Project Analysis of my Robotics Assignment

DUE: Wednesday 14th of June, 2023
Word Count:

Summary:

Over the past term of this robotics project, our class has been working on the fundamentals of wiring an Electronic Board for Arduino / Arduino Megas, learning the native language of Arduino, and learning the fundamentals of finding Inputs and Outputs. For my assignment I was given a board with an Arduino Mega. Despite the obstacles that have occurred throughout the term, namely teacher sickness, difficulty with setting up wiring on the boards and a new coding language taught very briefly, this project report summarises the term of our project learning, and what the final result turned out to be.

Initial theme:

Upon beginning the project, every student in the class was set to create a ‘Smart Device’, a device that could be automated and do some small task we have in everyday society. Examples of these automated things would be a Smart Fridge, Smart Car, Smart Door, etc.

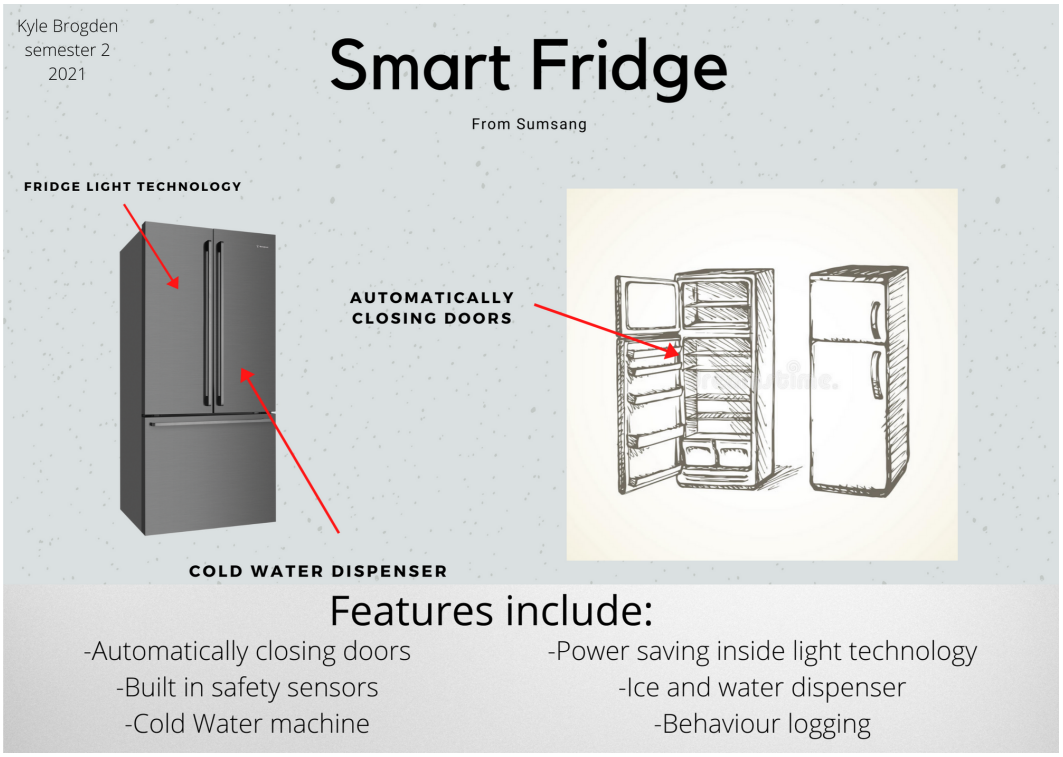


Figure 1. Kyle Brogden’s “Smart Fridge” Banner from Subject Website.

Figure 2. Jackson Dickie’s “Smart Coffee Machine” Banner from Subject Website.

My initial thought of what my project would be was a video game that could be affected by the inputs from the Arduino Board. My ‘banner’ does not showcase the wiring of the project, but it does showcase a few major inputs of the board and how they will impact the game on the monitor.

