The Exe.stentialist

S. Gmira & J. Kleine
s.gmira@student.rug.nl & j.kleine.4@student.rug.nl
S5058147 & S5152372

November 8, 2022

1 Project description

We have designed and programmed our own version of the classic Google Chrome dinosaur game. We have limited ourselves to using an Arduino Uno, a 2 pin push-button, and a 16x2 I^2C LCD display.

This project was lightly influenced by both our Introduction to Logic course and The Stranger (Albert Camus, 1942). The character that players control is named 'The Stranger', after the main character of the book aforementioned. The book's main themes include existentialism and absurdism. Additionally, to remain on the theme of existentialism, the character would be avoiding obstacles in the form of existential quantifiers.

2 Project analysis

Our program should output a working game on a 16x2 I^2C LCD display. The game we chose for this is a modified version of Google Chrome's Dinosaur game. The game starts when you activate the 2 pin push button; the same button is then used to make the character jump. The goal is to gain as many points as possible by avoiding obstacles for the longest amount of time.

These existential obstacles appear at random intervals. Additionally, they move from right to left, always approaching the character. The idea behind this is that it will seem like the character is running towards the obstacles.

When working on this project we broke it down into the following sub-problems:

- The wiring of the I^2C LCD display.
- The wiring of the 2 pin push button.
- Displaying of custom characters.
- Movement of obstacles across the display.
- Jumping and walking animation of 'The Stranger'.
- Checking if 'The Stranger' has hit an obstacle.

3 Design

As previously mentioned, the overall design of the project was broken down into separate sections.

As seen in the schematic below, and as described in the module in the back of the display, the wiring of the I^2C LCD consisted of connecting 4 wires.

Below can be seen the breaking down of the four possible inputs in the back of the display and what they each represent.

- GND: Connects to the ground pin on the Arduino Uno board.
- VCC: Connects to a 5V pin on the breadboard.
- SDA: Connects to the A4 pin on the Arduino Uno board.
- SCL: Connects to the A5 pin on the Arduino Uno board.

The push button is simply a way of connecting a circuit. Thus, to connect it, a 10 KΩ resistor to a 5V power source must be connected to it as well as a ground pin to create a voltage differential.

One thing that was noticed while making the wiring of the project is that the board only had a singular 5V pin. Since both the display and the push button needed a power source, it was necessary to connect them through the supplied breadboard.

