DEPARTMENT OF PHYSICS
UNIVERSITY OF COLOMBO

PH3032 - EMBEDDED SYSTEMS LABORATORY

MINI CAR GAME

Name: A.C. Senanayake
Reg No: 2021s19076
Index No: s16410

Table of Contents

Table of figures

1 Abstract

This project details the design and implementation of a mini car game using the ATmega328P microcontroller, programmed with AVR. The game, displayed on a 16x2 LCD screen, challenges players to control a car that must avoid randomly generated obstacles. The car movement is controlled by two push buttons, enabling up and down navigation across the display two rows. The main objective is to steer the car safely without colliding with the obstacles. This project demonstrates the integration of embedded systems, hardware interfacing, and software control, showcasing the ATmega328P capabilities in creating engaging, interactive applications.

2 Introduction

Embedded systems are widely used in various applications, from home electronics to industrial machines due to their ability to perform dedicated tasks efficiently with minimal hardware resources. The ATmega328P microcontroller known for its versatility and low power consumption, is a popular choice for projects requiring a balance between performance and cost. This project explains the design and development of a mini car game using the ATmega328P highlighting the microcontroller ability to handle real time control and interact with users.

The mini car game is designed to be simple yet engaging. It helps show how to combine game logic, hardware, and user controls on an embedded system. The car is shown as characters on a 16x2 LCD screen and obstacles appear randomly on the screen. The player uses two buttons to move the car up and down to avoid these obstacles.

The 16x2 LCD screen limited in graphics, is used well by showing the car and obstacles as characters on its two rows. The main challenge is moving the car and obstacles in real time which is managed with efficient AVR coding. Timers and interrupts in the microcontroller help make the gameplay smooth.

The two push buttons control the car movement working like a basic joystick. Each button is connected to the microcontroller GPIO pins and debounced to avoid wrong inputs and allow smooth control. Obstacles are placed randomly on the screen using the timer to create unpredictable positions and times, making the game more challenging for the player.

This project is not just an example of making a game using a microcontroller but also a way to learn about real time systems, user interfaces, and the basics of embedded programming.

3 Theory

3.1 ATmega328P Microcontroller

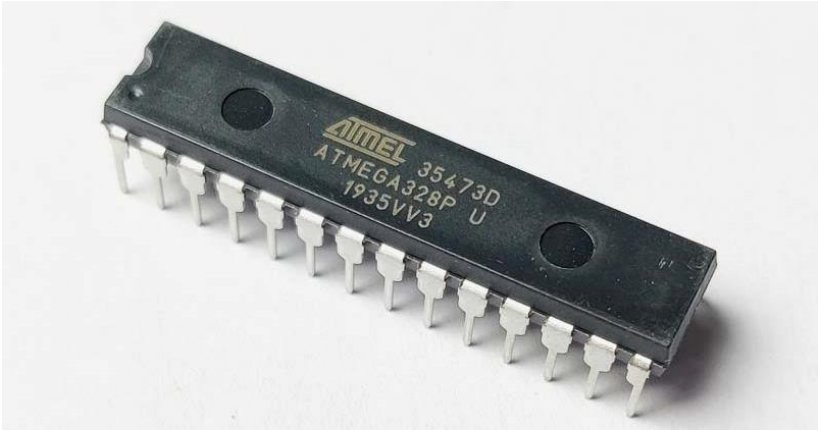

Figure 1: ATmega328p microcontroller created by Atmel

The ATmega328P is a small 8-bit microcontroller made by Atmel (now part of Microchip Technology). It is known for being efficient and versatile. It combined 32 KB of ISP Flash memory with read, write capabilities, 2 KB of memory of SRAM (Static Random Access Memory) for temporary data and 1 KB of EEPROM (Electrically Erasable Programmable Read Only Memory) memory that can save data even when the power is off. It can run at a top speed of 20MHz.

This microcontroller supports different ways of communicating like UART, SPI, and two wire serial communication. And it comes with 23 pins that can be used for general purposes. It also has 6 pins for controlling things like motors (PWM), and 6 pins for reading analog signals (like from a sensor).

It uses very little power, and it has modes that help save even more energy, making it good for battery powered devices. The ATmega328P is popular because it used in the Arduino Uno and is widely supported by a large community. It is commonly used in simple electronics projects, DIY builds, robotics, and IoT (Internet of Things) projects because it easy to work with and performs well.

3.2 Input/Output (I/O) Control

Input/Output (I/O) control on the ATmega328P microcontroller involves configuring and managing its GPIO (General Purpose Input/Output) pins. The ATmega328P has 23 GPIO pins organized into three categories.