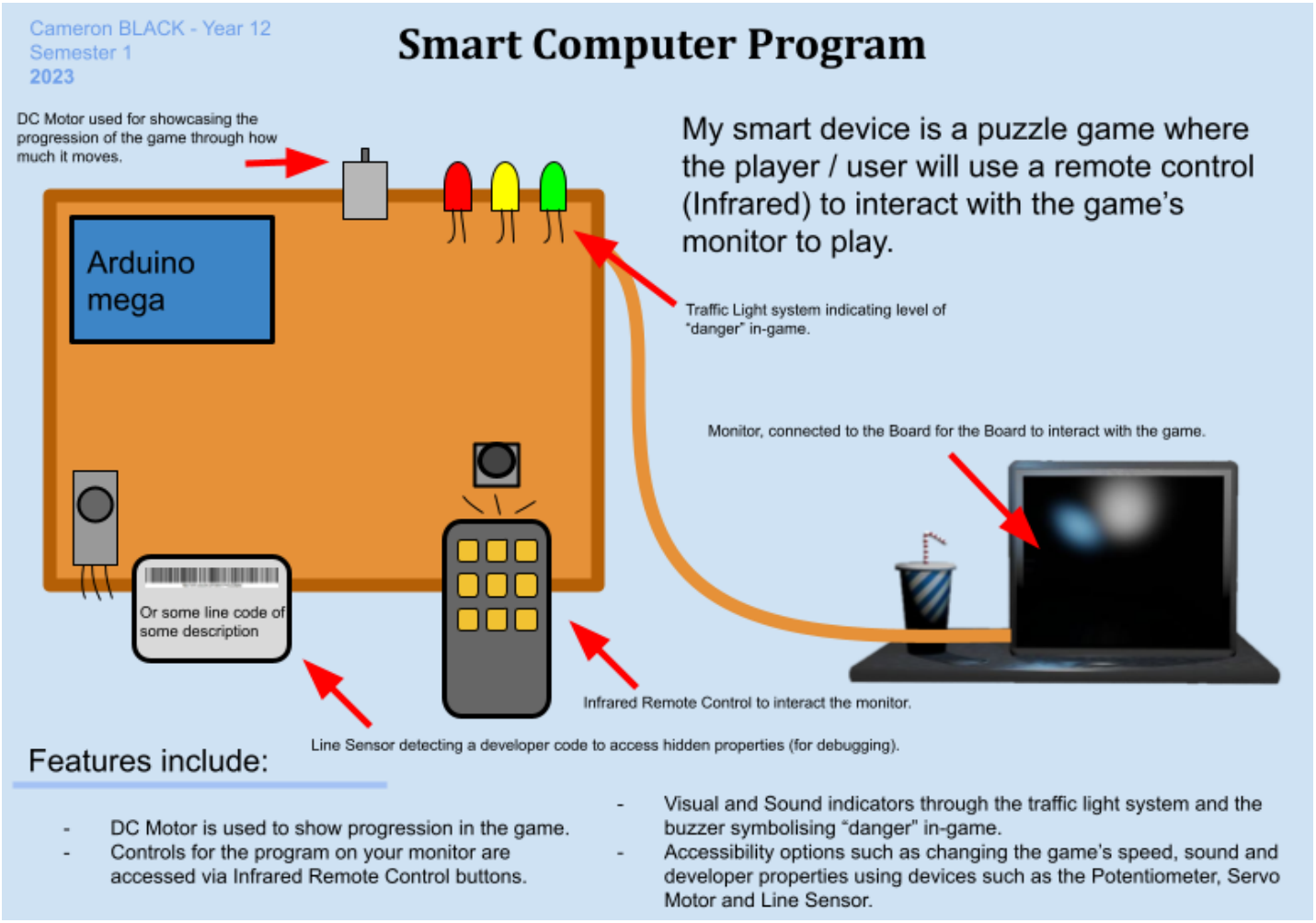


Figure 3. My banner for my ‘Smart Computer Program’.

Diagrams of devices:

Learning about how every device works, we have been tasked with using as many devices as we can for our project and setting all of them up with each of their own useful functions. These functions are called ‘behaviours’, displaying how they interact with each other.

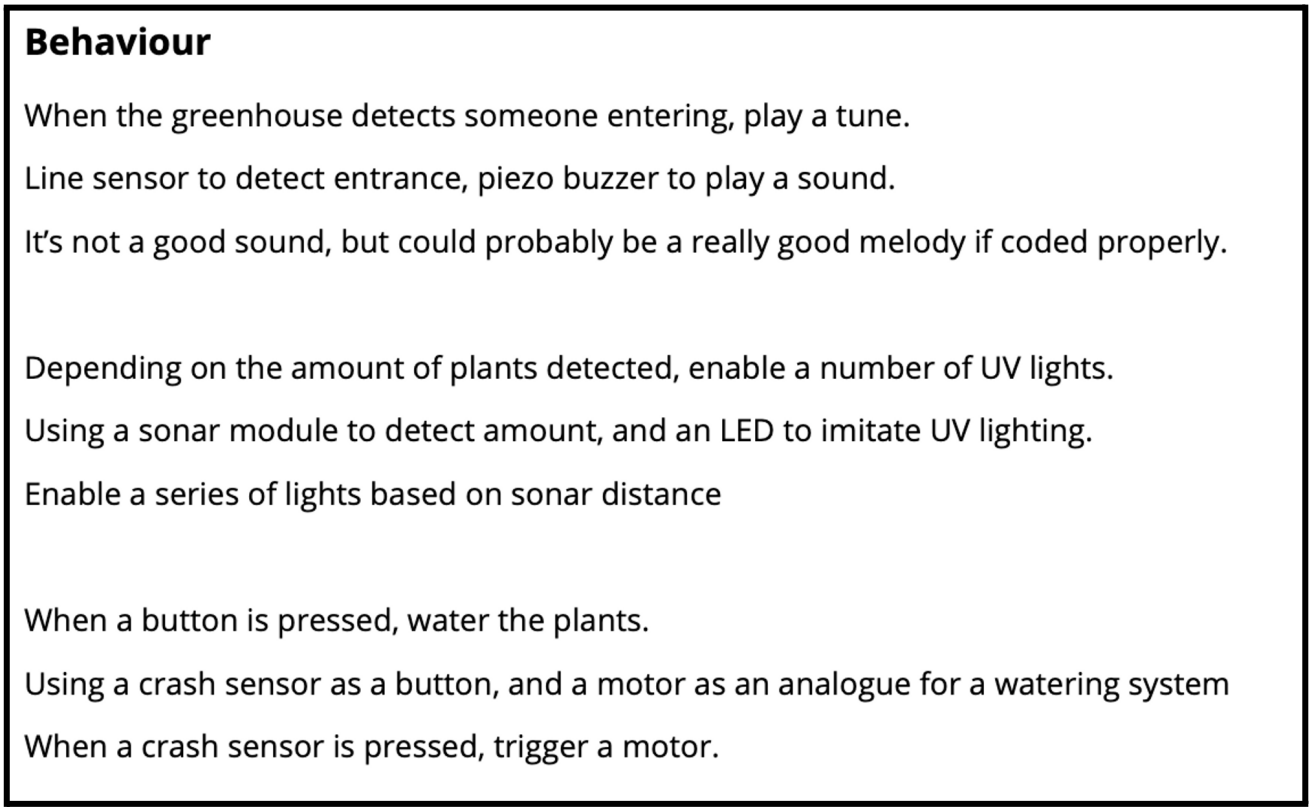


Figure 4. An example of a 'behaviour', a device's input and output connecting to the board. Taken from the Subject Website.

