Mecha Mayhem Final Report

Matthew Garcia, Anthony Gonzalez, Meghan Mancini, Vilasini Nathan

Northeastern University

GE1502: Cornerstone of Engineering II

Professor Joshua Hertz

Introduction

The Boston Children's Museum located in Seaport is dedicated to encouraging children to learn through play, and the museum has requested that Professor Joshua Hertz's sections of Cornerstone II at Northeastern create fun and educational games for a limited time exhibit on Saturday, April 15th. The theme of our section's exhibit is aviation, with the storyline involving aliens that are attacking the city of Boston and each game representing the different teams in our section coming together to defeat those aliens. Our team is the Space Huskies, and we are working together to create a game that will fit all these goals and have our own creative influence on the individual gameplay and theme. When we first started this project, we had a lot of ideas pertaining to what we wanted the children to learn and what we wanted our game to do. After we created our prototype, we were able to assess our work and decide what did and did not work. During the creation of the project itself, we were able to make changes and continually redefine what our project did and what we wanted the players to get out of it. Now, we have a product that fulfills our goals of creating a game that uses interactive mecha parts to answer questions pertaining to robotics. All of these goals were assessed quantifiably based on our own measurements or player assessment.

Problem Statement and Research Questions

First, this report will describe the problem statement and its associated research questions. Based on the development of our game and feedback received from our peers and Professor Hertz, we defined our final problem statement to be "How can we create an educational, fun exhibit where children learn about robotics through picking between a multitude of mecha components for the Boston's Children Museum". We believe this problem statement is interesting because it requires us to create a game that will be able to engage children in choosing between different parts of the mecha and also have the objective of completing the mecha within the allocated time. However, we also have to make sure the prompts provided are educational, while working around the technical complexities that come with connecting arduino code and electronics to a display on a screen. Based on this problem statement, certain research questions about how we wish to find a solution emerge. In this case, questions include, what is the budget? How will the player interact with the screen? How can we make the game both engaging and educational to the players? And how can we develop the theme and aesthetic of this game?

Characteristics of a Successful Solution

From these questions, we were able to identify specific functions that we want our project to perform. These functions include the project being able to combine differing electronic components such as photoresistors and a speaker with the processing software, being able to store certain mecha parts and electronic components in order to make the game compact, control the electronic feedback based on how the user responds to the game, and for the game to generate different text on screen that correlates to the player making some physical change. In addition to this, we were able to develop descriptions of what we wanted our project to be like. These descriptions, or objectives, include the project being transportable so that it can be transferred from campus to the museum, cost-efficient so that the cost is under a set budget, aesthetically-cohesive so the style of the project is in correlation with the overall aesthetic of the section while still having its own distinct style, and finally both educational and fun so the project is able to

provide a memorable and impactful experience for the children. Finally, we were able to identify certain criteria we wished our project to meet. These criteria, or constraints, include the project being able to display a digital game, the photoresistor controlling what appears on screen, and the speaker playing victory music when the player answers a question correctly.

Project Prototype

Based on these characteristics, our team first began to create a prototype game. To start, we created a 6 by 6 inch square box that represented what our actual game would look like. We then added electrical components representing what the electrical components that make up our game would consist of. In this sense, we used a potentiometer and a button to select between different components displayed on an LCD. Based on the option selected, a certain LED would light up, and either victory or defeat music would play. This represented both the components being selected in the game, and the game sending positive or negative feedback based on the component selected.

Game Modifications and Descriptions

In creating our real game, we decided to replace the idea of a points system with the goal of completing the mecha. By doing this, one player goes at a time rather than multiple, and it gives the players a more tangible objective to complete. While obtaining points without any clear reward can get tiring, actually completing something is much more engaging. Also, to make this game more challenging, we also decided to give the player a time limit of a minute. The timer is displayed on the screen, putting players under pressure when deciding what part to select. The game itself consists of 24 questions, with all 6 of the parts each having 4 questions associated with them. If the player gets the question right, they go to a screen stating they are correct; if they get it wrong, they go to a screen stating they are incorrect. After they move on, the screen shows one of the 24 questions again and again until either time runs out or the mecha is assembled.