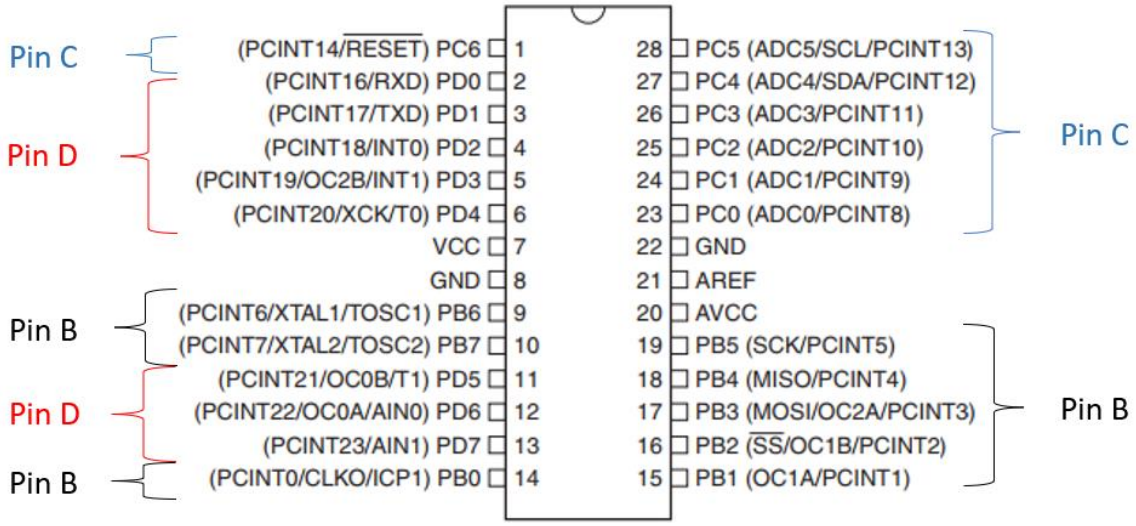

Figure 2: Categorized pin diagram of ATmega328P microcontroller

Those are PORTB, PORTC, and PORTD. These pins can be configured as input or output to interface with external devices such as sensors, LEDs, motors, switches, etc.

When setting pin as Input or Output the $DDRx$ register controls the direction of each pin on the port.

- $DDRx$ = 1 for output
- $DDRx$ = 0 for input

Push buttons are used to control the car and those are connected to the microcontroller GPIO pins. Input signals are provided by the buttons, which are then processed by the microcontroller to move the car on the LCD display.

3.3 LCD Interface and Display Control

16x2 LCD is used to display the game, with the car shown on one row and obstacles that can appear on both rows. The microcontroller talks to the LCD through a parallel connection, using control and data pins to show characters on the screen.

Each spot on the LCD has its own memory location. To move the car and show obstacles, the microcontroller changes these memory locations to update the display in real time.

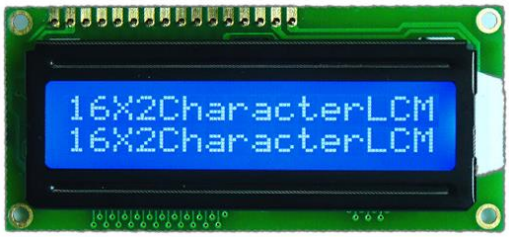

Figure 3: 16x2 Character liquid crystal display (LCD)

3.4 Timers and Interrupts

In this game, obstacles appear at random times. Timers in the ATmega328p, which allow to generate precise delays or trigger periodic events.

- Timer interrupts are used to control the time intervals between obstacles or other periodic events like screen updates.
- To make obstacles appear at random intervals, a pseudo random number generator (PRNG) can be used. The PRNG generates random numbers that vary the intervals between obstacles.

3.5 Push-Button Control Logic

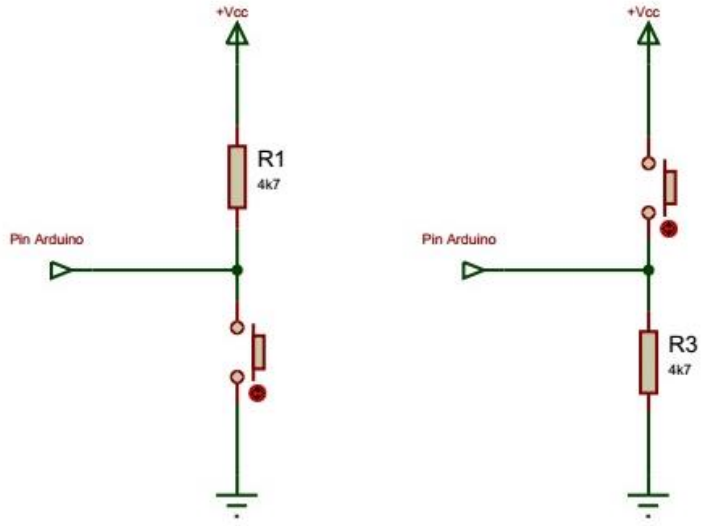

Figure 4: Circuit diagram of Pull Up (left) and Pull Down (right) resistors

The game checks the current position of the car in relation to the obstacles. When the player presses a button, the corresponding GPIO input pin is read, and the car position is updated on the LCD. Each button can connect to the microcontroller in one of two common configurations.

