Final Report MTE 100 - Monopoly ATM Group 846

Josh Zwiebel, 20772297, University of Waterloo

Soyazhe Khan, 20770492, University of Waterloo

Wais Shahbaz, 20782890, University of Waterloo

Leif Blake, 20766152, University of Waterloo

Date of Submission: December 3rd, 2018

Summary

In the game of Monopoly, the objective is to amass the highest amount of money possible, while bankrupting any opponents. Unfortunately, it is straightforward to cheat the financial system of the game, and therefore a need exists for a way to better manage the game's money. The proposed solution is an Automated Teller Machine (ATM) that would store players' money and allow players to complete the necessary transactions for the game's completion.

Constraints and criteria for designs are presented, as well as how they changed throughout the design process. Constraints consisted of:

- Restrictions on the movement of the gantry
- Size of the enclosure
- Project deadlines
- Color sensor tolerance

The final mechanical design was a Cartesian-based, pick and place, electromechanical system. It used a multitude of actuated movement, in conjunction with a pneumatic system, to fulfill its required tasks. It was built using a combination of Tetrix pieces, Lego Mindstorm pieces, and 3D printed parts.

The software design was implemented in a hierarchy of functions that allowed a member of the team to work on parts of the software independently and allowed for easier integration of the software. Details of each function are discussed in the report.

Verification of the ATM's success in meeting the constraints of the project during the demonstration is presented, as well as the success criterion of 97.5% success in the identification and distribution of bills. However, the ATM encountered a software bug during the demonstration.

The predicted project timeline is compared with the actual timeline. Tasks were initially broken up into smaller sub-tasks to allow members to work independently. Changes in the schedule mainly arose from unexpected delays in some sub-tasks and extraneous factors.

The conclusion of the project shows that the implementation of the mechanical features and the hierarchical software works with a high level of success, achieving all constraints through verification and nearly attaining all criteria for success of the project.

Finally, the following recommendations are discussed:

- Moving the money input and output system to the side of the gantry
- Decreasing the use of literals in code
- Achieving more organized segmentation of the code

Table of Contents

Table of Figures and Tables

1. Introduction

Often times the enjoyment that is found in a game of Monopoly can be ruined by one or two bad players. Monopoly is a game where the goal is to amass the most money by the end of the game and unfortunately, it is relatively simple for a player to mismanage finances and take more money from the bank than they are allotted, or neglect to pay another player. It can be seen that there is value to Monopoly players in a device that safeguards a game from potential cheating and organizes finances to ensure no mistakes are made while dealing with money in a given game. Any Monopoly money management system should allow players to:

1. Withdraw and deposit money from their accounts
2. Transfer funds between players
3. Allow a player to obtain money from the bank
4. Allow a player to declare bankruptcy

The Monopoly ATM is a proposed solution to this problem. If the Monopoly ATM can accomplish these tasks it will streamline the process of playing a game of monopoly and preserve the integrity of the game for all players involved.

2. Scope

The Monopoly ATM should store and keep track of all the game's money for the duration of the game, ensuring that it is available to players only when permitted by the rules of the game. Expanding on previous revisions of the scope, the ATM should determine how much money is left at the end of the game, in order to prevent players from keeping money for their next playing. This functionality can be broken down into four main tasks:

- Deposit
- Withdrawal
- Transfer
- Declaring Bankruptcy

2.1 Sub-Tasks

Internally, the ATM must complete a number of sub-tasks to fulfill the main tasks. These sub-tasks are listed below, as well as their required inputs if any, and what physical or digital system they will affect.

Table 2.1: Sub-Tasks and Inputs/Affected Systems

Sub-Task	Input(s)	Affected Systems
Detect player identity	Use the colour sensor to detect the colour of player's card and relate it to their personal account	n/a
Input/Output stack of bills	Receive motor encoder values to calculate distance travelled	Drive motor to push/pull transfer tray in and out of the ATM using a fishing line and pulley system, changed from crank mechanism
Detect bill value	Use the colour sensor to detect the colour of bill and compare to	n/a

	calculated values for each bill value ($1, $5, $10, $20, $50, $100, $500)	
Pick up bill	Receive motor encoder values to calculate distance travelled	Drive motor to actuate end effector down, pressing a suction cup on top bill. Drive motor to pull back on the pneumatic cylinder, a syringe, creating a vacuum and lifting the bill. Drive motor to actuate end effector up
Selecting Bills	Get button presses from user	Update array of bill value counts, new as of this revision due to decision on how to select withdraw values
Move the bill to the bin location	Receive motor encoder values to calculate distance travelled	Drive motor to move the carriage to the location above a designated bin or transfer tray
Drop bill	Receive motor encoder values to calculate the distance	Drive motor to actuate end effector down. Drive motor to slowly release vacuum using the pneumatic cylinder, releasing the bill. Drive motor to actuate the end effector up.
Update account balance	1. Read a value of deposit from detection of the bill value 2. Read a value of withdrawal/transfer from User input	Write an updated account balance to array, changed from earlier revisions where it would write to file due to simplicity of access to data
Reset motor encoders	Receive touch sensor input indicating gantry end mount has been reached	Drive motors to move the carriage to gantry end mount

Once a user has entered their player card and the ATM has identified them, each main task will be initiated by user input, indicating which type of transaction they wish to make. In this revision, if the "Monopoly Man[1]" card is scanned, users will only have the option to transfer money to a player, allowing them to receive funds from the game.

This differs from previous versions of the scope where transfers to the "Monopoly Man" would be internal to deposit and transfer functions, which was changed to simplify the code. Briefly, the main tasks will be completed by the sub-tasks as follows:

2.2 Deposit

1. ATM outputs empty transfer tray, the user inserts a stack of bills
2. ATM intakes transfer tray
3. Repeat until a stack of bills is consumed, i.e., the colour sensor reads the black of the input tray
    a. ATM detects the value of the top bill

[1] In this report, "Monopoly Man" refers to the character in the game representing the banking system

    b. ATM picks up a bill
    c. ATM moves the bill to correct bin for storage
    d. ATM drops bill
    e. ATM returns to the transfer tray
4. ATM updates the user account balance

2.3 Withdrawal

1. The user inputs the desired value of the withdrawal, up to the limit of their account balance
2. Until the desired withdrawal value is reached
    a. ATM moves to the highest possible value bill location
    b. ATM picks up the bill
    c. ATM moves to the transfer tray
    d. ATM drops bill

2.4 Transfer

1. User selects transferee
2. The user enters the value of the transaction, up to the limit of their account balance
3. ATM updates transferer's and transferee's account balances

2.5 Declaring Bankruptcy

1. User confirms that they are declaring bankruptcy
2. ATM prompts cash deposit, updating "Monopoly Man's" account instead of user's
3. Resets player's starting balance to initial value, with the difference being transferred to the "Monopoly Man"s account

In this revision of the scope, all main tasks will be considered complete when the user decides to end their turn on the ATM, implying that they are satisfied with the transactions they have completed and do not need to continue the transaction to correct any errors.

2.5 Startup and Shutdown

In addition to the main tasks, the ATM will have a task for starting the game, allowing users to input the number of players. In this revision of the scope, the "Monopoly Man"s account will have a starting value equal to the total amount of cash in the ATM in order to keep track of cash left outside the ATM in the endgame.

When all but one player have declared bankruptcy, the ATM declare the winner, accept all remaining cash, and display the outstanding cash outside the ATM.

2.6 Evaluating the Success of the Project

Success will be evaluated by the ATM's capacity to fulfill all financial transactions necessary for the game's completion without encumbering the players with tedious delays or frequently miscalculated transactions. In previochurvey, but this was deemed to be too subjective. The success will now be determined by its ability to successfully identify and deliver a bill to the desired location 97.5% of the time, as well as complete all software-based tasks without error.

3. Constraints

Each constraint that was decided upon during the planning stages of the project, as well as new constraints added during the building process, aided in guiding the project's final design.

3.1 Project Constraints

3.1.1 Gantry length must not be less than 560mm

The ATM must have compartments to store and organize the money it will be using. In order to store 7 different values of money, 7 different bins will need to be built to house each bill. Each bill is 50mm in width and will need 5mm clearance on either side of the bill, within the bin, to allow for slight variations in orientation. The bins will also require a “wall” between them of 5mm.

The total width of the bins in addition to the width of the intake tray (100mm) results in a total width of 560mm. This places a constraint on the design of the gantry it cannot be made smaller.[1]

3.1.2 Project timeline must not exceed 4 weeks

The time constraint on the project remained the same from the beginning of the project to the end as the project deadlines were set in stone before any planning began. This constraint helped with time budgeting for the project to ensure that everything was done on time. A Gantt chart was made to help see the project to fruition.

3.1.3 Movement of carriage must not be hindered

As stated in the interim report the success of the project was reliant on the ATM being able to move quickly and unhindered to transport money [1]. This was kept in mind when building the design process and became useful when considering how to wire the Lego EV3 robot to run code for the ATM. If the wires were haphazardly attached then the robot would become tangled when the gantry moved. In order to meet this constraint, cardboard slots were attached at points in the ATM interior to support wired and prevent any tangles.

3.1.4 Colour sensor can be no more than 20 mm from its target

This constraint was added during the testing phase of the project. It became clear that to obtain consistent results form the colour sensor the distance at which it was scanning would also have to remain consistent. Through experimentation, it was found that the best results were obtained at a distance of 20mm from a bill. Thus the constraint was set that a colour sensor could be no more than 20 mm away from its target.

3.1.5 Must not allow access to bills unless lid is opened

This constraint initially stated that no gaps in the enclosure could exceed 6.7cm. The constraint was updated with a new definition: no gap exceeding 6.7cm could allow direct access to funds, i.e. an internal component would be installed inside all gaps to block that path from any funds. This change gave the ATM more leniency in the construction of the enclosure.

3.1.6 Must not allow withdrawals exceeding account balance

In order to prevent cheating, it is evident that players should not be allowed to withdraw amounts exceeding their account balance, as this would allow them to hold more money than allowed by the rules of the game. This constraint remains unchanged from the software presentation.

3.2 Resource Constraints

3.2.1 The project cannot use Lego Mindstorm pieces for some mechanical functionality

The final project design ended up consisting of many 3D printed pieces as particular functionality was required. This restriction remained consistent throughout the project. As the construction of the mechanical design came closer to being completed the lego parts were used when possibly but often there was no alternative to the 3D printing.

4. Criteria

Competing design options were evaluated based on their cost, manufacturing time, and carriage weight. Each criterion received a weight from one to two depending on its significance, where the magnitude of the multiplier correlates to its magnitude of significance. All of the defining factors for each criterion were measured quantitatively, so data could be compared by magnitude.

Table 4.1: Updated Criteria used to Compare Alternate Designs

Criterion	**Weight**	**Critical Factors**	**Reason**
Cost ($)	1	Linear rail, linear bearing, 3D printing, linear fasteners	Project budget must be kept to a minimum due to lack of financial resources. By extension, the cost of the gantry system must be minimized
Manufacturing Time (hr)	2	Total fabrication time of 3D printed parts	Timeline of the project is constrained, so time must be used effectively. If a 3D printed part requires too much manufacturing time, the project may be behind schedule
Carriage Weight (g)	2	Mass of filament, bearings, motor, and fasteners	Weight must be minimized to reduce pressure on gantry motors and increase the speed of the arm

4.1 Criteria Reflection

4.1.1 Cost

During the initial planning stage, the team determined that cost would be the most important criteria, so it received a weight of 2.

As the project development continued, it became clear that functionality had to be prioritised over cost. Throughout the development of the system, additional parts needed to be purchased as needs arose. For example, the team bought foam boards and coloured paper a few days before Demo Day to create an enclosure for the robot in an aesthetic way. This purchase added $20 to the project.

