Final Report – Pinball Machine

Group 454

Nicholas Drazso, 20826832
Noah MacAskill, 20824705
Jia Sheng Lu, 20823525

MTE 100 and GENE 121

December 3rd, 2019

Summary

The addition of a pinball machine to the twelfth floor of Claudette Millar Hall was proposed as a solution to the lack of entertainment options available there. The Scope section will describe in full detail everything that the complete pinball machine will do. A discussion of changes made to the constraints and criteria of this project can be found in the Constraints and Criteria section. All details on the design and creation of the mechanical and software systems of the pinball machine can be found in the Mechanical Design and Implementation and the Software Design and Implementation sections respectively. The Verification section will explain how each of the constraints are met. The Project Plan section will discuss the distribution of tasks to each group member and compare the initial project plan to the actual project timeline. Finally, the recommendations section will present possible improvements and changes to certain mechanical and software designs that could potentially improve the overall project.

Acknowledgements

This project was possible with specific thanks to 2 members of the teaching team. Thanks to Nathan Klassen, who helped create a pinball machine layout plan that works with the limited amount of motors and sensors given. As well, thanks to Ryan Consell, who helped troubleshoot the coding of the Tetrix Prime standard servo motors. No other help was received for this project. Also, the code used for RobotC file input/output is acknowledged to be written by Professor Hulls, to whom we extend our gratitude.

Table of Contents

List of Tables

List of Figures

Introduction

There is a lack of entertainment options on the twelfth floor of CMH, which is a problem for the university students who live there and need a convenient source of fun and stress relieve. Thus, this project's aim was to create a pinball machine as a convenient source of entertainment for the residents of the twelfth floor of CMH, so that they can take a break from studying and have some fun.

Pinball is an arcade game which involves using flippers to flip pinballs up an inclined playing field and hitting special targets to gain points [1]. The game would end when all the pinballs on the playing field have fallen past the flippers and into an endzone. The pinball machine produced in this project is capable of launching pinballs autonomously on to the playing field, uses elapsed playing time to calculate points, and features two unique game modes on a space-themed playing field (shown in Figure 1).

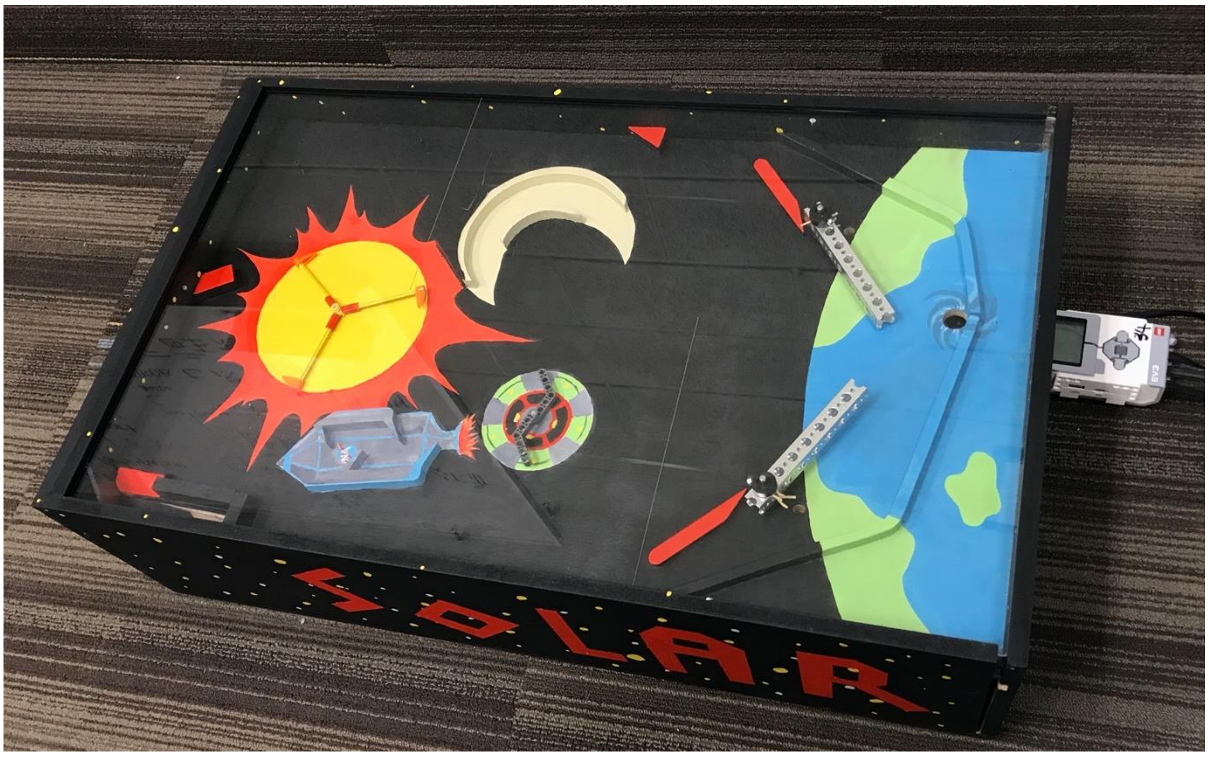


Figure 1: the complete pinball machine

Scope

Functionality and Interactions

Upon starting the program, the pinball machine prompts the user to select one of two game modes – "Classic" or "Balls of Fury" – by pressing the up or down button on the EV3 brick, respectively. Once a game mode is selected, the current score and number of live balls is displayed on the EV3 brick screen, and the two field motors on the playing field starts spinning their attached obstacles at 20% power.

In Classic mode, the ball launching mechanism motor gradually accelerates to 100% power, and through a series of large gear to small gear connections, turns the flywheel at a very fast speed. Next, the ball release mechanism motor turns an attached arm 27° upward, allows one pinball to roll down the ramp towards the ball launching mechanism, and then turns the arm back down to block the next pinball. The released pinball passes under the spinning flywheel at the bottom of the ramp and gets launched up a second ramp into the playing field. The ball launching motor decelerates until it is at 0% power. This marks the start of the game.

In Balls of Fury mode, the ball launching mechanism motor spins at 100% power for a long enough duration of time to allow the ball release mechanism motor to release 8 pinballs by turning its arm up and down 8 times. All 8 pinballs are launched into the playing field, the ball launching motor decelerates, and the game starts.

During the game, the pinball machine is performing multiple tasks at the same time. The user is able to control the left and right flippers by pressing their corresponding touch sensors. Holding a touch sensor turns its corresponding servo motor by 75°, which turns the attached Tetrix beam (the "flipper") to its raised position. Releasing that touch sensor returns the flipper to its initial position as seen in Figure 2 below.

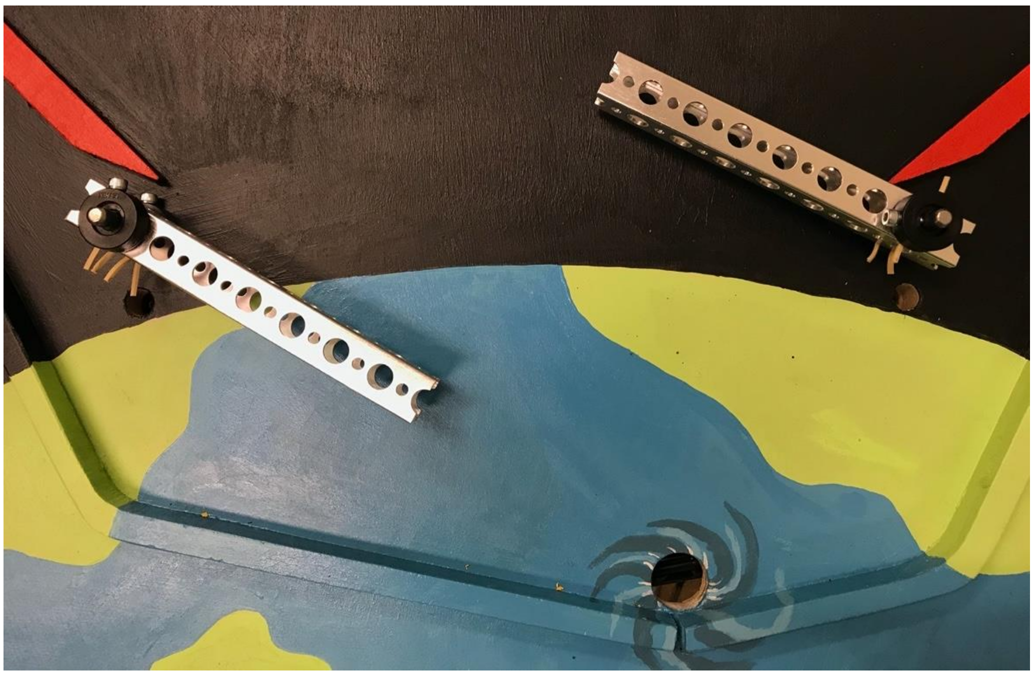


Figure 2: left flipper in its initial position, right flipper in its raised position

The ultrasonic sensor is polling for a measurement of 255cm or less than 10 cm, which results from a pinball falling into the endzone and activating the ball detection system (described in the Mechanical Design and Implementation section). Once such a measurement is detected, the number of live balls shown on the display screen is decremented by 1. To compensate for situations where the ultrasonic sensor only detects one such measurement for multiple balls or multiple measurements for one ball, the user may use the up and down buttons on the EV3 brick to manually adjust the number of live balls displayed on the screen. Pressing the up button increments the live ball count, and pressing the down button decrements the live ball count.

There are a few other tasks being performed during the game that differs between the 2 different game modes, summarized in Table 1 below:

Table 1: Unique Tasks Performed During the Game for Each Game Mode

Classic Mode	**Balls of Fury Mode**

• Every 5 seconds, the score is incremented by 10 • Every 30 seconds, a new ball will be launched into the playing field • Every 30 seconds, the field motors will both increase 20% in power and change direction (the speed no longer increases at the 150 second mark since it is at 100% power)	• Every 10 seconds, the "value" of each live ball is doubled, and the score is incremented by the product of the number of live balls and the "value" of each live ball • Every 10 seconds, the field motors will both increase 20% in power and change direction (the speed no longer increases at the 50 second mark since it is at 100% power)

The pinball machine recognizes the end of the game when the number of live balls become 0, as a result from balls dropping into the endzone. In Balls of Fury mode, the game would also end when the elapsed game time has surpassed one minute, even if there are still balls left on the playing field.

After the game ends, the 2 field motors will be set to 0% power and the user will no longer be able to control the flippers. A file containing the top 100 scores of the selected game mode is opened and the score obtained by the user is sorted into the correct spot within the scores (see appendix B7). The user's score and ranking relative to other scores is displayed on the screen, and "high score!" is displayed if the user's score is ranked first. Then the top 5 scores are displayed for 5 seconds before the program ends.

Changes in Scope

Initially, a touch sensor was planned to be situated in the middle of the playing field that the user could try to hit with the pinball. Every 3 hits of the touch sensor would award the user with an extra pinball being launched on to the play field. However, there were no extra sensor ports on the brick available for this additional touch sensor, since 2 ports are for the 2 touch sensors that control the flippers, 1 port is for the ultrasonic sensor, and the last port is for the Tetrix controller to which the 2 servo motors (for the flippers) are connected. This last port was initially planned for the touch sensor on the playing field, before it was realized that the Tetrix controller could only be connected to a sensor port. Thus, changes were made to have additional balls be launched every 30 seconds instead of using the touch sensor.

Another major change is the addition of another game mode, Balls of Fury, to the pinball game. This game mode was added after Classic mode was coded, and it was clear that another game mode that uses most of the functions already written for Classic mode could be easily added. This change makes the pinball machine more entertaining since having 8 balls on the playing field results in more interactions happening between the obstacles and the balls. It also increases the complexity of the program and highlights the transferability of many functions in the program as they are useable for 2 different game modes.