- Pull Up configuration - The button connects the GPIO pin to ground. When button pressed the pin reads LOW (0). When not pressed the pin reads HIGH (1) due to pull up resistor.
- Pull Down configuration - The button connects the GPIO pin to a supply voltage (5V). When pressed the pin reads a HIGH (1). When not pressed the pin reads LOW (0) due to a pull down resistor.

3.6 Randomness and Pseudo-Random Number Generation

Randomness is a crucial concept in many computational tasks such as simulations, gaming like that. True randomness is hard to achieve in digital systems so microcontrollers typically rely on pseudo random number generation (PRNG). Pseudo randomness refers to numbers that appear random but are generated by an algorithm. These numbers are not truly random they are based on an initial value called a seed. They are often good enough for many practical purposes where absolute unpredictability is not essential.

3.7 Game Logic and Collision Detection

- **Scene Update and Obstacle Movement**

The game creates a scrolling effect by constantly shifting the contents of both rows to the left. This gives the impression that the car is moving forward. As the content scrolls, the game randomly generates obstacles that appear on the right side of the screen. Obstacles can appear in the both top row and the bottom row. Every time the screen updates, any obstacles already on the screen move one position to the left, simulating forward motion.

- **Collision Detection**

As the car moves forward, the game checks if there is an obstacle directly in front of it. The car is always in the first column on the screen, and the game checks if there's an obstacle in the next column. If there is an obstacle in the car's path (on the same row), a collision is detected.

- **Game Speed and Difficulty**

As the game progresses, it gradually speeds up by reducing the time between updates. This means that obstacles will appear and move faster, making the game more challenging as player continue to play. This increase in speed happens at regular intervals adding to the difficulty over time.

- **Game Loop**

The game continuously checks for player inputs (moving the car), updates the screen to scroll the obstacles, and checks for collisions. If no collision occurs the game keeps running and the player must keep avoid obstacles by switching between the two rows.

4 Methodology

4.1 Materials

- ATmega328p microcontroller
- Created PCB
- 16 × 2 LCD display
- I2C module
- Buk Converter
- 9V battery
- Push buttons
- Jumper wires
- Resistors

4.2 Setup the circuit

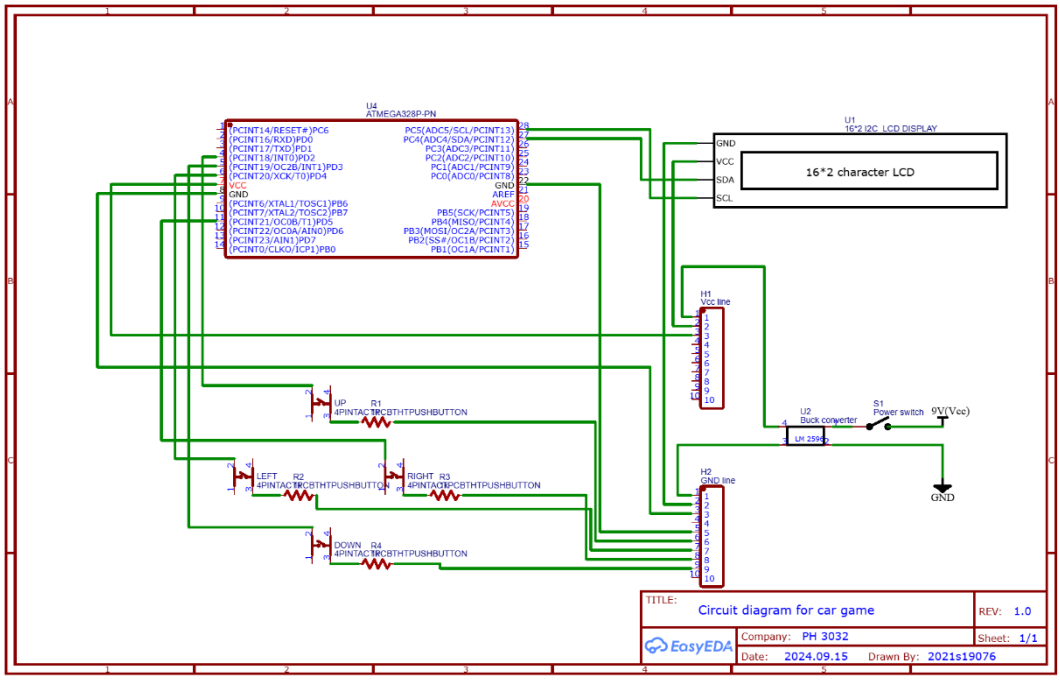

Figure 5: Circuit diagram of the mini car game. (Drawn by using easyEDA)

As above circuit diagram, a 9V battery was used as the power source for the system. However, most of the parts like the microcontroller and the LCD display needed a lower voltage (5V). To reduce the 9V input to the required 5V a buck converter was used.