Physical Components

In terms of physical components, we created a four-sided box with the screen centered towards the back of the station, making it able to adequately display the game to the player. The build was mainly held together with corner brackets, besides the front piece of wood which we used wood glue for. The piece of wood in the back was just slid in and out on top of the computer, giving us easy access to the computer if we need to stop the program. We implemented six photoresistors in the front of the project, each of them corresponding to one of the parts of the mecha. Placing the correct mecha part on the photoresistor correlates to selecting that part as the answer to the question. This makes the game physical in nature, with the players having to interact with the parts of the mecha in order to change the screen. By doing this, we believe the players will be engaged with the game rather than just passively interact with a screen. In addition, two buttons were added to the side of the project, with one of them used for starting the game and moving on to the next question, and the other used for stopping the game

Arduino Code

For the coding of the game, we created two programs: one in arduino and one in processing. In the arduino code, we defined inputs for the photoresistors and buttons, and then the variables correlating to the reading from the photoresistors. Initially, the readings of the photoresistor would correlate to a value sent to the serial monitor. However, this was found to be inadequate, as multiple parts will be placed on the photoresistors at a time. Instead, we calculated the derivative of the photoresistor by creating an array of the past 10 values from the photoresistors. The average value of this array was found, and the derivative was calculated by finding the change between the current reading and the average reading. When the derivative fell below 0, that meant there was some type of change in the light level of the photoresistor, and the code would print the current part being selected to the serial monitor. In addition, each time the start button was pressed, a count variable that was initialized at zero was increased. The current count value was sent to the serial monitor along with the part being selected. That way, the processing code would know what question it was on, and the answer being selected in response to that question.

Processing Code

In the processing code, functions for each question were defined and the timer was set. Then, the screen was initialized and set to the starting screen. Based on the value of the count sent from the arduino code, one of the questions was selected. So if the starting button was pressed at the beginning of the game, the count would increase to 1 and the first question was shown. Then the answer being selected was determined based on the value being sent from the change in the light levels of each of the photoresistors. If the answer is correct, then the correct screen is displayed and the counter for the part being selected is increased. If the answer is incorrect, then the incorrect screen is displayed. This process is repeated until either time runs out, or the counter for each of the parts is above zero, indicating that the player has completed the mecha. In addition, the computer would play music based on if they answered the question correctly or incorrectly, and if they won or lost the game.

Aesthetics

In terms of aesthetics, the project was painted light blue on the bottom, before gradually darkening in correlation with the height of the project. At the top, stars were painted in order to indicate the night sky. The back of the project was not painted, as it will be facing away from the players. The front piece of wood had a white silhouette of the mecha painted in order to guide the players as to where they should place the part currently selected. In addition, each of the mecha parts was painted a different color in order to give the players customizability when creating their mecha. For the display, the starting screen consisted of a background of space, with images of mechas and the title of the game overlaid onto this background. Each of the questions has a background of a space station, and both text and images of each of the parts of the mechas are displayed.

Analysis

Based on the clearly defined functions and objectives we established, we conducted a quantitative analysis to evaluate the performance of our game. One of the parameters we used to determine the success of our project was its transportability. We decided to quantify this aspect by measuring the dimensions of

the object. Our game measured 2 feet in width and 1.5 feet in length and height, which satisfied our transportability criteria as it could easily fit into our arms or a truck. In addition, we established a budget of $120 to ensure that we met the budget requirements. Overall, the total cost of our project was $119.25, coming just under budget. Our team found it challenging to assess the aesthetic appeal of the game, but we were able to do so by measuring the number of elements relating to the game's project-specific aesthetic. From this, we determined that all of our questions related to robotics, and we used six mecha parts.

In terms of data collected during the museum exhibit, we administered a survey to the children, asking whether or not they enjoyed the game after they finished playing. From a survey of twenty children, nineteen out of twenty children enjoyed the game, which helped us determine that our game did fulfill the goal of being entertaining. We also evaluated whether or not the children were able to understand the questions asked. From a total of six questions for each child, on average the children were able to understand 5.71 questions, so we could consider the questions overall appropriate for the children's age level and therefore the game was enjoyable. Finally, we recorded the functionality of the photoresistors based on how many photoresistors worked without any issues for each child. We recorded an average of 5.14 working photoresistors.

In terms of the functions for our projects, we were able to quantify whether the electronic parts could communicate with and make changes to the display in the screen by showing a number being generated from the arduino code being displayed on the console of the processing code. In terms of electronic feedback, we measured the consistency of the player being able to use the mecha parts to answer a question. The photoresistors we used were able to change processing 8 out of 10 times. Unlike the speaker, the value generated in the arduino code was received in processing, but some issues emerged surrounding the way the photoresistors were wired. Finally, for our constraints, the project is able to display a digital game, and the photoresistor is able to control what is on screen. However, the speaker still faces some difficulty playing music based on whether a question is answered correctly or incorrectly due to the difficulty between having both arduino and processing sending values to each other.

