode[S3] = modeEV3Color_Color;
	wait1Msec(50);
	SensorType[S4] = sensorEV3_Gyro;
	wait1Msec(50);
	nMotorEncoder[motorA] = 0;
	SensorMode[S4] = modeEV3Gyro_Calibration;
	wait1Msec(50);
	SensorMode[S4] = modeEV3Gyro_RateAndAngle;
	wait1Msec(50);
}


// determines the maximum number of cards per player
int maxCardsPerPlayer(int numPlayers, int numCards)
{
	int maxNumCards = 0;
	maxNumCards = (numCards / numPlayers);
	return maxNumCards;
}


// determines the turn angle when dealing radially
float determineAngle(int numPlayers)
{
	return 360 / numPlayers;
}


// deal cards
void dealCards(int numCards)
{
	nMotorEncoder[motorB] = 0;

	const int MED_SPEED = 12;
	const float CARD_LENGTH = 1.55;
```

```
	int card = numCards;
	int enc_limit = (card * CARD_LENGTH) * 180 / PI * 2.75;

	motor[motorB] = MED_SPEED;
	while (nMotorEncoder[motorB] < enc_limit)
	{}
	motor[motorB] = 0;
}


// Drive distance in metre
int driveDist(float dist_cm)
{
	const int MOTOR_POWER = 25;
	int enc_limit = dist_cm * 180 / (PI * 2.75);

	time1[T1] = 0;
	nMotorEncoder[motorA] = 0;

	if (dist_cm < 0)
		motor[motorA] = motor[motorD] = -MOTOR_POWER;
	else
		motor[motorA] = motor[motorD] = MOTOR_POWER;

	while (abs(nMotorEncoder[motorA]) < abs(enc_limit))
  {}
	motor[motorA] = motor[motorD] = 0;

	return time1[T1];
}


// Rotate Gyro
void rotateRobot(int angle)
{
	int motorPower = 25;
	tSensors gyroPort = S4;
	resetGyro(gyroPort);

	if (angle > 0)
	{
		motor[motorA] = -motorPower;
		motor[motorD] = motorPower;
	}
	else
	{
		motor[motorA] = motorPower;
		motor[motorD] = -motorPower;
	}

	angle = abs(angle);
	while (abs(getGyroDegrees(gyroPort)) < angle)
	{}
	motor[motorA] = motor[motorD] = 0;
```

```
}


// deals cards in a line
void dealLinear(int numPlayers, int cardsPerPlayer)
{
	const int DIST_BW_PILES = 20;
	// set distance between each pile (in cm)

	for (int pilesDealt = 0; pilesDealt < numPlayers; pilesDealt++)
	{
		dealCards(cardsPerPlayer);
		driveDist(DIST_BW_PILES);
	}
}


// deals cards in a circle
void radialDeal(int numPlayers, int numCards, int distance)
{
	int elapsedTime = 0;

	for (int counter = 1; counter <= numPlayers; counter++)
	{
		time1[T1] = 0;
		driveDist(distance);
		elapsedTime = time1[T1];

		wait1Msec(50);
		dealCards(numCards);
		wait1Msec(50);

		time1[T1] = 0;
		motor[motorA] = motor[motorD] = -25;

		while (time1[T1] < elapsedTime)
		{}

		motor[motorA] = motor[motorD] = 0;

		rotateRobot(determineAngle(numPlayers));
	}
}


// allows for user input
void manualMode()
{
	int numCards = 0, numPlayers = 0, cardsPerPlayer = 0;

	eraseDisplay();

	//are there jokers in the deck
	//one if statement
```

```
displayString(5, "Are there jokers in the deck?");
displayString(10, "Y/N (Left/Right buttons)");

while (!getButtonPress(buttonLeft) &&
       !getButtonPress(buttonRight))
{}
if (getButtonPress(buttonLeft))
{
       while (getButtonPress(buttonLeft))
       {}
       numCards = 54;
}
if (getButtonPress(buttonRight))
{
       while (getButtonPress(buttonRight))
       {}
       numCards = 52;
}

wait1Msec(50);