Constraints and Criteria

Constraints

The constraints stated in the preliminary design report were:

- The cost of extra materials for the pinball machine must not exceed $80 [1]
- The weight of the pinball machine must not exceed 20lbs [1]

Both constraints have been changed since the preliminary design report was written. The cost constraint was removed as it became more difficult to minimize as the project progressed. Much extra materials, such as wood, acrylic, paint, custom LEGO pieces, and pinballs had to be purchased, some of which came with additional shipping fees as well, leading to the total cost way exceeding the anticipated $80. A decision was made to instead focus on the quality of the project instead of spending time to search for the cheapest available option. Not only was this constraint unimportant in helping guide the project design, it was holding back many good design ideas that would elevate the quality of the project, because these ideas would lead to purchasing more expensive materials. For example, when deciding on the type of wood to buy for the frame of the pinball machine, which consists of four walls and a bottom board, ½'' MDF and ¼'' MDF were considered. The ½'' MDF would ensure a more stable frame that does not shake or break as easily as the ¼'' MDF, but it costs more than a ¼'' MDF of the same length and width [2] [3]. Thus, the cost constraint was ultimately removed as the quality of the project was deemed as more important by all group members.

The cost constraint was replaced by a constraint on the dimensions of the pinball machine: it must not be more than 3ft long, 2ft wide, and 1ft tall. These values were developed from measuring the dimensions of a table on the twelfth floor of CMH, on which the pinball machine would be most frequently placed. This constraint was put in place to ensure the pinball machine is well contained within the edges of that table, and the surface of the playing field is at roughly the waist level of the user. This ensures the pinball machine is not at risk of falling off the table, and the user can have a complete view of the entire playing field while playing the game. In addition, this constraint ensures the pinball machine can fit through any doors between the twelfth floor and the WEEF lab for transportation during presentations and Demo Day. Overall, this constraint was not very crucial in helping guide the project design, since it only serves to ensure the final product can be safely placed and transported. All components of the machine were able to fit within this dimensional constraint without concern.

The weight constraint changed from no more than 20lbs to no more than 35lbs. The first value was developed from weighing a sample table roughly the anticipated weight of the pinball machine, before its actual frame was built. The actual frame, plus the playing field board and acrylic cover, weighed 26lbs. After taking this measurement, some notebooks and small wooden pieces were placed on top of the frame to simulate the additional components yet to be integrated, and that weight measured 32lbs. Thus, 35lbs was decided to be the new weight constraint, with the extra 3lbs added to account for any errors in estimation. Again, this constraint was not very important in guiding the project design as it was easy to find light materials that would not contribute significantly to the overall machine's weight, but would still be reliable for their intended functions. The reason to have the weight constraint was to ensure the pinball machine can be moved easily without risk of injury [1].

In summary, the updated constraints for this project are:

- Dimensions must not exceed 3ft long by 2 ft wide by 1 ft tall
- Overall pinball machine must not exceed 35 lbs in weight

Criteria

There are no criteria listed in the preliminary design report due to a miscommunication of the rubric for that report, which required requirements and constraints. However, a list of criteria was developed for the formal presentation shortly afterwards:

- Aesthetics
  - Outer walls are painted and playing field is decorated [4]
- Entertainment Factor
  - 1 obstacle on average for every 20in^2 of the playing field [4]
- Mobility
  - No loose parts within or around the machine so that it is easy to carry [4]

These criteria mostly stayed the same throughout the project, the first one was improved slightly to be more measurable. A survey was created to see if the future users of this pinball machine think it is indeed aesthetically pleasing, and it will be deemed as such if 15 people think so.

The aesthetics and entertainment criteria played significant roles in guiding the design of the playing field. In order for the pinball machine to appear aesthetically pleasing and be as entertaining as possible, much effort was put into brainstorming the background of the playing field and designing the different obstacles that could be placed on it. In order to incorporate the mobility criteria, all the internal components of the pinball machine were secured to the frame using various techniques that will be discussed in the Mechanical Designs and Implementations section below.

In summary, the updated list of criteria is:

- Aesthetics
  - At least 15 people signed off on it being aesthetically pleasing
- Entertainment Factor
  - 1 obstacle on average for every 20in^2 of the playing field
- Mobility
  - No loose parts within or around the machine so that it is easy to carry

Mechanical Design and Implementation

Overall Mechanical Description

The pinball machine has 8 core hardware components. 6 of these components, the ball detection system, ball release mechanism, ball launching mechanism, field motors, flippers, and ramps, are contained within the frame and covered by the playing field board, the 2 other components. A view of how the internal components are placed within the frame is shown in **Error! Reference source not found.** below.

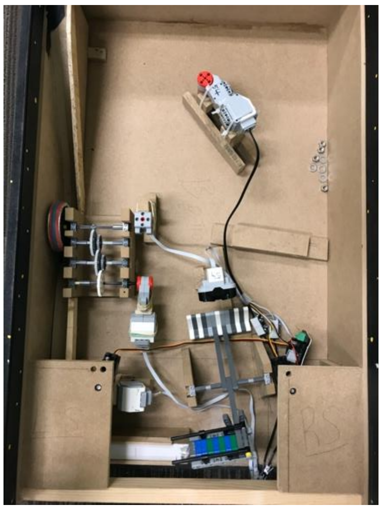


Figure 3: internal mechanical system layout

The ball detection system, ball release mechanism, ball launching mechanism, and ramps work together to autonomously collect balls from the endzone, and eventually release and launch them back into the playing field. The flippers allow the user to interact with the pinballs during the game, and the field motors spin obstacles on the playing field to make the game more exciting.

The Frame

Summary

The frame was the component of the pinball machine that housed all 6 internal mechanical components. It was constructed using mainly ½" MDF. The frame had an upper and a lower section which was divided by the playing field and the upper section was covered by a sheet of acrylic. It was held together using wood glue as well as screws.

Design Considerations

When designing the frame, there were 3 main considerations. First, the frame had to allow for quick and easy access to the lower section. To ensure this was possible, dado cuts were made along the inside walls of the frame to allow the playing field to slide in and out whenever necessary as well as for the acrylic top. Secondly, the front face had to be removable. To ensure this, hanger bolts were fastened into the front of the side pieces. Holes were cut in face, and nuts and washers were used to fasten the face to the frame. Lastly, to ensure the frame would be structurally sound throughout the duration of the project, 2 key supports were added. The first support was a piece of pine that was placed slightly under the dado cut in the front in order to strengthen frame since when the front face was off, the two side walls were not fully supported. In addition, a second back piece was used to help absorb any hard-hit balls and prevent the vibrations from effecting any internal mechanisms.

Manufacturing

The frame was constructed out of mainly ½" MDF. The pieces were cut using a table saw as well as a mitre saw. Many rips with the table saw were made to create the dados since there was no access to a proper dado saw or router with the correct bit. The side of the frame were fastened together with wood glue as well as #6 – 1 ¼" wood screws. The screws were counter sunk on each side to ensure the MDF would not crack when driving the screws in. In addition, hanger bolts were used on the front face to secure the front onto the frame.

Ball Detection System

Summary

The ball detection system was responsible for detecting when a ball had been lost in the game. The system relies on a see-saw mechanism in conjunction with an ultrasonic sensor to detect fallen balls, as shown in Figure 4. When the system is at its resting state, the sensor will read a value of approximately 27cm, and when a ball falls it falls on to a platform on one end of the see-saw causing the sensor to be blocked and read approximately 4cm for a small duration. The software will recognize this change and decrease the ball count.

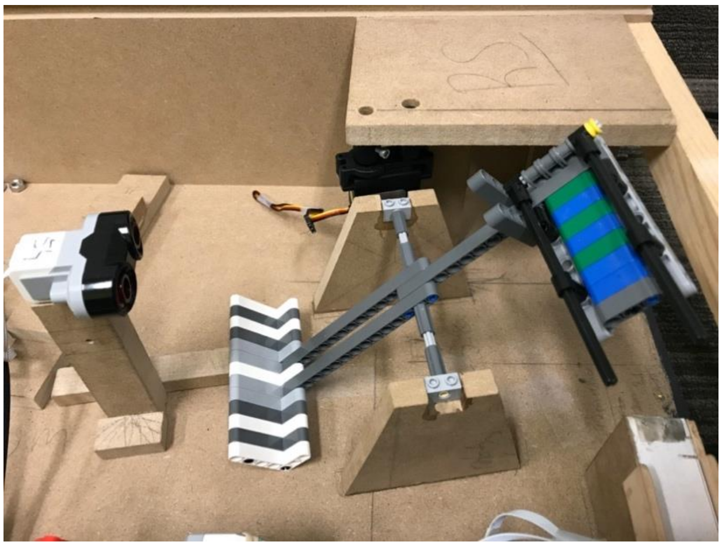


Figure 4: ball detection system, consisting of the ultrasonic sensor and see-saw

Design Considerations

When designing the ball detection system, there were 3 main considerations. First, that the ultrasonic sensor would always change its reading a substantial amount when a ball falls and that it would do it consistently. To ensure the sensor would have enough time to read a change in value, the weighting of the see-saw mechanism was essential. The end closer to the sensor had to be heavier than the platform side so the sensor's value at rest would be large. This side also had to be large to ensure the sensor would always be blocked upon a ball falling and there would be no chance the sensor would miss the wall and therefore not detect a dropped ball. To do both of these, the side closer to the sensor was made larger and heavier than the platform side.

Secondly, the platform had to be able to contain and guide the ball onto the ramp consistently and allow for multiple balls at once. To ensure this, the platform was first built independently from the rest of the mechanism in order to get good dimensions and build support around the edges of the platform, so the pinball did not roll off. In order for the ball to roll onto the ramp consistently, the see-saw needed to be slightly inclined towards the ramp, this was done by machining the centre wooden support opposite the ramp to be taller than the other centre support. The rails along the edges of the platform guided the ball on to the ramp as well. Also, to ensure that multiple balls could be handled at once, the rails were made tall enough that no balls would roll off. This was tested and was concluded to work.

Lastly, to ensure the system would be stopped at the perfect position during the rested and activated states, the bottom side of the playing board was used to stop both sides of the mechanism. When in the rested state, the top of the platform would rest against the bottom of the playing board and wait for a ball. Once a ball had fallen onto the platform, the top of the wall on the sensor side would hit the bottom of the playing board, ensuring the platform would stop slightly above the ramp so the ball could roll off onto the ramp with a small boost of speed given from the small fall, the ultrasonic sensor was positioned accordingly.

Manufacturing

The majority of the system was constructed with Lego because it was easy to modify and test with. The supports for the ultrasonic sensor and the see-saw mechanism were made using ½" MDF because it allowed for the easiest mounting to the frame as well as its increased rigidity compared to a Lego mounting system. In addition to the structural benefits of MDF, it was also easier to glue MDF supports to the board compared to trying to mount Lego pieces temporarily. The entirety of the see-saw mechanism was built using Lego and epoxied to the MDF support pieces.

Ball Release Mechanism

Summary

The ball release mechanism was responsible for the release of a single ball to the ball launching mechanism. It would lift a Lego arm up at a slow speed until a desired encoder value to allow a singular ball to roll down to the ball launching mechanism, before dropping a fast speed to contain the ball behind it. This mechanism can be seen below in Figure 5.

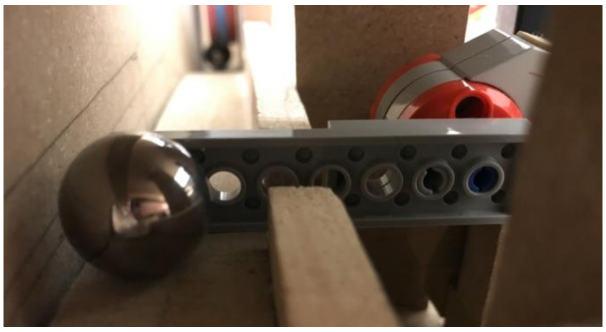


Figure 5: ball release mechanism

Design Considerations

When designing the ball release mechanism, there were 3 main design considerations. To ensure accurate zeroing of the encoder, the arm had to be parallel with the ramp. To ensure this, the motor was mounted at the perfect height so that the arm would be parallel to the ramps surface, and so that through software, the discrepancy between the arm's real location and its location in the software would be minimized. Secondly, the motor had to be extremely secured to the mounting bracket so that the encoder value would be accurate for all balls. To ensure this, 5 heavy duty elastic bands were used to secure the motor to the mount. Lastly, the system had to be small in order to allow space for other systems. To ensure this, a small and simple design was chosen over a more complex and potentially more reliable design.

Manufacturing

The mechanism was built using Lego and mounted to MDF. Since there was a constraint on the size of the mechanism, a Lego arm was used because it allowed for the simplest and the smallest mechanism possible. An MDF mounting bracket was made because of the rigidity it offered compared to a Lego one as well as the ease of being able to glue the MDF directly to the base of the frame.