Future Work

If we were to continue work on our project in the future, our team would want to move away from processing to a software that is able to display advanced graphics and animation with more versatility, allowing us to randomize our questions for a more enjoyable experience. In addition, we would like to move away from photoresistors, as they proved to be faulty and would often pick up on changes that were not there. By using distance sensors or some other type sensor, we would be able to calculate what part is being selected based on its distance to the sensor, rather than changes in light levels which can be highly variable. Finally, we would rework the game a bit in order to make it less competitive and more focused on pure education. To do this, we would remove the timer and allow the player to keep going if they got a question wrong. We would also add a button that goes to a screen that gives a player a hint if they are unsure.

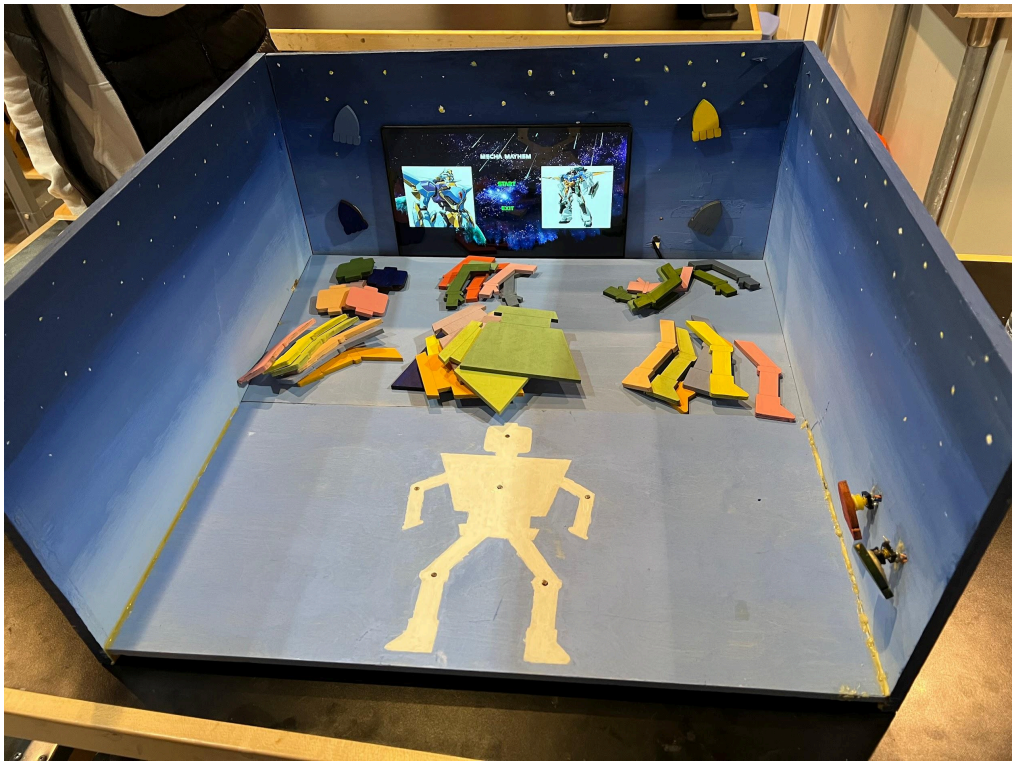
Conclusion

To restate our problem statement, we wished to create an educational game where players utilized mecha parts to become more educated on the subject of robotics while also having fun. To do this, we created a project that used a screen to display questions, and had the player select the part that correlated to the answer of that question. All in all, we believe we were mainly successful in achieving our goals, as