Alongside short sentence responses to summarise the behaviours all of our boards should have to explain each device's use, flowcharts have also been used to describe the process of each device's behaviour. Below are a few flowchart diagrams and descriptions of my project's behaviours.

All of the flowcharts were made using an extension to Visual Studio Code called 'Mermaid'. Which can be found at this link: https://mermaid.js.org/

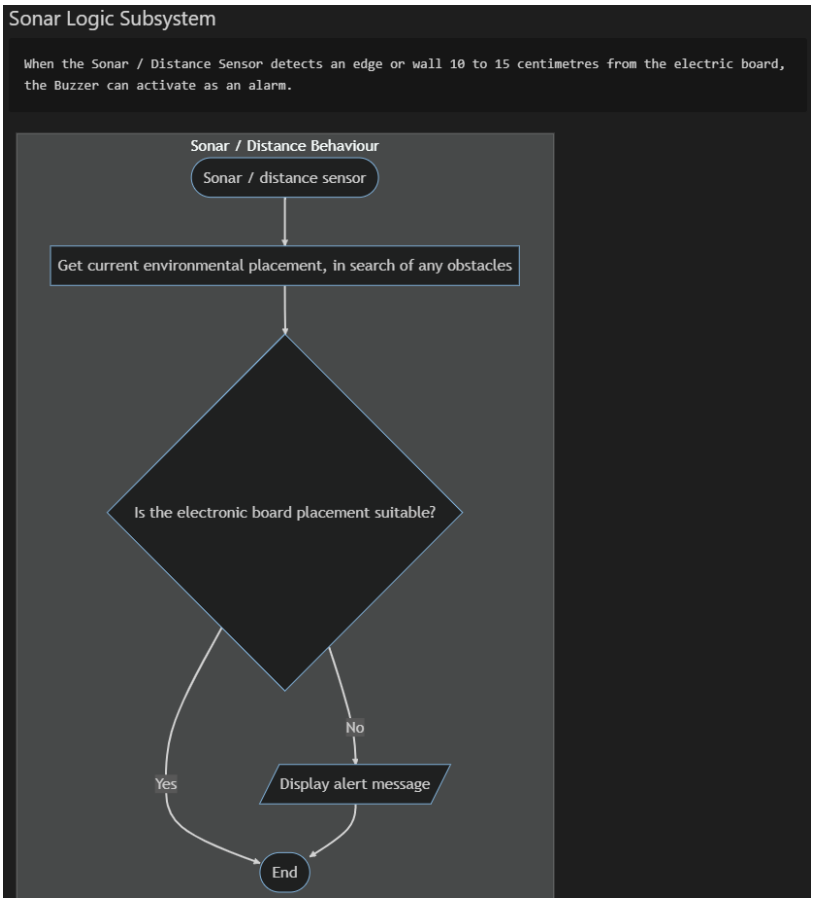


Figure 5. Flowchart Diagram of my ‘Sonar Device Subsystem’.