Ball Launching Mechanism

Summary

The ball launching mechanism was responsible for launching a single ball up a ramp and into the playing field. It was driven by a large motor connected to a gear box which was attached to a flywheel. When activated, the flywheel would spin, and a ball would be guided under the spinning flywheel and launched up the ramp. This can be seen below in Figure 6.

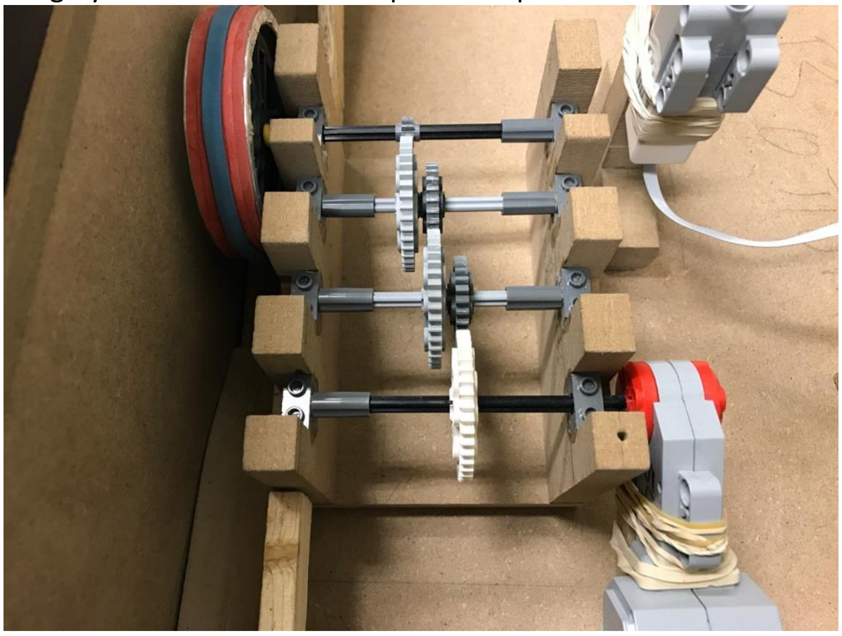


Figure 6: ball launching mechanism

Design Considerations

When designing the ball launching mechanism, there were five main design considerations.

First, to ensure that the flywheel would spin fast enough to propel the ball up the ramp, a gear box was used. The gear box used 3 x 24 tooth, 2 x 16 tooth, and 1 x 8 tooth gears in order to increase the rpm of the final axle and spin the flywheel fast enough. In addition, the location on the flywheel that struck the ball was also important to ensure the ball made it up the ramp. A groove was sanded along the outside edge of the base in order to keep the ball along the wall of the main frame so the location on the flywheel that struck each ball was consistent.

Secondly, the gears had to be both aligned along the axle and interlocked with each other enough to prevent unnecessary wear and prevent as much loss of energy as possible.

For these reasons, the accuracy during the manufacturing process was essential to the systems ability to function at its maximum potential.

Third, in order to stabilize the system as much as possible, the frame for the mechanism was over engineered. The two walls were screwed and glued to the base and slots were made for the axle mounting blocks so that they could be epoxied on 3 sides instead of just 1.

Fourth, the driving motor had to be mounted so that it would lose as little power as possible while it was running. To do this, a mounting bracket was made, and the motor was secured to it with 12 elastic bands. After testing, it was concluded that the motor did successfully fulfill this requirement.

Fifth, the distance between the base of the mechanism and the bottom of the flywheel had to be slightly less than the diameter of the ball in order for the flywheel to apply enough power to the ball to get up the ramp. This distance was controlled by the height that the slots in the frame walls were cut to. A prototype was built with Lego and tested until an appropriate distance was achieved, then measurements were taken of the prototype and reproduced on a wooden frame. The overengineered frame for the mechanism also aided in ensuring that this distance would always remain consistent.

Manufacturing

There were 2 main components to this mechanism. The gear box was made using Lego gears, axles, connectors, adaptors, and mounting blocks. These were used instead of 3D-printed parts due to the time constraint and ease of testing. The frame for this mechanism was built using ½" MDF because structural stability was a key requirement for this mechanism. To manufacture the frame for this mechanism, the walls and base were cut with a bandsaw and the slots were cut using a bandsaw and scroll saw. The walls were mounted to the base with wood glue and countersunk screws.

Field Motors

Summary

The field motors were responsible for spinning an obstacle on the field. They were mounted below the playing field and lined up with the holes in the playing field to allow an axle to go up through it. The field motors can be seen mounted below in Figure 7, and integrated on the playing field in Figure 8.

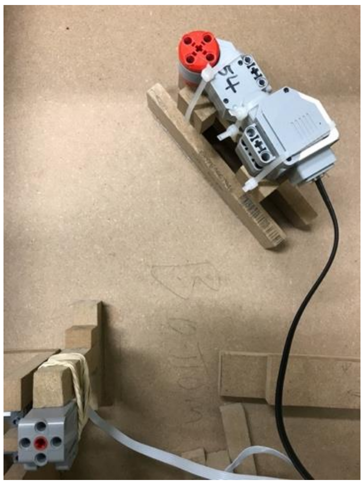


Figure 7: two field motors

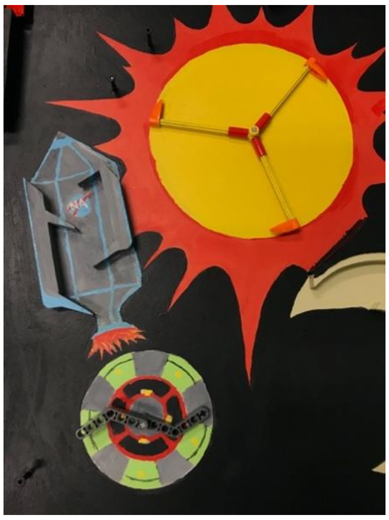


Figure 8: the two LEGO obstacles attached to the field motors (top right and bottom left)

Design Considerations

When designing the field motor mounts, there were 2 main design considerations. First, the motors had to be at the proper height so that the tops of the axles would be above the playing field board. To ensure this, the mounting brackets were manufactured at the proper height so that enough of the axle was exposed above the field to mount an obstacle to.

Secondly, the mounting brackets had to ensure that the motors would not shift after the obstacle struck the ball. To meet this requirement, the mounts were designed with multiple flat securing points and secured with many elastic bands.

Manufacturing

The mounts were made of ½" MDF because it allowed for the most rigidity and ease of mounting to the base.

Flippers

Summary

The flippers were responsible for launching the ball up the playing field. They were built using Tetrix Prime components and controlled via touch sensors. The flippers can be seen integrated on the playing field below in Figure 9, and one of their corresponding servos mounted in Figure 10.

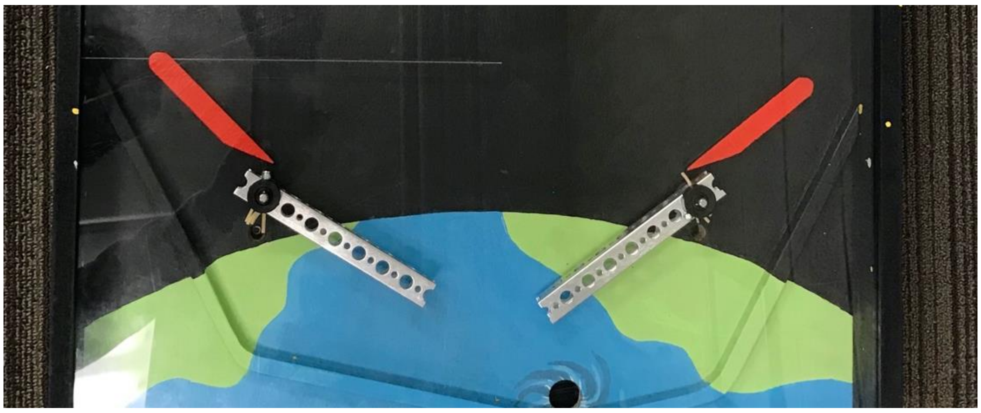


Figure 9: the two flippers on the playing field

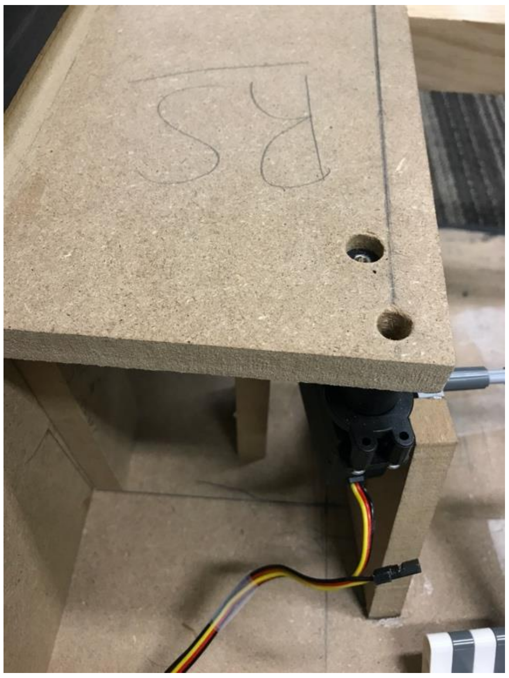


Figure 10: right servo motor and its supports

Design Considerations

When designing the flippers, there were 3 main design considerations. First, to ensure the flippers would not break, Tetrix Prime beams were used as the flippers. Since they were made from aluminum, they were light and strong. Second, the flippers had to be responsive and accurate, so Tetrix servo motors were used over LEGO motors or the continuous motors. Finally, the motors had to be mounted in the lower section and the mounts could not affect the other systems. A design where the servo motors were mounted to a plate and the plate was raised, as seen in Figure 10, was the best option due to it’s strength, and how it doesn’t interfere with other nearby systems as it’s mounted out of the way.

Manufacturing

The mounts for the flippers were built out of ½” MDF and manufactured using a bandsaw. The holes in the top of the mount (as seen in Figure 10) were cut for the rod that comes out of the servo motor and for the mounting bolt, so that the flippers could be taken out from the top. The hole for the rod lined up with a hole on the playing field and the hole for the mounting bolt was counter sunk to allow the playing field to still slide in to the frame.

Ramps

Summary

The ramps were responsible for accepting the balls from the ball detection system, guiding them to the ball launching mechanism, and up to the playing field. They can be seen below in Figure 11 and Figure 12.

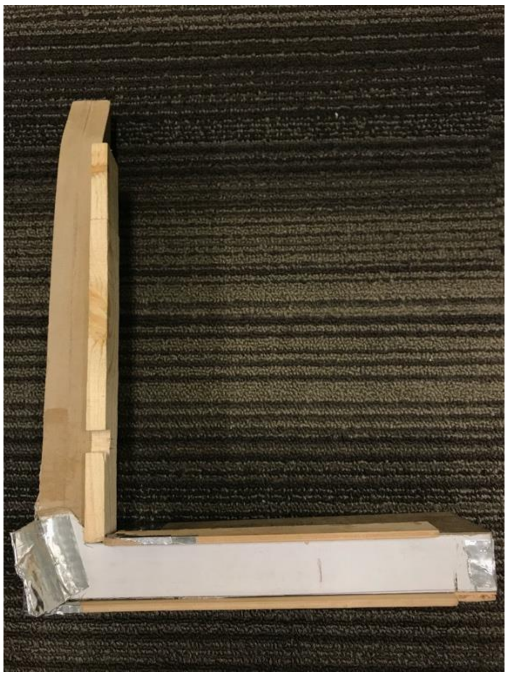


Figure 11: ramp leading from ball detection system to ball launching mechanism

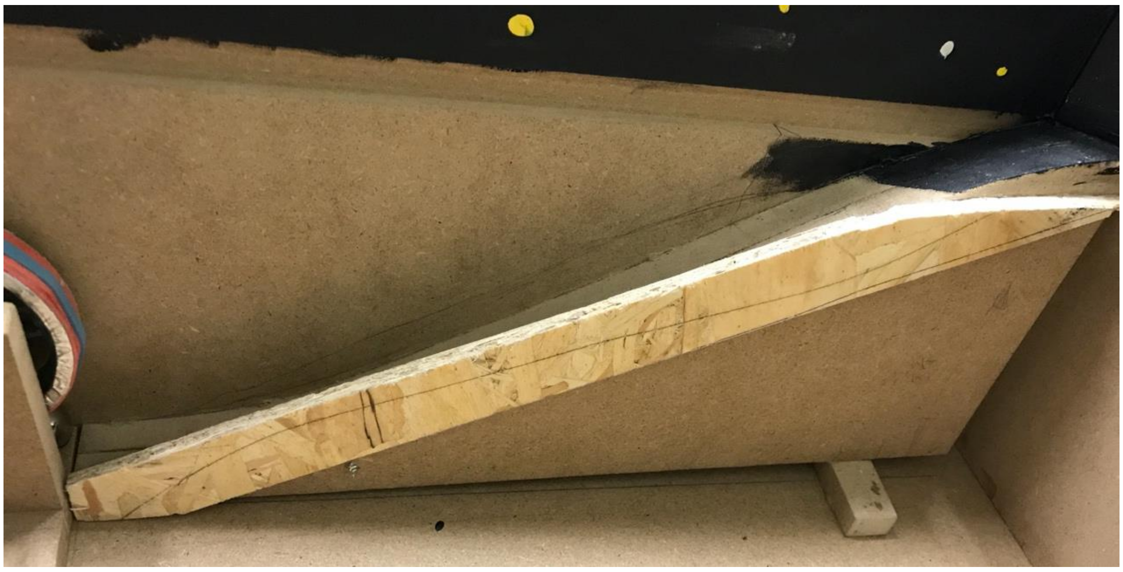


Figure 12: ramp leading from ball launching mechanism to playing field

Design Considerations

When designing the ramps, there were 2 main design considerations. First, the design had to ensure that the balls would approach the ball release and ball launching mechanisms in the desired position consistently. To achieve this, the ramp before the ball release and ball launching mechanisms was bevelled towards the frame wall. In addition to the bevel, a fence was glued onto the open sides of each ramp in order to contain the balls at all times.