For the completion of the design, 26 3D printed components were required. The cost was not a factor in the decision to 3D print pieces to the extent initially predicted due to access to a personal 3D printer. There were no service charges associated with using the 3D printer, and only filament costs had to be taken into account. As a result, using the printer was affordable.

Overall, cost was irrelevant in the final design choices, so it received an updated criteria weight of 1.

4.1.2 Carriage Weight

In the interim report the carriage weight was deemed significant to the design process due to its effect on the gantry motors, so it received a criteria weight of 1.5. The carriage weight was deemed necessary as a lower carriage weight would result in less pressure on the gantry motor and increase its speed.

Although, as the development of the robot continued, it was found that the weight of the carriage became increasingly important. The carriage weight had a more substantial impact on the robot than expected. Initially, it was stressed that a lower carriage weight reduces pressure on the gantry motors and increases the speed of the carriage.

As the project progressed the reasoning behind the constraint changed to more accurately reflect the design of the gantry. The benefit of lighter carriage would be to reduce drift in the gantry transversal and ensure more accurate encoder values.

It became evident that that motor encoders of the gantry motor are more accurate the less stress is placed on the carriage. Thus, the carriage weight was found to affect the precision of the gantry motor, which proved to be significant as the robot was required to move bills to their proper bin location.

The small change in the motor encoders that the carriage weight made was significant since bills had only 2.5 mm clearance in width and 5 mm clearance in length from the edges of the bins. The change in carriage weight resolved difficulties regarding the accuracy of bill placements.

In retrospect, the carriage weight criteria affected the final product more than initially believed, so it received an updated criteria weight of 2.

4.1.3 Manufacturing Time

When considering design options, the fabrication time of 3D parts was one of the determining criteria in deciding between alternate design options. This criterion encompassed strictly 3D printing time, and not designing time. It received a criteria weight of 1 as fabrication times were thought to have little impact on project time constraints.

After completing the project, it was evident that the manufacturing time was an essential criterion for design since the project consisted of 26 3D printed parts in total. Due to the project time constraints, with enough time to complete the system.

As the parts were designed with minimal support and part material, printing times were minimized. Due to the number of printed pieces, if the fabrication time for each part had not been considered, there would be less time for project testing.

Thus, due to the importance of time in this project, the manufacturing time received an updated criteria weight of 2.

5. Mechanical Design and Implementation

In order to fulfill the functionality described in the project scope, the final design was a cartesian based, pick and place electromechanical system. It primarily consisted of a gantry, carriage, end effector, pneumatic system, and money input and output system. The final design had other secondary components that were required for the housing of bills, overall enclosement of the system, and user interface.

The carriage transversed along the gantry with the end effector mounted on it. With a multitude of actuated movement, the end effector, in conjunction with the pneumatic system and carriage movement, acquired and reallocated bills to and from designated bin locations. One such bin location was the transfer tray - an integral part of the money input and output system. Through the use of the transfer tray, bills were deposited and withdrawn from the ATM, completing the mechanical system.

The final design was built using a combination of Tetrix pieces, Lego Mindstorm pieces and 3D printed parts. Primary consideration was given to 3D printed parts, whenever any custom part was required as a team member had a four 3D printers on standby.

5.1 Gantry

The gantry system consisted of two linear rails, two end mounts, and one end stop. It facilitated controlled, bi-directional, movement of the carriage along its axis.

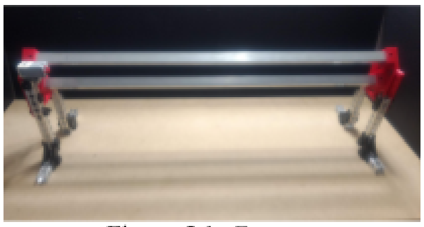


Figure 5.1: Gantry

5.1.1 Linear Rail

Two sections of 20mm by 20mm square aluminum tubing were used as linear rails of the gantry. A constraint was set for the minimal length of the gantry to be 560 mm. As such, the length of these sections were 580mm each.

Alternative options such as linear rods and V-slot aluminum extrusions were also considered. The merits of each option were placed into a decision matrix and evaluated accordingly [1]. Due to the cost benefit and easy availability of square aluminum tubing, it was considered the best option and chosen to be the linear rail.

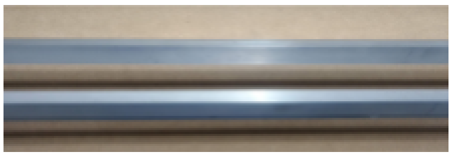


Figure 5.2: Linear Rail of Gantry

The linear rails were vertically aligned and placed with their edges oriented upwards to facilitate the use of the bearing system, described in Section 4.2.3.

These aluminum sections were set 80mm apart in height, from outer edge to outer edge. This was done to accommodate the carriage design specifications, described in Section 4.2.1.

5.1.2 End Mount

The primary functionality of the end mount was to maintain the position of the linear rails, while interfacing with Tetrix bars to create a rigid frame for the gantry. In order to achieve this, the end mounts were 3D printed and designed to accept the linear rails with a transitional fit.

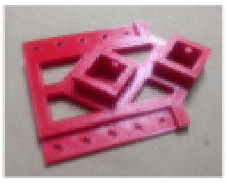


Figure 5.3: Gantry End Mount

8-hole Tetrix bars were used as the stand-offs for the gantry system. As per the functionality of the end mounts, they were also designed to be interfaceable with Tetrix bars to allow for variability in height, as shown in Figure 5.4.

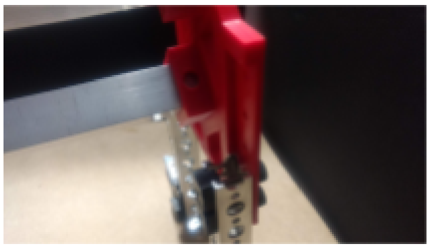


Figure 5.4: End Mount fitting

In the final mechanical design, a gantry height of 140mm was used because through testing, this height allowed the end effector to acquire and dispatch bills without any interference from the money bins.

5.1.3 End Stop System

A mechanical end stop system was used for homing the carriage after every movement along the linear rail.

A touch sensor was used, on one end of the gantry, as a switch for the end stop. This sensor was attached to the stand-offs using a designed 3D printed mount to interface the Tetrix and the Lego part, as shown in Figure 5.5.

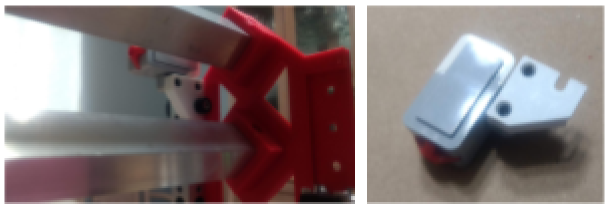


Figure 5.5: End Stop Mounted on Gantry (left) and Touch Sensor Mount (right)

No alternative designs were considered due to the time constraints of the project, and the simplicity of the designed mount.

5.2 Carriage

The carriage consists of one front plate, one back plate, one Large EV3 motor, one medium Lego EV3 motor, and 4 sets of the bearing system. It also houses the end effector which acquires and dispatches bills to and from designated destinations.

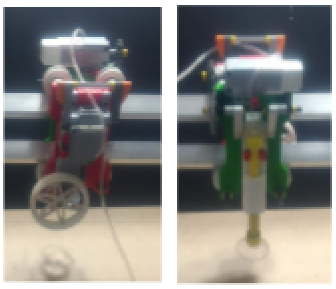


Figure 5.6: Carriage Back View (left) and Carriage Fight View (right)

5.2.1 Front & Back Plates

These 3D printed plates were the main housing components of the carriage. They both have two fixed top holes and two slotted bottom holes to attach to each other using four M6X60mm socket head bolts. The bottom two holes were slotted to allow tightness adjustability between the bearing system and the linear rail.

The back plate was designed and sized to accept a Large EV3 Motor, using standard Lego connectors, as shown in Figure 5.7.

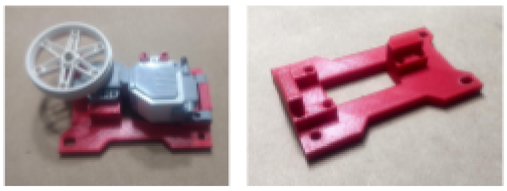


Figure 5.7: Back Plate with Motor (left) and Back Plate (right)

This resulted in the back plate dimensions being 170mm by 72mm. The sizing of the back plate dominated the dimensioning of the whole carriage, as well as the spacing of the linear rails; all other parts were dimensioned with relation to the sizing of the backplate to maintain design consistency.

The front plate was designed to accept a Medium EV3 Motor, using standard Lego connectors, as shown in Figure 5.8. It was also designed to facilitate the cranking mechanism to actuate the end effector.

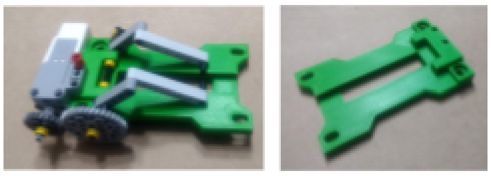


Figure 5.8: Front Plate with Motor (left) and Front Plate (right)

In order to maintain the strict vertical sliding motion of the end effector, a dovetail groove was integrated into the front plate, as shown in Figure 5.9.

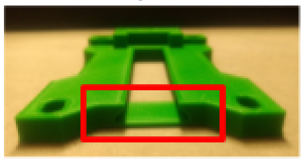


Figure 5.9: Dovetail Groove on Front Plate

5.2.2 Bearing System

The four sets of the bearing system allowed for the carriage to freely move along the linear rails. Each set consisted of one 626ZZ radial bearing, one 3D printed v-groove wheel, two 3D printed spacers. Each v-groove wheel had one radial bearing at its centre to facilitate carriage motion, as shown in Figure 5.10.

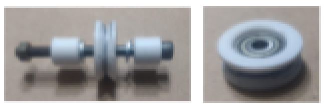


Figure 5.10: Exploding Bearing View (left) and V-Groove Wheel (right)

The front and back plates were connected via four M6X60mm socket head bolts, with one v-groove wheel placed between the plates on every bolt. 3D printed spacers are used to centre the wheels between the two plates. This configuration, along with the design of the v-groove wheels, constrained the carriage in 5 directions of motion; allowing only horizontal movement.

Figure 5.11: Carriage on Gantry

A tradeoff in wheel quality was made by 3D printing them instead of buying very expensive, commercially available, counterparts. This tradeoff did not undermine the required functionality of the bearing system as it did not require a great amount of precision. Through testing, it was found that the precision provided by the 3D printed wheel was sufficient for the project.

5.2.3 Gantry Drive System

The gantry drive system moved the carriage along the linear rails. It consisted of one Large EV3 motor, one Lego wheel with grooves, and fishing line. Taking design inspiration from Fusion F400 3D Printer [2], a fishing line drive mechanism was implemented.

The motor was mounted to the back plate of the carriage and the wheel was mounted to the motor. The fishing line was fixed at either end of the gantry, as shown in Figure 5.12.

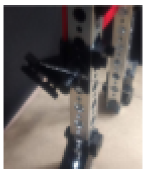


Figure 5.12: Standoff Fishing Wire Mount

In order to convert radial motion of the motor to linear motion of the carriage, the fishing line was looped in the groove of the wheel. A double loop was used at the wheel to minimize the risk of belt slippage.

Figure 5.13: Gantry drive system wheel

Using a timing belt and pulley was considered as an alternative design option. This would have been more precise in movement as the belting system would have prevented any drift during carriage motion. However, it was rejected due to the complicated interfacing required between EV3 motor and belt. Due to the timeline constraint, designing these parts was not feasible.

