

# Sprawozdanie Systemy Wbudowane

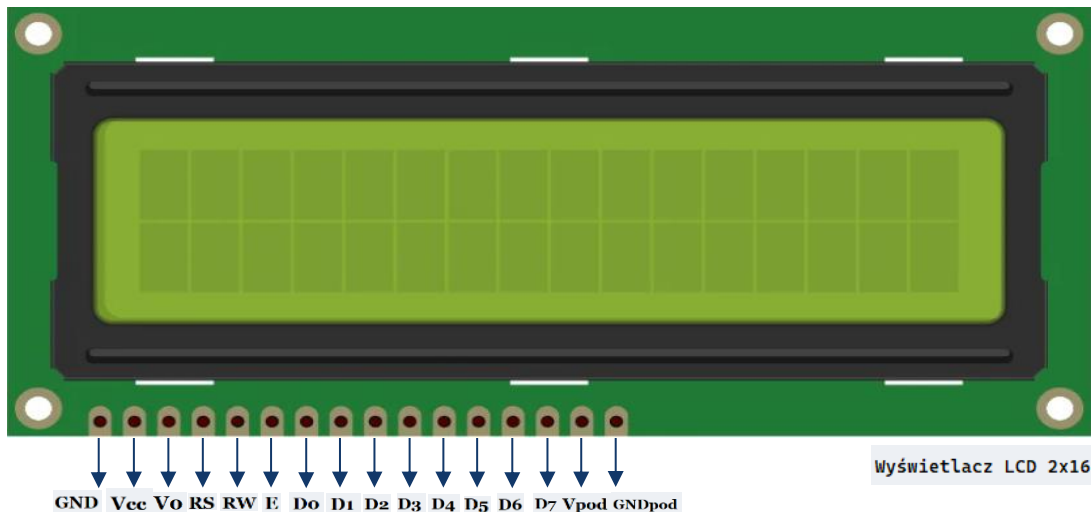
- *CRACKED FLAPPY BIRD* -

*Sebastian Feleńczak*

*Paweł Wiszniewski*

*Michał Fila*

# DOKUMENTACJA SPRZĘTOWA



## ☑ Wyświetlacz LCD 2x16

Wyświetlacz LCD składa się z interfejsu równoległego, co oznacza, że mikrokontroler musi manipulować kilkoma pinami interfejsu równocześnie, aby kontrolować wyświetlacz. Interfejs zawiera następujące piny:

- ⌘ Pin select register (RS), który kontroluje, do którego obszaru pamięci LCD wprowadzamy dane. Możesz wybrać rejestr danych, który przechowuje to, co jest wyświetlane na ekranie, lub rejestr instrukcji, gdzie kontroler LCD oczekuje na instrukcje dotyczące kolejnych operacji.
- ⌘ Pin Read/Write (R/W), który wybiera tryb odczytu lub zapisu.
- ⌘ Pin Enable, który umożliwia zapis do rejestrów.
- ⌘ 8 pinów danych (D0 - D7). Stan tych pinów (wysoki lub niski) to bity, które zapisujesz do rejestru podczas zapisu lub wartości, które odczytujesz podczas odczytu.
- ⌘ Istnieje również pin kontrastu wyświetlacza (Vo), piny zasilania (+5V i GND) oraz piny podświetlenia LED (Bkl+ i Bkl-), które można użyć do zasilania wyświetlacza, regulacji kontrastu oraz włączania i wyłączania podświetlenia LED.

Proces sterowania wyświetlaczem polega na umieszczeniu danych, które tworzą obraz tego, co chcemy wyświetlić, w rejestrach danych, a następnie umieszczeniu instrukcji w rejestrze instrukcji.

W ramach użycia wyświetlacza LCD 2x16 wykorzystujemy konwerter I2C wraz z biblioteką LCD\_I2C autorstwa "blackhack".

Autor: [https://github.com/blackhack/LCD\\_I2C](https://github.com/blackhack/LCD_I2C)



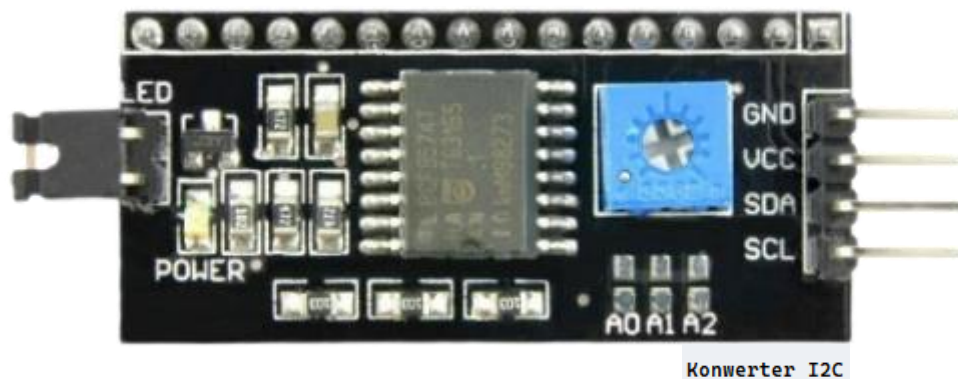
## ☑ Konwerter I2C dla wyświetlacza LCD 2x16 I2C

Konwerter I2C dla wyświetlaczy ze sterownikiem HD44780 w znaczący sposób ułatwia podłączenie ich do Arduino czy Raspberry Pi przez użycie tylko dwóch pinów i zasilania. Zamiast podłączyć do Arduino czy Raspberry z użyciem 6 pinów sterujących w wersji minimalnej (4 bity danych oraz RS i Enable) podłącz używając tylko SDA i SCL (interfejs I2C).