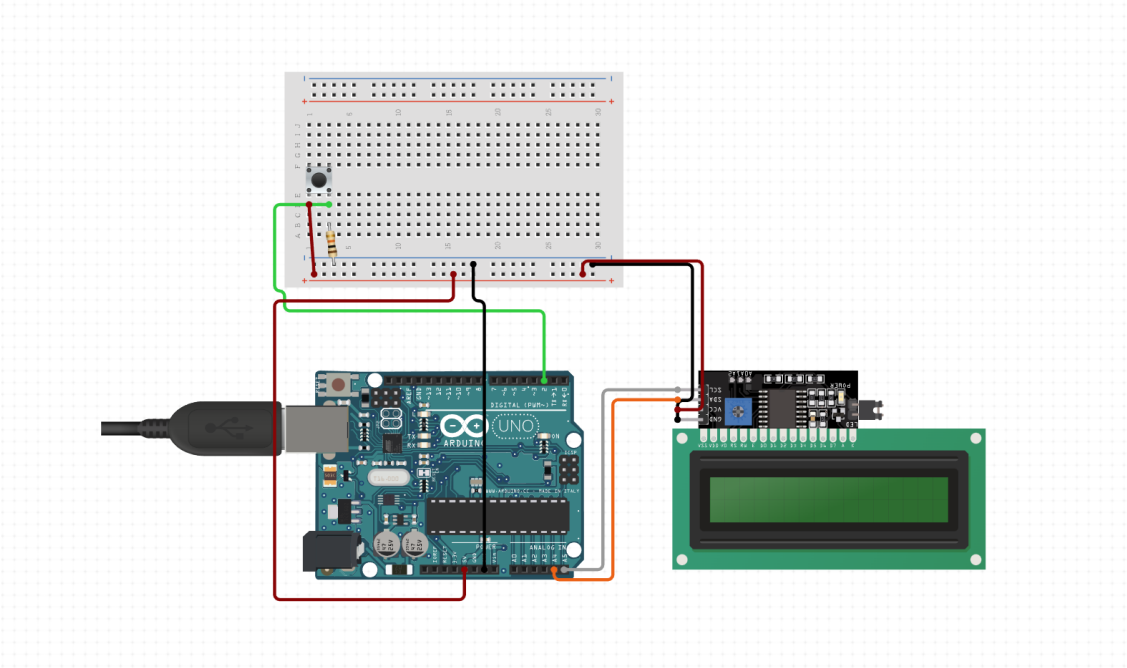

Figure 1: Wiring schematic.

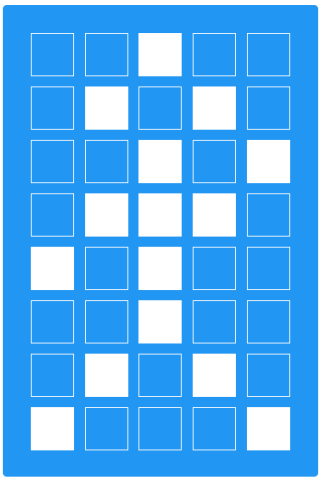

Figure 2: First walking animation.

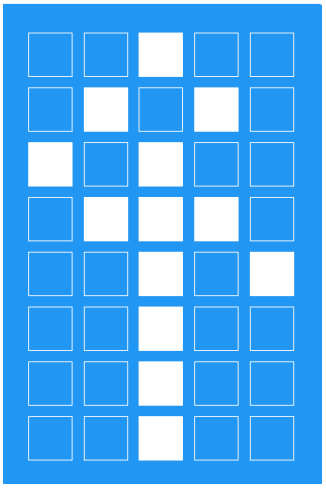

Figure 3: Second walking animation.

In order to personalize our project, we decided to create custom characters which would fit with out imagined idea of the project. The two figures above represent the two possible walking states of the character. By alternating between the two, it creates the illusion that the character is walking across the screen.

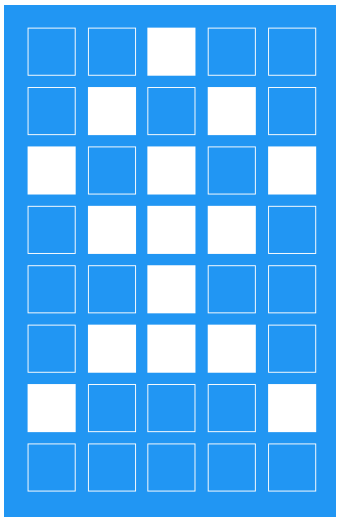

Figure 4: Jumping animation.

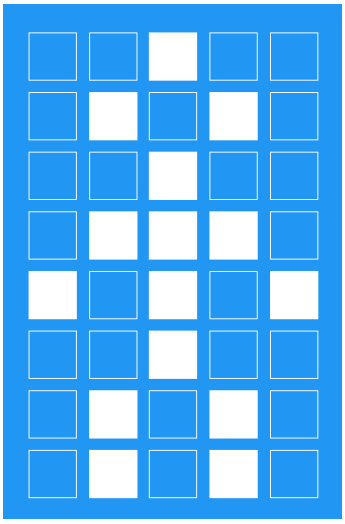

Figure 5: Neutral animation.

Additionally, we created 2 more custom characters which we would use when the character is jumping (Figure 4), or when the game has yet to start (Figure 5). By adding an animation for when the character jumps, it gives players the illusion that he is jumping over the obstacles rather than floating over them.

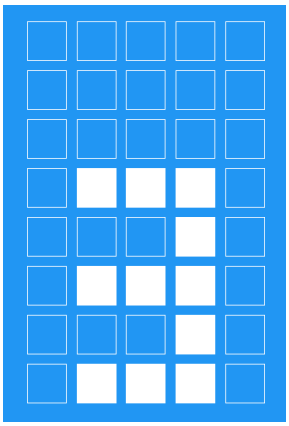

Figure 6: Obstacle.

The last custom character that we implemented was the obstacle. This took on the shape of an existential quantifier as previously mentioned.