A tradeoff in movement speed was made due to the final drive system design. The carriage could not operate at very high speeds because, through testing, it was found that high-speed movement caused drift. Any drift in the carriage movement made the stop location of the carriage unpredictable.

<u>5.2.4 End Effector Actuation Mechanism</u>

A crank mechanism was used to actuate the end effector into and out of designated bin locations. The crank was driven by the Medium EV3 motor, mounted on the front plate of the carriage, and a system of gears. The gears were primarily used to create space between the crank mechanism and the linear rail, rather than any mechanical advantage; calculation of gear ratios were thus irrelevant. This is shown in Figure 5.14.

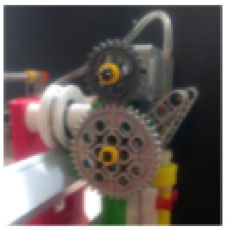


Figure 5.14: End effector actuation mechanism gears

The crank was double-sided to prevent any rotational moment during actuation. To maximize the range of motion within the space of the front plate, 5-hole Lego pieces were used as the crank arms and 7-hole Lego pieces were used as the linkage arms. (See Appendix 1.1 for Calculation)

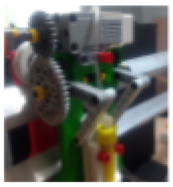


Figure 5.15: End Effector crank mechanism

An alternate design option of using a parallelogram linkage system was also considered because, through trial, it was found that this actuation system would have been mechanically easy to implement. However, it was found that the parallelogram linkage system had circular movement as it actuated. Inferring that this would interfere with acquiring and dispatching bills into bins, the crank was chosen as the best option.

5.3 End Effector

The end effector was an integral part of acquiring and dispatching money within the entire system. It consisted of a plunger, plunger guide, spring, 35mm diameter suction cup, and two end caps.

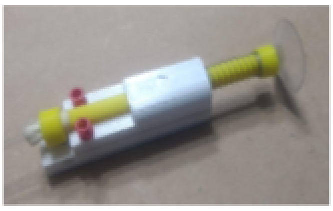


Figure 5.16: End Effector Assembly

The plunger was a 3D printed shaft with a 4mm diameter hole for the vacuum line to pass through, connecting the pneumatic system to the suction cup. This part slid within the plunger guide - a 3D printed holder with a 12mm diameter slotted-hole to facilitate the strict vertical sliding motion of the plunger. The plunger guide required to be a length of 72mm as the dovetail groove implemented into the front was 72mm in length. The plunger required to be a length of 116mm. (See Appendix 1.2 for Calculations)

The guide was connected to the crank mechanism of the carriage and was the only part of the end effector that was actuated vertically. It also had dovetailed grooves that maintained the strict vertical sliding motion along the front plate of the carriage, as shown in Figure 5.17.

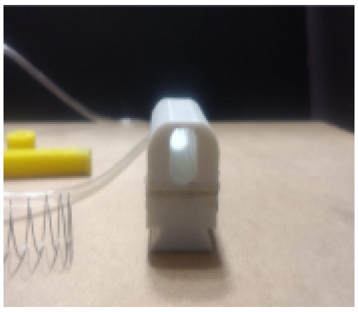


Figure 5.17: Plunger Guide
Dovetail Groove

Once the plunger was placed in the guide, along with a suspension spring, two end caps were used as stopping points for the plunger movement via cam lock fastening.

The suction cup that was used was commercially available. It was attached at the bottom end cap of the plunger. The end caps were designed to house the suction cup. As the plunger lowered, the suspension spring created a seal between the suction cup, at the end of the plunger, and the bill being picked up, as shown in Figure 5.18.

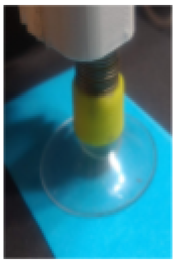


Figure 5.18: Suction
Cup Seal

While designing the end effector, a point of concern was whether or not the bill would bend due to the vacuum suction, causing it to release from the single suction cup. As such, an alternate design option that was considered included having two 20mm diameter suction cups since this would provide more even suction along the length of the bill during pickup and limit bill dropping.

This design was not implemented because, through testing, it was found that the bills did not bend under the vacuum suction of a single suction cup. Since a single suction cup was much simpler to integrate into the plunger and had the same functionality as the alternate design option, the single suction cup was implemented.

5.4 Pneumatic System

The pneumatic system consisted of one pneumatic cylinder, and one Large EV3 motor, housed in a frame made of Lego. It facilitated the acquiring and dispatching of bills by creating vacuum suction at the end effector.

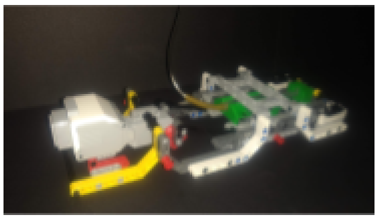


Figure 5.19: Pneumatic System
Assembly

5.4.1 Pneumatic Cylinder

A syringe was used as the pneumatic cylinder to create vacuum suction at the end effector. Through testing, it was found that 2 cm^3 of vacuum was required to properly lift a bill. As such, a 3 cm^3 syringe was utilized.

A commercially available pneumatic cylinder was also considered as it would have provided smoother actuation. However, due to its large size, complicated interfacing design requirement, and its high cost, the syringe was considered the best option..

Since there were no options available to attach the syringe to Lego pieces, a 3D printed holder was designed to interface the syringe and the pneumatic system housing, as shown in Figure 5.20.

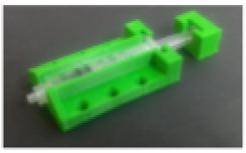


Figure 5.20: Pneumatic Cylinder Holder

5.4.2 Pneumatic System Housing

The frame of the pneumatic system was made out of Lego. The design was inspired by a Lego Mindstorm design available online [3]. Due to the time constraint, for a relatively large component, such as the housing, the best use of resources provided by the teaching team was considered foremost as it would limit designing and manufacturing time.

Figure 5.21: Pneumatic System Housing (right) and Slide (left)

The frame housed the Large EV3 motor, and also had a sliding guide built into it. A tray, made of appropriate Lego pieces, was built to slide along the guide. The syringe was mounted on the tray using the 3D printed holder. This is shown in Figure 5.23.

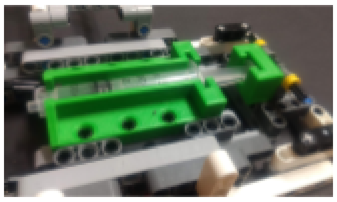


Figure 5.23: Holder Mounted on Side

Due to the simplicity of the Lego structure, no alternative options were considered, in spite of minute roughness in linear motion.

5.4.3 Pneumatic Actuation Mechanism

A crank mechanism was used to actuate the tray, with the mounted syringe, along with the sliding guide. This mechanism was used because it provided strict linear motion, which was required to actuate the pneumatic cylinder.. The crank was driven by the Large EV3 motor, housed in the frame.

Figure 5.24: Pneumatic Actuation System

The crank was double-sided to prevent any rotational moment during actuation. Two 5-hole Lego pieces were used as the crank arms and two 5mm Lego shafts were used as the linkage arms. No calculations were required as the range of motion was controlled through code - optimal actuation distances were found through testing.

5.5 Money Input & Output System

The money input and output system consisted of two linear rails, one transfer tray, two end mounts, one Medium EV3 motor, and one colour sensor, mounted on the linear rails. Other than the EV3 Brick, this system was the primary mechanical interface with the user as it was used to complete deposits and withdrawals.

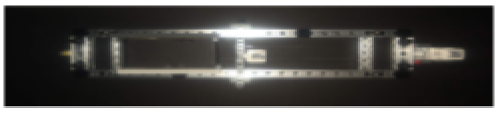


Figure 5.25: Money Input Output

5.5.1 Transfer Tray

The transfer tray was the sliding component of this system that held bills during the transfer and withdrawal functions of the ATM. The inner dimensions of the tray were made 50mm by by 100mm to house bills, as defined in Section 4.6.1. This bin was made out of foam board, as this material was aesthetic, and through testing, it was found that it provided structural rigidity to the tray.

A frame, made of Lego, was made to hold the bin during the movement of the tray along the linear rails.

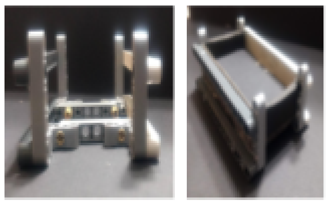


Figure 5.26: Transfer Tray

5.5.2 Linear Rail

The linear rails required to be a length of at least 285mm because through testing, it was found that the transfer tray needed a 285mm range of motion to be able to complete its functions.

Each of the two linear rails consisted of one 15-hole bar taped together with one 6-hole bar. This configuration provided a structurally sound linear rail of length 350mm, with 65mm of space on either end for connections to the stand-offs and to each other. The stand-offs, also made out of Tetrix, were 36mm off the base to provide space for the drive system of this component as described in Section 4.5.5.

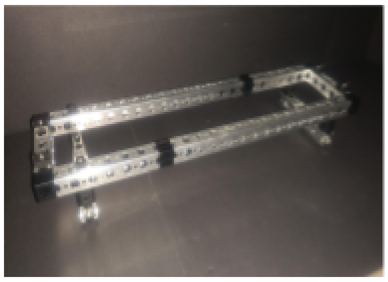


Figure 5.27: Tetrix Linear Rail System

The Tetrix bars were taped together because, through testing, it was found that using standard Tetrix connecting components from the kit would have blocked the movement of the tray sliding on the rails.

Sections of 20mm by 20mm square aluminum tubing were considered as an alternative option since this would result in a single-piece linear rail; it would be more structurally rigid and allow for smoother travel of the transfer tray. However, for the purposes of the money input and output system, smooth travel of the transfer tray was irrelevant. Utilizing the square aluminum tubing would have required specifically designed mounts and hardware. As such, Tetrix rails were considered the better option since all mountings and hardware were available in the kit.

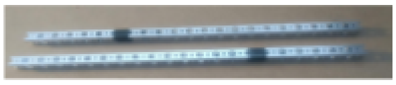


Figure 5.28: Tetrix Rails

The linear rails were horizontally aligned to facilitate sliding of the transfer tray. These rails were also set 75mm apart, in width, due to tray design specifications described in Section 4.5.1.

Due to the friction of sliding on this designs linear rails, a tradeoff in structural rigidity of the tray was made. To add more structural rigidity meant to increase the tray's weight. The tray needed to be light enough so that the friction, caused by the tape on the linear rail, did not hinder its movement. The maximum allowable weight of the tray was not tested due to project time constraints. Through testing, however, it was found that keeping the tray as light as possible was the best solution.

5.5.3 Colour Sensor Mount & Colour Sensor

A colour sensor was used to identify bill types during the deposit function of the ATM. A constraint was set for the maximum height of the colour sensor to be 20mm off what it was scanning. As such, the colour sensor was positioned 20mm from the transfer tray..

3D printed mounts were used to interface the sensor, along with other Lego pieces, to the Tetrix rails, as there was no other way to interface the two resources.

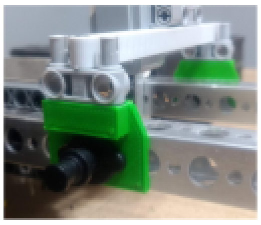


Figure 5.30: Colour Sensor Mount

The colour sensor mount was positioned 145mm away from the gantry to prohibit interference between the colour sensor and any movement of bills, into and out of the tray.