Biblioteka, z której można skorzystać przy podłączeniu do Arduino dostępna jest w serwisie GitHub. Na płytce znajduje się zworka LED, służąca do włączania lub wyłączania podświetlenia wyświetlacza, potencjometr do regulacji kontrastu, oraz zworki A1, A2 i A3 służące do zmiany adresu magistrali.

### Specyfikacja:

- ⌘ Konwerter oparty na układzie PCF8574 lub PCF8574A
- ⌘ Zasilanie 5V
- ⌘ Domyślny adres magistrali 0x27 (wersja z PCF874) lub 0x3F (wersja z PCF8574A)
- ⌘ Wbudowane rezystory podciągające dla magistrali I2C
- ⌘ Wymiary 42x20 mm



W zależności od użytego chipsetu moduł ten może mieć różne przestrzenie adresowe:

- ⌘ Chipset PCF8574 - 0x20
- ⌘ Chipset PCF8574T - 0x27
- ⌘ Chipset PCF8574A - 0x38
- ⌘ Chipset PCF8574AT - 0x3F

Układ posiada cztery wyprowadzenia oraz zworkę, której wyciągnięcie powoduje wyłączenie podświetlenia ekranu. Na płytce znajduje się także listwa goldpin, która jest dopasowana do złącz popularnych wyświetlaczy opartych na sterowniku HD44780. Wszystkie wyprowadzenia są przylutowane.

### Opis wyprowadzeń:

Nr	Nazwa	Opis
1	GND	Masa
2	VCC	Zasilanie +5V
3	SDA	Linia magistrali danych I2C
4	SCL	Linia zegarowa magistrali I2C

## ☑ Sensor ultradźwiękowy HC-SR04

Czujnik HC-SR04 używa sonaru do określania odległości do obiektu, podobnie jak robią to nietoperze lub delfiny. Oferuje on doskonałe wykrywanie odległości bez kontaktu, wysoką dokładność i stabilne odczyty. Zakres pomiarowy wynosi od 2 cm do 450 cm lub od 1 cala do 16 stóp. Jego działanie nie jest zaburzone przez światło słoneczne ani czarne materiały, tak jak jest to w przypadku czujników Sharp (choć akustycznie miękkie materiały, takie jak tkaniny, mogą być trudne do wykrycia).

### Cechy:

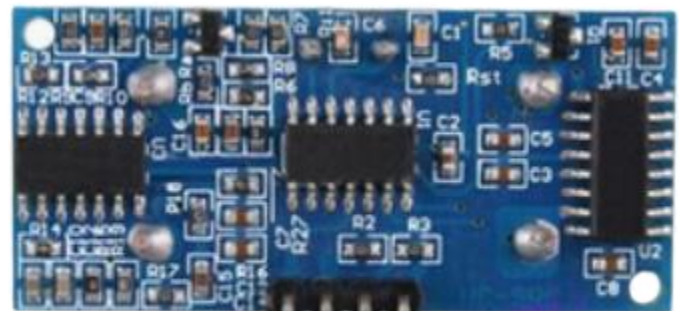
- ⌘ Zasilanie: +5V DC
- ⌘ Prąd spoczynkowy: <2 mA
- ⌘ Prąd roboczy: 15 mA
- ⌘ Kąt efektywny: <15°
- ⌘ Zakres pomiarowy: 2cm – 400cm / 1" – 13ft
- ⌘ Rozdzielczość: 0,3cm
- ⌘ Kąt pomiaru: 30°
- ⌘ Szerokość impulsu wejściowego wyzwania: 10μs
- ⌘ Wymiary: 45mm x 20mm x 15mm

Pomiar czasowy takiego czujnika rozpoczyna się w momencie otrzymania wysokiego impulsu (5V), który trwa co najmniej 10μs, co spowoduje, że sensor wyśle 8 cykli fali ultradźwiękowej o częstotliwości 40kHz i będzie oczekiwał na odbity sygnał ultradźwiękowy. Gdy wykryje ultradźwięki na odbiorniku, ustawia wyjście Echo w stan wysoki (5V) i oczekuje przez pewien czas (szerokość impulsu), który jest proporcjonalny do odległości, aby otrzymać odległość. Mierzmy szybkość (Ton) impulsu na pinie Echo.