The overall design of the program was straightforward. The most difficult part to implement was the scrolling of the obstacles. However, once that was figured out, we managed to implement other vanity features such as the custom characters. The logic of the program was the following:

1. Check if the game is running.
2. Wait for player input to play the game.
3. When the game is running, move the obstacles from right to left.
4. If an object has reach the leftmost wall, reset its position randomly.
5. When the player presses the button, make the character jump.
6. Check if the player is at the same location as the obstacles.
7. If that is the case then end the game.
8. Restart from the beginning.

4 Program code

existentialist.ino

```
#include <LiquidCrystal_I2C.h> //Including libraries to communicate with the display
#include <Wire.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // Set the LCD address to 0x27 for a 16 chars and 2 line display
const int buttonPin = 8; // The number of the pushbutton pin

int currState = 0; //Current button state
int prevState = 0; //Previous button state
int touchingGround = 0; //Check for whether or not the character is touching the floor

int highestScore = 0; //The highest score that has been acheived so far
int curScore = 0; //The player's current score

boolean playing = false; //Boolean for checking whether a player is playing or not

int distance1 = 0; //First distance of obstacles
int distance2 = 0; //Second distance of obstacles

int speed = 400; //Obstacle movement delay
int isJumping = 0; //Character in air delay
int walkingState = 0; //Used to switch between walking 2 different walking animations

byte jumping[8] = { //Map for the character jumping animation
  B00100,
  B01010,
  B10101,
  B01110,
  B00100,
  B01110,
  B10001,
  B00000
};

byte neutral[] = { //Map for character before starting game
  0x04,
  0x0A,
  0x04,
  0x0E,
  0x15,
  0x04,
  0x0A,
  0x0A
};

byte walk1[] = { //Map for character while playing the game
  0x04,
  0x0A,
  0x05,
  0x0E,
  0x14,
  0x04,
  0x0A,
  0x11
};
```

```
54
55 byte existentialism[] = { //Map for obstacles
56   0x00,
57   0x00,
58   0x00,
59   0x0E,
60   0x02,
61   0x0E,
62   0x02,
63   0x0E
64 };
65
66 byte walk2[8] = { //Map for the character walking animation (second part)
67   B00100,
68   B01010,
69   B10100,
70   B01110,
71   B00101,
72   B00100,
73   B00100,
74   B00100
75 };
76
77 void setup() {
78   lcd.init();
79   lcd.clear();
80   lcd.backlight(); //Initializing and setting up display
81   lcd.createChar(0, neutral); //Creating custom characters
82   lcd.createChar(1, walk1);
83   lcd.createChar(4, walk2);
84   lcd.createChar(3, jumping);
85   lcd.createChar(2, existentialism);
86 }
87
88 void loop() {
89   speed = 400;
90   if (playing == false) { //Setting up before the game starts
91     lcd.setCursor(1, 0);
92     lcd.print("PRESS TO START");
93     lcd.setCursor(2, 1);
94     lcd.write(0);
95     lcd.setCursor(4, 1);
96     lcd.write(2);
97     currState = !digitalRead(buttonPin); //Read user button input
98     if (currState == HIGH) {
99       playing = true;
100    }
101    if (playing) {
102      game();
103    }
104  }
105  delay(100);
106 }
107
108 void game() {
109   lcd.clear();
110   distance1 = random(4, 9);
111   distance2 = random(4, 9); //Create difference in distances
```

```
112   for (int i = 16; i >= -(distance1 + distance2); i--) { //For loop that moves the
          obstacles towards the player
113     lcd.setCursor(11, 0);
114     lcd.print(curScore); //Printing out of current player score
115     currState = !digitalRead(buttonPin); //Read user button input
116
117     if (((currState == HIGH) && (touchingGround == 1)) || isJumping > 0) { //If the button
            is pressed make the character jump
118       if (isJumping == 0) {
119         isJumping = 2; //Reset value for character in air time
120       }
121       lcd.setCursor(1, 0);
122       lcd.write(3);
123       lcd.setCursor(1, 1);
124       lcd.print(" "); //Clear where the character was
125       touchingGround = 0;
126       isJumping--;
127     } else { //If the button is not pressed then the character is on the ground
128       if (walkingState == 0) { //Check for character walking animation
129         lcd.setCursor(1, 1);
130         lcd.write(1);
131         lcd.setCursor(1, 0);
132         lcd.print(" ");
133         touchingGround = 1;
134         walkingState = 1;
135       } else {
136         lcd.setCursor(1, 1);
137         lcd.write(4);
138         lcd.setCursor(1, 0);
139         lcd.print(" ");
140         touchingGround = 1;
141         walkingState = 0;
142     }
143
144     lcd.setCursor(i, 1); //Printing out of the first obstacle
145     lcd.write(2);
146     lcd.setCursor(i + 1, 1); //Removing where the obstacle was in the previous loop
147     lcd.print(" ");
148
149     lcd.setCursor(i + distance1, 1); //Printing out of the second obstacle
150     lcd.write(2);
151     lcd.setCursor(i + distance1 + 1, 1); //Removing where the obstacle was in the previous
            loop
152     lcd.print(" ");
153
154     lcd.setCursor(i + distance1 + distance2, 1); //Printing out of the third obstacle
155     lcd.write(2);
156     lcd.setCursor(i + distance1 + distance2 + 1, 1); //Removing where the obstacle was in
          the previous loop
157     lcd.print(" ");
158
159     if ((i + distance1 + distance2) == 0) { //Checks if an obstacle has reached the end of
            the screen
160       i = 11; //Reset obstacle position
161       lcd.setCursor(0,1);
162       lcd.print(" "); //Clear out the obstacle position
163       lcd.setCursor(12,1);
164       lcd.print(" "); //Clear out the obstacle position
```

```
165     }
166
167     if (((i == 1) || (i + distance1 == 1) || (i + distance1 + distance2 == 1)) && (
            touchingGround == 1)) { //Checks if the character and the obstacle are in the same
            spot
168       lcd.clear();
169       lcd.setCursor(3,0);
170       lcd.print("YOU LOST");
171       lcd.setCursor(3,1);
172       lcd.print("TRY AGAIN");
173       delay(2500);
174
175       if (curScore > highestScore) { //Checking if the current score is greater than the
              previous score
176         highestScore = curScore;
177       }
178       lcd.clear();
179       lcd.setCursor(0,0);
180       lcd.print("HIGHEST: ");
181       lcd.print(highestScore);
182
183       lcd.setCursor(0,1);
184       lcd.print("CURRENT: ");
185       lcd.print(curScore);
186       delay(2500);
187
188       playing = false; //Reset values for the next round
189       curScore = 0;
190       lcd.clear();
191       break;
192     }
193
194     if ((curScore%20 == 0) && (speed > 200)) { //Increase speed at which obstacles move by
            25ms if the score is a multiple of 20
195       speed -= 25;
196     }
197     if ((curScore > 200) &&(curScore%100 == 0) && (speed > 50)) { //Increase speed at
            which obstacles move by 25ms if the score is a multiple of 100
198       speed -=25;
199     }
200     curScore++; //Score increases evertime the forLoop runs
201     delay(speed);
202   }
203 }
```