5.5.4 End Mount

The end mounts were used in conjunction to enable the transfer tray drive system. 3D printed mounts were used to interface the the Medium EV3 motor, along with other Lego pieces, with the Tetrix rails, as there was no other way to interface the two resources.

One end mount was designed to accept the Medium EV3 motor using standard Lego connectors, as shown in Figure 5.31.

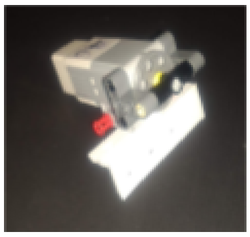


Figure 5.31: Motor End Mount

The other end mount was designed to accept a Lego pulley, attached using appropriate Lego pieces, as shown in Figure 5.32.

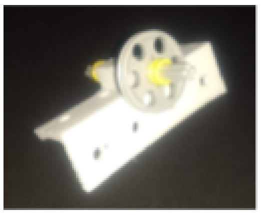


Figure 5.32: Pulley End Mount

5.5.5 Transfer Tray Drive System

The transfer tray drive system moved the tray along the Tetrix rails. It consisted of one Medium EV3 motor, two Lego pulleys, with grooves, and fishing line.

In order to convert radial motion of the motor to linear motion of the tray, the fishing line was looped around the pulleys at each end mount and secured onto the tray to maintain tension, as shown in Figure 5.33.

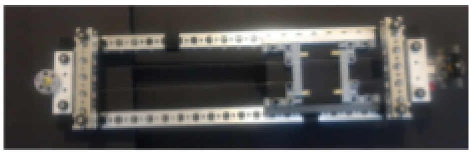


Figure 5.33: Transfer Tray Drive System Bottom View

The motor was not attached to the tray in order to minimize the weight of the tray. The tray needed to be light enough so that the friction, caused by the tape on the linear rail, did not hinder its movement. The maximum allowable weight of the tray was not tested due to project time constraints. Through testing, however, it was found that keeping the tray as light as possible was the best solution.

Appropriate Lego pieces were added to maintain the straightness of the pulley, located on the motor, as shown in Figure 5.34. The tension of the fishing line, when the system was put together, often removed the pulley from the motor. The additional support prevented this.

Figure 5.34: Motor End Mount Support

5.6 Miscellaneous Components

5.6.1 Money Bins

Seven money bins housed 25 bills of each bill type within the ATM. Due to the 50mm by 90mm Monopoly money dimensions, the inner dimensions of the bins were made 55mm by 100mm. This 2.5 mm clearance in width and 5 mm clearance in length allowed for some slight variations in money placement orientation.

Through trial, it was found that 25 bills were 15 mm in height. To accommodate this height of bills, the walls of the bins were made 20 mm in height.

These bins were constructed of foam board as this material was aesthetic and structurally rigid.

The seven bins were divided into two sets; one set of 4 bins and one set of 3 bins. This allowed the money input and output system to be placed in the centre of the ATM, shortening the time it took to deposit or collect bills to and from the tray. This is shown in Figure 5.35.

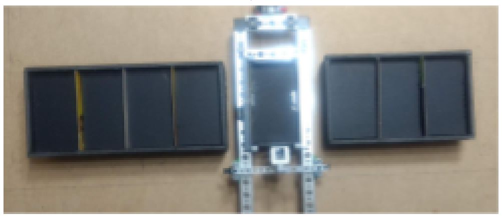


Figure 5.35: Bin Configuration

The bases of all bins were made horizontally aligned so that the same function could be used to code the pickup of bills from bins. The actual height of the bins from the base was not important as the gantry height was made variable, described in Section 4.1.1. This meant that the end effector could reach the bottom of the bins at its extended position while clearing the bins at its retracted position.

5.6.2 Enclosure

A constraint was set for the ATM to be fully enclosed. As such, an enclosure was required around the entire system. The enclosure consisted of six dimensioned pieces of foam board, glued together. A lid was also made to satisfy the same constraint, as shown in Figure 5.36.

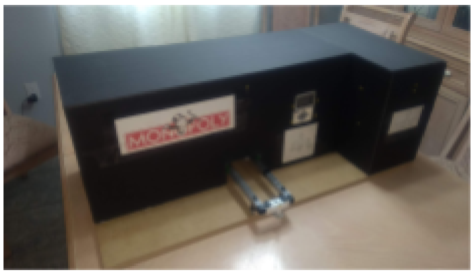


Figure 5.36: Fully Enclosed ATM

The enclosure required to be higher than 28mm - the highest point of the ATM. As such, the height of the enclosure was set at 30mm.

The front piece of the enclosure required a 110mm by 75mm slot removed so that the piece could fit over the money input and output system. This system needed to be partially outside the enclosure to allow users to deposit money into the transfer tray. This is shown in Figure 5.37.

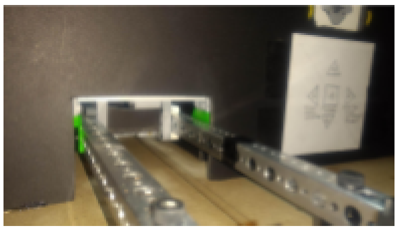


Figure 5.37: Transfer Tray Slot

5.6.3 EV3 Brick Slot & Mount

The ATM enclosure required a slot for the users to be able to access all buttons and display screen on the EV3 Brick; this was the main interface of the entire system. As such, a 65mm by 105mm slot was cut on the enclosure. Through group survey, the location of the slot was selected, as highlighted in Figure 5.38.

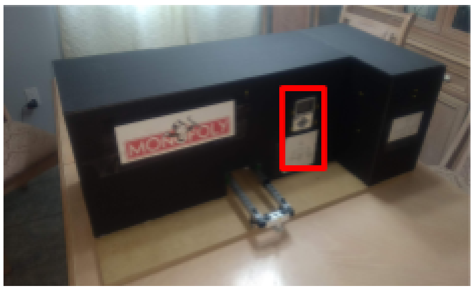


Figure 5.38: EV3 Brick Location

The EV3 Brick was mounted to the enclosure through the use of zip ties since zip ties were cheap and readily available. This is shown in Figure 5.39.

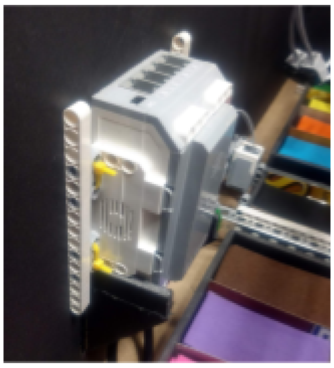


Figure 5.39: EV3 Mount

5.6.4 Player Colour Sensor

The ATM enclosure required a slot for the players to insert their player card and have the system recognize them, through the use of a colour sensor. This required a 80mm wide slot on the enclosure as the player cards were 70mm in width.

A support plate, made from foam board, was added on the inside of the enclosure so that players do not accidentally drop their player cards into the ATM.

Through group survey, the location of the slot was selected, highlighted in Figure 5.40.

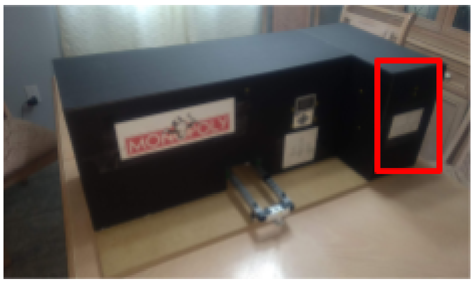


Figure 5.40: Colour Sensor Slot Location

A constraint was set for the maximum height of the colour sensor to be 20mm off what it was scanning. As such, the colour sensor was mounted, using zip ties, 10mm from the support plate.

5.7 Overall Assembly

The layout of the system's assembly is shown in Figure 5.41.

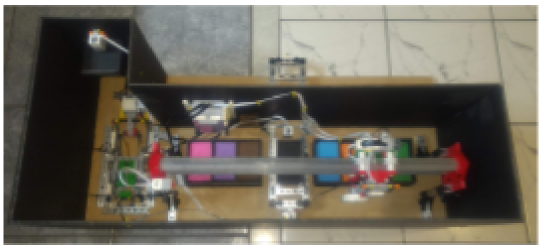


Figure 5.41: Overall ATM Layout

The entire mechanical system was mounted on a base (400mm by 800mm MDF board) using #8 - 30mm drywall screws. All designed parts, except for the pneumatic system, had Tetrix bases that accepted these screws without any physical damage to the part. This is highlighted in Figure 5.42

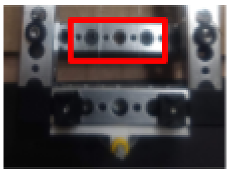


Figure 5.42: Transfer Tray Mounting

To mount the pneumatic system, Tetrix pieces were used to surround it and restrict all motion; essentially securing the system onto the base. For additional security, zip ties were added to the pneumatic system and surrounding Tetrix bars, as shown in Figure 5.43.

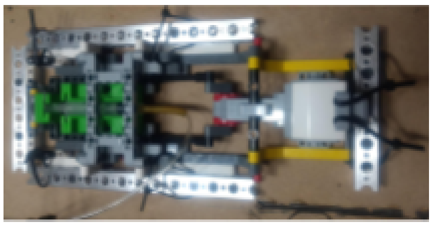


Figure 5.43: Pneumatic System Housing

Foam board was an alternative option considered for the base material. While foam board is cheap and stacking enough layers would provide similar structural strength to the MDF board, the components of the mechanical system were unable to be mounted to the foam board. As such the MDF board was used.

6. Software

6.1 Overall Program Design

The Monopoly banking system was expected to be a large code file due to the range of available functions offered. Every user action had to be checked to ensure they weren't breaking any rules. As well, a friendly user interface had to be implemented for ease of use.

It was decided that the program would be function based due to having such a broad base of tasks to complete, and the RobotC programming language does not support class data types. The software was coded in a hierarchy of functions, where each main task had its own master function, which called a series of smaller functions to complete subtasks in a task.

The program is coded in a hierarchy style: each function calls smaller functions until reaching a trivial function, where a calculation is performed, input from the user is converted to data, or the robot makes a mechanical movement.

This programming approach was preferred over other styles as it makes the code easier to debug; errors will only require a change in one spot of code rather than multiple spots because every main task shares similar functions.

One tradeoff to this programming approach is that it makes the code difficult to read since there is a large number of lines and some functions end up being far away from where they are called. This problem was mediated by labelling function prototypes with the line that their function body is found in the file.

6.2 Program Breakdown

The program was coded in a function hierarchy style to simplify debugging and make the code more efficient. Every function calls another function until reaching trivial functions, calculations, user input processing, or updating data.

The subtasks were chosen so that they would simplify code debugging as each function essentially has one purpose, so when something went wrong the program, it was clear which subtask had failed. As well, when there were errors in the code, minimal changes in the code were required since in most cases a single subtask would be responsible for the error, so only a small portion of code was required to be changed to alter the functionality of the entire program.