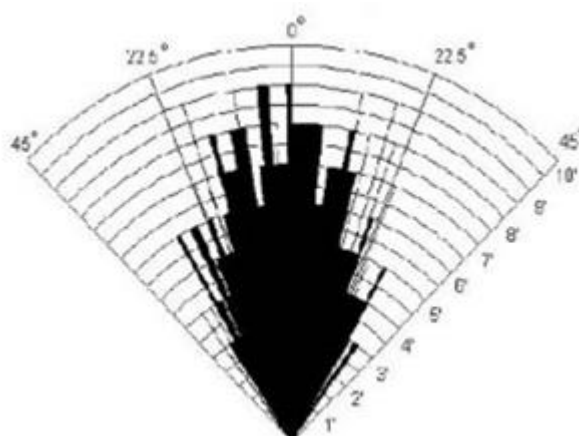
Czas = Szerokość impulsu Echo, w μs (mikrosekundy)

- ⌘ Odległość w centymetrach = Czas / 58
- ⌘ Odległość w calach = Czas / 148
- ⌘ Można również skorzystać ze znajomości prędkości dźwięku, która wynosi 340 m/s.

Parametr	Min	Avg	Max
Napięcie pracy	4,50 V	5,0 V	5,5 V
Prąd spoczynkowy	1,5 mA	2 mA	2,5 mA
Prąd roboczy	10 mA	15 mA	20 mA
Częstotliwość	40 kHz		



Sensor ultradźwiękowy HC-SR04





Moduł Arduino Nano

## ☑ Mikrokontroler Arduino® Nano

Arduino Nano to klasyczna płyta prototypowa firmy Arduino o najmniejszych wymiarach (45mm x 18mm). Arduino Nano jest wyposażone w złącza pinów umożliwiające łatwe podłączenie do płytki prototypowej.

### Specyfikacja:

<b>Płyta</b>	Nazwa	<a href="#">Arduino® Nano</a>
	SKU	A000005
<b>Mikrokontroler</b>	ATmega328	
<b>Podłączenie USB</b>	Mini-B USB	
<b>Piny</b>	Wbudowane piny LED	13
	Cyfrowe piny wejścia/wyjścia	14
	Analogowe piny wejścia	8
	PWM piny	6
<b>Komunikacja</b>	UART	RX/TX
	I2C	A4 (SDA), A5 (SCL)
	SPI	D11 (COPI), D12 (CIPO), D13 (SCK)
<b>Moc</b>	Napięcie wejścia/wyjścia	5V
	Nominalne napięcie wejścia	7V – 12V
	Prąd stały na pin wejścia/wyjścia	20mA
<b>Prędkość zegara</b>	Procesor	Atmega328 16MHz
<b>Pamięć</b>	Atmega328P	2KB SRAM, 32KB flash 1KB EEPROM
<b>Wymiary</b>	Waga	5g
	Szerokość	18mm
	Długość	45mm

### Kompatybilność:

Sam mikrokontroler kompatybilny jest z takimi środowiskami jak:



Arduino IDE



Arduino CLI



Web Editor

My tworząc grę na zajęciach z modułu Systemy Wbudowane korzystaliśmy z Arduino IDE.

## ☑ Procesor Atmega328

Parametr	Wartość
Typ pamięci programu	Flash
Rozmiar pamięci programu (KB)	32
Prędkość CPU (MIPS/DMIPS)	20
Pamięć danych EEPROM (bajty)	1024
Timery	2 x 8-bit - 1 x 16-bit
Samodzielne PWM	6
Liczba przetworników ADC	0
Liczba kanałów ADC	8
Maksymalna rozdzielczość ADC (bitów)	10
Liczba komparatorów	1
Zakres min. temperatury	-40
Zakres max. temperatury	85
Maksymalne napięcie zasilania (V)	5.5
Minimalne napięcie zasilania (V)	1.8
Liczba pinów	32
Niskie zużycie energii	Nie
I2C	1 – I2C

Procesor Atmega328



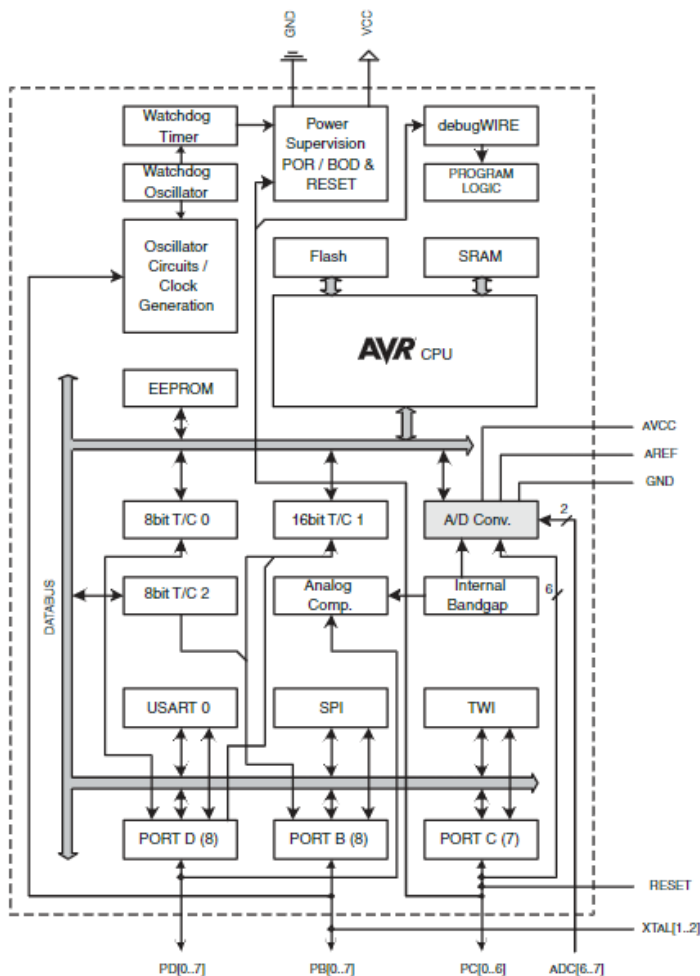
**ATmega328** to popularny mikrokontroler z rodziny AVR, produkowany przez firmę **Microchip Technology** (dawniej **Atmel Corporation**). Jest oparty na architekturze RISC z 8-bitowym rdzeniem AVR. ATmega328 obsługuje interfejsy komunikacyjne, takie jak UART, SPI i I2C, umożliwiając komunikację z innymi urządzeniami. Posiada 3 16-bitowe timery/komparatory, które mogą być wykorzystywane do generowania przerwań, mierzenia czasu i generowania sygnałów PWM. Procesor obsługuje również przerwania sprzętowe, które reagują na zdarzenia zewnętrzne.