The buck converter is a DC-DC step down voltage regulator. It took the 9V input from the battery and provided a steady 5V output. The input of the converter was connected directly to the 9V battery terminals, and its output (+) terminal was connected to the power line and output (–) terminal was connected to the GND line of the circuit. The microcontroller, LCD display, I2C module, and push buttons are all powered from output of buck converter.

The microcontroller was the central processing unit of the system. It was responsible for running the game logic, managing input from the buttons, and controlling the output on the display. Several digital pins of the microcontroller were used to connect with components like push buttons and the I2C module, which was linked to the LCD display. These pins were set up in the software to perform tasks like reading inputs from the buttons and sending data to the display.

A 16x2 LCD display is used to provide visual feedback and output for the game. To minimize the number of connections and simplify the wiring, an I2C (Inter-Integrated Circuit) communication module is attached to the LCD. The I2C module allows communication between the microcontroller and the LCD using only two lines. Microcontroller has dedicated pins for I2C communication, typically labeled as SDA and SCL. The SDA and SCL lines are connected to the corresponding pins.

User interaction with the game is enabled through four push button switches, which are connected to four different digital pins of the microcontroller. These buttons serve as inputs for controlling various aspects of the game. One terminal of the button was connected to a digital input pin of the microcontroller. The other terminal was connected to ground (GND). When the button was pressed, the corresponding pin was pulled to a low logic level (0V), allowing the press to be detected by the microcontroller. And the associated action to be performed.

Table 1: Table of Push buttons configuration of the circuit.

Button	Pin	Objective
Up button	PD2	Move the car to top raw of screen
Down button	PD3	Move the car to bottom raw of screen
Right button	PD4	Start the game
Left button	PD5	

Two separate PCBs were designed and made, with one dedicated to the switches and the other allocated for the remaining components. All relevant considerations and requirements were incorporated into the design and layout of each PCB to ensure functionality.

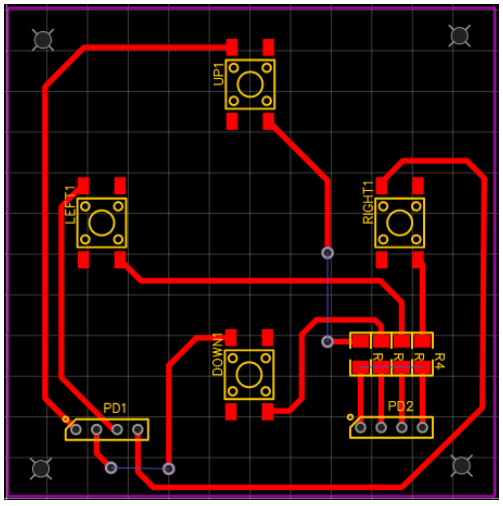

Figure 6: PCB for switches (designed by using easyEDA)

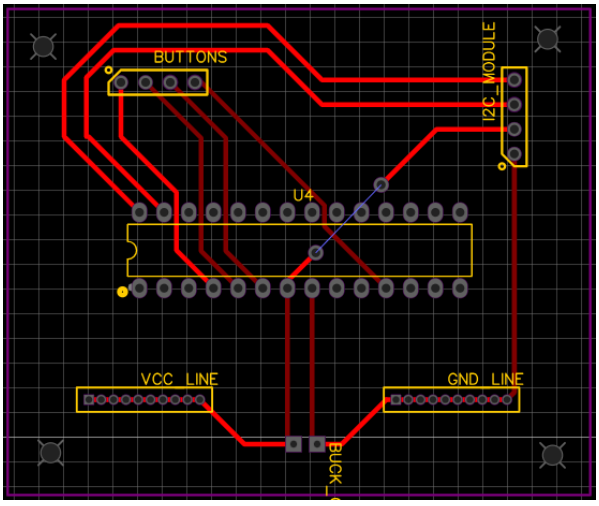

Figure 7 : PCB for the device (designed by using easyEDA)

A key part of the circuit design was ensuring that all components shared a common ground and had proper power connections. The following connections were made:

- Ground (GND) - A common ground line was created by connecting all the GND pins of the components (microcontroller, I2C module, LCD, buttons) to the (-) terminal of the buck converter.
- Power (V_{cc}) - The 5V output from the buck converter was connected to the V_{cc} pins of the microcontroller, I2C module, and LCD, supplying the necessary power to each component.