Secondly, the ramp could not impede the balls ability to roll. This was ensured by sanding the surface of the ramp as well as adding cardboard pieces to improve the guidance of the ramps and to prevent the ball from getting stuck anywhere on the ramp.

Manufacturing

The ramps were constructed using ½" MDF and cut using a band saw. Each ramp was built to a width of 1" by gluing two identical pieces together in order to take into account for the diameter of the ball being ¾". Ideally, the ramps could have been manufactured to be exactly ¾" wide but due to the time constraint and the materials available, this was not possible. The ramps were cut so the slope changed at rate of 0.5" down vertically per 5" down the length of the ramp so the ball would have had enough speed to roll down the ramp. This was constrained by the height that the ball detection systems platform would be at during the active state and the location of the flippers and ball release mechanism. The curve that the final ramp was cut at was determined through prototyping. The only factor that was considered during prototyping was how the initial slope of the ramp would affect the balls path up the ramp and if it could potentially launch the ball up and hit the bottom of the playing field.

Playing Field

Summary

The game of pinball takes place on the space-themed playing field, as shown in **Error! Reference source not found.**. Pinballs enter the playing field through the entrance hole on the top left corner of the field. There are obstacles on the playing field which interact with the pinballs, including the 2 revolving obstacles spun by the field motors. The flippers and the endzone are located at the bottom of the playing field, near the player.

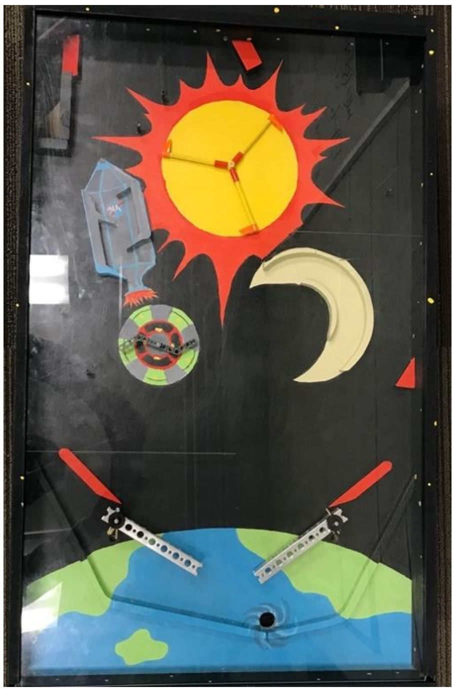


Figure 13: playing field

Another important feature of the playing field is the mini entrance ramp located at the entrance hole. This ramp acts as a one-way gate that only allows pinballs to enter the playing field, and blocks any pinballs trying to go down the ramp towards the ball launching mechanism.

Design Considerations

Aesthetics and the type, amount, and placement of obstacles are the 2 most important factors to consider when designing the playing field. Since this is the main component through which the player experiences the game of pinball, it must be made as entertaining and as aesthetically pleasing as possible. Thus, different types of obstacles, such as a curved rail that changes the direction of a pinball, a zig-zagging pair of railings that slows down the pinball, and spinning obstacles were incorporated onto the playing field to make the game more interesting. Then the field was painted to be space-themed, so that each obstacle can blend into the theme: the curved railings were painted to be the moon, the zig-zagging railings were decorated as a rocket ship, the small spinning obstacle was a part of an UFO and the large one was a part of the Sun. In this way, the obstacles could synergize with the paintings on the playing field to deliver a fun playing experience for the user.

Another important factor to consider is the location of the holes for the entrance ramp, spinning obstacles, flippers, and the endzone. All these holes must line up with the positions of their respective systems in the lower section of the frame. Thus, these positions were carefully planned during the design of the layout of the lower section to ensure that no components conflict with each other, and the positions of the holes were carefully measured on the playing field.

Manufacturing

A ½'' MDF was cut to be the playing field. All the stationary obstacles were separately made from scrap wood and glued onto the playing field.

Software Design and Implementation

Software Description and Functions

There were many unique software tasks that were to be performed in order to run a successful pinball machine. The software was written in a way that no events were missed during gameplay. For example, it did not miss any balls falling through the end zone and it was able to react to the user activating the flippers almost simultaneously. For this reason, no waits or loops were used throughout gameplay. Instead, the game updated according to many different booleans and timers. A variety of functions were called where inputs were checked for, and the appropriate booleans and timers were changed.

As mentioned, there was a lot of activity to be checking for and updating throughout gameplay. These were divided into distinct well named functions for each distinct action. Having these distinct functions was chosen for clarity and for ease of testing, as each individual function of the code could be further analysed to test for one specific thing. This allowed the code to be manipulated easily which helped create other game modes. All functions are laid out in Table 2.

Table 2: Functions

Function Name	Parameters	Return Type	Description	Writer
initialize	none	void	Configures all sensors	Noah

flippers	bool leftFlipperRaised bool rightFlipperRaised	void	Controls the flippers (see Appendix B1)	Noah
ultrasonicCheck	bool ballLost int ballCount	void	Detects when balls fall through the endzone (see Appendix B2)	Noah
ballCountUpdate	int ballCount bool downButtonPressed bool upButtonPressed	void	Detects manual ball increments and decrements (see Appendix B3)	Noah
ballLaunch	bool ballLaunchAccel bool ballLaunchDecel bool ballLaunching int ballsLaunched string gameMode	void	Launches balls into the playing fields (see Appendix B4)	Nicholas
ballRelease	none	void	Releases balls down ramp towards ball launching system (see Appendix B5)	Nicholas
setFieldMotors	int motorPower	void	Sets field motors to a desired power (see Appendix B6)	Jerry
sortScore	int score int * topScores	int	Sorts score through topScores array, returns leaderboard placement (see Appendix B7)	Jerry

Tasks

The software task list for demo began with start-up. The user was prompted to select either the Classic or Balls of Fury game mode. If Classic mode was selected, the score, 0, and ball count, 1, were displayed on the EV3 screen and one ball was launched into the playing field. If Balls of Fury was selected, then the score, 0, and the ball count, 8, were displayed on the screen, and then all 8 balls were launched into the playing field. The two field motors began to spin at 20% power.

During regular operation, the left and right flippers were raised and lowered through the user pressing and releasing the left and right touch sensors. The ball count was decremented by one every time a ball fell into the endzone. In Classic mode, the score was incremented by 10 every 5 seconds, and every 30 seconds a new ball was launched into the playing field. The field motors both increased their power by 20%, and switched directions at this time as well. In Balls of Fury mode, the value of each live ball, which started at 10, was doubled every 10 seconds and the score was incremented by the product of the number of balls that were alive by the current value of a ball. The field motors both increased their power by 20%, and switch directions as well. Once the field motors hit 100%, they were no longer incremented, but they continued to switch directions at their regular intervals.

To handle unexpected events, a manual ball increment/decrement feature was added so that the user could manually increment or decrement the ball count if the ultrasonic misread the number of fallen balls. If the up button was pressed on the EV3, then the ball count was incremented by 1 and if the down button was pressed on the EV3, then the ball count was decremented by 1.

As for the shutdown, once the ball count was 0 in Classic mode, the game would end. In Balls of Fury mode, once the ball count was 0 or 1 minute of gameplay elapsed, the game ended. The field motors were set to a power of 0, then the flipper control was deactivated, and the score stops increasing. The user's placement on the leaderboard was determined by a sorting algorithm, see Appendix B7. The user's score and placement were displayed on the screen and "high score!" was displayed if a high score was achieved. The top 5 scores were displayed to the user, and the program ended.

Data Storage in the Program

The data for the previous 100 high scores were stored on a text file on the EV3 brick. There was a file for both Classic mode and Balls of Fury mode. Once the game was completed, these scores, depending on the game mode, were sorted into an array. The score achieved by the user was then sorted into the array using the sortScore function, see Appendix B7, to determine the user's placement on the leaderboard. The updated scores were then outputted back to the text file.

Design Decisions

Various software design decisions were made throughout the project in order to optimize the program functionality and readability. In the beginning, the main program consisted of various loops that controlled the flippers, with conditions that were constantly checking for certain inputs (fallen balls, manual ball increment/decrement, score updates, etc.). It was decided that this format had too much repetition and should be improved (thanks to Ryan Consell for helping realize a more efficient format). Instead of having loops within loops where certain inputs are constantly acting as the exit condition, the aforementioned system where functions are constantly being called within one big outer loop (which checks for end of game) was chosen. This system uses booleans to let the program know when to do what task or check for what condition (launch new balls, decrement the ball count, raise/lower flippers, etc.).

An example of how this system works can be seen in the flippers function (see Appendix B1). There are two booleans (rightFlipperRaised and leftFlipperRaised). The program is constantly checking for touch sensor inputs (corresponding to the left and right flippers). Once this is detected, the corresponding boolean is made true and the flipper is raised. The program then checks for when the touch sensor is released, and it will then turn the corresponding boolean back to false and lower the flipper. This system can be seen fully implemented in Appendix A.

Testing

Testing procedures (including the reason for testing, test cases and expected behaviour) can be seen below in Table 3.

Table 3: Software Testing

Test	Reason	Test Cases	Expected Behaviour
Flippers	To ensure flippers are fully functioning	• Holding flipper buttons • Spamming flippers buttons	• Both flippers are in raised position • Flippers react responsively
Ball release mechanism	To ensure all balls can be released	• Releasing different amounts of balls	• All balls are released, one by one
Ball launching mechanism	To ensure balls are launched on to playing field	• Launching balls on to the playing field	• Balls are launched up ramp and on to playing field
Sorting/displaying score	• Ensure score is sorted properly • Ensure leaderboard is displayed properly	• Testing good scores (on the leaderboard) • Testing bad scores (off the leaderboard)	• The score is sorted into the array, leaderboard is updated, and user's placement is displayed • "u r pathetic" is displayed to user as they haven't made the leaderboard

Problems

The biggest issue that was encountered had to do with the timing of the ball release mechanism. This system raises and lowers an arm to release one ball at a time down a ramp to the ball launching mechanism, as described in the Mechanical Design and Implementation section. The issue was that as more balls got released into the playing field, there would be less force pushing against the front ball (as there were less balls behind it). This meant that the ball would move slower than balls previous, and an up and down arm speed for the first ball was found to be too quick for balls further down the line. Resolving this issue took patience and lots of trial and error. The timing was eventually perfected by deciding to raise the arm at a slow speed until it's high enough to let the ball through, then lowering it quickly. This optimized timing led up to 8 balls to be launched into the playing field most of the time. The EV3 motor still behaves inconsistently despite all factors being carefully controlled, perhaps due to the precise encoder count that it has to turn every time.