Standardowe napięcie zasilania dla ATmega328 wynosi 5 V, ale może działać w zakresie od 1,8 V do 5,5 V. Istnieje również wersja zasilana napięciem 3,3 V (ATmega328P-3.3V).

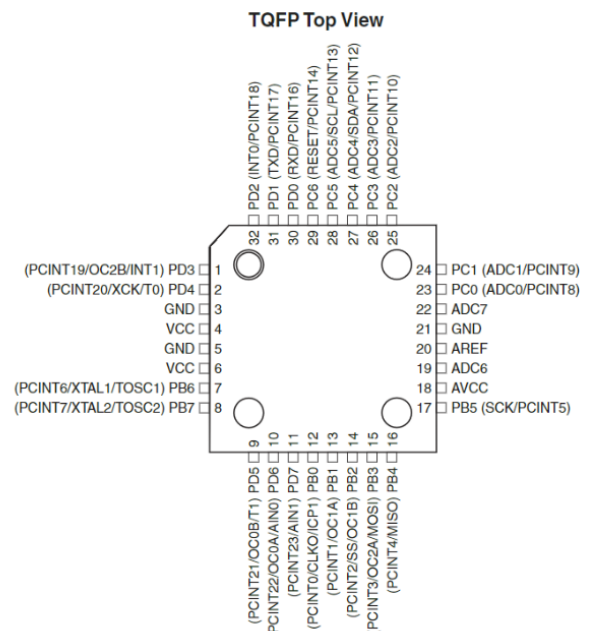
s

ATmega328 jest powszechnie stosowany w projektach elektronicznych, prototypowaniu, robotyce, automatyce, systemach wbudowanych i wielu innych dziedzinach ze względu na swoją prostotę, wszechstronność i dostępność.

Schemat blokowy procesora Atmega328



Widok z góry z opisem pinów procesora





## OPIS GRY

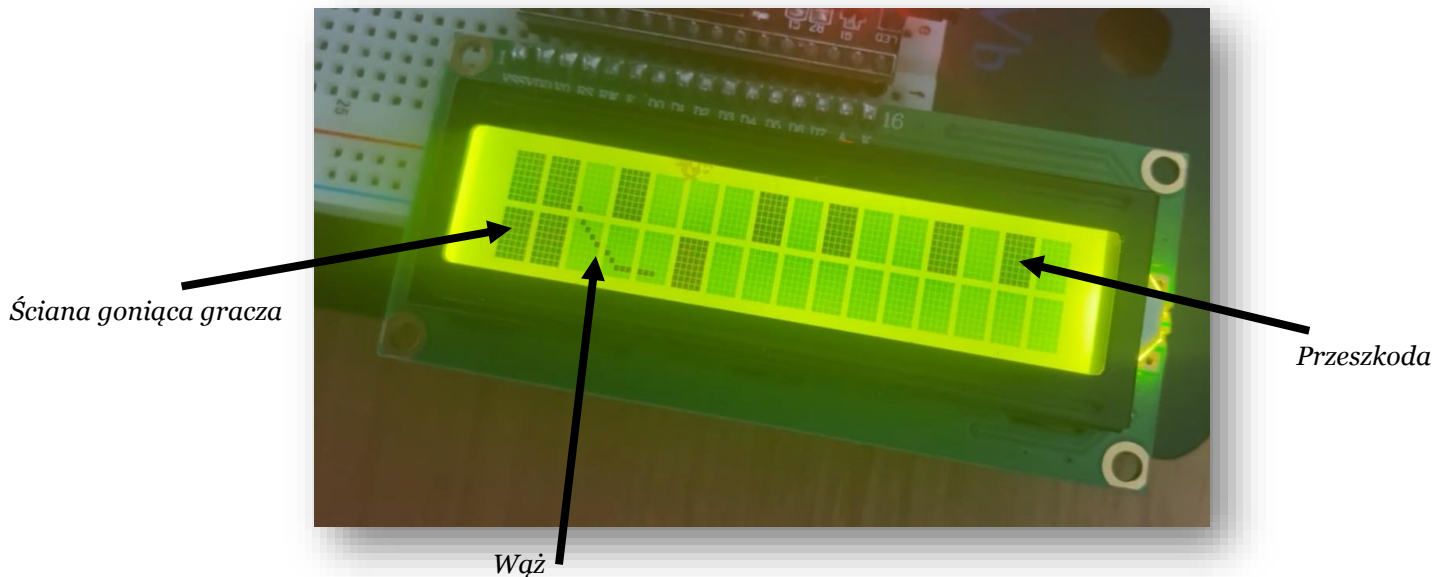
Gra "Cracked Flappy Bird" to wciągająca i ekscytująca zabawa, która wykorzystuje sensor ultradźwiękowy, wyświetlacz LCD 16x2 oraz mikrokontroler Arduino Nano. Celem gry jest prowadzenie węża przez kolejne etapy, pokonując przeszkody i zdobywając jak najwięcej punktów.