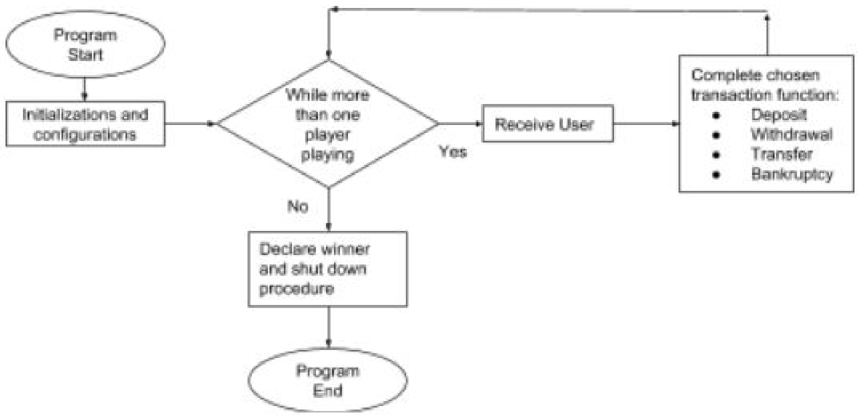

Program Start
Initializations and configurations
While more than one player playing
Yes
No
Receive User
Complete chosen transaction function:
• Deposit
• Withdrawal
• Transfer
• Bankruptcy
Declare winner and shut down procedure
Program End

Figure 6.2: High Level Flowcharts of Main Functions

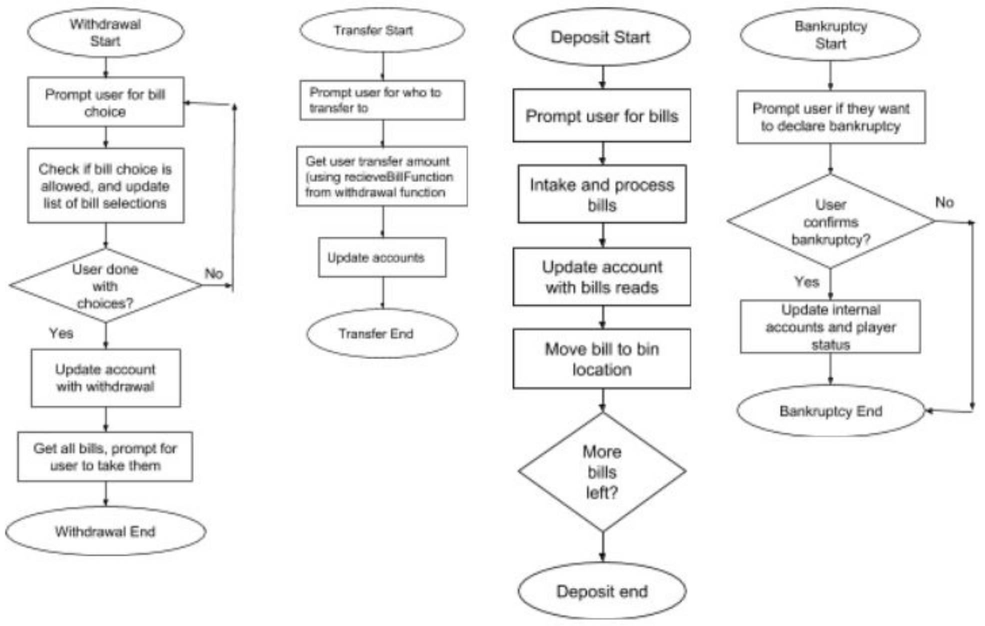

Withdrawal Start
Prompt user for bill choice
Check if bill choice is allowed, and update list of bill selections
User done with choices?
No
Yes
Update account with withdrawal
Get all bills, prompt for user to take them
Withdrawal End
Transfer Start
Prompt user for who to transfer to
Get user transfer amount (using recieveBillFunction from withdrawal function
Update accounts
Transfer End
Deposit Start
Prompt user for bills
Intake and process bills
Update account with bills reads
Move bill to bin location
More bills left?
Deposit end
Bankruptcy Start
Prompt user if they want to declare bankruptcy
User confirms bankruptcy?
No
Yes
Update internal accounts and player status
Bankruptcy End

Figure 6.3: Flowcharts of Main Functions

6.3 Function Tables

Table 6.1: Startup and Main

Function, Author(s)	**Parameters**	**What it does**
task main, Leif	-	See Fig. 6.3
void sensorConfig, Leif	-	Sets up sensors with appropriate mode and port
void setupPlayers, Leif	-int &numPlayers -int *accountBalance -bool *isPlaying	Prompts users to select number of players, updates numPlayers, updates elements in accountBalance at player indices to starting balance, and updates player indices in isPlaying array to true

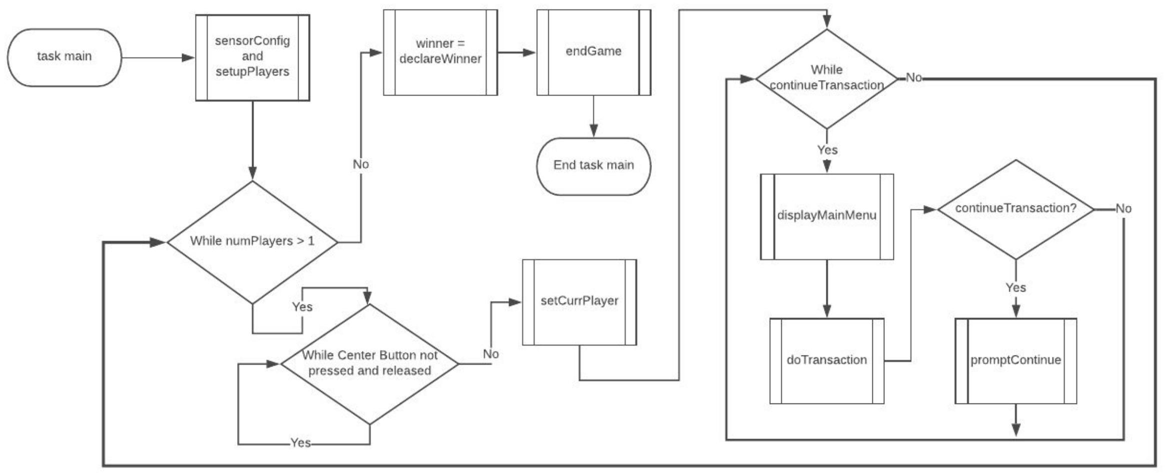


Figure 6.3: Task Main Flowchart

Table 6.2: Player Turn

Function, Author(s)	**Parameters**	**What it does**
void setCurrPlayer, Leif	-int &currPlayer -bool *isPlaying	While no valid player card has been read, calls senseCard function to get a player card value and checks it against the player index in isPlaying to determine if the player card is valid. Updates currPlayer to the scanned player card
void displayMainMenu, Leif	-int currPlayer -int *accountBalance	Displays options for transactions. If currPlayer is the Monopoly Man (0), only displays tranfer and cancel transaction options
void doTransaction, Leif	-int currPlayer -int &numPlayers -bool *isPlaying -int *accountBalance -bool &continueTransaction	Waits for user input to select transaction option, then calls transaction function. If currPlayer is the Monopoly Man (0), only allows for transfer option and cancelling transactions
void promptContinue, Leif	-bool &continueTransaction	Prompts user if they wish to continue doing transactions, waits for user input, and updates continueTransaction accordingly
void promptCancel, Leif	-bool &continueTransaction	Prompts user to confirm cancelling their transactions, waits for user input, and updates continueTransaction accordingly

Table 6.3: General User Interface

Function, Author(s)	**Parameters**	**What it does**
void displayText_NoWait, Wais	-string *text	Erases display, then displays text
void displayText_Wait, Wais	-string *text	Erases display, displays text, then waits for 1 second

Table 6.4: Withdrawal

Function, Author(s)	**Parameters**	**What it does**
void withdraw, Wais	-int CurrPlayer -int *accountBalance	Master Withdraw function, calls all subtasks in withdraw function
int receiveWithdrawBills, Wais	-int &playerBalance -int *transactionBills -bool &isCancelled	Receives user's withdraw bill choices, updates array of bill choices. Calls getLowerOptions.
void getLowerOptions, Wais	-int playerBalance -int *transactionBills -bool &isCancelled	Prompts user for lower value bills for selection in withdraw. Allows user to proceed to higher values by calling getHigherOptions. Refer to Fig. 6.4

void getHigherOptions, Wais	-int playerBalance -int *transactionBills -bool &isCancelled	Prompts users for higher value bills. Allows user to confirm withdraw. Similar to getLowerOptions, except returns state of transaction as boolean
bool isClearorCancel, Wais	-	Checks whether user presses button so that they want to cancel or clear transaction while selecting bills
void clearChosenBills, Wais	-int *transactionBills	Clears arrays of chosen bills for transaction
void cancelTransaction, Wais	-int *transactionBills -bool &isCancelled	Cancels withdraw/transfer function; clears array of chosen bills and sets transaction state to false
bool isValidTransaction, Wais	-int playerBalance -int totalTransaction -int bill	Checks whether a chosen bill for withdraw is valid, denies addition to chose bills array if invalid
int calcTransactionAmount, Wais	-int *transactionBills	Calculates total amount of transaction (withdraw or transfer)
int calcRemainingCash, Wais	-int transactionAmount -int balance	Calculates user's remaining cash during a withdraw/transfer function
void completeWithdrawal, Wais	-int currentPlayer -int *accountBalance -int withdraw -int *transactionBills	Finishes withdraw function by updating user account and moving bills
void displayLowerOptions, Wais	-int *transactionBills -int playerBalance	Displays lower value bill options
void displayHigherOptions, Wais	-int *transactionBills -int playerBalance	Displays higher value bill options

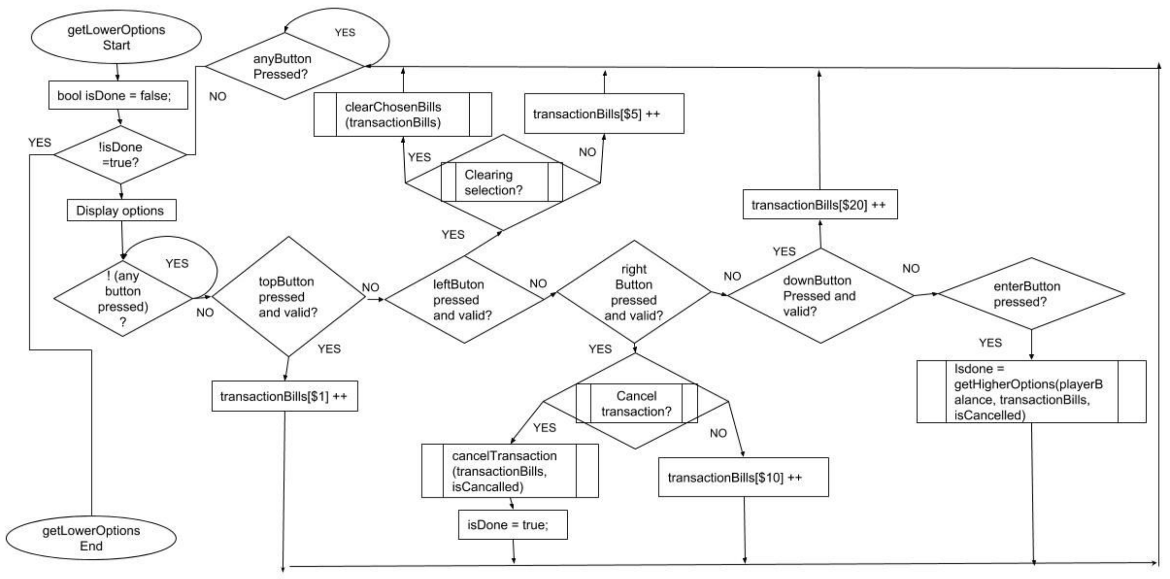


Figure 6.4: getLowerOptions Flowchart

Table 6.5: Deposit

Function, Author(s)	**Parameters**	**What it does**
void deposit, Josh	-int currPlayer -int *accountBalance -bool isPlayerDone	Prompts user to place bills on tray, calls processDeposit() when user clicks button to proceed
void processDeposit, Josh	-int *transactionBills	While the colour sensor sees a bill on the tray, it reads it, updates user's account, and places bill in corresponding bin location