5 Evaluation

Initially, we thought of designing and making a "Currently Playing" display that would work in conjunction with the Spotify API. However, after having looked through it, we came to the conclusion that it would be far too difficult to implement due to different languages that are used. Instead, we opted to create our own version of the Google Chrome dinosaur game.

The goal for the project was to create a fun game that could be played between friends, in order to see who is more skilled at evading existential quantifiers as 'The Stranger'. It would create competition whilst still being entertaining to play on your own. At the end of each game, your score as well as the current highest score is displayed.

So far we have found three unintended features in the game. The first of which is the ability to infinitely fly as long as the push button is activated. This would allow the player to gain an unlimited amount of points as long as the button was activated. We corrected this by adding an amount of time for which 'The Stranger' must be in the air for. Once this was implemented, it forced the player to anticipate the obstacle which required a degree of skill.

The second unintended feature that was found consisted of a slight delay when displaying obstacles on the screen every thirty points. What would happen is that there would be two different obstacles that would appear one next to the other. After having looked through the source code, we determined that this phenomena occurred when the indices of the blocks were reset. To remedy this, we reset what was displayed(as seen on 157-161 of the source code).

The last minor error that was found was that the variable that dictated the speed at which obstacles moved was not reset at the end of every game. To fix this we included a line that would reset the value of the speed every time the game restarts.