/*
  * set num players
  * left to decrease, right to increase
  * down button sets value and exits loop
  * in a loop so that the display is always updated
*/
bool numPlayersSelected = false;

while (!numPlayersSelected)
{
       eraseDisplay();
       displayString(5, "Number of players: %d", numPlayers);
       displayString(10, "Left to decrease");
       displayString(11, "Right to increase");
       displayString(12, "Down to enter");

       while (!getButtonPress(buttonLeft) &&
              !getButtonPress(buttonRight) &&
              !getButtonPress(buttonDown))
       {}

       if (getButtonPress(buttonLeft) && numPlayers > 1)
       {
              while (getButtonPress(buttonLeft))
              {}
              numPlayers--;
       }

       if (getButtonPress(buttonRight))
       {
              while (getButtonPress(buttonRight))
              {}
              numPlayers++;
```

```
        }

        if (getButtonPress(buttonDown))
        {
                while (getButtonPress(buttonDown))
                {}
                numPlayersSelected = true;
        }
}

wait1Msec(50);

/*
  * set cardsPerPlayer
  * left to decrease, right to increase
  * in a loop so that the display is always updated
  * down button sets value and exits loop if value satisfies
    conditions
  * checks if cardsPerPlayer*numPlayers >= numCards &
    cardsPerPlayer > 0
*/
bool numCardsSelected = false;

while (!numCardsSelected)
{
        eraseDisplay();
        displayString(5, "Cards per player: %d", cardsPerPlayer);
        displayString(10, "Left to decrease");
        displayString(11, "Right to increase");
        displayString(12, "Down to enter");

        while (!getButtonPress(buttonLeft) &&
               !getButtonPress(buttonRight) &&
               !getButtonPress(buttonDown))
        {}

        if (getButtonPress(buttonLeft) && cardsPerPlayer > 1)
        {
                while (getButtonPress(buttonLeft))
                {}
                cardsPerPlayer--;
        }

        if (getButtonPress(buttonRight) && cardsPerPlayer <
            maxCardsPerPlayer(numPlayers, numCards))
        {
                while (getButtonPress(buttonRight))
                {}
                cardsPerPlayer++;
        }

        if (getButtonPress(buttonDown))
        {
                while (getButtonPress(buttonDown))
```

```
                {}
                numCardsSelected = true;
            }
        }

        wait1Msec(50);

        // deal in circle or deal in line
        eraseDisplay();
        displayString(5, "Deal in circle (Left)");
        displayString(10, "Deal in line (Right)");

        while (!getButtonPress(buttonLeft) &&
               !getButtonPress(buttonRight))
        {}

        if (getButtonPress(buttonLeft))
        {
            while (getButtonPress(buttonLeft))
            {}
            eraseDisplay();
            const int DIST_TO_PLAYER = 50;
            radialDeal(numPlayers, cardsPerPlayer, DIST_TO_PLAYER);
        }

        if (getButtonPress(buttonRight))
        {
            while (getButtonPress(buttonRight))
            {}
            eraseDisplay();
            displayString(0, "woo linear mode!");
            dealLinear(numPlayers, cardsPerPlayer);
        }
}


void playPoker()
{
        // declaring constants and ints
        int numPlayers = 4;
        const int MAX_PLAYERS = 10;
        const int MIN_PLAYERS = 2;

        const int NUM_CARDS = 2;
        const int DIST_TO_PLAYER = 100;

        int distToDiscard = 30;
        const int DIST_BW_CARDS = 10;

        const int NUM_ROUNDS = 3;
        int cardsPerRound[NUM_ROUNDS] = {3, 1, 1};
        string roundName[NUM_ROUNDS] = {"flop", "turn", "river"};

        displayString(4, "Select the number of players");
```

```
displayString(6, "(Press the enter button");
displayString(7, "to confirm)");

// user input for number of players
while (!getButtonPress(buttonEnter))
{
	displayString(3, "%d", numPlayers);

	if (getButtonPress(buttonUp) && numPlayers < MAX_PLAYERS)
	{
		while (getButtonPress(buttonUp))
		{}
		numPlayers++;
	}

	if (getButtonPress(buttonDown) && numPlayers > MIN_PLAYERS)
	{
		while (getButtonPress(buttonDown))
		{}
		numPlayers--;
	}
}

eraseDisplay();

// deal 2 cards to each player
radialDeal(numPlayers, NUM_CARDS, DIST_TO_PLAYER);

// deal the community cards
for (int dealRound = 0; dealRound < NUM_ROUNDS; dealRound++)
{
	// discard 1 card
	driveDist(-distToDiscard);
	dealCards(1);
	driveDist(distToDiscard);