Table 6.6: Transfer

Function, Author(s)	**Parameters**	**What it does**
void transfer, Soyazhe	-int transferor -int playersInGame -int *playerBalance -bool *isPlaying	Creates array transferOption to store the players associated with each button. Calls displayTransferOptions, then sets transferee to return value of getTransferee. If transferee is not -1, calls transferAmount. Else, or if transaction in cancelled, cancels transfer.
int getTransferee, Soyazhe	-int playersInGame -int *transferOption	Calls waitButtonPress with number of *playersInGame*, then determines which player has been chosen from *transferOption* and returns

		the player index.
void transferAmount, Soyazhe	-int transferor -int transferee -int *accountBalance -bool &isTransferCancelled	Prompts user to pick bills, then calls getTranserAmount and updates transferor and transferee balances in accountBalance accordingly
int getTransferAmount, Soyazhe	-int playerBalance -bool &isTransferCancelled	Calls getLowerOptions to allow user to input bill values, returns total amount calculated by calcTransactionAmount
void waitButtonPress, Soyazhe	-int playersInGame	See Fig. 6.5
void buttonPressValid, Soyazhe	-int playersInGame	See Fig. 6.5
void displayTransferOptions, Soyazhe	-int transferor -int *transferOption -bool *isPlaying	See Fig. 6.6

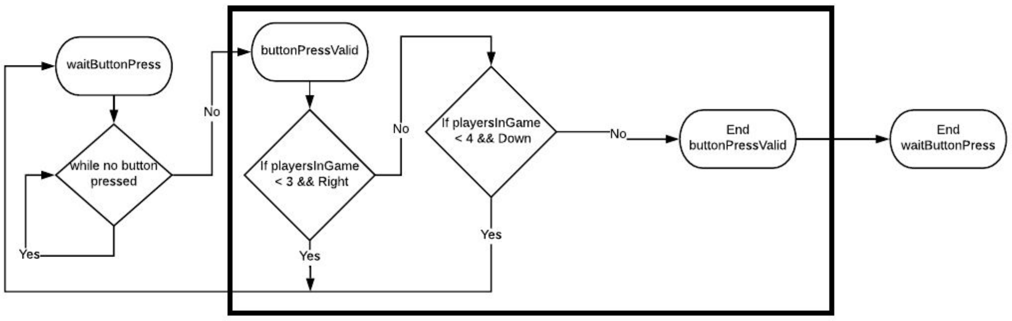


Figure 6.5: waitButtonPress and buttonPressValid Flowchart

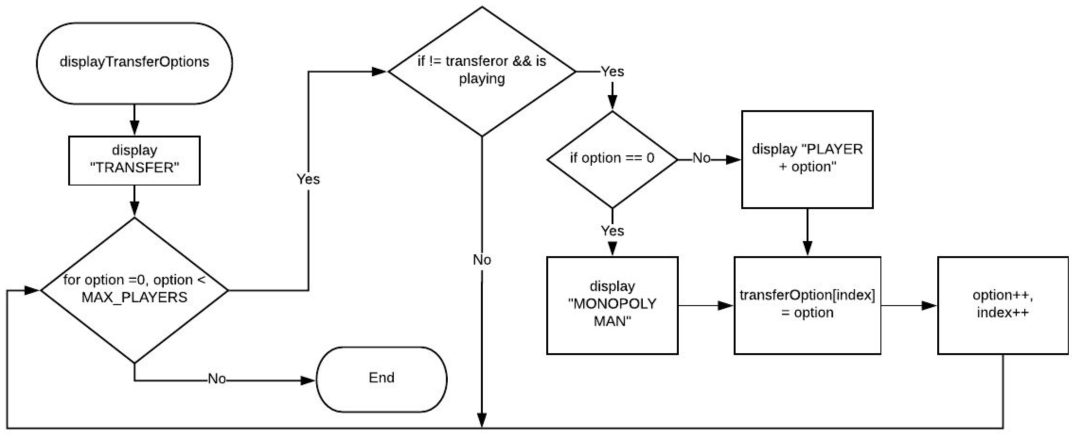

displayTransferOptions
display "TRANSFER"
for option =0, option < MAX_PLAYERS
Yes
No
End
if != transferor && is playing
Yes
No
if option == 0
Yes
No
display "PLAYER + option"
display "MONOPOLY MAN"
transferOption[index] = option
option++, index++

Figure 6.6: displayTransferOptions Flowchart

Table 6.7: Shutdown

Function, Author(s)	Parameters	What it does

void declareBankruptcy, Leif	-int currPlayer -int &numPlayers -bool *isPlaying -int *accountBalance -bool &continueTransaction	Prompts user to confirm delcaring bankruptcy, calls resetPlayerBalance and deposit (with isPlayerDone set to true), sets currPlayer index in isPlaying to false, decrements numPlayers, and sets continueTransaction to false
void resetPlayerBalance, Leif	-int currPlayer -int *playerBalances	Adds playerBalances[currPlayer] minus the starting balance to the Monopoly Man's account balance, then sets currPlayer's balance to starting balance
int declareWinner, Josh	-bool *isPlaying	Searches through isPlaying array, where winning player equals true. Displays the winning player, then returns the winning player index
void endGame, Josh	-int currPlayer -int *playerBalances	Calls resetPlayerBalance on winning player, calls deposit (with isPlayerDone set to true), displays amount cash outside of the ATM then waits 5s

Table 6.8: Bill Movement

Function, Author(s)	**Parameters**	**What it does**
void moveBillsOut, Josh, Wais	-int *transactionBills	For the quantity of bills at each bill index in transactionBills, calls masterTransverse on that bill and the output tray location
void masterTransverse, Josh	-int initial -int final	Calls gantryTransverse on the intial bin location, call pickUpBill, calls gantryTransverse on the final bin location, then calls dropBill
void pickUpBill, Josh	-	Zeroes the end effector actuator motor encoder, calls moveSelectMotor to actuate the vertical actuator down, then to engage suction with the end effector motor, then to actuate the vertical actuator up
void GantryTransverse, Josh	-int position	Calls zeroGantry, zeroes the gantry motor encoder, then calls moveSelectMotor with the gantry motor and a specified distance based on the bin position
void dropBill, Josh	-	Calls moveSelectMotor to actuate the end effector down, then release suction, then to actuate the end effector up, then to set the pneumatic crank motor to its neutral suction position
void zeroGantry, Josh	-	Engages the gantry motors in reverse until the end stop sensor is hit, then sets the gantry motor to 0

void sendTray, Josh	-int trayLocation	If the transfer tray location is 0, sends the transfer tray to the colour sensor position by calling moveSelectMotor, else moves the transfer tray to the user position
void conveyorReturn, Josh	-	Calls moveSelectMotor with the conveyor motor to move the transfer tray from the user positon to the start position
void moveSelectMotor, Wais	-int motorPort -int power -float encoderDistMult -float encoderDistLimit -int waitTime -int direction	Engage motorPort at power. If direction is negative (1), loop while the encoder value multiplied by encoderDistMult is greater than encoderDistLimit. Else, loop while less than. Set motorPort to 0, then wait for waitTime.

Table 6.9: Colour Sensing

Function	**Parameters**	**What it does**
int senseBill, Josh, Leif, Wais	-	Takes 30 colour readings in loop and stores count of bill values in array. Then, loops through array to determine the modal bill value, and returns that bill's index. Returns -1 if no colours were sensed
int senseCard, Josh, Leif, Wais	-	Takes 10 colour readings in loop and stores count of player cards in array. Then, loops through array to determine the modal player card, and returns that bill's index. Returns -1 if no colours were sensed

6.3 Function Testing

Given the hierarchical nature of the program, only the key functions of the program needed to be tested. This is because each of the main functions called many smaller functions, so if the main function produced the desired result, then all of the smaller functions composing that main function must have performed successfully as well. The sum of this testing forms the entirety of the game's operations, so the success of these functions dictate the overall software performance.

Table 6.10: Function Tests

Function	**Test**	**Expected Result**	**Achieved Expected Result (yes/no)**
setupPlayers	Select 2, 3, and 4 player mode	Confirmation message	yes
getCurrPlayer	Scan player card 4 in 3 player mode	Continues to prompt for player card	yes
getCurrPlayer	Scan player card 0,1,2,3,4 in 4 player	Display main menu and correct player with account balance	yes

	mode		
displayMainMenu	Scan player card 0	Main menu only allows for transfer and cancel	yes
withdraw	Select one of each bill	ATM outputs one of each bill in cash, account balance is reduced by $686	yes
withdraw	Select bills exceeding account balance	Stops accepting input once account balance is reached	yes
clearOrCancel	Hold left button and release, hold right button and release	Bills entered are cleared, then transaction is cancelled and user is prompted to continue turn	yes
deposit	Put one of each bill in tray and deposit	Bills are sorted to correct location, account balance increases by $686	yes
transfer	Select transfer in 2, 3, and 4 player modes	Transferee options are restricted to players in game	yes
transfer	Select one of each bill value to transfer	Both transferor and transferee account balances are updated by ±$686	yes
declareBankruptcy	Confirm declaring bankruptcy	ATM requests deposit of remaining funds, player card no longer scans	yes
delcareWinner	Declare bankruptcy on all but one player	Correct winning player is displayed, ATM requests deposit of remaining cash	yes
endGame	Keep one bill of each value after endgame deposit	ATM displays that $686 are missing from the ATM	no

6.4 Task List

Start-Up

- Prompts for a number of players
- Initialize players with starting cash balances
- Initializes monopoly man with starting bank money

Sub-tasks necessary for all functions

- Identifying a user card
- Identifying bill colours
- Picking up a bill
- Transporting a bill
- Depositing money into a users account

Withdrawal

- Prompts user for bills they want to withdraw
    - Checks that bill selections are valid before updating
- Updates user's account with withdrawal
- Commands mechanical system to grab selected bills and dispense to user

Deposit

- Prompts user to place bills on transfer tray, intakes transfer tray for processing
- For each bill on transfer tray:
    - Reads bill
    - Updates account with bill value
    - Transfers bill to proper storage location

Transfer

- Prompts user to select transferee
- Prompts user for transfer amount, using bill withdrawal function to get amount of transfer
- Updates transferer's and transferee's accounts

Bankruptcy

- Request return of funds
- Set user account to default

Shut down

- Declares a winner of the game
- Requests for all money to be returned
- Terminates after endgame procedure

No changes have been made from earlier task lists since all tasks were manageable to code and functioned properly. As well, no errors arose that were beyond the team's ability to corect.

6.5 Data Storage

The program was designed with variables being declared and used as locally as possible. In the main body, the program's two defining variables, accountBalance[ ] and isPlaying[ ] arrays keep track of each player's balance and whether they are playing or not. These variables govern all transaction functions as they are used to check the validity of transactions and they keep track of the overall movement of bills in and out of the system.

Table 6.11: Data Storage

Variable	**Purpose**
int accountBalances[]	Holds balance total for each player
bool isPlayerPlaying[]	Keeps track of which players are still in the game
int numPlayers	The number of players currently playing used to check whether the game is still progressing
int transactionBills[]	Keeps track of a user's bill selections during withdrawal and transfer functions
bool isTransactionCancelled	Checks whether the user has decided to cancel a transaction at any point during the transaction, the stops update of accounts

int currPlayer	Stores index position of player who is currently in transaction

6.6 Significant Problems and Resolutions

While developing the program code for the Monopoly ATM, errors were common due to the length and complexity of the program.

The vast majority of logic errors were easily fixed due to the program design. Since each task was split into many smaller subtasks, logic errors were usually found in a subtask function, and was fixed without breaking the entire program.