Na początku gry wąż pojawia się z lewej strony na środku ekranu. Gracz steruje wężem, poruszając go w górę lub w dół, przesuwając swoją dłoń lub jakimś przedmiotem w lewo lub w prawo (Lewo = Góra, Prawo = Dół).

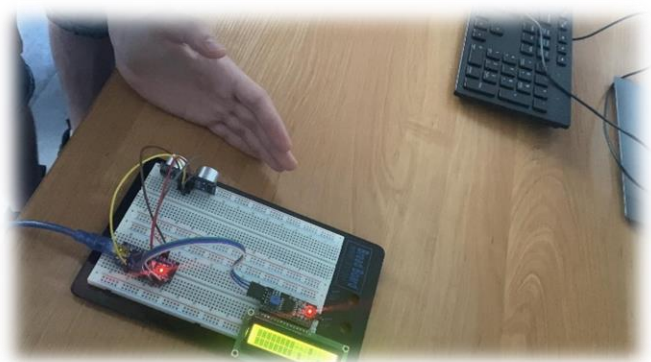
Wąż porusza się po planszy, która jest wyświetlana na ekranie LCD 16x2. Plansza składa się z kwadratowych pól, na których znajdują się przeszkody. Przeszkody są reprezentowane jako przeszkadzające obiekty, przez które wąż nie może przechodzić. Gracz musi unikać tych przeszkód, aby nie zderzyć się z nimi. Dodatkowo gracza goni przesuwająca się za nim ściana. Wąż po zderzeniu umiera, gracz traci wszystkie punkty i gra rozpoczyna się od nowa.

Wraz z postępem gracz zdobywa punkty za każdy pokonany etap. Punkty są wyświetlane na ekranie, umożliwiając graczowi śledzenie swojego postępu. Wraz z przejściem na kolejne etapy, wąż przyspiesza o 20% maksymalnego przyspieszenia, co czyni grę coraz bardziej wymagającą. Gracz musi utrzymać skupienie i zręczność, aby unikać przeszkód i zdobyć jak najwięcej punktów.

Gra "Cracked Flappy Bird" to wciągająca przygoda, która wymaga zręczności i koncentracji. Zastosowanie sensora ultradźwiękowego, wyświetlacza LCD 16x2 i mikrokontrolera Arduino Nano umożliwia interaktywność i sprawia, że gra jest jeszcze bardziej atrakcyjna.



Ekran zakończenia gry z wynikiem gracza




Przykład gracza sterującego dłonią

# DZIAŁANIE GRY

Gra została napisana w środowisku Arduino IDE 2.1.0.

Początkowo program zaczyna się od deklaracji wszystkich niezbędnych zmiennych, które potem będą wykorzystywane razem z importem biblioteki `LCD_I2C.h`, która potrzebna jest do obsłużenia wyświetlacza.



```
1  #include <LCD_I2C.h>
2
3  LCD_I2C lcd(0x3F, 16, 2);
4
5  const int trigPin = 2;
6  const int echoPin = A0;
7
8  long duration;
9  int distance;
10 int directionS = 1;
11
12 using byte = unsigned char;
13
14 int const DEFAULT_ROW_NUM = 2;
15 int const DEFAULT_COLUMN_NUM = 16;
16 int const DEFAULT_LINE_NUM = 8;
17
18 int gameState = 2; // 0 - przegrana, 1 - wygrana, 2 - gra idzie dalej
19
20 bool bombs[DEFAULT_ROW_NUM][DEFAULT_COLUMN_NUM + 2];
21
22 int nextByte;
23 int currentRow;
24 int currentColumn;
25 int currentLine;
26 int velocity = 0;
27 int points = 0;
```