	// prompt for user input
	displayString(4, "Press the enter button");
	displayString(5, "to deal the %s", roundName[dealRound]);

	// wait for user input
	while (!getButtonPress(buttonEnter))
	{}
	while (getButtonPress(buttonEnter))
	{}

	// clear display
	eraseDisplay();

	for (int cardsDealt = 1; cardsDealt <=
	     cardsPerRound[dealRound]; cardsDealt++)
	{
		driveDist(DIST_BW_CARDS);
		distToDiscard += DIST_BW_CARDS;
```

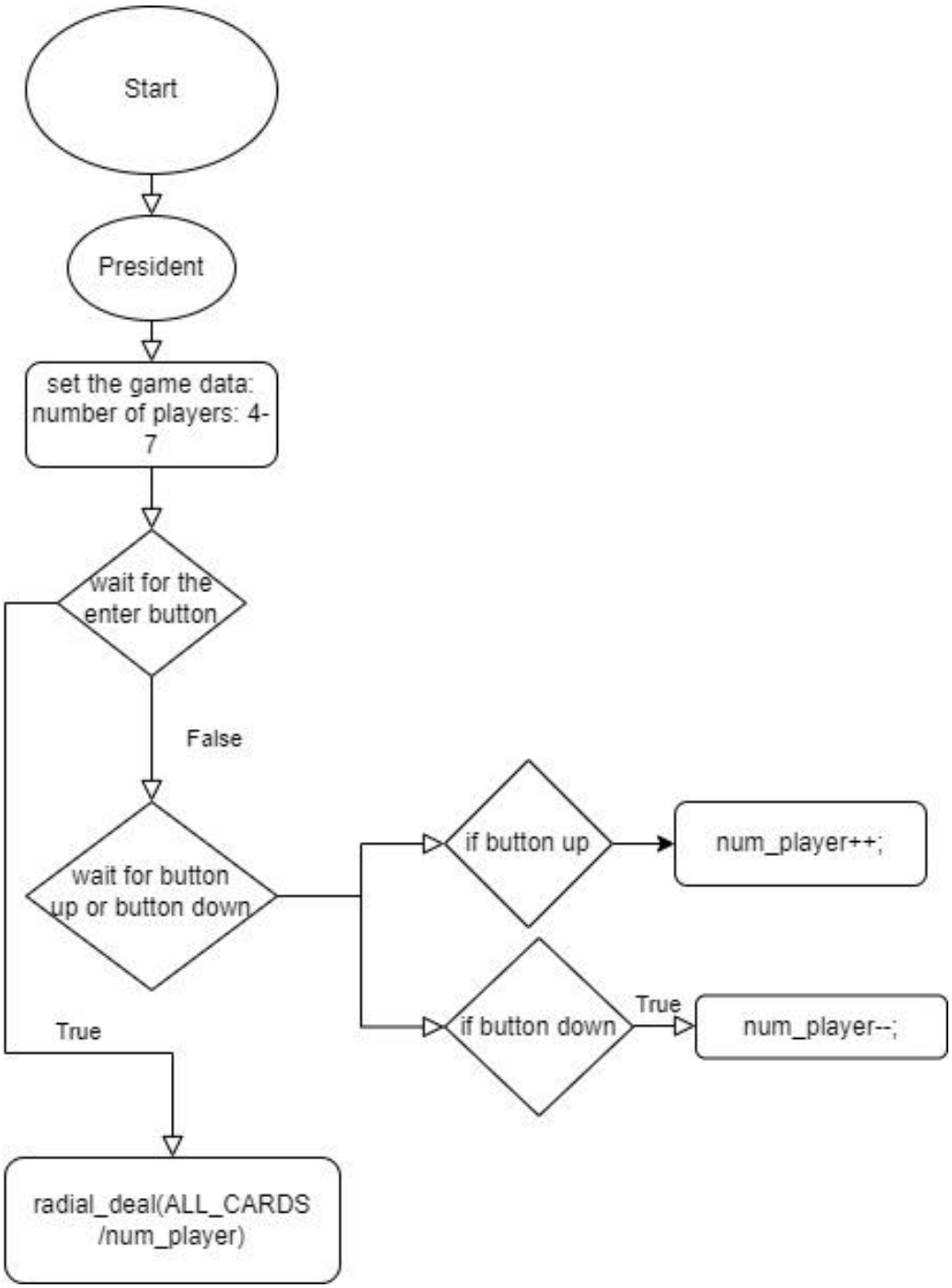
```
                        dealCards(1);
                }
        }
}