However, problems relating to sensor output and motor control were the biggest problem. For each colour sensor, due to the dynamic lighting environment and the low quality of the colour sensor, readings for colours were very inconsistent.

In order to resolve the inconsistency, thorough testing was done on each coloured paper available to get RGB ranges for each colour. 7 colours were selected to be money bills, and 5 were selected to be player cards. The selection of colours for each set was chosen to limit any colours with similar RGB values so that the colour readings would be more accurate. Colour sensing was implemented through a combination of absolute ranges of RGB values, and ranges relative to the strongest value for each colour. In this way, the colour sensing could accurately account for changes in reflective intensity resulting from the proximity of the sensed paper.

As well, It was found that the motor encoders values as a function of distance travelled were inconsistent throughout the design. Hence, thorough testing was done to ensure that the set encoder limits would accurately move various parts to their desired location.

7. Verification

7.1 Verifying Project Constraints

7.1.1 Gantry length must not be less than 560mm

This constraint was trivial to verify as once the gantry system was complete and its total length was greater than 560mm, the constraint was automatically met.

7.1.2 Timeline of project must not exceed 4 weeks

This constraint was trivial to verify as the project was completed in four weeks before the demonstration deadline.

7.1.3 Movement of carriage must not be hindered

The gantry was tested extensively once it was complete. All wires were attached as to cause the least hindrance in the operation of the carriage. By performing manual testing and verifying success of the carriage being able to transport bills successfully, this constraint was verified.

7.1.4 Colour sensor can be no more than 20 mm from its target

This constraint was trivial to verify, as the measured distance between the two colour sensors and their targets was less than 20mm.

7.1.5 Must not allow access to bills unless lid is opened

The ATM was constructed with a lid that prevented any user interference with funds unless opened. The only gap in the enclosure that exceeded the initial gap limit of 6.7cm was blocked internally by the structure of the input/output tray, meeting the updated constraint.

7.1.6 Must not allow for withdrawal greater than the account balance

This constraint was easy to verify as it was software dependent. Any attempt made to withdraw more funds than a player had available was rejected which met the constraint.

7.5 Success Criteria

It was necessary to test the success criteria of 97.5% success in the identification and distribution of bills and a software implementation that kept track of finances without error. Through 84 trials, 12 for each bill type, the system correctly identified all bills but was unable to pick up 2.

The source of the error was seemingly random and was not identified. Regardless, the trials met had a success rate of 98%, meeting the 97.5% success rate requirement.

A bug in the software during demonstration led to an inaccurate value of cash outside of the ATM at the end of the game. This was found to be an issue in calling the deposit function during bankruptcy, and was corrected after the demonstration.

8. Project Plan

8.1 Overview

In this project, the tasks were heavily considered with respect to time due to the short timeframe. Major components of the design were split into smaller tasks so that team members could efficiently work together to finish the project within the time limit.

The main milestones for this project included a fully functioning mechanical design, followed by a set of code that could make the robot operate autonomously with success. Throughout the development of the robot, the majority of the mechanical design was completed by Leif and Soyazhe, and the software was primarily coded by Josh and Wais.

8.2 Task Management

The development of the robot was broken down into many small tasks to ensure that team members remained focused on the task at hand rather than the robot as a whole. With this approach, the team was able to produce great quality parts and software, which functioned with minimal error.

With each task, a specific team member was held responsible for that task. As well, other team members were assigned secondary responsibility for each task in order to help one another when difficulties arose. With this approach, team members gained an understanding about different aspects of the robot as it developed, ensuring everyone was knowledgeable of the system as a whole. However, all team members were work collaboratively to ensure success in all tasks.

Table 8.1: Project Task Management

Task	Team Member Mainly Responsible	Additional Responsible Team Members
Gantry	Soyazhe	Leif
Carriage	Soyazhe	Leif
Vertical Actuator	Soyazhe	Leif
End Effector	Soyazhe	Leif
Bill Processing System	Leif	Soyazhe, Josh
Robot Base and Enclosure	Soyazhe	Wais, Josh, Leif
System Startup	Leif	Wais

System Shutdown	Josh	Wais
Bill Deposits	Josh	Wais, Soyazhe
Bill Withdrawals	Wais	Josh
Money Transfers	Soyazhe	Leif, Wais
Software Polishing	Wais	Soyazhe, Josh, Leif

8.3 Project Plan Revisions

During the development of the robot, some tasks become evidently more important or complex than expected and required additional time and effort. Similarly, some tasks were simpler or less important and than expected, so less time and effort were used on those aspects.

8.3.1 Gantry

It was predicted that the gantry would be simple to design, but precision of the mounts and linear rail measurements were underestimated while planning, and upon development, the gantry took a whole weekend to develop rather than only a Saturday alone. It pushed back development, slowing down the production of the robot since an insufficient amount of time was allocated to its development.

8.3.2 End Effector

Since it only had one small motion, it was expected that the end effector would take minimal effort to construct. Although, upon reflecting about the purpose of this mechanism, it was determined that it required a spring, to allow its actuation mechanism to push the component flush with the tray upon activation. As well, it had to be able to mount to the carriage and had to connect to the pneumatic system.

After the first test, the vertical actuator was deemed a failure since it was not moving along a strictly vertical axis. As a result, the vertical actuator had to be redesigned using dovetail grooves, and it took a total of two weeks to have a complete functioning end effector, rather than the expected single weekend.

8.3.3 Money Input & Output System

The money input and output system for the ATM was expected to be a motor connected to the transfer tray that would pull the transfer tray along its linear rails, in and out the ATM enclosure. However, after considering the mechanical model at this stage of development, it was determined that this approach was no longer applicable since it would interfere with carriage movement due to its the height of the motor.

As a result, an additional weekend had to be committed to designing a compatible money input and output system, rather than the projected three days. Since development was delayed to the last weekend, it delayed development of the ATM. As a result, testing of the overall system was pushed back by two days.

8.3.4 Software Polishing

Being the last component of the project, the code polishing did not receive enough attention. The entire mechanical design was complete five days before demo day, leaving a limited amount of time to produce working code, and then polishing that code. Since the deadline was strict, it was agreed upon that the team would focus on having a complete set of code and only clean up the display of code, rather than refactor it and optimize it.

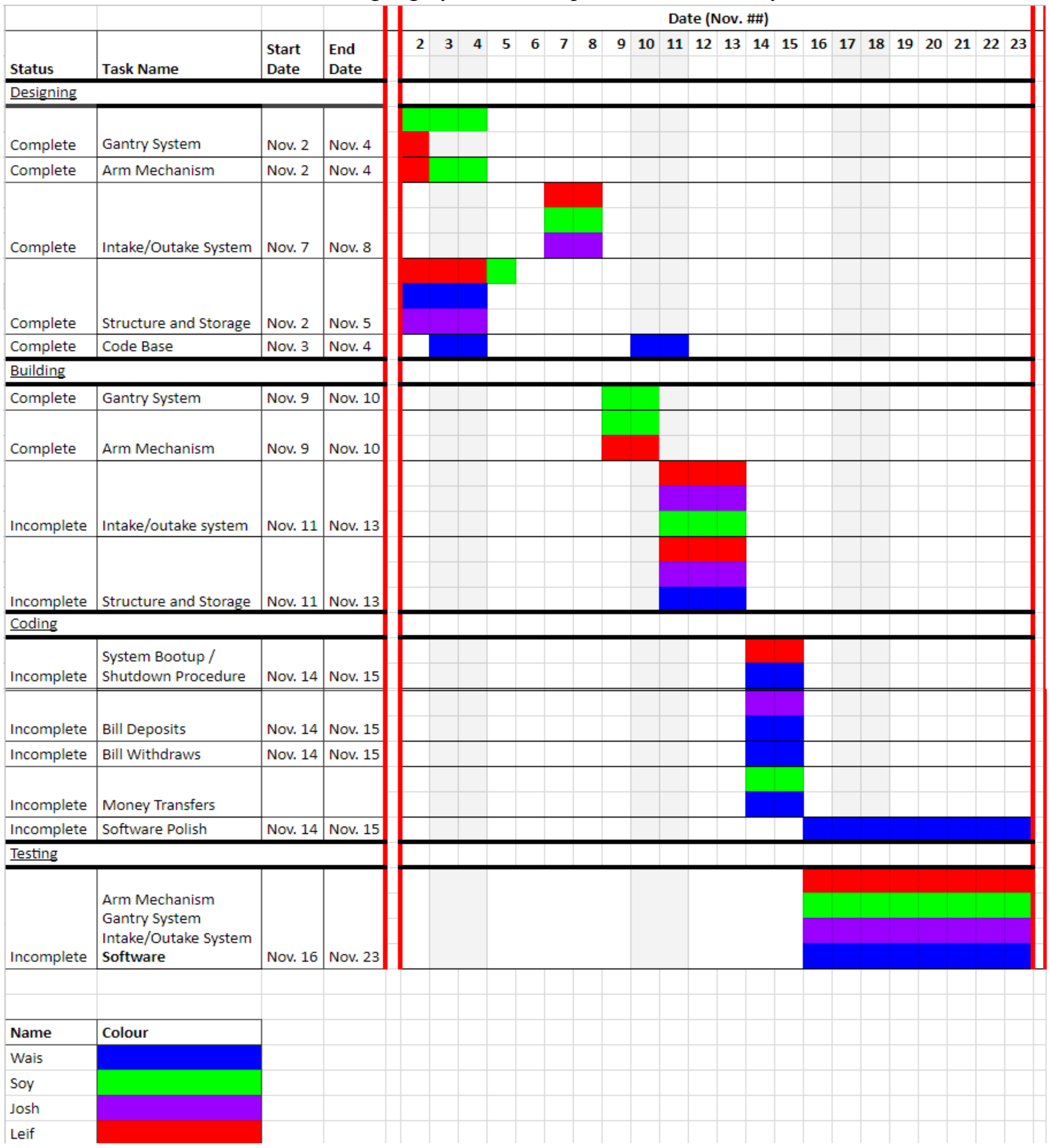
8.4 Deviations from the Planned Schedule and Actual Schedule

Due to the overall time constraint of four weeks for the project, each task had tight time allocations, and it was expected that the project would be under development until demo day.

Along with the project plan of splitting up large tasks into many smaller tasks, it left a short amount of time to complete each task, so team members had to remain focussed at all times to ensure the completion of the robot in time.