Verification

Dimensional Constraint

49'' by 97'' (roughly 4ft by 9ft) MDF's were purchased from Home Depot, which were used to construct the frame of the pinball machine. The left and right pieces of the frame were cut to 2.5ft (for the frame's length) by 0.7ft (for the frame's height); the front and rear pieces of the frame were cut to 1.5ft (for the frame's width) by 0.7ft (for the frame's height); the top and bottom pieces of the frame were cut to fit the frame's length and width accordingly. In this way, the frame's overall dimensions are 2.5ft long by 1.5ft wide by 0.7ft (measured by a tape measure), which meets the constraint on the dimensions of the board stated in the Constraints and Criteria section.

Weight Constraint

Since the frame of the pinball machine was constructed from ½'' MDF for stability, it is quite heavy and leaves 9lbs of weight for the rest of the components to be integrated to the frame (see Constraints and Criteria section). Thus, lighter materials such as thin plywood and small pieces of ½'' MDF were used to construct the various systems placed beneath the playing field board, and LEGO pieces were used to construct the see-saw balance of the ball detection system to minimize additional weight. The estimated amount of materials to be used to construct the extra components were also weighed beforehand to ensure their weight does not exceed 9lbs. In the end, the entire pinball machine weighed 34lbs (just below the 35lbs limit) on a scale as a result of this careful planning and use of lighter materials.

Discarded Constraint - Cost

As per mentioned in the Constraints and Criteria section, the constraint that "the cost of extra materials for the pinball machine must not exceed $80" was discarded. It quickly became clear that this constraint was not going to be met once more planning was done to determine the amount of extra materials that had to be purchased, and more research was done to determine the costs of each material. For example, the amount of wood that had to be purchased costs $25, the acrylic cover alone costs $26, and the paint supplies cost $25.

A design change that can address the issue is purchasing lower quality materials in general, such as purchasing ¼'' MDF instead of ½'' MDF for the frame or using low quality paint to decorate the machine. However, as discussed before, the focus for this project is more about quality than cost, thus this constraint was discarded.

Project Plan

Task Distribution

All team members worked together to design the mechanical and software components of the pinball machine. Since only one of the group members, Nicholas Drazso, has had extensive experience in woodworking, he constructed the majority of the hardware. Only Nicholas and Jia Sheng Lu had ESMS machine shop training, so Jia Sheng also helped manufacture some pieces needed for various components. This leaves Noah MacAskill to code the majority of the software, while Jia Sheng and Nicholas each wrote a few functions as well. Overall, every group member participated in some aspect of the design and creation of both mechanical and software systems, and worked together to assemble the pinball machine in the end.

Deviations from Project Plan

There were no changes made to the project plan presented in the preliminary design report [1], however the actual project timeline deviated significantly from it due to many unforeseen issues. Table 4 compares the planned dates of completion for all tasks (can be found on the preliminary design report as well) to their actual dates of completion.

Table 4: Anticipated vs Actual Task Completion Dates

	Anticipated Completion Date	**Actual Completion Date**
Frame Construction	Nov. 4th	Nov. 3rd
Ball Launching Mechanism Construction	Nov. 7th	Nov. 6th
Ball Release Mechanism Construction	Nov. 7th	Nov. 11th
Flippers System Construction	Nov. 13th	Nov. 16th
Start Game Code	Nov. 7th	Nov. 19th
Ball Launching Code	Nov. 7th	Nov. 15th
Points Assignment Code	Nov. 10th	Nov. 16th
Highscore File I/O Code	Nov. 11th	Nov. 19th
Field Motor Control Code	Nov. 15th	Nov. 16th
Flippers Control Code	Nov. 15th	Nov.17th
Installation of Obstacles onto Playing Field	Nov. 17th	Nov. 20th
Decorating and Painting Pinball Machine	Nov. 19th	Nov. 21st
Testing and Refining of Overall System	Nov. 20th	Incomplete

The planned timeline was followed accordingly for only the first 2 tasks – covering frame construction and ball launching mechanism construction. All other tasks were completed after the anticipated completion date, one reason being the team's inability to receive the parts requested in the extra parts request form until November 12th. The extra parts requested included the servo motors needed to be the flippers, extra touch sensors needed to control the flippers, extra motors needed to be the field motors, and extension cords, thus the tasks that depended on these components were bottlenecked. For example, without the servo motors, the code written to control them couldn't be tested and the structure to hold them in place couldn't be designed, thus flipper system construction and flippers control code completion dates were both delayed.

The completion dates of high score file I/O code, ball launching code, and start game code marked the largest deviations from their planned completion dates, since the team decided to focus on the more important tasks such as constructing the ball launching and release mechanisms and troubleshooting the setting up of the servo motors. These large deviations are also a result of unthorough planning. It is unreasonable to have the code for ball launching be done on the same day that the construction of the ball launching mechanism is done, since the code needs to be tested on the hardware first and then further debugged and refined. It is also unreasonable to program the start game and file I/O code before the flippers control (which is more important to the functionality of the pinball machine) is coded.

Additionally, many important, time consuming tasks were not even included in the project plan due to oversight, which is another reason for the majority of the planned tasks being pushed back. Such tasks include designing and constructing the ball detection system, coding a function for the ultrasonic sensor to detect a ball in the endzone, designing and constructing all the internal ramps, and affixing all components to the frame of the pinball machine.

Conclusions

The most important features of the pinball machine include launching pinballs onto the playing field via its ball releasing and launching mechanisms, allowing the user to control the flippers via touch sensor presses, detecting balls that fall through the endzone, and ranking the scores of players. Each feature has its associated hardware system and RobotC function, which work together to accommodate for two distinct game modes. In general, the pinball machine executes all of its tasks successfully, with the exception of the ball release mechanism, which is inconsistent in its ability to release balls despite all hardware and software factors being controlled very carefully.

The pinball machine project has successfully provided a fun source of entertainment to the residents of the twelfth floor of CMH, bringing joy to everyone who played it. The constraints on the pinball's dimensions (no more than 3ft long by 2ft wide by 1ft tall) and weight (no more than 35lbs) were both met. As for the criteria, at least 15 other people thought it was nice aesthetically, there was measured to be at least 1 obstacle on average for every 20in^2 of the playing field, and there were no loose parts in and around the pinball machine, resulting in an aesthetically pleasing, entertaining, and easy to carry product.

Possible Improvements for Mechanical Design

Ball Release Mechanism

The ball release mechanism was the most inconsistent system of the pinball machine. As mentioned in the Software section, the ball release arm was incredibly difficult to control precisely. Sometimes it would release one ball as expected, but other times it would release 2 balls or no balls at all, despite all factors being carefully controlled.

An alternate design for the ball release mechanism, where the EV3 motor rotates a board with a pinball-sized hole cut into it (see Figure 14), could increase the consistency and reliability of this system. Every time a pinball is to be released, the motor would turn the board 360° at a rate just slow enough for one pinball to roll through the hole cut-out in the board. This eliminates the need for the current ball release arm to precisely turn 27° repeatedly, which is a possible source of its inconsistency since the motor encoder is not very precise and error will eventually build up.

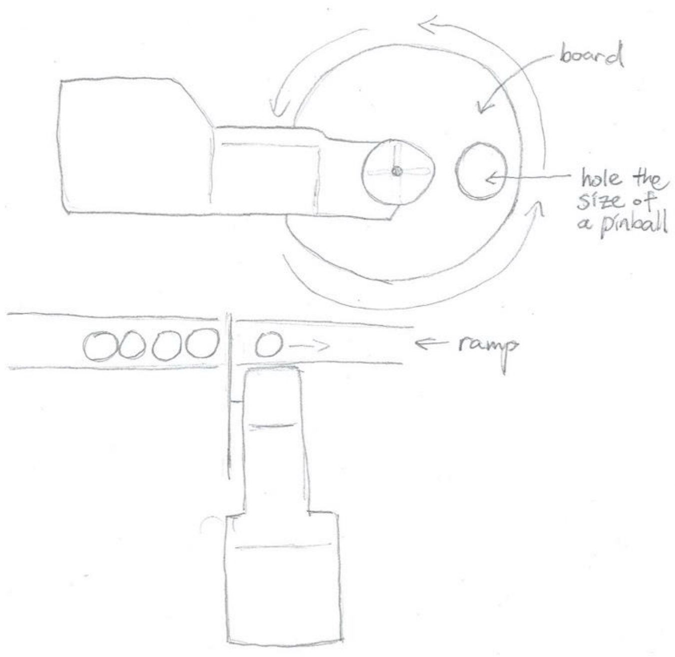


Figure 14: design for improved ball release mechanism

Ball Launching Mechanism

This system experiences the most amount of mechanical stress due to the high gear ratios used to turn each axle, and the high speed at which the gears are turned, which could gradually wear out the gears. In addition, the axle onto which the flywheel is attached bends every time a pinball passes under the flywheel, which wears the axle and wastes energy which could be used to further propel the pinball.

A potential improvement is to replace the LEGO gears and axles with ones made from alloy steel, which is a far stronger material than plastic and suitable for gears and axles [5]. The EV3 motor should also be replaced with a high torque motor compatible with the new axles, perhaps coded via an Arduino board. This would result in a much higher quality system that does not wear easily and launches the pinball much more powerfully.

Internal Ramps

The ramps inside the pinball machine are 0.25'' wider than the diameter of the pinball and have small bumps and unlevel portions which added inconsistencies to the movement of balls on it. This sometimes results in a pinball not rolling toward the ball release mechanism because it is stuck behind a bump, or a pinball not being released or launched properly because it had too much sideways movement on the ramps.

Narrowing down ramps to being 0.125'' or even 0.063'' wider than the diameter of the pinball would minimize sideways movements of the pinball, and using the width of one MDF piece instead of gluing 2 of them together would avoid unevenness throughout the ramp surface. This design better regulates the movement of pinballs inside the pinball machine and further improves the consistency of the ball release and launching mechanisms.

Endzone Ball Management

There is currently no system in place to prevent 2 or more balls falling into the endzone consecutively, which may result in the ultrasonic sensor detecting the 2 lost balls as only 1 ball. Although this situation does not happen often, having a feature in place to prevent this situation would further sophisticate the pinball machine.

One recommendation is to use a system similar to the newly recommended design for the ball release mechanism as mentioned above. A board, which has a pinball-sized hole on it and is attached to an EV3 motor, would block the bottom of the endzone hole. During the game, the motor would continuously spin the board so that every 5 seconds or so the pinball-sized hole would match up with the endzone hole and allow only one pinball to slip through, and the ultrasonic would have sufficient time to distinguish between this lost pinball and the next one. This solution could use the same function that would be used to control the newly recommended ball release mechanism as well, since the two systems behave identically, just (possibly) with different motor powers.

Possible Improvements for Software Design

Using Multiple Tasks

If this project were to be used in industry, its software aspect would be easier to read and maintain if a separate task was used to contain the code for each of the tasks that the pinball machine is executing during the game. As mentioned in the Software Design and Implementation section, many different conditions are being checked simultaneously during gameplay. The program currently does this by iterating through a while loop, within which are many if statements and booleans to check for specific conditions. While both software designs would achieve the same result, using different tasks to categorize what the pinball machine is doing during the game would make the code easier to read and debug in industry.

Player Profile Ranking System

An additional software feature that can make the game more professional and closer to industry standard is a player profile ranking system. This feature would allow each new player to set up a profile within the game which has their name and high scores in each game mode. At the end of each game, the highest scores for that game mode would be displayed alongside the player who obtained it. This would make the game more personal to each player and further encourage competitiveness as each player can try to beat their friends' scores.

References

[1] J. S. Lu, N. Drazo and N. MacAskill, "Preliminary Design Report for Pinball Machine," Waterloo, 2019.

[2] Rona, "MDF Panel - Natural," Rona, [Online]. Available: https://www.rona.ca/en/mdf-panel-1-4-x-49-x-97-49585310. [Accessed November 2019].

[3] Home Depot, "Metrie MDF Premium," Home Depot, [Online]. Available: https://www.homedepot.ca/product/metrie-mdf-premium-1-2-x-49-x-97/1000167402. [Accessed November 2019].

[4] J. S. Lu, N. Drazo and N. MacAskill, *Pinball Machine,* Waterloo, 2019.

[5] C. Gonzalez, "Gears Look to the Future for Material," MachineDesign, 8 December 2015. [Online]. Available: https://www.machinedesign.com/engineering-essentials/gears-look-future-material. [Accessed 1 December 2019].