Z racji ograniczeń pamięciowych kontrolera *LCD\_I2C* dopuszcza się maksymalnie do 8 znaków własnych, które są zapamiętywane na odpowiednie pozycje na wyświetlaczu oraz do pamięci kontrolera wyświetlacza. Takie rozwiązanie pozwala na dynamiczne zarządzanie polami na których dany znak się znajduje. Z racji wystąpienia tego ograniczenia oraz potrzeby czytelności kodu deklaracja tych znaków w funkcjonalnej, bądź pustej formie została umieszczona w początkowej części kodu.

```
1 //deklaracja początkowa tabel znaków
2 uint8_t clean[8] =
3 {
4     0b000000,
5     0b000000,
6     0b000000,
7     0b000000,
8     0b000000,
9     0b000000,
10    0b000000,
11    0b000000
12 };
13 uint8_t full[8] =
14 {
15     0b111111,
16     0b111111,
17     0b111111,
18     0b111111,
19     0b111111,
20     0b111111,
21     0b111111,
22     0b111111
23 };
24 uint8_t snake_top_first[8] =
25 {
26     0b000000,
27     0b000000,
28     0b000000,
29     0b000000,
30     0b000000,
31     0b000000,
32     0b000000,
33     0b100000
34 };
35 uint8_t snake_down_first[8] =
36 {
37     0b000000,
38     0b000000,
39     0b000000,
40     0b000000,
41     0b000000,
42     0b000000,
43     0b000000,
44     0b000000
45 };
```

```
1 uint8_t snake_top_second[8] =
2 {
3     0b000000,
4     0b000000,
5     0b000000,
6     0b000000,
7     0b000000,
8     0b000000,
9     0b000000,
10    0b000000
11 };
12 uint8_t snake_down_second[8] =
13 {
14     0b000000,
15     0b000000,
16     0b000000,
17     0b000000,
18     0b000000,
19     0b000000,
20     0b000000,
21     0b000000
22 };
23 uint8_t snake_top_last[8] =
24 {
25     0b000000,
26     0b000000,
27     0b000000,
28     0b000000,
29     0b000000,
30     0b000000,
31     0b000000,
32     0b000000
33 };
34 uint8_t snake_down_last[8] =
35 {
36     0b000000,
37     0b000000,
38     0b000000,
39     0b000000,
40     0b000000,
41     0b000000,
42     0b000000,
43     0b000000
44 };
```

Potem mamy funkcję `void setup()`, która rozpoczyna pracę z wyświetlaczem LCD, ustawia `trigPin` jako wyjście oraz `echoPin` jako wejście i co najważniejsze używamy naszych tabel znaków do wygenerowania znaków na ekranie.

```
1  void setup() {
2      lcd.begin();
3      lcd.backlight();
4
5      pinMode(trigPin, OUTPUT); // Ustawia trigPin jako wyjście
6      pinMode(echoPin, INPUT); // Ustawia echoPin jako wejście
7
8      //uruchomienie zadeklarowanych tabel znaków jako znaki
9      lcd.createChar(1, clean);
10     lcd.createChar(2, full);
11     lcd.createChar(3, snake_top_first);
12     lcd.createChar(4, snake_down_first);
13     lcd.createChar(5, snake_top_second);
14     lcd.createChar(6, snake_down_second);
15     lcd.createChar(7, snake_top_last);
16     lcd.createChar(8, snake_down_last);
17 }
```

## Funkcje generacyjne:

```
1  //-----
2  // funkcje generacyjne
3  //-----
```

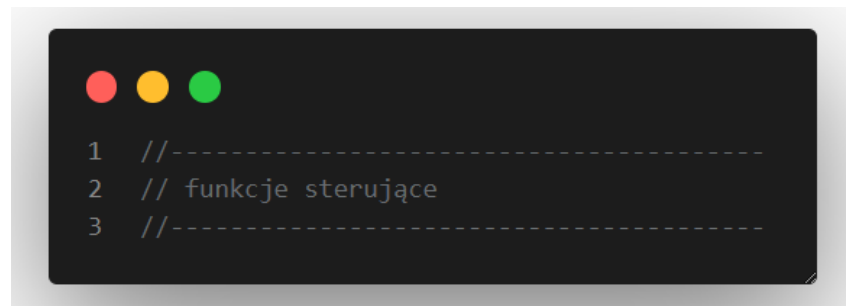
1. Pierwszą z nich jest `void reset()`, która odpowiada za tworzenie pierwszego i kolejnych etapów gry. Generowanie przeszkód oraz resetowanie ścieżki naszego węża.

```
1 void reset() // tworzenie pierwszego i kolejnych ekranów(poziomów) gry
2 {
3     generateBombs();
4
5     // reset ścieżki
6     for (int i = 0; i < DEFAULT_ROW_NUM; i++) {
7         for (int j = 0; j < DEFAULT_COLUMN_NUM; j++) {
8             if (bombs[i][j])
9             {
10                lcd.setCursor(j, i);
11                lcd.write(2);
12            }
13            else
14            {
15                lcd.setCursor(j, i);
16                lcd.write(1);
17            }
18        }
19    }
20
21    // reset wartości
22    nextByte = 16;
23
24    currentRow = 0;
25    currentColumn = 0;
26    currentLine = DEFAULT_LINE_NUM - 1;
27    lcd.setCursor(currentRow, currentColumn);
28    lcd.write(3);
29 }
```