Figure 8.1: Gantt Chart of Planned Project Timeline

Note that light grey columns represent weekend days



Status	Task Name	Start Date	End Date
Designing			
Complete	Gantry System	Nov. 2	Nov. 4
Complete	Arm Mechanism	Nov. 2	Nov. 4
Complete	Intake/Outake System	Nov. 7	Nov. 8
Complete	Structure and Storage	Nov. 2	Nov. 5
Complete	Code Base	Nov. 3	Nov. 4
Building			
Complete	Gantry System	Nov. 9	Nov. 10
Complete	Arm Mechanism	Nov. 9	Nov. 10
Incomplete	Intake/outake system	Nov. 11	Nov. 13
Incomplete	Structure and Storage	Nov. 11	Nov. 13
Coding			
Incomplete	System Bootup / Shutdown Procedure	Nov. 14	Nov. 15
Incomplete	Bill Deposits	Nov. 14	Nov. 15
Incomplete	Bill Withdraws	Nov. 14	Nov. 15
Incomplete	Money Transfers		
Incomplete	Software Polish	Nov. 14	Nov. 15
Testing			
Incomplete	Arm Mechanism Gantry System Intake/Outake System **Software**	Nov. 16	Nov. 23

Date (Nov. ##): 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Name	Colour
Wais	🟦
Soy	🟩
Josh	🟪
Leif	🟥

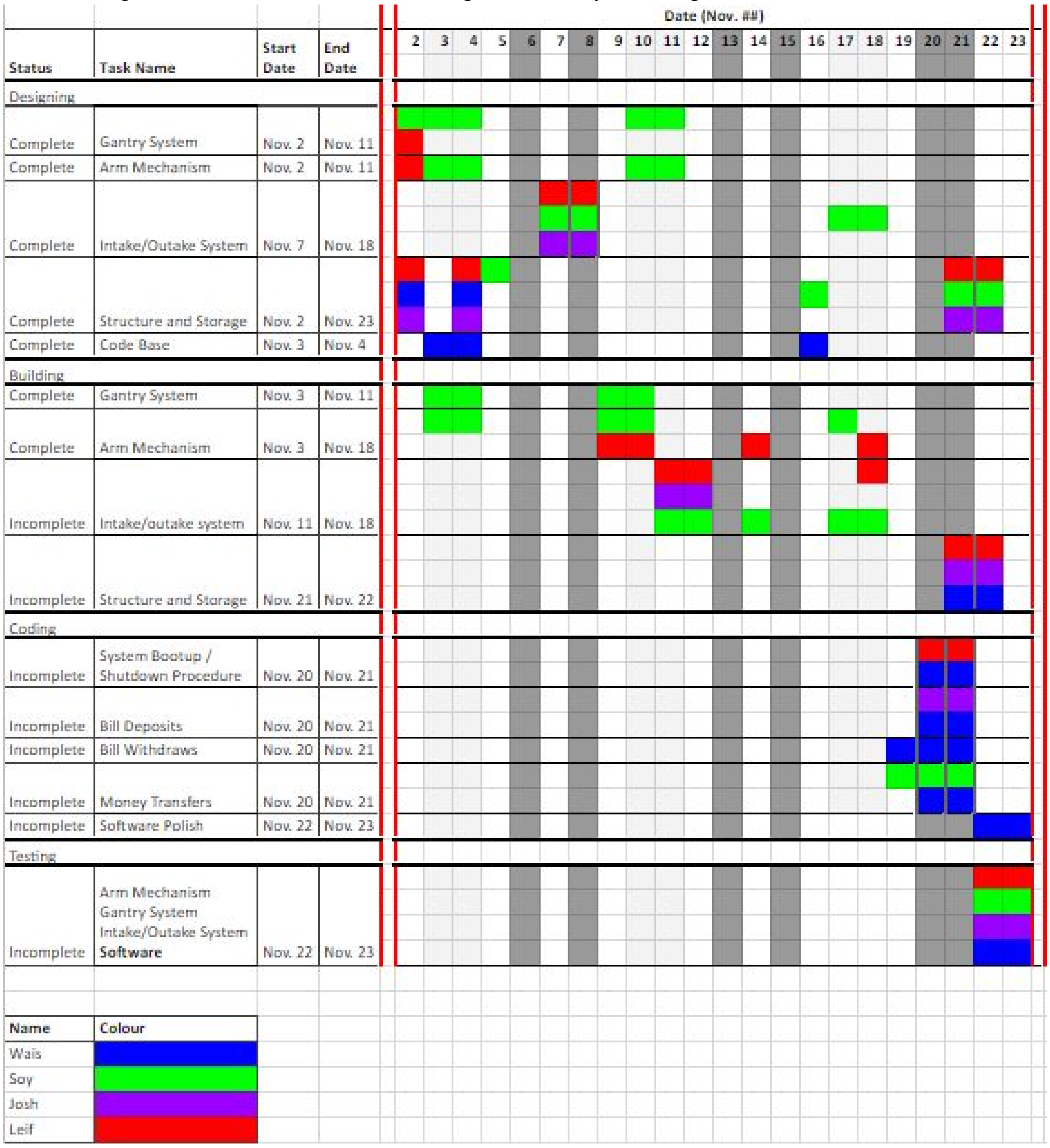
The planned schedule for the project shows the ideal time and task management for the project.

However, due to alterations in the project plan, many tasks were pushed beyond their due date. This delayed the start of subsequent tasks since each task required the completion of the previous task. Such alterations in the project plan include the gantry, the vertical actuator (as part of the arm mechanism) and the bill processing system (intake/outtake tray). More detail about each project plan revision can be found above under Section 8.3.

As well, there were select dates where team members were not available to work due to their other course loads and conflicting schedules. The real project plan differs from the projected plan as limited work or no work was completed on Tuesdays and Thursday, with no work being completed on the last Wednesday.

Figure 8.2: Gantt Chart of Actual Project Timeline

Note that light grey columns represent weekend days, while dark gray lines represent days where limited or no work was completed on the robot. Notice in the significant delay to testing and limited time to test.



Status	Task Name	Start Date	End Date
Designing			
Complete	Gantry System	Nov. 2	Nov. 11
Complete	Arm Mechanism	Nov. 2	Nov. 11
Complete	Intake/Outake System	Nov. 7	Nov. 18
Complete	Structure and Storage	Nov. 2	Nov. 23
Complete	Code Base	Nov. 3	Nov. 4
Building			
Complete	Gantry System	Nov. 3	Nov. 11
Complete	Arm Mechanism	Nov. 3	Nov. 18
Incomplete	Intake/outake system	Nov. 11	Nov. 18
Incomplete	Structure and Storage	Nov. 21	Nov. 22
Coding			
Incomplete	System Bootup / Shutdown Procedure	Nov. 20	Nov. 21
Incomplete	Bill Deposits	Nov. 20	Nov. 21
Incomplete	Bill Withdraws	Nov. 20	Nov. 21
Incomplete	Money Transfers	Nov. 20	Nov. 21
Incomplete	Software Polish	Nov. 22	Nov. 23
Testing			
Incomplete	Arm Mechanism Gantry System Intake/Outake System **Software**	Nov. 22	Nov. 23

Name	Colour
Wais	🟦
Soy	🟩
Josh	🟪
Leif	🟥

9. Conclusions

The ATM was designed to solve the problem of cheating Monopoly's financial system. The design was able to successfully meet the set constraints of:

1. Gantry length less than 560mm
2. Movement of carriage not hindered
3. Not allowing withdrawals exceeding account balance
4. Not allowing access to bills unless the lid is open

Additionally, the success criteria of 97.5% success in the identification and distribution of bills was met during verification. However, a bug in the software miscalculating the outstanding cash outside the ATM at the end of the game was present during demonstration, failing to meet the success criteria of zero error in the software implementation. This bug was fixed after demonstration.

The criteria of cost, carriage weight, and manufacturing time were also accounted for in the selection of designs.

The final mechanical design implemented several important features:

1. Gantry constructed with square aluminum tubing
2. Carriage propelled along fishing line
3. Vertical actuator propelled by crank mechanism
4. Pneumatic system controlled by syringe, propelled by crank mechanism
5. Input/Output tray propelled by fishing line and pulley mechanism

The software design was implemented through a hierarchy of functions, with the player's turns looping until only one player remained. Player financial data was stored in arrays contained within the main task, which were then passed by reference to the main transaction functions for modification. The main tasks were as follows:

1. Withdrawal
2. Deposit
3. Transfer
4. Bankruptcy

The ability to distinguish consistently between the 7 bill colours and 5 player cards was achieved through the implementation of RGB colour sensing. Finally, the user interface was implemented with specific buttons corresponding to lettered lists of options and was consistent throughout the program for ease of use.

10. Recommendations

10.1 Move transfer tray to side of gantry

The transfer tray was positioned at the fourth bin location, roughly in the centre of the bins, in order to maximize the overall efficiency of distributing bills to the bins. However, during testing, it was found that in order to satisfy the success criterion of delivering a bill to its correct location 97.5% of the time, the carriage had to be zero after every movement to a bin location. If this were not done, unexpected drift in the carriage's movement would result in bills not being dropped into the bins correctly.

Hence, moving the transfer tray to the side of the gantry would lower the time required to move bills between their storage location and the transfer tray - the carriage could zero its motion overtop of the transfer tray, eliminating the extra movement of moving between the zero position and the transfer tray.

10.2 Decrease the use of literals in the code

The implementation of constants and enumerations was inconsistent throughout the code. Expanding their use, especially early on in the software design process, would have made editing the code easier and would have facilitated the integration of the multiple functions.

Specific instances of this include the declaration of player indices, particularly the "Monopoly Man" as constants, and the enumeration of bill values. The bill location indices were coded as an using enumerated data types, but only later in the design process, making the earlier integration of some functions problematic.

10.3 Segment code for better organization

The Monopoly ATM is able to support four main task functions: deposit, withdrawal, transfer, and bankruptcy. The code could be much easier to maintain if these four functions were coded in separate files and imported into the main file. This approach would reduce the number of lines in the main program file, and finding functions related to a specific task would be easily located by accessing source file rather than scrolling through all of the lines of code together.

Furthermore, any following projects involving an ATM concept could implement parts one of the main four tasks by simply importing its source file, making the code more applicable and versatile.

References

[1] L. Blake et. al., “Interim Report”, 2018, [Online] Available: UW LEARN. [Accessed Dec 2, 2018]

[2] Fusion3, *F400 USER MANUAL*, Revision 3, July 6 2016. [Online]. Available: https://www.fusion3design.com/wp-content/uploads/2016/07/F400-User-Manual-Rev-3-7-6-16.pdf. [Accessed Nov 12, 2018]

[3] Jason, “EV3 Cookie Icing Machine”, *jkbrickworks.com*, 2015. [Online]. Available: https://jkbrickworks.com/ev3-cookie-icing-machine#1517. [Accessed Nov 17, 2018]

Appendix 1

1.1 End Effector Crank Arm Length Calculations:

- 3-Hole Lego Piece Length = 16mm
- 5-Hole Lego Piece Length = 32mm
- 7-Hole Lego Piece Length = 48mm

The maximum length of motion possible is the length of the dovetail groove on the front carriage, 72mm. The minimum length of motion was determined to be 56mm since a 16mm overlap was necessary when the end effector is at its fully extended position. Therefore,

56mm ≤ Length of Actuation ≤ 72mm.

Due to the Medium EV3 motor mounted at the top of the front plate, the range of the crank arm was found limited to 37 degrees above the horizontal for the retracted position. However, the crank arm could rotate until it was flush with the front carriage at its extended position. The length of actuation, then, was defined:

$$Length\ of\ Actuation\ =\ Length + (Length) tan ø$$, where ø is 37 degrees

Using 3-Hole Lego Piece for Crank Arm:

$$Length\ of\ Actuation\ =\ 16\ +\ 16tan(37)\ =\ 28$$

Using 5-Hole Lego Piece for Crank Arm:

$$Length\ of\ Actuation\ =\ 32\ +\ 32tan(37)\ =\ 56$$

Using 7-Hole Lego Piece for Crank Arm:

$$Length\ of\ Actuation\ =\ 48\ +\ 48tan(37)\ =\ 84$$

The only viable crank arm length was the 5-hole Lego piece.

1.2 Plunger Length Calculations:

- Plunger Guide Length: 72mm
- End Cap Length: 6mm each
- Extension Past Guide: 20 mm (since bins had 20mm high walls)
- Compressed Spring Length: 12mm

The plunger length required had as long as all components listed above to complete its functionality.

$$\begin{aligned} Plunger\ Length\ &=\ Plunger\ Guide\ Length\ +\ 2(End\ Cap\ Length)\ +\ Extension\ Past\ Guide \\ &\quad +\ Compressed\ Spring\ Length \\ &=\ 72\ +\ 2(6)\ +\ 20\ +\ 12 \\ &=\ 116 \end{aligned}$$

The plunger length was 116mm.