4.3 Programming the ATmega328p

- **External Interrupt Configuration (For buttons)**

Push buttons that are connected to $PD2$ ($INT0$) and $PD3$ ($INT1$) control the car movement for up and down. External interrupts are used to detect when these buttons are pressed.

DDRD - $PD2, PD3 \rightarrow 0$ (as input and the rest of pins remain unchanged)

PORTD - This enables pullup resistors on $PD2$ and $PD3$. When the button is not pressed pull up resistors hold the pin at HIGH. When the button is pressed, the pin is pulled LOW.

EIMSK - This enables external interrupts $INT0$ ($PD2$) and $INT1$ ($PD3$). When these bits are set to 1, the corresponding interrupts are enabled. Interrupts are triggered when the state of the associated pin changes according to the configuration in $EICRA$.

EICRA - $ISC01$ (for $INT0$) and $ISC11$ (for $INT1$) are set to 1, triggers the interrupts on the falling edge of the signal. This means the interrupt will be triggered when the button is pressed (the pin goes from HIGH to LOW).

- **I2C Configuration (for LCD Communication)**

The I2C module is used for communication with the LCD display. The SDA and SCL lines of the I2C connected to the corresponding pins on the microcontroller.

SDA (Serial Data Line) - This pin is responsible for transmitting data between the microcontroller and the I2C module. ($SDA \rightarrow PC4$)

SCL (Serial Clock Line) - This pin controls the timing of communication between the microcontroller and the I2C module by providing a clock signal to keep everything in sync. ($SCL \rightarrow PC5$)

- **LCD Initialization and Control**

The LCD uses I2C communication, so all configuration related to the LCD (such as clearing the display, setting the cursor position, and printing characters) is done through the I2C interface. A header file was used for I2C.

- **Score Calculating**

The score is calculated based on the time spent playing. The timer begins as soon as the game starts and stops once a collision is detected. Score is generated by converting the total time survived into points, rewarding players for longer survival periods.

4.4 Construct the device

The device was assembled by connecting all necessary components using jumper wires. Each component was carefully linked to ensure proper functionality, and jumper wires were used to establish secure connections throughout the PCBs and other components.

A wooden box was crafted using easily available wood found at home. The decision to create the box was driven by the low cost and the easy accessibility of materials. It features a lid that opens and closes effortlessly, ensuring convenient use.

All the necessary parts have been placed inside the box. Six holes were added to the lid to fit the push buttons, the LCD display, and the power switch. Power switch can be easily turned on and off the device whenever needed. A rechargeable 9V battery was used to provide power for the device, which allows it to work without being plugged into an external power source. This makes the device portable, meaning it can be moved and used in anywhere without needing to stay connected to an outlet.

Figure 8: Fully assembled device with wooden box.

4.5 Testing

The game is tested by running it multiple times to ensure it functions properly. Each gameplay session is observed carefully with particular attention given to key features such as scoring accuracy, collision detection, and timer functionality. Any detected issues are documented and adjustments are made as necessary. After each modification the game is re tested to confirm that the changes resolve the issues without introducing new ones. This iterative process is repeated until the game operates smoothly, providing players with good gaming experience.

5 Results and Analysis

The mini car game developed using the ATmega328p microcontroller was successfully implemented and tested. The game displayed on LCD screen operates smoothly, allowing the player to control the car movement up and down within two rows avoiding obstacles. Two push buttons were used to move the car between the rows. Obstacles were randomly generated at different intervals, making the game challenging and engaging for players. The system responded reliably to the push button inputs and the car movement was displayed accurately on the LCD screen. The score calculation worked as expected.

Multiple runs were conducted to evaluate the system performance.

Table 2 : Table of Scores obtained when playing the game.

Attempts	Scores obtained
1	68
2	134
3	116
4	296
5	208

These results show that the scoring system works consistently. The score increasing proportionally to the played time.

The game provided an interactive experience. The difficulty level controlled by the randomness of the obstacle appearance and speed of the obstacle moving. The implementation of this game demonstrated the effective use of the ATmega328p microcontroller.

6 Discussion

The mini car game developed using the ATmega328P microcontroller successfully demonstrates the potential of embedded systems in creating interactive applications. The project integrates multiple hardware components, such as the 16x2 LCD display and push buttons, along with AVR programming to provide a functional and engaging user experience. The game is designed that the player controls a car to avoid obstacles that appear randomly. This shows important ideas like handling player controls in real time, creating random events, and using external devices to give visual feedback to the player.

One of the main challenges during development was making sure the obstacles appeared at random unpredictable times. To solve this, the microcontroller timer and random number functions were used to change when and where obstacles showed up on the screen. However, sometimes the random placement made it impossible for the player to avoid a crash. To fix this, the timing and position of the obstacles were adjusted to always leave a way to escape, making the game more fair and fun. Another important part of the development was making sure the car moved quickly and accurately when the player pressed the buttons, and that this movement showed correctly on the LCD screen. At first, there was a small delay in the car movement because the push buttons needed to be debounced to avoid accidental triggers. This problem was fixed by using a software debouncing method, which removed false button presses and made the car respond smoothly to player actions.