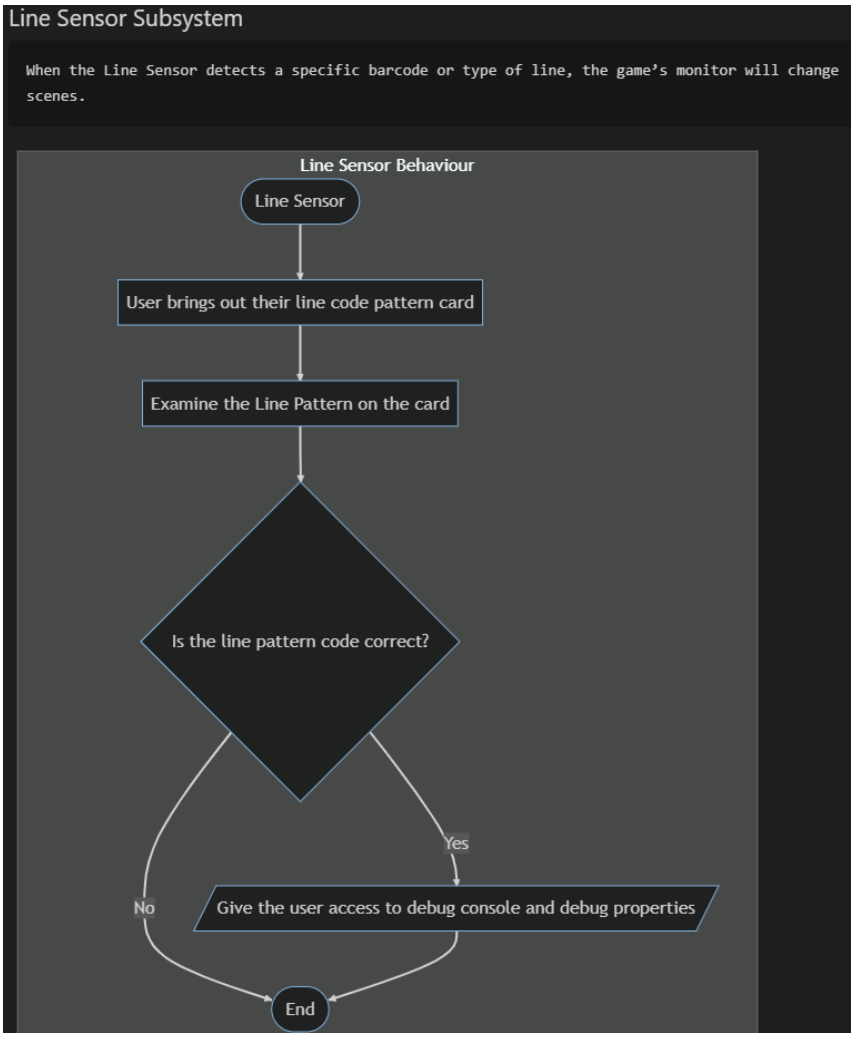


Figure 6. Flowchart Diagram of my ‘Line Sensor Subsystem’.

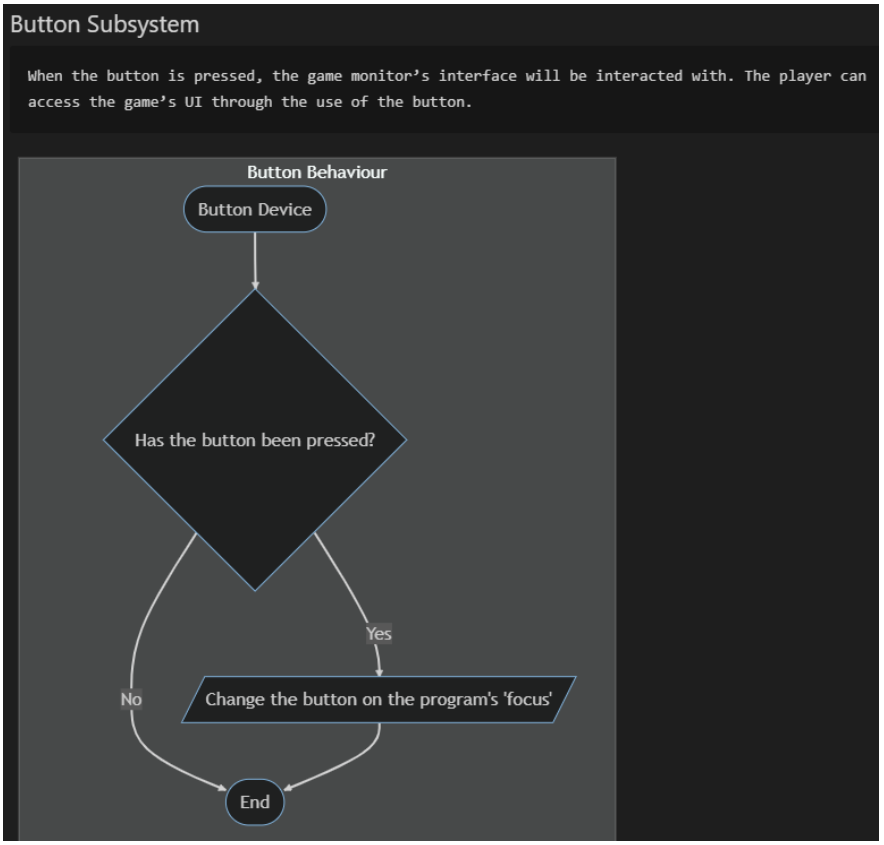


Figure 7. Flowchart Diagram of my 'Button Device Subsystem'.

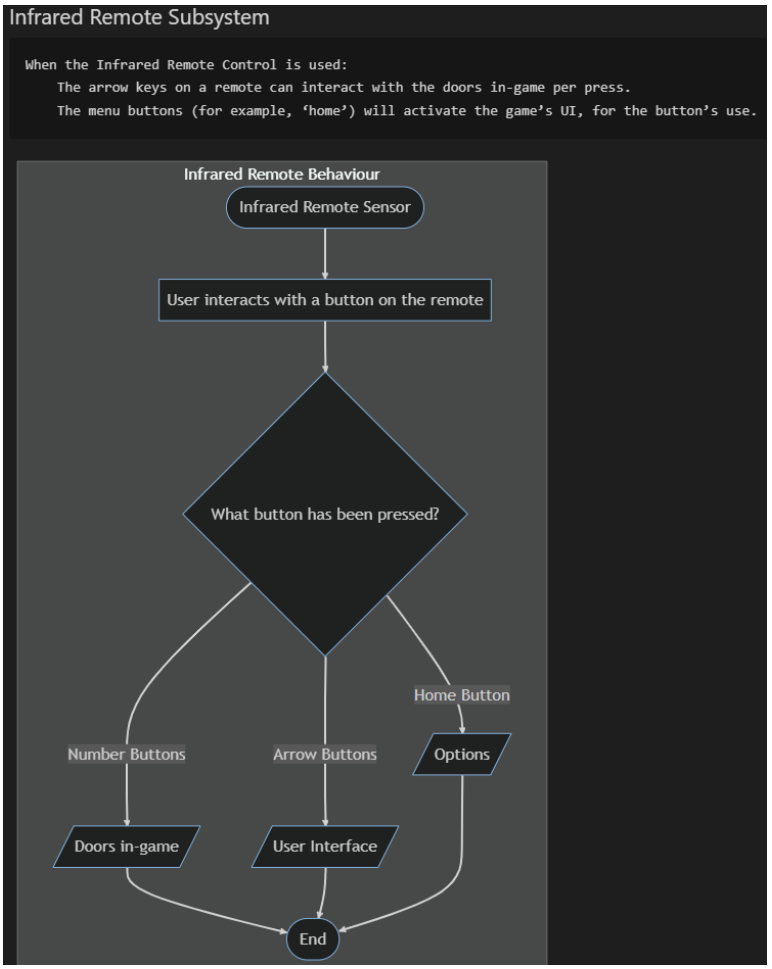


Figure 8. Flowchart Diagram of my 'Infrared Remote Subsystem'.