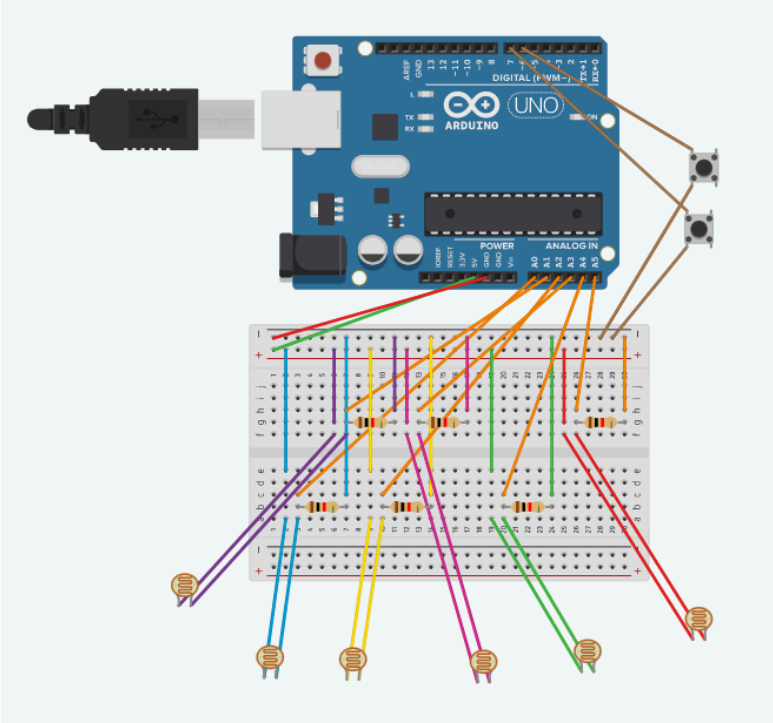
despite some bugs, our project was able to engage the players and have the majority of them take away an educational, fun experience.

Images

Physical Product:



Wiring:



Code

Arduino Code:

```
const int start_button = 7; // Pin for the start button
const int stop_button = 6; // Pin for the stop button
const int resistor0 = A0; // Pin for the first photoresistor (analog)
const int resistor1 = A1; // Pin for the second photoresistor
const int resistor2 = A2; // Pin for the third photoresistor
const int resistor3 = A3; // Pin for the fourth photoresistor
const int resistor4 = A4; // Pin for the fifth photoresistor
const int resistor5 = A5; // Pin for the sixth photoresistor
int lightlevel0; // Integer value of the light level for the first photoresistor
int lightlevel1; // Integer value of the light level for the second photoresistor
int lightlevel2; // Integer value of the light level for the third photoresistor
int lightlevel3; // Integer value of the light level for the fourth photoresistor
int lightlevel4; // Integer value of the light level for the fifth photoresistor
int lightlevel5; // Integer value of the light level for the sixth photoresistor
int start_button_state; // The value of whether or not the start button is pressed
int stop_button_state; // The value of whether or not the stop button is pressed
int count = 0; // The current question

const int numReadings = 10;  // Number of previous values to use
int lightlevel0Readings[numReadings]; // Array to store previous values
int lightlevel0Index = 0;  // Index of the next reading to be added to the array
int lightlevel0Total = 0;  // Running total of previous values

int lightlevel1Readings[numReadings]; // Same as above for the second photoresistor
int lightlevel1Index = 0;
int lightlevel1Total = 0;

int lightlevel2Readings[numReadings]; // Same as above for the third photoresistor
int lightlevel2Index = 0;
int lightlevel2Total = 0;

int lightlevel3Readings[numReadings]; // Same as above for the fourth photoresistor
int lightlevel3Index = 0;
int lightlevel3Total = 0;

int lightlevel4Readings[numReadings]; // Same as above for the fifth photoresistor
int lightlevel4Index = 0;
int lightlevel4Total = 0;

int lightlevel5Readings[numReadings]; // Same as above for the sixth photoresistor
int lightlevel5Index = 0;
int lightlevel5Total = 0;

void setup() {
  pinMode(resistor0, INPUT); // Sets the first photoresistor as an input value
  pinMode(resistor1, INPUT); // Sets the second photoresistor as an input value
  pinMode(resistor2, INPUT); // Sets the third photoresistor as an input value
  pinMode(resistor3, INPUT); // Sets the fourth photoresistor as an input value
  pinMode(resistor4, INPUT); // Sets the firth photoresistor as an input value
  pinMode(resistor5, INPUT); // Sets the sixth photoresistor as an input value
  pinMode(start_button, INPUT_PULLUP); // Sets start button as an input pullup value
  pinMode(stop_button, INPUT_PULLUP); // Sets stop button as an input pullup value
  Serial.begin(9600); // Begins the serial monitor
}

void loop() {
  start_button_state = digitalRead(start_button); // Sets the start button state as the digital read of the start button
  stop_button_state = digitalRead(stop_button); // Sets the stop button state as the digital read of the stop button
  lightlevel0 = analogRead(resistor0); // Sets the light level as the analog read of the first photoresistor
  lightlevel1 = analogRead(resistor1); // Sets the light level as the analog read of the second photoresistor
  lightlevel2 = analogRead(resistor2); // Sets the light level as the analog read of the third photoresistor
  lightlevel3 = analogRead(resistor3); // Sets the light level as the analog read of the fourth photoresistor
  lightlevel4 = analogRead(resistor4); // Sets the light level as the analog read of the fifth photoresistor
  lightlevel5 = analogRead(resistor5); // Sets the light level as the analog read of the sixth photoresistor

  lightlevel0Total = lightlevel0Total - lightlevel0Readings[lightlevel0Index] + lightlevel0; // Adds the current reading to the array and update the running total
  lightlevel0Readings[lightlevel0Index] = lightlevel0;
  lightlevel0Index = (lightlevel0Index + 1) % numReadings;
  int lightlevel0_avg = lightlevel0Total / numReadings; // Calculates the average of the previous readings
  int lightlevel0_diff = lightlevel0 - lightlevel0_avg; // Calculates the derivative based on the average of the previous readings
  float lightlevel0_derivative = (float)lightlevel0_diff / 1000;

  lightlevel1Total = lightlevel1Total - lightlevel1Readings[lightlevel1Index] + lightlevel1; // Same as above for the second photoresistor
  lightlevel1Readings[lightlevel1Index] = lightlevel1;
  lightlevel1Index = (lightlevel1Index + 1) % numReadings;
  int lightlevel1_avg = lightlevel1Total / numReadings;
  int lightlevel1_diff = lightlevel1 - lightlevel1_avg;
  float lightlevel1_derivative = (float)lightlevel1_diff / 1000;

  lightlevel2Total = lightlevel2Total - lightlevel2Readings[lightlevel4Index] + lightlevel2; // Same as above for the third photoresistor
  lightlevel2Readings[lightlevel2Index] = lightlevel2;
  lightlevel2Index = (lightlevel2Index + 1) % numReadings;
  int lightlevel2_avg = lightlevel2Total / numReadings;
  int lightlevel2_diff = lightlevel2 - lightlevel2_avg;
  float lightlevel2_derivative = (float)lightlevel2_diff / 1000;

  lightlevel3Total = lightlevel3Total - lightlevel3Readings[lightlevel3Index] + lightlevel3; // Same as above for the fourth photoresistor
  lightlevel3Readings[lightlevel3Index] = lightlevel3;
  lightlevel3Index = (lightlevel3Index + 1) % numReadings;
  int lightlevel3_avg = lightlevel3Total / numReadings;
  int lightlevel3_diff = lightlevel3 - lightlevel3_avg;
  float lightlevel3_derivative = (float)lightlevel3_diff / 1000;

  lightlevel4Total = lightlevel4Total - lightlevel4Readings[lightlevel4Index] + lightlevel4; // Same as above for the fifth photoresistor
  lightlevel4Readings[lightlevel4Index] = lightlevel4;
  lightlevel4Index = (lightlevel4Index + 1) % numReadings;
  int lightlevel4_avg = lightlevel4Total / numReadings;
  int lightlevel4_diff = lightlevel4 - lightlevel4_avg;
  float lightlevel4_derivative = (float)lightlevel4_diff / 1000;

  lightlevel5Total = lightlevel5Total - lightlevel5Readings[lightlevel5Index] + lightlevel5; // Same as above for the sixth photoresistor
  lightlevel5Readings[lightlevel5Index] = lightlevel5;
  lightlevel5Index = (lightlevel5Index + 1) % numReadings;
  int lightlevel5_avg = lightlevel5Total / numReadings;
  int lightlevel5_diff = lightlevel5 - lightlevel5_avg;
  float lightlevel5_derivative = (float)lightlevel5_diff / 1000;
```

```
  if (start_button_state == 0 ) { // When the start button is pressed
    count++;   // Increase the count by 1 (This tells Processing which Question number we're on)
    Serial.print("Start"); // Prints "Start" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    delay(1000);
  }

  if (stop_button_state == 0) { // When the stop button is pressed
    Serial.print("Stop"); // Prints "Stop" to the serial monitor
    delay(1000);
  }

  if (lightlevel0_derivative < -.01) { // When the derivative of the first light level is under -.01
    Serial.print("Head"); // Prints "Head" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    }

  if (lightlevel1_derivative < -.01) { // When the derivative of the second light level is under -.01
    Serial.print("Body"); // Prints "Body" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    }

  if (lightlevel2_derivative < -.01) { // When the derivative of the third light level is under -.01
    Serial.print("Right Arm"); // Prints "Right Arm" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    }

  if (lightlevel3_derivative < -.01) { // When the derivative of the fourth light level is under -.01
    Serial.print("Right Leg"); // Prints "Right Leg" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    }

  if (lightlevel4_derivative < -.001) { // When the derivative of the fifth light level is under -.001
    Serial.print("Left Arm"); // Prints "Left Arm" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    }

  if (lightlevel5_derivative < -.001) { // When the derivative of the sixth light level is under -.001
    Serial.print("Left Leg"); // Prints "Left Leg" to the serial monitor
    Serial.println(count); // Prints the current question number to the serial monitor
    }
}
```