The small screen size imposed certain limitations on the complexity of the game. The car could only move between two rows, which limited the range of motion available to the player. However, within these constraints, the game remained challenging due to the randomness of the obstacles and the speed at which they appeared.

An interesting observation during testing was how the speed of obstacle appearance affected the player experience. When the intervals between obstacles were too short, the game became too difficult and frustrating for play. Conversely, when obstacles appeared too infrequently, the game lacked challenge and became boring. A balance was achieved by fine tuning the obstacle generation algorithm, ensuring that the game remained challenging but not overly difficult.

This project could start as a foundation for more complex embedded system applications. With further improvements this mini car game could develop into a more advanced embedded system application.

7 Conclusion

In conclusion, the mini car game designed using the ATmega328P microcontroller successfully demonstrates the application of embedded systems in game development. The combination of push button controls, a 16x2 LCD display, and random obstacle generation created a simple interactive game where players could move the car to avoid collisions. The project highlights the importance of integrating hardware and software components effectively to ensure smooth performance, especially in real time systems where user inputs need to be processed right away.

This project could serve as a foundation for more complex embedded system applications like more advanced gaming mechanics, real time monitoring systems, interactive control systems. With additional enhancements such as a larger display or more complex game logic, this mini car game could develop into a more complex embedded system program.

8 Appendix

8.1 Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "I2C.h"    //I2C header file

#define F_CPU 1000000UL

//define buttons
#define DOWN_BUTTON PD2       //INT0
#define UP_BUTTON PD3         //INT1
#define RIGHT_BUTTON PD4      //Start buttons
#define LEFT_BUTTON PD5


//Variables to track button presses
volatile uint8_t downButtonPressed = 0;
volatile uint8_t upButtonPressed = 0;



//Parameters of games

volatile long frameStepMs = 300;
volatile long prevFrame = 0;
volatile long totalFrame = 0;
volatile uint8_t gameRunning = 0;
volatile unsigned long gameStartTime = 0;  //Variable to store the start time of game

//game components
char car = '>';                 //represent the car.
char obstacle = 'O';   //represent obstacle.
char contents[2][16] = { ' ' }; //2D array represent the LCD screen contents
int carPosRow = 0;
int carPosCol = 0;

void sceneRender() {
    LCD_Command(0x80); //set cursor to the beginning of the first row
    for (int i = 0; i < 16; i++) {
        LCD_Char(contents[0][i]);
    }
    LCD_Command(0xC0); //set cursor to the beginning of the second row
    for (int i = 0; i < 16; i++) {
        LCD_Char(contents[1][i]);
    }
    _delay_ms(10);
}



//show countdown before game starts

void showCountdown() {
    for (int i = 3; i >= 0; i--) {
        LCD_Command(0x01); //clear display
        LCD_String("Starting in...");
        LCD_Command(0xC0); //move to the second line
        LCD_Char('0' + i); //display countdown
        _delay_ms(1000);
```

```
    }
    LCD_Command(0x01); //dlear display
}

//show countdown after score display
void showCountdown2() {
    for (int i = 3; i >= 0; i--) {
        LCD_Command(0x01); //clear display
        LCD_String("Restarting...");
        LCD_Command(0xC0); //move to the second line
        LCD_String("Start in ");
        LCD_Char('0' + i);
        _delay_ms(1000);
    }
    LCD_Command(0x01); //clear display
}

//Restart the game
void restartGame() {
    //reset game variables
    gameRunning = 1;
    frameStepMs = 300;
    prevFrame = 0;
    totalFrame = 0;
    gameStartTime = millis();

//clear game
    for (int r = 0; r < 2; r++) {
        for (int c = 0; c < 16; c++) {
            contents[r][c] = ' ';
        }
    }

//place car in initial position
    carPosRow = 0;
    carPosCol = 0;
    contents[carPosRow][carPosCol] = car;

    //Render initial scene
    sceneRender();
}

void showDeathMessage() {
    LCD_Command(0x01); //clear display
    LCD_String("Car crashed...");
    LCD_Command(0xC0); //move to the second line
    LCD_String("Score: ");

//Calculate and display the elapsed time as score
    unsigned long elapsedTime = (millis() - gameStartTime) / 100;
    char scoreStr[10];
    itoa(elapsedTime, scoreStr, 10);
    LCD_String(scoreStr);

    _delay_ms(3000);
    showCountdown2(); // Display countdown for restart

    restartGame();     //restart game
}

void updateScene() {
    //check for collision
    for (int r = 0; r < 2; r++) {
        if (contents[r][carPosCol] == car && contents[r][carPosCol + 1] != ' ') {
            showDeathMessage();
            return;
```