The language of Arduino:

The native coding language that Arduino uses is C#, and requires a direct connection to the Arduino Board for the code to work. The baud rate is set to 9600 bits per second pre-set with the project file. A major part of the assignment has been the SD Card code which had come pre-set with the project file, set up to ‘log’ the functions’ results in the Serial Monitor for debugging purposes, and save them directly to the SD Card.

SmartDevice.ino commonCode.ino

```
1  void logEvent(String dataToLog) {
2    /*
3      Log entries to a file on an SD card.
4    */
5    // Get the updated/current time
6    DateTime rightNow = rtc.now();
7
8    // Open the log file
9    File logFile = SD.open("events.csv", FILE_WRITE);
10   if (!logFile) {
11     Serial.print("Couldn't create log file");
12     abort();
13   }
```

```
// Log the event with the date, time and data
logFile.print(rightNow.year(), DEC);
logFile.print(",");
logFile.print(rightNow.month(), DEC);
logFile.print(",");
logFile.print(rightNow.day(), DEC);
logFile.print(",");
logFile.print(rightNow.hour(), DEC);
logFile.print(",");
logFile.print(rightNow.minute(), DEC);
logFile.print(",");
logFile.print(rightNow.second(), DEC);
logFile.print(",");
logFile.print(dataToLog);
```

```
30    // End the line with a return character
31    logFile.println();
32    logFile.close();
33    Serial.print("Event Logged: ");
34    Serial.print(rightNow.year(), DEC);
35    Serial.print(",");
36    Serial.print(rightNow.month(), DEC);
37    Serial.print(",");
38    Serial.print(rightNow.day(), DEC);
39    Serial.print(",");
40    Serial.print(rightNow.hour(), DEC);
41    Serial.print(",");
42    Serial.print(rightNow.minute(), DEC);
43    Serial.print(",");
44    Serial.print(rightNow.second(), DEC);
45    Serial.print(",");
46    Serial.println(dataToLog);
47  }
```

Figure 9. The “common code” portion of the Assignment project.

Over the next few weeks from the beginning of Term 2, the class was set towards just implementing the wiring of their own boards, and the foundation for the functions to be made for each device.

Figure 10. Picture of the setup code for each device taken from the Subject Website.

Because of how the setup code has been given to the class, the class had not learned how to expertly use the language C#, but instead have succeeded in learning how to find Inputs and Outputs in substituting code to achieve their goals.

For example, an excerpt from my own project, which uses the Infrared Remote to log what button has been pressed on the remote through a specific code that goes to the Infrared Sensor for each type of button pressed.

```
void handleReceivedTinyIRData(uint16_t aAddress, uint8_t aCommand, bool isRepeat) {
  logEvent("Infrared Code received: " + aCommand);
  if (aCommand == 70) {
    logEvent("IR Command - Up Pressed - Light Off");
    digitalWrite(ledRed, HIGH);
  }
  if (aCommand == 21) {
    logEvent("IR Command - Down Pressed - Light Off");
    digitalWrite(ledRed, LOW);
  }
  if (aCommand == 68) {
    Serial.println("Left");
  }
  if (aCommand == 67) {
    Serial.println("Right");
  }
}
```

Figure 11. “handleRecievedTinyIRData” function from my Arduino project.

Methods and what they do (Inputs and Outputs):

Line Sensor

How the Line Sensor works within my project (according to the initial theme) is: When the Line Sensor detects a specific barcode or type of line (in this instance, a developer card or pattern), the game’s monitor will change scenes to assist in debugging.

The input for this device is the pattern/code that the user will show to the sensor for it to fulfil its function, and the output it exports is the access it grants the user to enable debugging properties and developer scenes. This output does correspond with other functions.

```
110  void lineSensorDebugMode() {
111    /*
112    when the line sensor interacts with a pattern of lines, the sensor will determine if the user can access debug properties of the program.
113    @params none
114    @return lineSensorValue
115  */
116    int lineSensorValue = digitalRead(lineSensorPin);
117
118    // Unfortunately due to time restraints, this function is unfinished, and does not provide any Output or Input.
119  }
```

Figure 12. Current build ‘state of code’ for the Line Sensor function.

Servo Motor

How the Servo Motor works within my project (according to the initial theme) is: When the Potentiometer's state has changed, and the Line Sensor Debug Value is initialised, the motor can change the overall speed of executing the program.

The input for this device roots back to the potentiometer, as you will need to change the value of that device to change the value of this device. The output of this device will be the change in volume of the monitor's program, which does not correspond with other functions.

```
void servoMotorMonitorSpeed() {
  /*
  When the potentiometer is changedwhen the servo motor's state is changed, the speed of the program on the monitor will increase or decrease accordingly.
  @params Potentiometer state, line Sensor 'Debug' Priveleges.
  @return none
*/
  int servoPos = 100;
  myservo.write(servoPos);

  // Unfortunately due to time restraints, this function is unfinished, and does not provide any Output or Input.
}
```

Figure 13. Current build 'state of code' for the Servo Motor function.