void playPresident()
{
        int numPlayers = 4;
        int cardsPerPlayer = 0;
        const int MAX_PLAYERS = 7;
        const int MIN_PLAYERS = 3;
        const int DIST_TO_PLAYER = 50;
        const int ALL_CARDS = 54;

        displayString(4, "Select the number of players");
        displayString(7, "(Press the enter button");
        displayString(8, "to confirm)");

        // user input for number of players
        while (!getButtonPress(buttonEnter))
        {
                displayString(3, "%d", numPlayers);

                if (getButtonPress(buttonUp) && numPlayers < MAX_PLAYERS)
                {
                        while (getButtonPress(buttonUp))
                        {}
                        numPlayers++;
                }

                if (getButtonPress(buttonDown) && numPlayers > MIN_PLAYERS)
                {
                        while (getButtonPress(buttonDown))
                        {}
                        numPlayers--;
                }
        }

        cardsPerPlayer = ALL_CARDS/numPlayers;
        // integer division; ensures num of cards per player rounds down

        eraseDisplay();

        radialDeal(numPlayers, cardsPerPlayer, DIST_TO_PLAYER);
}


void playGoFish()
{
  int numPlayers = 4;
        const int MAX_PLAYERS = 7;
        const int MIN_PLAYERS = 2;
        const int ONE_CARD = 1;
```

```
	// user input for number of players
	while (!getButtonPress(buttonEnter))
	{
		displayString(3, "Number of players: %d", numPlayers);

		if (getButtonPress(buttonUp) && numPlayers < MAX_PLAYERS)
		{
			while (getButtonPress(buttonUp))
			{}
			numPlayers++;
		}

		if (getButtonPress(buttonDown) && numPlayers > MIN_PLAYERS)
		{
			while (getButtonPress(buttonDown))
			{}
			numPlayers--;
		}
	}

	eraseDisplay();


if (numPlayers > 3)
{
	dealLinear(numPlayers, 5); // deal 5 cards per player
}
else if (numPlayers <= 3)
{
	dealLinear(numPlayers, 7); // deal 7 cards per player
}


bool gameActive = true;

while (gameActive)
{
	displayString(4, "Game has begun!");
	displayString(5, "Press the enter button");
	displayString(6, "to deal a card");
	displayString(10, "Press the down button to quit");