Appendix A – Project Source Code

```
//import libraries
#include "EV3Servo-lib-UW.c" //to control Tetrix Standard Servo Motors
#include "PC_FileIO.c" //for file i/o with EV3 Brick

//motors
const tMotor BRM = motorA; //ball release mechanism
const tMotor BLM = motorB; //ball launching mechanism
const tMotor FIELD_MOTOR_C = motorC; //motor on playing field
const tMotor FIELD_MOTOR_D = motorD; //motor on playing field

//sensors
const tSensors TOUCH_L = S1; //touch sensor to control left flipper servo
const tSensors TOUCH_R = S2; //touch sensor to control right flipper servo
const tSensors SERVOS = S3; //servo controller board
const tSensors ULTRASONIC = S4; //ultrasonic sensor

//servo configurations
const int SERVO_L = 3; //left flipper servo connected to SV3 port
const int SERVO_R = 4; //right flipper servo connected to SV4 port
const int INIT_POSITION = 0; //initial position of the servo motors
const int RAISED_POSITION = 75; //the position that the servos will turn to

//ball release
const int ENCODER = 27; //orginally 27, 32 let 2 balls go
const int UP_SPEED = 5; //originally 5
const int DOWN_SPEED = 22;

//array size of topScores
const int NUM_SCORES = 100;

//maximum number of balls that can be launched
const int NUM_BALLS = 8;

//configures all sensors
void initialize()
{

	//Flipper System
	SensorType[TOUCH_L] = sensorEV3_Touch; // left Touch Sensor
	SensorType[TOUCH_R] = sensorEV3_Touch; //right touch sensor
	SensorType[SERVOS] = sensorI2CCustom9V; // Tetrix Input

	//ensures servo positions are set to initial
	setServoPosition(SERVOS, SERVO_L, INIT_POSITION);
	wait1Msec(100);
	setServoPosition(SERVOS, SERVO_R, INIT_POSITION);
	wait1Msec(100);

	//Ultrasonic
	SensorType[ULTRASONIC] = sensorEV3_Ultrasonic;
}

//controls the flippers in correspondence to touch sensor presses
void flippers(bool & leftFlipperRaised, bool & rightFlipperRaised)
{
	//controls left flipper
	if(!leftFlipperRaised && SensorValue[TOUCH_L] == true)
	{
		setServoPosition(SERVOS, SERVO_L, -RAISED_POSITION);
		leftFlipperRaised = true;
	}
	else if(leftFlipperRaised && SensorValue[TOUCH_L] == false)
	{
		setServoPosition(SERVOS, SERVO_L, INIT_POSITION);
```

```
			leftFlipperRaised = false;
		}

		//controls right flipper
		if(!rightFlipperRaised && SensorValue[TOUCH_R] == true)
		{
			setServoPosition(SERVOS, SERVO_R, RAISED_POSITION);
			rightFlipperRaised = true;
		}
		else if(rightFlipperRaised && SensorValue[TOUCH_R] == false)
		{
			setServoPosition(SERVOS, SERVO_R, INIT_POSITION);
			rightFlipperRaised = false;
		}
}

//endzone ball detection through ultrasonic sensor
void ultrasonicCheck(bool & ballLost, int & ballCount)
{
		if(!ballLost &&(SensorValue[ULTRASONIC] == 255 ||
			SensorValue[ULTRASONIC] < 10))
		{
			ballCount--;
			displayBigTextLine(8, "Ball count: %d", ballCount);
			ballLost = true;
			time1[T2] = 0;
		}
		//prevents the ultrasonic from counting the same ball in the endzone twice
		else if(ballLost && time1[T2] > 1050)
		{
			ballLost = false;
		}
}

//increases field motor speed by 20 (up to 100), and reverses their speed
void setFieldMotors(int & motorPower)
{
		motor[FIELD_MOTOR_C] = motorPower;
		motor[FIELD_MOTOR_D] = -motorPower;

		motorPower *= -1;

		if(abs(motorPower) != 100)
		{
			if(motorPower < 0)
				motorPower -= 20;
			else
				motorPower += 20;
		}
}

//checks for manual ball count increment/decrement and displays ball count
void ballCountUpdate(int & ballCount, bool & downButtonPressed,
		bool & upButtonPressed)
{
		if(getButtonPress(buttonDown) && !downButtonPressed)
		{
			downButtonPressed = true;
			ballCount --;
			displayBigTextLine(8, "Ball count: %d", ballCount);

		}else if (!getButtonPress(buttonDown) && downButtonPressed )
		{
			downButtonPressed = false;
		}
		if(getButtonPress(buttonUp) && !upButtonPressed)
```

```
	{
		upButtonPressed = true;
		ballCount ++;
		displayBigTextLine(8, "Ball count: %d", ballCount);

	}else if (!getButtonPress(buttonUp) && upButtonPressed )
	{
		upButtonPressed = false;
	}
}

//controls the ball release mechanism to release one ball at a time
void ballRelease()
{
	if(motor[BRM] == -UP_SPEED && nMotorEncoder(BRM) <= -ENCODER)
	{
		motor[BRM] = DOWN_SPEED;
	}
	else if(motor[BRM] == DOWN_SPEED && nMotorEncoder(BRM) >= 0)
	{
		motor[BRM] = 0;
		nMotorEncoder(BRM) = 0;
	}
}

//controls the ball launching mechanism to launch balls into the playing field
void ballLaunch(bool & ballLaunchAccel, bool & ballLaunchDecel,
	bool & ballLaunching, int & ballsLaunched, string gameMode)
{
	//acelerating the ball launching wheel
	if(ballLaunchAccel && time1[T3] > 250)
	{
		motor[BLM] += 5;
		time1[T3] = 0;
		if(motor[BLM] == 100)
		{
			ballLaunchAccel = false;
			motor[BRM] = -UP_SPEED;
		}
	}

	//keeping the ball launching wheel at max speed
	//behaviour differs depending on gamemode
	if(motor[BLM] == 100)
	{
		//launches one ball only in classic mode
		if(gameMode == "Classic" && time1[T3] > 3500)
		{
			ballLaunchDecel = true;
			time1[T3] = 0;

		//launches all balls in Balls of Fury mode
		}else if(gameMode == "Balls of Fury")
		{
			if(time1[T3] > 1800 && ballsLaunched != NUM_BALLS)
			{
				motor[BRM] = -UP_SPEED;
				time1[T3] = 0;
				ballsLaunched ++;
			}else if(time1[T3] > 2000)
			{
				ballLaunchDecel = true;
			}
		}
	}
```

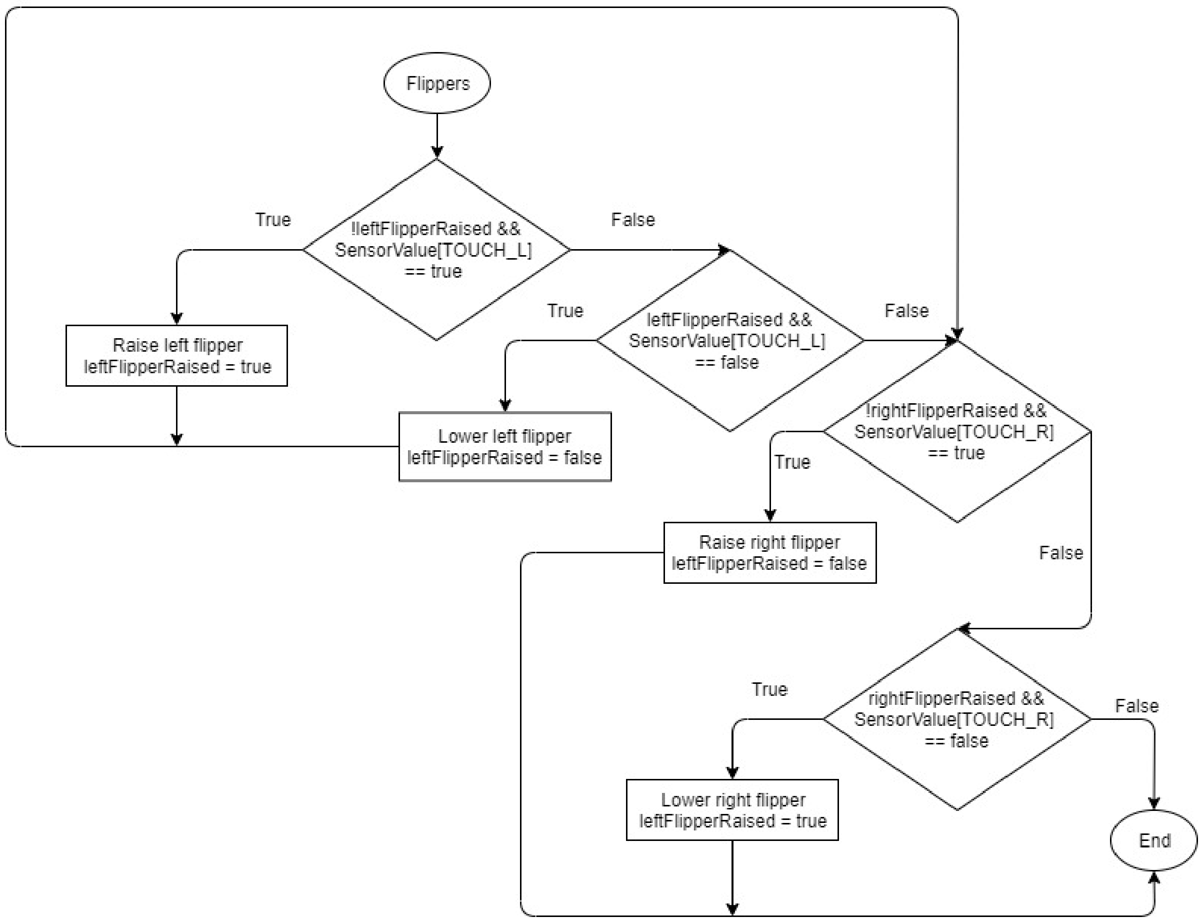
```
        //ball is release while ball launching wheel at max speed
        ballRelease();

        //decelerating the ball launching wheel
        if(ballLaunchDecel && time1[T3] > 250)
        {
                motor[BLM] -= 5;
                time1[T3] = 0;
                if(motor[BLM] == 0)
                {
                        ballLaunchDecel = false;
                        ballLaunching = false;
                }
        }
}

//sorting the obtained score into the pre-existing list of scores
//returns the ranking of the obtained score when compared with other scores
int sortScore(int score, int * topScores)
{
        int index = 0;
        bool sortComplete = false;

        while(!sortComplete && index < NUM_SCORES)
        {
                if(score > topScores[index])
                {
                        for(int j = NUM_SCORES - 1; j > index; j--)
                        {
                                topScores[j] = topScores[j-1];
                        }
                        topScores[index] = score;
                        sortComplete = true;
                }
                index++;
        }

        //if the obtained score does not rank within the top 100 scores
        if(index >= NUM_SCORES)
                index = -1;

        return index;
}

task main()
{
        initialize();

        //general variables
        int score = 0;
        int motorPower = 20; //field motor power
        int ballCount = 0; //the number of live balls on the playing field
        string gameMode = "";

        //ball launching mechanism variables
        bool ballLaunching = true;
        bool ballLaunchAccel = true;
        bool ballLaunchDecel = false;

        //flippers system variables
        bool leftFlipperRaised = false;
        bool rightFlipperRaised = false;

        //endzone ball detection variables
        bool ballLost = false;
        bool downButtonPressed = false;
bool upButtonPressed = false;

//balls of fury mode variables
int ballValue = 10;
int ballsLaunched = 1;

//initializes BRM
nMotorEncoder[BRM] = 0;

//menu - selecting game modes
displayBigTextLine(0, "MAIN MENU");
displayBigTextLine(3, "UP:");
displayBigTextLine(5, "CLASSIC");
displayBigTextLine(8, "DOWN:");
displayBigTextLine(10, "BALLS OF FURY");

while(!getButtonPress(buttonAny))
{}
if(getButtonPress(buttonUp))
{
	gameMode = "Classic";
}
else if(getButtonPress(buttonDown))
{
	gameMode = "Balls of Fury";
}
while(getButtonPress(buttonAny))
{}

eraseDisplay();

//starting the game
ballLaunch(ballLaunchAccel, ballLaunchDecel, ballLaunching, ballsLaunched,
	gameMode); //starts launcing ball(s)
setFieldMotors(motorPower); //activates field motors
displayBigTextLine(5, "score: %d", score); //displays current score
time1[T1] = 0; //setting the game timer to 0

//during the game
if(gameMode == "Classic")
{
	ballCount = 1;
	displayBigTextLine(8, "Ball count: %d", ballCount);

	while(ballCount > 0)
	{
		//updates score
		if(time1[T1] >= 5000)
		{
			time1[T1] = 0;
			score += 10;
			displayBigTextLine(5, "score: %d", score);

			//releases new ball, changes speed/direction of field motors
			if(score % 60 == 0)
			{
				setFieldMotors(motorPower);
				if(ballCount < NUM_BALLS)
				{
					ballLaunchAccel = true;
					ballLaunching = true;
					ballCount++;
				}
				displayBigTextLine(8, "Ball count: %d", ballCount);
				time1[T3] = 0;
			}
```

```
                }

                //controls flippers
                flippers(leftFlipperRaised, rightFlipperRaised);

                //checks for ultrasonic ball drop
                ultrasonicCheck(ballLost, ballCount);

                //Checks for manual ball decrement
                ballCountUpdate(ballCount, downButtonPressed, upButtonPressed);

                //manages ball launching system
                if(ballLaunching)
                {
                        ballLaunch(ballLaunchAccel, ballLaunchDecel, ballLaunching,
                                ballsLaunched, gameMode);
                }
        }
        //updates score based on extra time remaining
        score += time1[T1]/500;

}else if(gameMode == "Balls of Fury")
{
        ballCount = NUM_BALLS;
        displayBigTextLine(8, "Ball count: %d", ballCount);

        time1[T4] = 0;

        while(ballCount > 0 && time1[T1] < 61000)
        {
                //manages ball launching system
                if(ballLaunching)
                {
                        ballLaunch(ballLaunchAccel, ballLaunchDecel, ballLaunching,
                                ballsLaunched, gameMode);
                }

                //updates the score & changes field motor powers
                if(time1[T4] > 10000)
                {
                        score += ballCount * ballValue;
                        ballValue *= 2;
                        setFieldMotors(motorPower);
                        displayBigTextLine(5, "score: %d", score);
                        time1[T4] = 0;
                }

                //controls flippers
                flippers(leftFlipperRaised, rightFlipperRaised);

                //checks for ultrasonic ball drop
                ultrasonicCheck(ballLost, ballCount);

                //Checks for manual ball decrement
                ballCountUpdate(ballCount, downButtonPressed, upButtonPressed);
        }

}

//ending the game
//stop all motors
motorPower = 0;
setFieldMotors(motorPower);
motor[BLM] = motor[BRM] = 0;

//reset servo positions
```

```
setServoPosition(SERVOS, SERVO_L, INIT_POSITION);
wait1Msec(100);
setServoPosition(SERVOS, SERVO_R, INIT_POSITION);
wait1Msec(100);

eraseDisplay();

//read in prev top 100 scores from file
TFileHandle fin;
if(gameMode == "Classic")
{
	bool fileOkay = openReadPC(fin, "highscoreClassic.txt");

}else if(gameMode == "Balls of Fury")
{
	bool fileOkay = openReadPC(fin, "highscoreBOF.txt");
}

int topScores[NUM_SCORES];
int prevTopScore = 0;

for(int count = 0; count < NUM_SCORES; count++)
{
	readIntPC(fin, prevTopScore);
	topScores[count] = prevTopScore;
}

//sorts the current score into the array if it's greater than any top score
int rank = sortScore(score, topScores);

//writes the updated score list into file
TFileHandle fout;
if(gameMode == "Classic")
{
	bool fileOkay2 = openWritePC(fout, "highscoreClassic.txt");

}else if(gameMode == "Balls of Fury")
{
	bool fileOkay2 = openWritePC(fout, "highscoreBOF.txt");
}

for(int count = 0; count < NUM_SCORES; count++)
{
	writeLongPC(fout, topScores[count]);
	writeEndlPC(fout);
}

//displays the obtained score and the rank of that score
displayBigTextLine(0, "Score: %d", score);

if(rank == -1)
{
	displayBigTextLine(3, "u r pathetic");
}
else
{
	displayBigTextLine(3, "You placed %d !", rank);
	if(rank == 1)
	{
		displayBigTextLine(6, "HIGHSCORE!");
	}
}

wait1Msec(3000);

//shows the top 5 scores
```

```
        eraseDisplay();
        displayBigTextLine(0, "Leaderboard");
        for(int count = 0; count < 5; count++)
        {
                displayBigTextLine(2*count+3, "%d: %d", count + 1, topScores[count]);
        }

        wait1Msec(5000);
}
```