DC Motor

How the DC Motor works within my project (according to the initial theme) is: When the program progresses, the DC Motor moves a small clock hand to show the in-game passage of time.

There is no input from the user for this device, as the device depends on the program's initialisation to operate. The output of this device will be the engine movement of the DC Motor, which does not correspond with other functions.

```
void ingameProgressionThroughDCMotorMovement() {
  /*
  over time when the game is playing, the DC motor will move as a visual aid for the player on their progress. dependent on the elapsed time in the game, the difficulty may increase or "dangers" may appear.
  @params none
  @return none
*/
  motor.forward();
  delay(1000);
  motor.stop();
  delay(1000);
  motor.backward();
  delay(1000);

  // Unfortunately due to resource difficulties, this function is unfinished, and does not provide any Output or Input.
}
```

Figure 14. Current build 'state of code' for the DC Motor function.

Potentiometer

How the Potentiometer works within my project (according to the initial theme) is: When the potentiometer's state is changed (angle is changed), and debug privileges have been granted from the Line Sensor, the Potentiometer will move the Servo Motor, adjusting the master volume of the program.

The input for this device is the state change of the Potentiometer. The output of this device will be the volume adjustment of the monitor's program.

```
void potentiometerVolumeAdjust() {
  /*
  when the potentiometer's state is changed, the volume of the program on the monitor will increase or decrease accordingly.
  @params line Sensor 'Debug Privileges
  @return Potentiometer state
*/
  int potValue = analogRead(pot);
  //Serial.println(potValue);
  delay(500);

  // Unfortunately due to time restraints, this function is unfinished, and does not provide any Output or Input.
}
```

Figure 15. Current build ‘state of code’ for the Potentiometer function.

Traffic Light System

How the Traffic Light works within my project (according to the initial theme) is: When there is a ‘dangerous’ element in the program’s game, a corresponding LED will turn on and off. There is no user input for this system, as it is reliant on the program. The output for this system is the initialised glow of each LED from the program.

```
void trafficLightVisualDangerSystem() {
  /*
  when the program receives "danger" variables from ingame "danger", the colour of the corresponding LED will turn on (red = high danger, yellow = medium danger, green = low danger
  @params none
  @return none

  // Unfortunately due to technical difficulties, this function is unfinished, and does not provide any Output or Input.
*/
}
```

Figure 16. Current build ‘state of code’ for the Traffic Light System.

Button (Crash Sensor)

How the Crash Sensor works within my project (according to the initial theme) is: When the button is pressed, the ‘focus’ in the program changes (in which the focus is what button is highlighted at a given time). The input of the function is the crash sensor (button) and the output for this function is the change in the monitor’s user interface ‘focus’.

```
void userInterfaceButton() {
  /*
  when the 'crash sensor' button is pressed, the user interface will progress to other options (and while in-game, will enable the pause menu/options)
  @params none
  @return -
*/
  int crashSensorValue = digitalRead(crashSensor);
  if (crashSensorValue == HIGH) { // Detects when the light of the Crash Sensor's LED is low (off) or high (on).
    //Serial.println(crashSensorValue);
    logEvent("Button Activated"); // When the light of the crash sensor is on (the button is pressed), log the event and say "Button Activated"

    delay(1000);
    logEvent("Button Deactivated"); //After one second, say that the button has been deactivated.
  }

  // Unfortunately due to resource difficulties, this function is unfinished, and does not provide a consistent Output.
}
```

Figure 17. Current build ‘state of code’ for the Crash Sensor System.

Infrared Controller

How the Infrared Remote Controller works within my project (according to the initial theme) is: When a specific button is pressed from the remote (numbers, arrows, etc), an input will go to the program, impacting an in-game element or mechanic.

The input for this function will be from the IR remote controller, and the output will be from the program's player inputs.

```
void infraredRemoteControllerInput() {
  /*
  when the remote's (connected to the electronic board) presses specific buttons (inputs, ie. button 1,2,3,FUNC/STOP,VOLUME), specific outputs will occur.
  @params none
  @return infrared Remote Player Inputs
*/

  int c = remote.listen(1);  // waits 1 second before timing out!
  //int c = remote.listen();  // Without a number, it means 'wait forever'
  if (c >= 0) { // The variable 'c' is the number of the code recieved from a button pressed of an IR remote.

    switch (c) {
      // Top keys
    case 70: // case "70" means that if the code that comes through the IR Remote is 70, then log the event as UP, as 70 is the code recieved when the remote's UP button is pressed.
      logEvent("UP");
      break;
    case 21: // Similarly to case "70", case "21" logs the event as DOWN when the DOWN button on the IR remote is pressed.
      Serial.println("DOWN");
      break;
    case 68:
      Serial.println("LEFT");
      break;
    case 67:
      Serial.println("RIGHT");
      break;
    case 64:
      Serial.println("OK");
      break;
```

```
      // Numbers
    case 22:
      Serial.println("1");
      break;
    case 25:
      Serial.println("2");
      break;
    case 13:
      Serial.println("3");
      break;
    case 12:
      Serial.println("4");
      break;
    case 24:
      Serial.println("5");
      break;
    case 94:
      Serial.println("6");
      break;
    case 8:
      Serial.println("7");
      break;
    case 28:
      Serial.println("8");
      break;
    case 90:
      Serial.println("9");
      break;
    case 82:
      Serial.println("0");
      break;
```

```
      // # and *
    case 66:
      Serial.println("*");
      break;
    case 74:
      Serial.println("#");
      break;

      // otherwise...
    default:
      Serial.println("Code is :" + c);
      break;
    }
  }

  // Unfortunately due to technical difficulties, this function is unfinished, and does not provide a consistent Output
}
```

Figure 18. Current Build ‘state of code’ of the Infrared Remote Controller function.