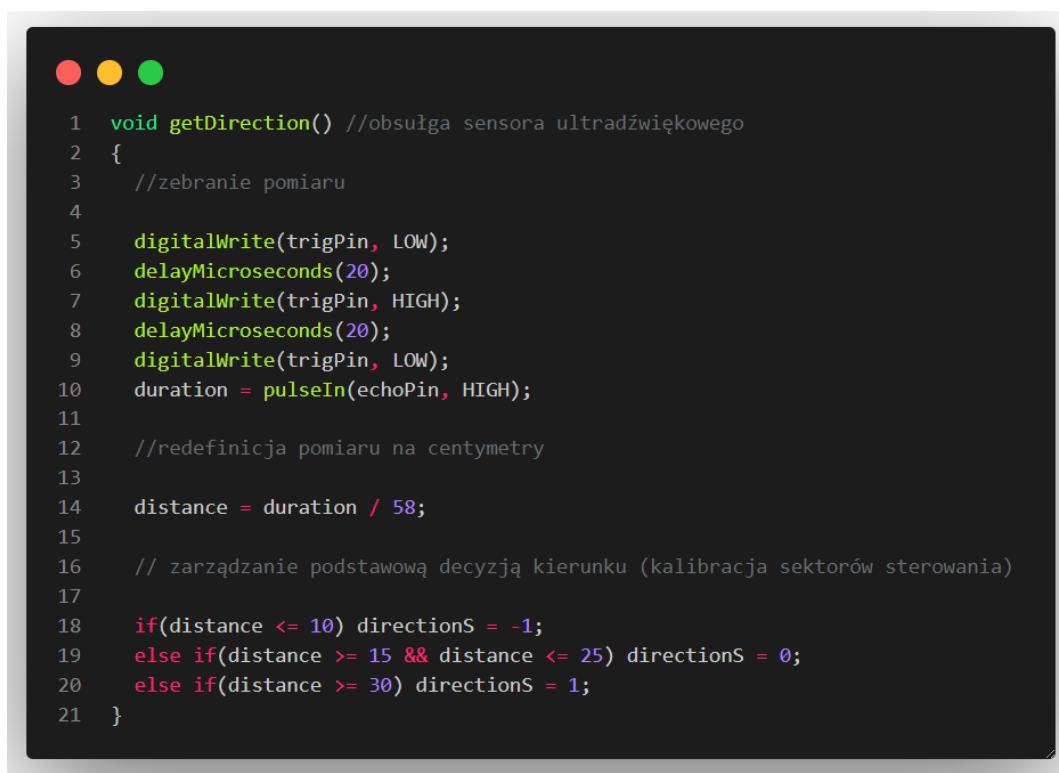
2. Drugą jest zarazem właśnie generacja bomb / ścian, która została użyta już w funkcji wyżej.

```
1 void generateBombs()
2 // generacja przeszkód(Bomb/ścian)
3 // jako tablica boolowska, która pełni rolę funkcji sprawdzającej
4 {
5     int x;
6     int y;
7     int chance;
8     bombs[0][0] = 0;
9     bombs[0][1] = 0;
10    bombs[1][0] = 0;
11    bombs[1][1] = 0;
12
13    for (x = 2; x < DEFAULT_COLUMN_NUM; x++)
14    {
15        chance = rand() % 3;
16        if (chance == 0)
17        {
18            bombs[0][x] = 1;
19            bombs[1][x] = 0;
20            bombs[0][x + 1] = 0;
21            bombs[1][x + 1] = 0;
22            ++x;
23        }
24        else if (chance == 1)
25        {
26            bombs[0][x] = 0;
27            bombs[1][x] = 1;
28            bombs[0][x + 1] = 0;
29            bombs[1][x + 1] = 0;
30            ++x;
31        }
32        else
33        {
34            bombs[0][x] = 0;
35            bombs[1][x] = 0;
36        }
37    }
38 }
39
```

## Funkcje sterujące:



1. Pierwszą i zarazem bardzo ważną funkcją jest `void getDirection()`, która obsługuje sensor ultradźwiękowy HC-SR04. Zbiera pomiar odległości i zamienia ją na jednostkę centymetra oraz zarządza decyzją kierunku, w którym wąż ma się poruszać w zależności od zbadanego wcześniej pomiaru odległości.



2. Kolejną i ostatnią z rodziny funkcji sterujących jest `void goAnywhere()`, która przyjmuje jako parametr wartości całkowite zmiennej `directionS` odpowiadającej za kierunek ruchu węża. Cała funkcja odpowiada za cykl rysowania naszego bohatera gry oraz działa jako kontroler decyzyjny ruchu.

```
1 void goAnywhere(int directionS)
2 // kontroler decyzyjny ruchu i dopasowania wartości/cyklu rysowania
3 {
4     nextByte /= 2;
5
6     if (nextByte == 0) {
7         nextByte = 16;
8         moveChainOfSnake();
9         currentColumn++;
10
11         if (currentColumn > 2) {
12             hideColumn(currentColumn - 3);
13         }
14
15         if (isWin()) {
16             gameState = 1;
17             return;
18         }
19     }
20
21     currentLine = currentLine - directionS;
22
23     if (directionS == 1)
24     {
25         if (currentLine < 0) {
26             currentLine = DEFAULT_LINE_NUM - 1;
27             currentRow--;
28         }
29     }
30     else if (directionS == -1)
31     {
32         if (currentLine > DEFAULT_LINE_NUM - 1) {
33             currentLine = 0;
34             currentRow++;
35         }
36     }
37
38     if (isLose()) {
39         gameState = 0;
40         return;
41     }
42     if (currentRow==0)
43     {
44         snake_top_first[currentLine] += nextByte;
45         printSnakeHead();
46     }
47     else
48     {
49         snake_down_first[currentLine] += nextByte;
50         printSnakeHead();
51     }
52 }
```