Processing Display Code:

```
import processing.serial.*; // Allows processing to eastablish contact with the serial monitor
Serial myPort;
import processing.sound.*; // Allows processing to import sound files
SoundFile correct; // Creates a sound file that plays when the correct answer is chosen
SoundFile incorrect; // Creates a sound file that plays when the incorrect answer is chosen
SoundFile victory; // Creates a sound file that plays when the player wins
SoundFile failure; // Creates a sound file that plays when the player loses
String val = ""; // Creates a string
boolean firstContact = false;
int n1 = 0; // Counters for the answer screens of the questions
int n2 = 0;
int n3 = 0;
int n4 = 0;
int n5 = 0;
int n6 = 0;
int n7 = 0;
int n8 = 0;
int n9 = 0;
int n10 = 0;
int n11 = 0;
int n12 = 0;
int n13 = 0;
int n14 = 0;
int n15 = 0;
int n16 = 0;
int n17 = 0;
int n18 = 0;
int n19 = 0;
int n20 = 0;
int n21 = 0;
int n22 = 0;
int n23 = 0;
```

```
int n24 = 0;
int GUIn = 0; // Counter for the starting screen
int q1n = 0; // Counters for the question screens
int q2n = 0;
int q3n = 0;
int q4n = 0;
int q5n = 0;
int q6n = 0;
int q7n = 0;
int q8n = 0;
int q9n = 0;
int q10n = 0;
int q11n = 0;
int q12n = 0;
int q13n = 0;
int q14n = 0;
int q15n = 0;
int q16n = 0;
int q17n = 0;
int q18n = 0;
int q19n = 0;
int q20n = 0;
int q21n = 0;
int q22n = 0;
int q23n = 0;
int q24n = 0;
int h_counter = 0; // Counter for the head
int b_counter = 0; // Counter for the body
int la_counter = 0; // Counter for the left arm
int ra_counter = 0; // Counter for the right arm
int ll_counter = 0; // Counter for the left leg
int rl_counter = 0; // Counter for the right leg
```

```
Timer timer; // Initializes the timer
int currentTime; // Creates an integer value for the current time
PFont scifiFont; // Creates the font used
PImage head; // Creates an image for the head
PImage left_arm; // Creates an image for the left arm
PImage body; // Creates an image for the body
PImage right_arm; // Creates an image for the right arm
PImage left_leg; // Creates an image for the left leg
PImage right_leg; // Creates an image for the right leg
PImage mecha1; // Creates an image for one of the mechas
PImage mecha2; // Creates an image for one of the mechas
PImage mecha3; // Creates an image for one of the mechas
PImage mecha4; // Creates an image for one of the mechas
PImage space; // Creates an image of space
PImage station; // Creates an image of a space station

void setup() {
  String portName = Serial.list()[0]; // Sets the serial monitor to the one on my computer
  myPort = new Serial(this, "COM3", 9600); // Starts the serial monitor
  myPort.bufferUntil('\n'); // Doesn't work until new line
  timer = new Timer(1000); // Sets the time interval
  fullScreen(); // Sets the display to the full screen of the computer
  fill(0);
  scifiFont = createFont("Space Crusaders.otf", 30); // Loads the font
  head = loadImage("Head.jpg"); // Loads the image of the head
  left_arm = loadImage("Left Arm.jpg"); // Loads the image of the left arm
  body = loadImage("Body.jpg"); // Loads the image of the body
  right_arm = loadImage("Right Arm.jpg"); // Loads the image of the right arm
  left_leg = loadImage("Left Leg.jpg"); // Loads the image of the left leg
  right_leg = loadImage("Right Leg.jpg"); // Loads the image of the right leg
  mecha1 = loadImage("Mecha1.jpg"); // Loads the image of one of the mechas
  mecha2 = loadImage("Mecha2.jpeg"); // Loads the image of one of the mechas
  mecha3 = loadImage("Mecha3.jpg"); // Loads the image of one of the mechas
  mecha4 = loadImage("Mecha4.jpg"); // Loads the image of one of the mechas
  space = loadImage("Space.jpg"); // Loads the image of space
  station = loadImage("Station.jpg"); // Loads the image of the space station
  correct = new SoundFile(this, "level-up-sound-effect.mp3"); // Loads the sound played when the player gets a question correct
  incorrect = new SoundFile(this, "computer-fail-sound-effect.mp3"); // Loads the sound played when the player gets a question incorrect
  victory = new SoundFile(this, "level-fail-sound-effect.mp3"); // Loads the sound played when the player win
  failure = new SoundFile(this, "win-sound-effect.mp3"); // Loads the sound played when the player loses
}

void serialEvent(Serial myPort) {
  val = myPort.readStringUntil('\n'); // The value is what is printed to the console until there is a line break
}

void draw() {