Appendix B1 – flippers Function Flowchart



Flippers
!leftFlipperRaised && SensorValue[TOUCH_L] == true
True
False
Raise left flipper leftFlipperRaised = true
leftFlipperRaised && SensorValue[TOUCH_L] == false
True
False
Lower left flipper leftFlipperRaised = false
!rightFlipperRaised && SensorValue[TOUCH_R] == true
True
False
Raise right flipper leftFlipperRaised = false
rightFlipperRaised && SensorValue[TOUCH_R] == false
True
False
Lower right flipper leftFlipperRaised = true
End

Figure 15: flippers function flowchart

Appendix B2 – ultrasonicCheck Function Flowchart

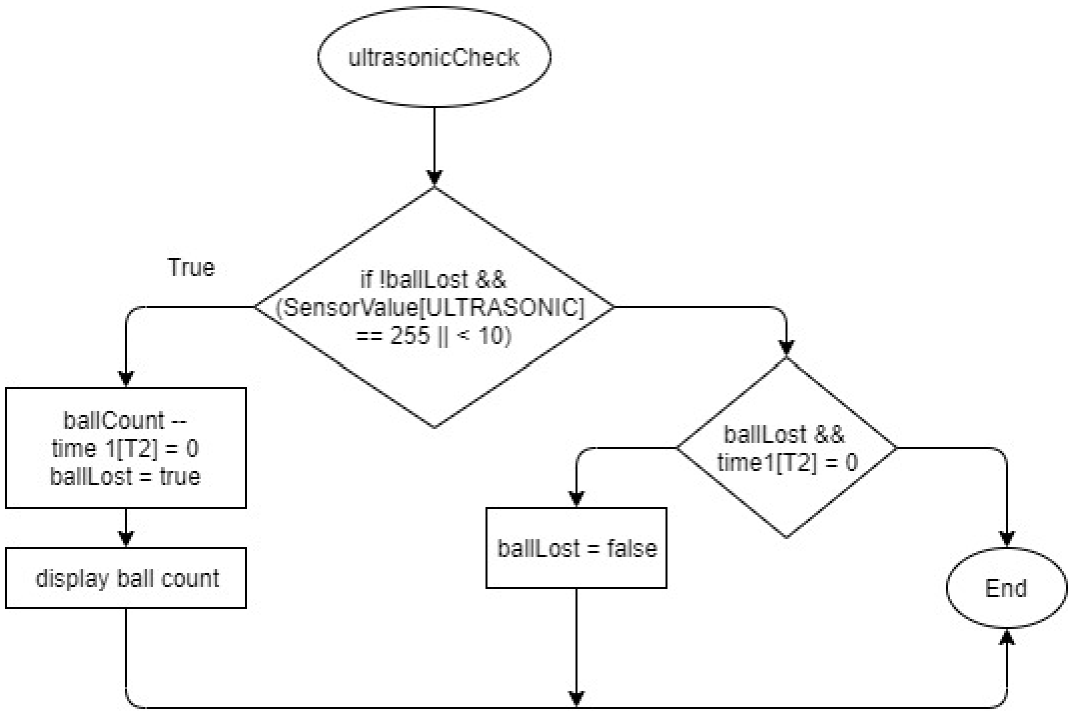

ultrasonicCheck
True
if !ballLost && (SensorValue[ULTRASONIC] == 255 || < 10)
ballCount -- time 1[T2] = 0 ballLost = true
display ball count
ballLost && time1[T2] = 0
ballLost = false
End

Figure 16: ultrasonicCheck function flowchart

Appendix B3 – ballCountUpdate Function Flowchart

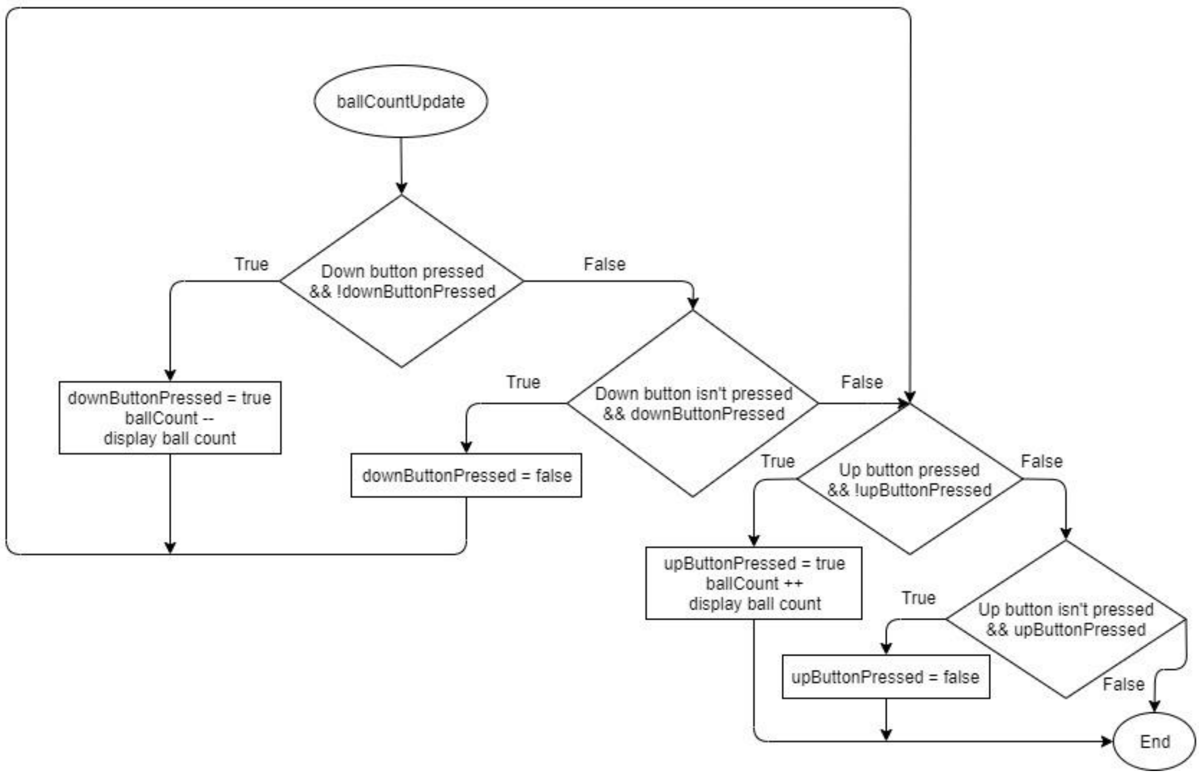

ballCountUpdate
True
Down button pressed && !downButtonPressed
False
downButtonPressed = true
ballCount --
display ball count
True
Down button isn't pressed && downButtonPressed
False
downButtonPressed = false
True
Up button pressed && !upButtonPressed
False
upButtonPressed = true
ballCount ++
display ball count
True
Up button isn't pressed && upButtonPressed
upButtonPressed = false
False
End