Piezo Buzzer

How the Piezo Buzzer works within my project (according to the initial theme) is: When the Sonar / Distance Sensor returns an environmental error (as part of its function), or when a high ‘danger’ element of the program’s game is present, the Piezo Buzzer will go off / buzz.

There are no user inputs for the Piezo Buzzer function, as the buzzer is dependent on the Sonar Device and the program to operate. The output for this function is the tone buzzer of the Piezo Buzzer.

```
void piezoBuzzerAlert() {
  /*
  INITIAL THEME: when the buzzer (piezo) gets a parameter variable of something "dangerous" in the program's game, sound the alert.
  @params none
  @return none

  ADJUSTED THEME: when the Infrared Remote outputs a specific code (ARROWS, NUMBERS, ETC.), play a melody (a group of tone commands, delayed in particular ways to form music).
  @params IR remote code
  @return none
*/

/* This tone system which has been commented out was supposed to be the melody of the Default Dance from Fortnite.
  tone(piezoPin, 349);  // Send 1KHz sound signal... NOTE F4
  delay(200);
  noTone(piezoPin);
  delay(500);
  tone(piezoPin, 349);  // Send 1KHz sound signal... NOTE F4
  delay(100);
  noTone(piezoPin);
  delay(100);
  tone(piezoPin, 392);  // Send 1KHz sound signal... NOTE G4
  delay(100);
  noTone(piezoPin);
  delay(100);
  tone(piezoPin, 440);  // Send 1KHz sound signal... NOTE A4
  delay(100);
  noTone(piezoPin);
  delay(500);
  tone(piezoPin, 392);  // Send 1KHz sound signal... NOTE G4
  delay(100);
  noTone(piezoPin);
  delay(1000);
*/ // This function has been commented out for the moment being because of the tone looping during the debugging period.
}
```

Figure 19. Current Build ‘state of code’ of the Piezo Buzzer function.

Sonar / Distance Sensor

How the Sonar / Distance Sensor works within my project (according to the initial theme) is: When the Distance Sensor detects an environmental hazard (when the board is less than 10 centimetres from a wall or monitor, call upon the Piezo Buzzer to alert the user of danger to the Electronic System.

The user input to this function is the location of the Electronic Board, sounding the buzzer if the board is too close to an obstacle. The output is the buzzer system that is connected to the Sonar / Distance system.

```
322  void environmentalAlarmSystem() {
323    /*
324    when the distanceSensorEnvironmentalCheck function calls this function (in the instance of a danger to the electronic board's placement), sound the alarm
325    @params 'distanceSensorEnvironmentalCheck' output
326    @return -
327
328    // Unfortunately due to time restraints, this function is unfinished, and does not provide any Output or Input.
329  */
330  }
331
332  void distanceSensorEnvironmentalCheck() {
333    /*
334    when the the base of the arduino is too close to an obstacle (for instance a desk or a wall), send a parameter to the piezo buzzer to sound the alarm.
335    @params none
336    @return -
337  */
338    digitalWrite(trigPin, LOW);
339    //Serial.println("Trig Pin (Sonar) set to LOW");
340    delayMicroseconds(2);
341    // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
342    digitalWrite(trigPin, HIGH);
343    //Serial.println("Trig Pin (Sonar) set to HIGH");
344    delayMicroseconds(10);
345    digitalWrite(trigPin, LOW);
346    //Serial.println("Trig Pin (Sonar) set to LOW");
347    // Reads the echoPin, returns the sound wave travel time in microseconds
348    long duration = pulseIn(echoPin, HIGH);
349    // Calculating the distance
350    int distance = duration * 0.034 / 2;  // Speed of sound wave divided by 2 (go and back)
351
352    // Unfortunately due to resource difficulties, this function is unfinished, and does not provide a consistent Output.
353  }
```

Figure 20. Current Build ‘state of code’ of the Sonar / Distance function.

Code and Schedule inconsistencies:

Throughout the term there have been a variety of cases of our teacher Ryan Cather calling in sick, and as such we have been slightly behind on setting up our Electronic Boards. Thankfully, a majority of the classwork that our Robotics class researches is on the Subject Website.