  println(val); // Prints the value out in the console
  textFont(scifiFont, 50); // Sets the text font to the one loaded earlier
  //text(mouseX/width + "    " + mouseY/height, mouseX/width, mouseY/height); // Used to find coordinates of mouse
  if (timer.complete() == true) { // When a second has passed
    currentTime++; // Increase the time
    timer.start(); // Start the time
  }
  if (GUIn == 0) { // When the GUI counter is 0
    show_GUI(); // Calls the starting screen function
  }
  if (val == null) { // If nothing is printed to the console (never should happen)
    background(0, 0, 0);
  }
  else { // If something is always printed to the console (should always be happening)
    if (val.contains("Start") && q1n == 0) { // If the starting button is pressed for the first time and the question 1 counter is 0
      GUIn++; // Increases the GUI counter so the starting screen no longer displays
      q1(); // Calls the first question function
    }
    if (val.contains("1") && n1 == 0) { // If the count is 1 (starting button is pressed for the first time) and the answer 1 counter is 0
      if (val.contains("Head")) { // If the head is pressed down
        h_counter++; // The head counter is increased
        q1n++; // Increases the first question counter so the first question screen no longer displays
        correct(); // Calls the correct function
        correct.play(); // Plays the correct music
        delay(2000);
        correct.stop();
      }
      if (val.contains("Body") || val.contains("Left Arm") || val.contains("Right Arm") || val.contains("Left Leg") || val.contains("Right Leg")) { // If any part that is
        incorrect(); // Calls the incorrect function
        incorrect.play(); // Plays the incorrect music
        delay(1000);
        incorrect.stop();
        q1n++; // Increases the first question counter so the first question screen no longer displays
      }
    }
    if (val.contains("2") && n2 == 0) { // Does the same as above for each succesive question
      if (val.contains("Start") && q2n == 0) {
        q2();
      }
      if (val.contains("Body")) {
        b_counter++;
        q2n++;
        correct();
        correct.play();
        delay(2000);
```

```
      n8++;
      n9++;
      n10++;
      n11++;
      n12++;
      n13++;
      n14++;
      n15++;
      n16++;
      n17++;
      n18++;
      n19++;
      n20++;
      n21++;
      n22++;
      n23++;
      n24++;
      complete(); // Calls the complete function
      victory.play(); // Plays the victory music
      delay(2000);
      victory.stop();
    }
    if (currentTime >= 60) { // If the time exceeds one minute
      q1n++; // Increases the value of all of the question counters so no question screen is displayed
      q2n++;
      q3n++;
      q4n++;
      q5n++;
      q6n++;
      q7n++;
      q8n++;
      q9n++;
      q10n++;
      q11n++;
      q12n++;
      q13n++;
      q14n++;
      q15n++;
      q16n++;
      q17n++;
      q18n++;
      q19n++;
      q20n++;
      q21n++;
      q22n++;
      q23n++;
      q24n++;
      n1++; // Increases the value of all of the answer counters so no answer screen is displayed
      n2++;
      n3++;
      n4++;
      n5++;
      n6++;
      n7++;
      n8++;
      n9++;
      n10++;
      n11++;
      n12++;
      n13++;
      n14++;
      n15++;
      n16++;
      n17++;
      n18++;
      n19++;
      n20++;
      n21++;
      n22++;
      n23++;
      n24++;
      incomplete(); // Calls the incomplete function
      failure.play(); // Plays the failure music
      delay(2000);
      failure.stop();
    }
    if (val.contains("Stop")) { // If the stop button is pressed
      exit(); // Stop the game
    }
  }
}
```

```
void show_GUI() { // Is shown is the GUI function is called
  background(space); // Sets the background to space
  fill(255, 255, 255);
  textAlign(CENTER);
  text("Mecha Mayhem", width/2, (height*.1875)); // Creates text of the title of the game and sets its cooridnates
  fill(51, 229, 49);
  textAlign(CENTER);
  text("Start", width/2, (height*.4));  // Creates "Start" text and sets its coordinates
  image(mecha4, width*.05, height*.25, width*.3, height*.5); // Imports one of the mecha images and sets its coordinates
  image(mecha1, width*.65, height*.25, width*.3, height*.5); // Imports one of the mecha images and sets its coordinates
  textAlign(CENTER);
  text("Exit", width/2, (height*.6)); // Creates "Exit" text and sets its coordinates