	while (!getButtonPress(buttonEnter) &&
		!getButtonPress(buttonDown))
	{}

		if (getButtonPress(buttonEnter))
		{
			while(getButtonPress(buttonEnter))
			{}
			dealCards(ONE_CARD);
		}
		if (getButtonPress(buttonDown))
```

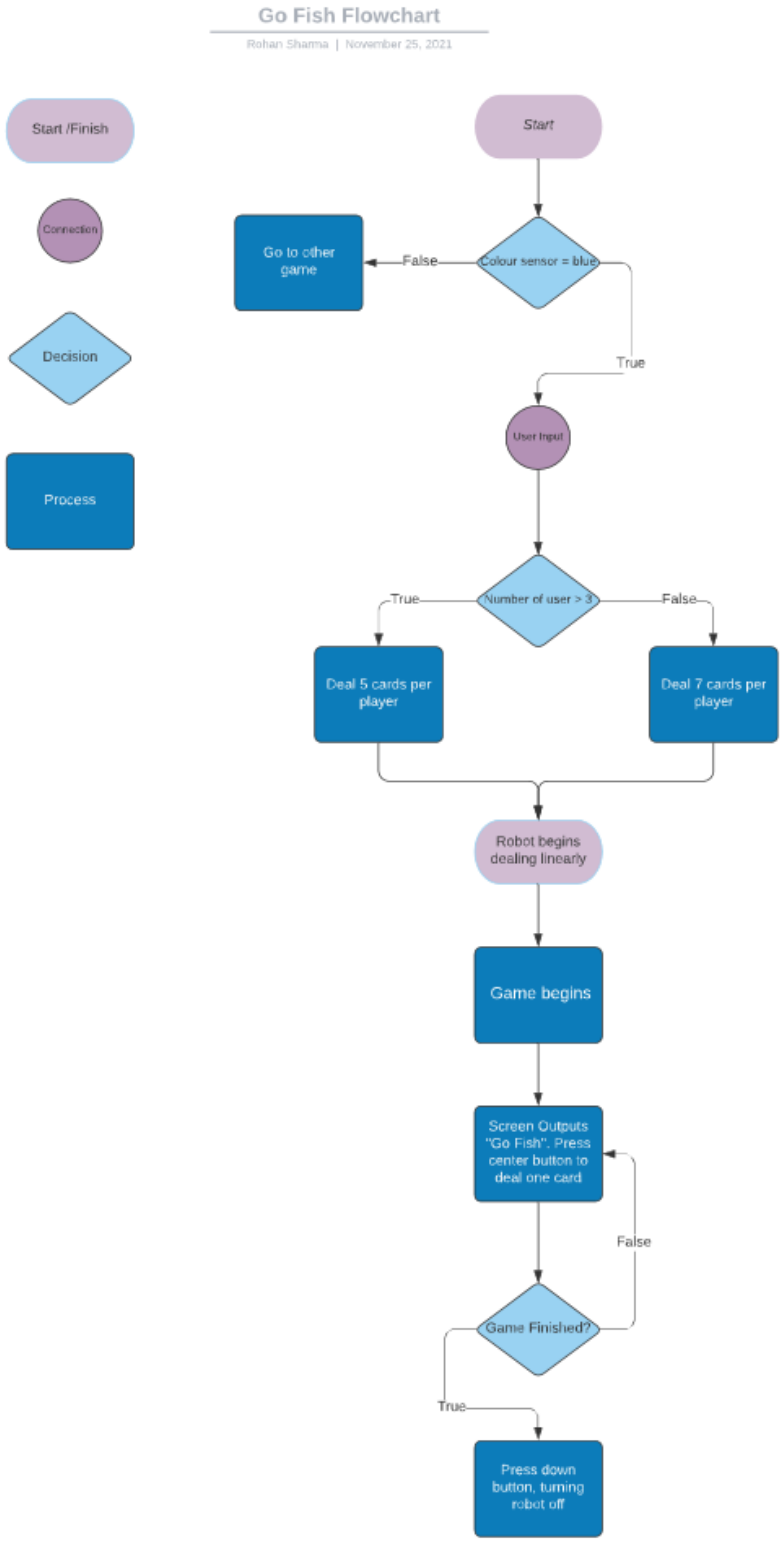
```
                {
                        while(getButtonPress(buttonDown))
                  {}
                        gameActive = false;
                }
  }
}


// selects a game
void colourMode()
{
        eraseDisplay();
        displayString(4, "Colour mode selected");

        // waits until user shows a colour
        while (SensorValue[S3] != (int)colorGreen && SensorValue[S3] !=
               (int)colorBlue && SensorValue[S3] != (int)colorRed)
        {}

        eraseDisplay();

        if (SensorValue[S3] == (int)colorGreen)
        {
           displayString(1, "Poker");
           playPoker();
        }
        else if (SensorValue[S3] == (int)colorBlue)
        {
           displayString(1, "Go Fish");
           playGoFish();
        }
        else if (SensorValue[S3] == (int)colorRed)
        {
           displayString(1, "President");
           playPresident();
        }
}


task main()
{
        configSensors();

        bool keepRunning = true;

        while (keepRunning)
        {
                eraseDisplay();
                displayString(5, "Manual Mode (Left) or");
                displayString(6, "Colour Mode (Right)?");
                displayString(10, "press the down button to quit");

                while (!getButtonPress(buttonLeft) &&
```

```
                !getButtonPress(buttonRight) &&
                !getButtonPress(buttonDown))
            {}

            if (getButtonPress(buttonLeft))
            {
                while (getButtonPress(buttonLeft))
                {}
                manualMode();
            }

            if (getButtonPress(buttonRight))
            {
                while (getButtonPress(buttonRight))
                {}
                colourMode();
            }

            if (getButtonPress(buttonDown))
            {
                while (getButtonPress(buttonDown))
                {}
                keepRunning = false;
            }
        }
}
```

Appendix B: Flowchart for President



Appendix C: Flowchart for Go Fish



Appendix D: Flowchart for Poker