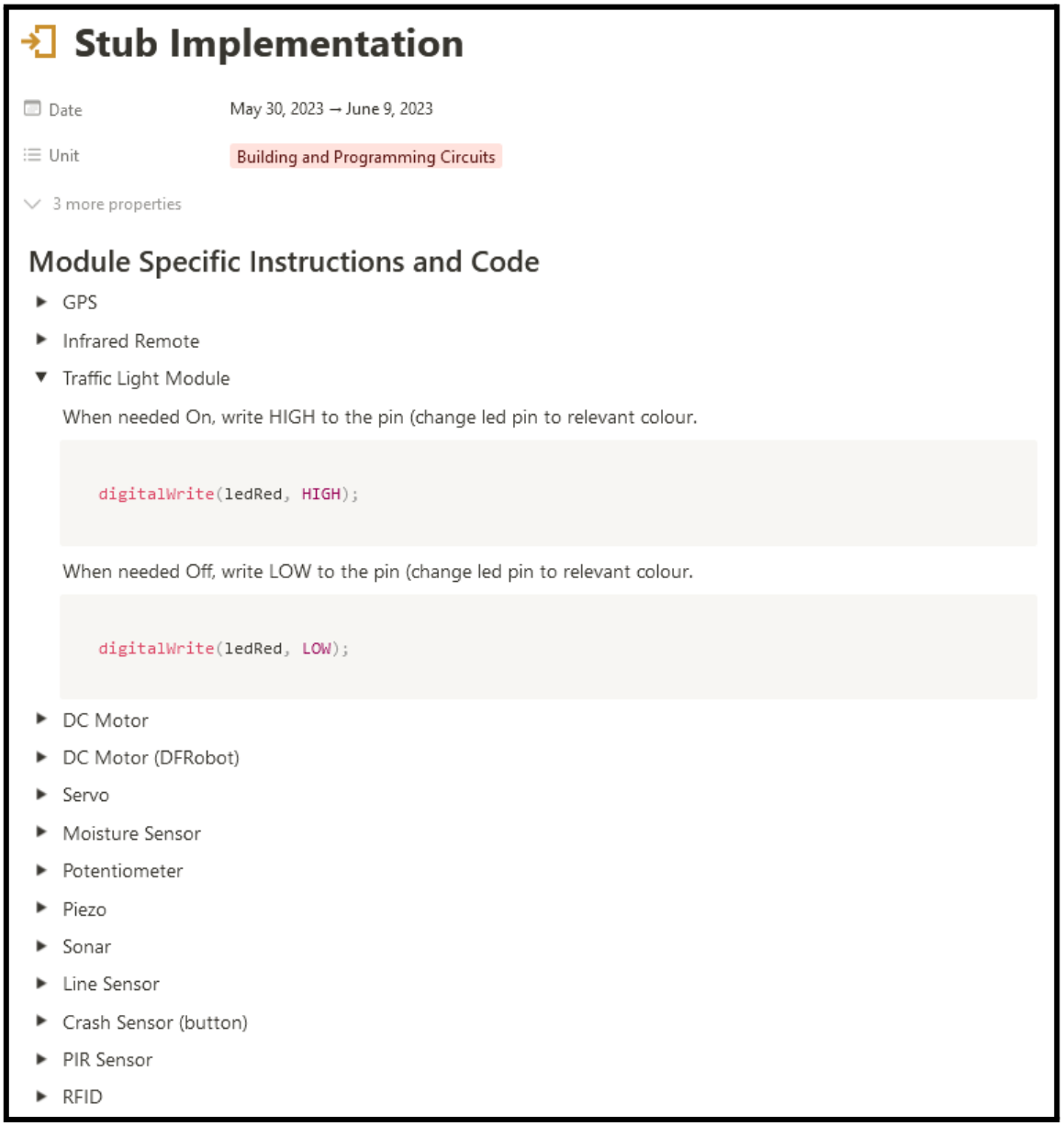


Figure 21. ‘Stub Implementation’ from the subject website.

Although, the subject website which houses all of the tutorials for the course does not have any code in C# that tells the students about how to use their devices for more purposes other than initialisation. Notably, this is referred to as the “Stub Implementation” of the assignment, and such Smart Device projects cannot be finalised until there is further research of how to use the language is available for students to access.

Finalised Theme ideas, adapted from obstacles:

Because of a few inconsistencies with the C# code and schedule, a few minor changes could be made to the theme of the assignment, repurposing a few implemented functions.

Notable Changes:

Piezo Buzzer - Infrared Remote changes:

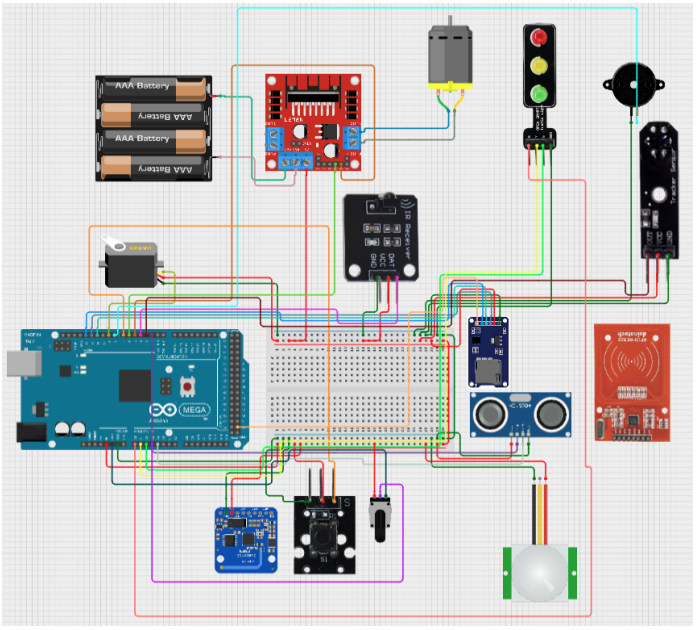
When the Infrared Remote presses a certain button type (Arrows, Numbers, etc), the IR remote function will call upon the piezo buzzer to play a variety of different melodies. One of

these melodies is the already implemented Fortnite Default Dance. This is used to get a ‘useful’ output from the finalised assignment Electronic Board.

Traffic Light System changes:

The initial purpose of the traffic light system is to visualise a danger element. However after adapting from this due to technical difficulties, the traffic light LEDs activate upon IR remote button presses (i.e When ‘HOME’, ‘BACK’ or ‘#’ & ‘*’ are pressed). This is used to verify that the IR remote function works.

Wiring Diagram and Final Project board:

	
WiringDiagram.ckt, a file designed in “Cirkit Designer” to formulate how to wire Electronic Boards in class.	A picture of the finalised Electronic Board that I have made for my assignment.