Figure 17: ballCountUpdate function flowchart

Appendix B4 - ballLaunch Function Flowchart

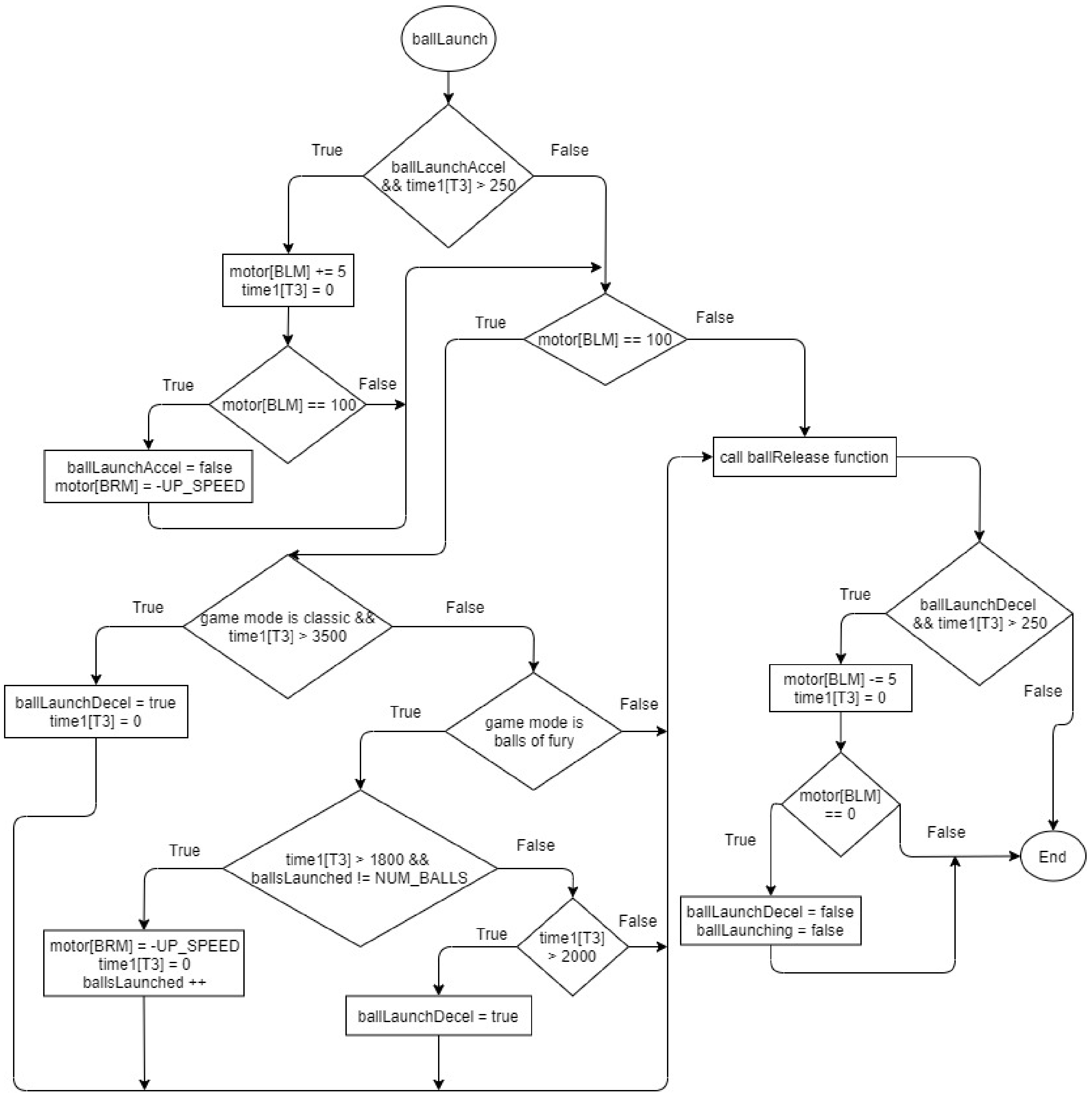


Figure 18: ballLaunch function flowchart

Appendix B5 – ballRelease Function Flowchart

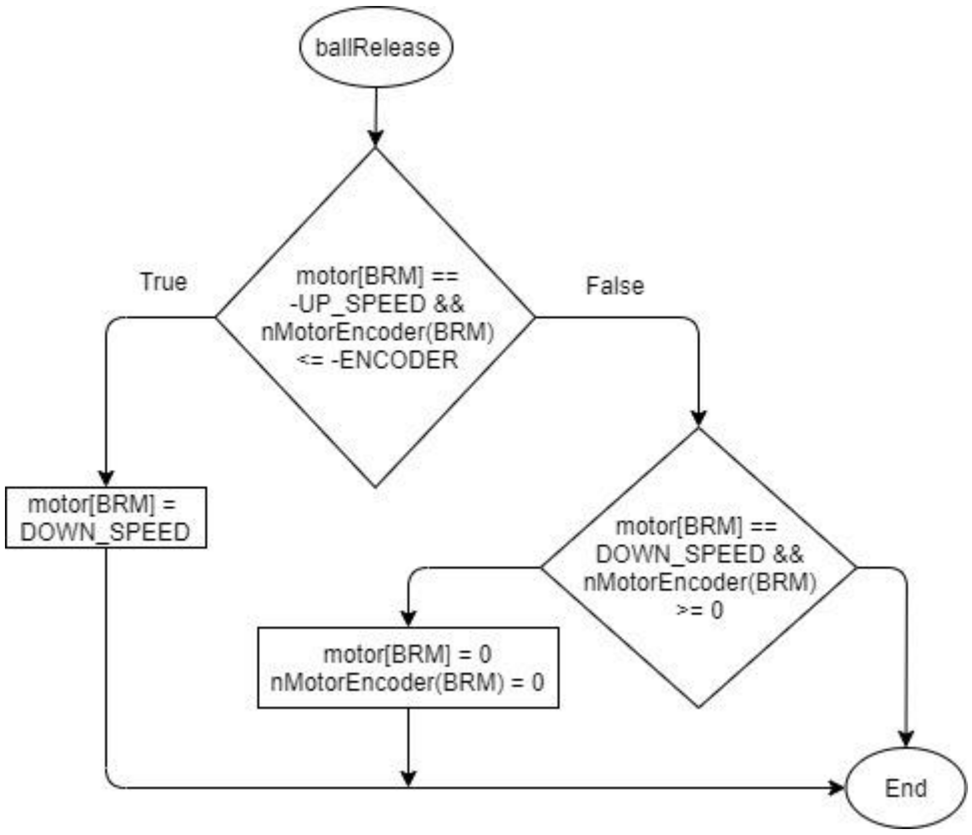

ballRelease
motor[BRM] == -UP_SPEED && nMotorEncoder(BRM) <= -ENCODER
True
False
motor[BRM] = DOWN_SPEED
motor[BRM] == DOWN_SPEED && nMotorEncoder(BRM) >= 0
motor[BRM] = 0
nMotorEncoder(BRM) = 0
End

Figure 19: ballRelease function flowchart

Appendix B6 – setFieldMotors Function Flowchart

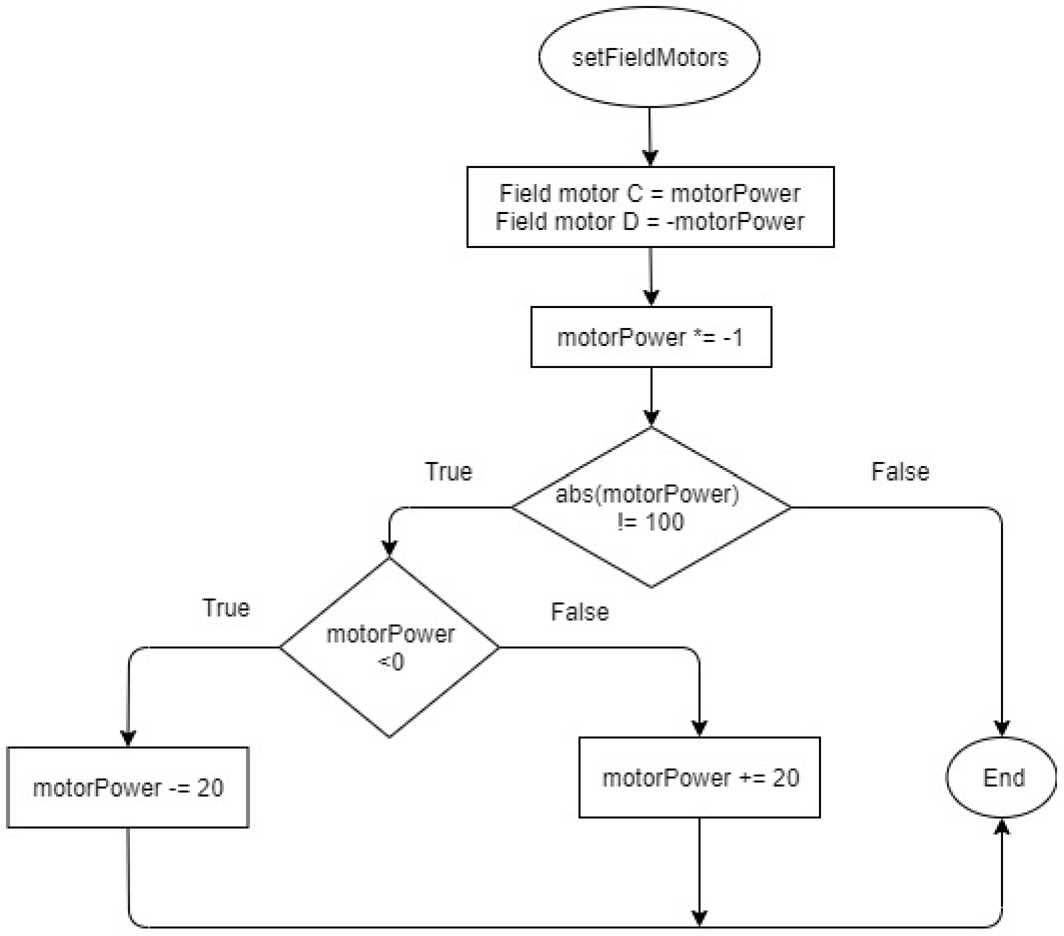


Figure 20: setFieldMotors function flowchart

Appendix B7 – sortScore Function Flowchart

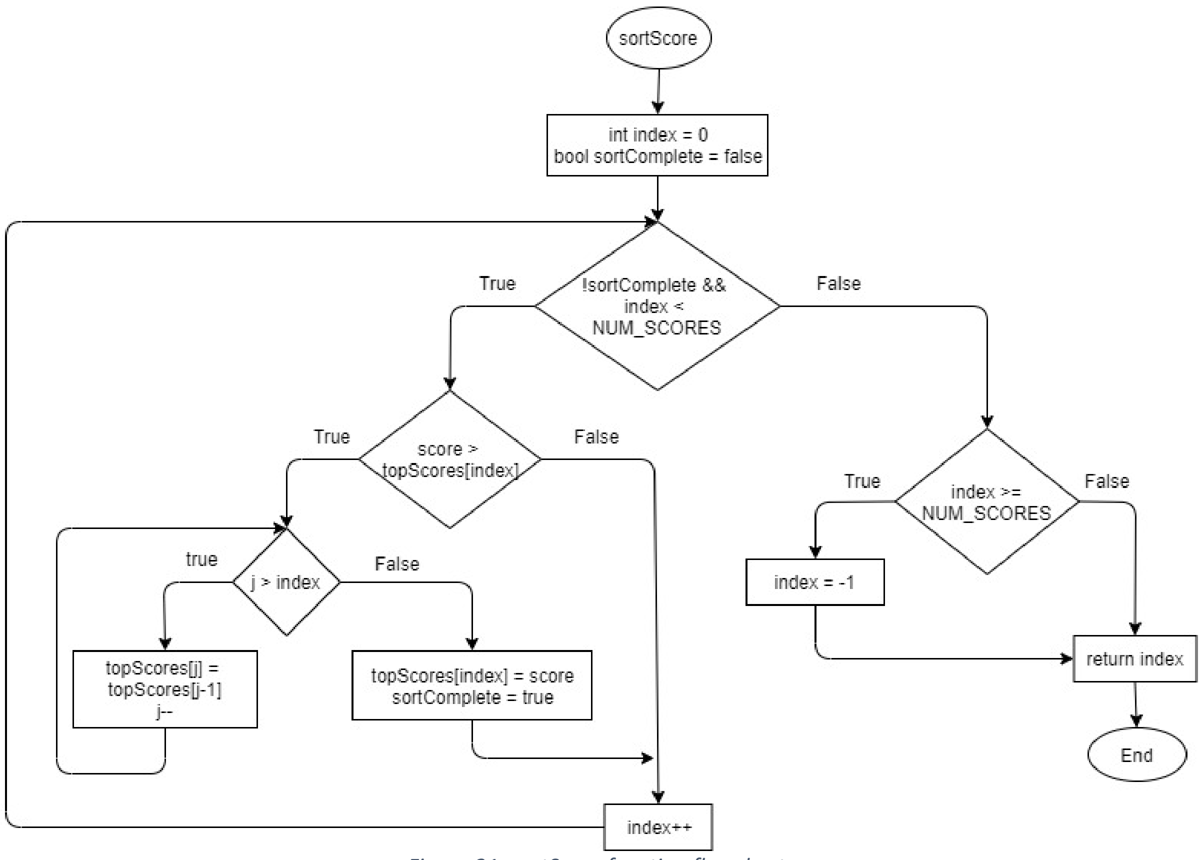


Figure 21: sortScore function flowchart