}

void q1() { // Is shown if the first question function is called
  background(station); // Sets the background to the space station
  fill(255, 255, 255);
  textAlign(CENTER);
  text("Which part of the robot helps it see?" + " " + currentTime, width/2, (height*.15)); // Creates text of the question and the current time and sets its coordinates
  fill(51, 229, 49);
  text("Head", width/2, height*.2); // Creates "Head" text and sets its coordinates
  image(head, width*.4, height*.25, width*.2, height*.2); // Imports the head image and sets it right below the "Head" text
  text("Body", width/2, height*.5); // Creates "Body" text and sets its coordinates
  image(body, width*.4, height*.55, width*.2, height*.2); // Imports the body image and sets it right below the "Body" text
  text("Left Arm", width/4, height*.5); // Creates "Left Arm" text and sets its coordinates
  image(left_arm, width*.15, height*.55, width*.2, height*.2); // Imports the left arm image and sets it right below the "Left Arm" text
  text("Right Arm", 3*width/4, height*.5); // Creates "Right Arm" text and sets its coordinates
  image(right_arm, width*.65, height*.55, width*.2, height*.2); // Imports the right arm image and sets it right below the "Right Arm" text
  text("Left Leg", width/4, height*.8); // Creates "Left Leg" text and sets its coordinates
  image(left_leg, width*.15, height*.85, width*.2, height*.15); // Imports the left leg image and sets it right below the "Left Leg" text
  text("Right Arm", 3*width/4, height*.8); // Creates "Right Leg" text and sets its coordinates
  image(right_leg, width*.65, height*.85, width*.2, height*.15); // Imports the right leg image and sets it right below the "Right Leg" text

void correct() { // Is shown if the correct function is called
  background(station); // Sets the background to the space station
  fill(255, 255, 255);
  textAlign(CENTER);
  text("That's correct, good job" + " " + currentTime, width/2, height/2); // Creates text telling the player they are correct as well as their current time and sets its c
}

void incorrect() { // Is shown if the incorrect function is called
  background(station); // Sets the background to the space station
  fill(255, 255, 255);
  textAlign(CENTER);
  text("That's not quite right, try again" + " " + currentTime, width/2, height/2); // Creates text telling the player they are incorrect as well as their current time anc
}

void complete() { // Is shown if the complete function is called
  background(space); // Sets the background to space
  fill(255, 255, 255);
  textAlign(CENTER);
  text("Congratulations, you completed the mecha!", width/2, (height*.1875)); // Congratulates the player and sets the coordinates of the text
  text("Your game code is GPGJ", width/2, (height*.25)); // Tells they player their game code and sets the coordinates of the text
  image(mecha2, width*.25, height*.4, width*.5, height*.5); // Imports one of the mecha images and sets its coordinates
}

void incomplete() { // Is shown if the incomplete function is called
  background(space); // Sets the background to space
  fill(255, 255, 255);
  textAlign(CENTER);
  text("Too bad, time ran out!", width/2, (height*.1875)); // Tells the player they lost and sets the coordinates of the text
  text("Try again", width/2, (height*.25)); // Tells the player to try again and sets the coordinates of the text
  image(mecha3, width*.25, height*.4, width*.5, height*.5); // Imports one of the mecha images and sets its coordinates
}
```

Processing Timer Class:

```
class Timer { // Creates the class
  int startTime; // Creates an integer value for the start time
  int interval; // Creates an integer value for the time interval
  Timer(int timeInterval) { // The interval is set in the display code
    interval = timeInterval; // Sets interval defined in the display code equivalent to the time interval
  }
  void start() { // Creates a start function
    startTime = millis(); // Makes the start time the current time when the function is called
  }
  boolean complete() { // Creates a boolean complete function
    int elapsedTime = millis() - startTime; // Sets the elapsed time eguivalent to the current time minus the starting time
    if (elapsedTime >= interval) { // If the elapsed time is greater than the time interval
      return true;
    }
    else { // If the elapsed time is not greater than the time interval
      return false;
    }
  }
}
```