```
125.         }
126.     }
127.
128.     //shift obstacles left
129.     for (int r = 0; r < 2; r++) {
130.         for (int c = 0; c < 15; c++) {
131.             if (contents[r][c] != car) {
132.                 contents[r][c] = contents[r][c + 1];
133.             }
134.         }
135.     }
136.
137.     //generate new obstacle in the rightmost column
138.     int num = rand() % 4;
139.     if (num == 0) {
140.         contents[0][15] = obstacle;
141.         contents[1][15] = ' ';
142.     } else if (num == 1) {
143.         contents[0][15] = ' ';
144.         contents[1][15] = obstacle;
145.     } else {
146.         contents[0][15] = ' ';
147.         contents[1][15] = ' ';
148.     }
149. }
150.
151. //move the car between two rows.
152. void moveCarDown() {
153.     if (carPosRow == 0) {
154.         contents[carPosRow][carPosCol] = ' ';
155.         carPosRow = 1;
156.         contents[carPosRow][carPosCol] = car;
157.     }
158. }
159.
160. void moveCarUp() {
161.     if (carPosRow == 1) {
162.         contents[carPosRow][carPosCol] = ' ';
163.         carPosRow = 0;
164.         contents[carPosRow][carPosCol] = car;
165.     }
166. }
167.
168. //Wait for either the left or right button to start the game
169. void waitForStart() {
170.     LCD_Command(0x01); // Clear display
171.     LCD_String("Welcome.. Press");
172.     LCD_Command(0xC0);
173.     LCD_String("Lft/Rgt to start");
174.
175.     // Wait until either left or right button is pressed
176.     while ((PIND & (1 << LEFT_BUTTON)) && (PIND & (1 << RIGHT_BUTTON))) {
177.         _delay_ms(100);
178.     }
179.
180.     showCountdown(); //show countdown before starting
181. }
182.
183. // Interrupt service routines for button presses
184. ISR(INT0_vect) {  //ISR for DOWN button (PD2)
185.     downButtonPressed = 1;
186. }
187.
188. ISR(INT1_vect) {  //ISR for UP button (PD3)
189.     upButtonPressed = 1;
190. }
```

```
191.
192. void setup() {
193.     LCD_Init();
194.     waitForStart();  //wait for player to press start button and countdown
195.
196.     //Set button pins as input with pull up
197.     DDRD &= ~((1 << DOWN_BUTTON) | (1 << UP_BUTTON) | (1 << LEFT_BUTTON) | (1 <<
RIGHT_BUTTON));
198.     PORTD |= (1 << DOWN_BUTTON) | (1 << UP_BUTTON) | (1 << LEFT_BUTTON) | (1 <<
RIGHT_BUTTON);
199.
200.     //Enable interrupts
201.     EIMSK |= (1 << INT0) | (1 << INT1);  //enable INT0 and INT1
202.     EICRA |= (1 << ISC01) | (1 << ISC11); //falling edge triggers
203.
204.     sei();  //Enable global interrupts
205.
206.     //Initialize game start time and start game
207.     gameStartTime = millis();
208.     restartGame();  //set up initial game state
209. }
210.
211. void loop() {
212.     if (downButtonPressed) {
213.         downButtonPressed = 0;
214.         if (gameRunning) moveCarDown();
215.     }
216.     if (upButtonPressed) {
217.         upButtonPressed = 0;
218.         if (gameRunning) moveCarUp();
219.     }
220.
221. //handle frame updates
222.     long frame = millis();
223.     if (frame - prevFrame >= frameStepMs) {
224.         updateScene();
225.         totalFrame++;
226.         prevFrame = frame;
227.         if (totalFrame % 10 == 0 && frameStepMs > 200) {
228.             frameStepMs -= 20;
229.         }
230.     }
231.
232.     //render the scene if the game is running
233.     if (gameRunning) {
234.         sceneRender();
235.     }
236. }
237.
238.
239. //main function
240. int main(void) {
241.     setup();  //initialize and display
242.
243.     while (1) {
244.         loop();  //run the game
245.     }
246.
247.     return 0;
248. }
249.
```

8.2 Header file for i2c module

```
#ifndef I2C_H
#define I2C_H

#include <avr/io.h>           /* Include AVR std. library file */
#include <util/delay.h>       /* Include Delay header file */

/* LCD I2C address */
#define LCD_I2C_ADDRESS 0x27  /* Define I2C address of the LCD, often 0x27 or 0x3F */
#define LCD_BACKLIGHT 0x08    /* Backlight control bit */
#define ENABLE 0x04           /* Enable bit */
#define READ_WRITE 0x02       /* Read/Write bit */
#define REGISTER_SELECT 0x01  /* Register select bit */

/* I2C Functions */
void I2C_Init(void) {
    TWSR = 0x00;              /* Set prescaler bits to zero */
    TWBR = 0x46;              /* SCL frequency = 50kHz for F_CPU = 8MHz */
    TWCR = (1<<TWEN);         /* Enable TWI */
}

void I2C_Start(void) {
    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);  /* Enable TWI, generate start condition */
    while (!(TWCR & (1<<TWINT)));            /* Wait for TWINT flag to set */
}

void I2C_Stop(void) {
    TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);  /* Enable TWI, generate stop condition */
    _delay_us(100);
}

uint8_t I2C_Write(uint8_t data) {
    TWDR = data;              /* Copy data to TWI data register */
    TWCR = (1<<TWINT)|(1<<TWEN); /* Enable TWI and clear interrupt flag */
    while (!(TWCR & (1<<TWINT)));  /* Wait for TWINT flag to set */

    // Check status
    uint8_t status = TWSR & 0xF8;
    if (status == 0x28 || status == 0x18) { /* 0x28 = TW_MT_DATA_ACK, 0x18 =
TW_MT_SLA_ACK */
        return 0;  // ACK received
    } else {
        return 1;  // NACK or error
    }
}

/* LCD Functions */
void LCD_EnablePulse(uint8_t data) {
    I2C_Write(data | ENABLE);  /* Enable bit high */
    _delay_us(1);              /* Enable pulse width */
    I2C_Write(data & ~ENABLE); /* Enable bit low */
    _delay_ms(2);              /* Wait for the command to execute */
}

void LCD_Command(uint8_t cmnd) {
    uint8_t highNibble = (cmnd & 0xF0) | LCD_BACKLIGHT;
    uint8_t lowNibble = ((cmnd << 4) & 0xF0) | LCD_BACKLIGHT;

    I2C_Start();
    I2C_Write(LCD_I2C_ADDRESS << 1);  /* Send the I2C address with write mode */

    LCD_EnablePulse(highNibble);      /* Send the upper nibble */
    LCD_EnablePulse(lowNibble);       /* Send the lower nibble */
```

```
62.
63.     I2C_Stop();
64. }
65.
66. void LCD_Char(uint8_t data) {
67.     uint8_t highNibble = (data & 0xF0) | REGISTER_SELECT | LCD_BACKLIGHT;
68.     uint8_t lowNibble = ((data << 4) & 0xF0) | REGISTER_SELECT | LCD_BACKLIGHT;
69.
70.     I2C_Start();
71.     I2C_Write(LCD_I2C_ADDRESS << 1);  /* Send the I2C address with write mode */
72.
73.     LCD_EnablePulse(highNibble);      /* Send the upper nibble with RS=1 for data */
74.     LCD_EnablePulse(lowNibble);       /* Send the lower nibble with RS=1 */
75.
76.     I2C_Stop();
77. }
78.
79. void LCD_Init(void) {
80.     I2C_Init();                 /* Initialize I2C */
81.     _delay_ms(20);              /* LCD Power ON delay */
82.
83.     LCD_Command(0x02);          /* Initialize for 4-bit mode */
84.     LCD_Command(0x28);          /* 2 lines, 5x7 matrix in 4-bit mode */
85.     LCD_Command(0x0C);          /* Display ON, Cursor OFF */
86.     LCD_Command(0x06);          /* Auto increment cursor */
87.     LCD_Command(0x01);          /* Clear display */
88.     _delay_ms(2);
89. }
90.
91. void LCD_String(char *str) {
92.     while (*str) {
93.         LCD_Char(*str++);
94.     }
95. }
96.
97. #endif
98.
```

9 References

(No date) Design and implementation of Arduino microcontroller based automatic lighting control with I2CLCDdisplay.Availableat: https://www.researchgate.net/publication/326512260_Design_and_Implementation_of_Arduino_Microcontroller_Based_Automatic_Lighting_Control_with_I2C_LCD_Display (Accessed: 20 October 2024).

Gaspar, W. (2023) An LCD Arduino game, Medium. Available at: https://medium.com/@williangaspar360/an-lcd-arduino-game-a11fdf1e2d4a (Accessed: 20 October 2024).

Benne de BakkerBenne is professional Systems Engineer with a deep expertise in Arduino and a passion for DIY projects. (2024) Character I2C LCD with Arduino Tutorial (8 examples), Makerguides.com. Available at: https://www.makerguides.com/character-i2c-lcd-arduino-tutorial/ (Accessed: 20 October 2024).

star, H. (2022) A simple LCD game using Arduino Uno and character display, Hackster.io. Available at: https://www.hackster.io/Hack-star-Arduino/a-simple-lcd-game-using-arduino-uno-and-character-display-ae01f9 (Accessed: 20 October 2024).