## Funkcje sprawdzające:



```
1 //-----  
2 // funkcje sprawdzające  
3 //-----
```

1. Funkcja `isWin()` sprawdza czy wąż przeszedł na drugą stronę ekranu (czy ukończył poziom).



```
1 bool isWin() // sprawdzenie czy wąż ukończył poziom(prawostronę ekranu)  
2 {  
3     return (currentColumn > DEFAULT_COLUMN_NUM - 1);  
4 }
```

2. Funkcja `isLose()` analogicznie do pierwszej sprawdza czy nastąpiła kolizja węża z jakąś przeszkodą znajdującą się na ekranie bądź z górną lub dolną ścianą.



```
1 bool islose() // sprawdzenie czy nastąpiła kolizja węża z przeszkodą/ścianą  
2 {  
3     return (currentRow < 0 || currentRow > DEFAULT_ROW_NUM - 1 || bombs[currentRow][currentColumn]);  
4 }  
5
```



## Funkcje rysowania i pamięciowe węże:



```
1 //-----  
2 // funkcje rysowania i zarządzania pamięcią znaków węża  
3 //-----
```

1. Funkcja `hideColumn()` odpowiada za chowanie kolumn za wężem, który idzie do przodu (to jest ta ściana, która za nim podąża).



```
1 void hideColumn(int columnNum) // chowanie kolumn za wężem  
2 {  
3     lcd.setCursor(columnNum, 0);  
4     lcd.write(2);  
5     lcd.setCursor(columnNum, 1);  
6     lcd.write(2);  
7 }
```

2. Funkcja `moveChainOfSnake()` pokazuje jak przebiega proces rysowania i aktualizowania węża podczas trwania gry.



```
1 void moveChainOfSnake() //zarządzanie pełnego cyklu  
2 {  
3     copySnake();  
4     clearSnake();  
5     updateSnake();  
6     printSnakeHead();  
7     printSnakeTail();  
8 }
```

3. Funkcja `copySnake()` odpowiada za przesunięcie pamięci w pełnym cyklu, który znajduje się w funkcji `moveChainOfSnake()`.



```
1 void copySnake() //przesunięcie pamięci w cyklu pełnym(zarządzanie znakami)  
2 {  
3     memcpy(snake_top_last, snake_top_second, sizeof clean);  
4     memcpy(snake_down_last, snake_down_second, sizeof clean);  
5     memcpy(snake_top_second, snake_top_first, sizeof clean);  
6     memcpy(snake_down_second, snake_down_first, sizeof clean);  
7 }
```

4. Funkcja `clearSnake()` czyści aktywną część węża na początku pełnego cyklu.

```
1 void clearSnake() //czyszczenie aktywnej części węża na początku cyklu pełnego(zarządzanie znakami)
2 {
3     memcpy(snake_top_first, clean, sizeof clean);
4     memcpy(snake_down_first, clean, sizeof clean);
5 }
```

5. Funkcja `updateSnake()` redefiniuje znaki w pełnym cyklu na podstawie tabel znakowych, które definiowaliśmy na samym początku programu.

```
1 void updateSnake() //redefinicja znaków w pełnym cyklu na podstawie tabel znakowych
2 {
3     lcd.createChar(3, snake_top_first);
4     lcd.createChar(4, snake_down_first);
5     lcd.createChar(5, snake_top_second);
6     lcd.createChar(6, snake_down_second);
7     lcd.createChar(7, snake_top_last);
8     lcd.createChar(8, snake_down_last);
9 }
```

6. Funkcja `printSnakeHead()` jak sama nazwa wskazuje rysuje głowę węża, czyli aktywną jego część.

```
1 void printSnakeHead() //rysowanie głowy(aktywnej części)węża
2 {
3     lcd.createChar(3, snake_top_first);
4     lcd.createChar(4, snake_down_first);
5
6     if (!bombs[0][currentColumn])
7     {
8         lcd.setCursor(currentColumn, 0);
9         lcd.write(3);
10    }
11
12    if (!bombs[1][currentColumn])
13    {
14        lcd.setCursor(currentColumn, 1);
15        lcd.write(4);
16    }
17 }
```

7. Funkcja `printSnakeTail()` analogicznie do głowy rysuję ogon, lecz jest to pasywna część węża.

```
1 void printSnakeTail() //rysowanie ogona(pasywnej części) węża
2 {
3     if (!bombs[0][currentColumn])
4     {
5         lcd.setCursor(currentColumn, 0);
6         lcd.write(5);
7     }
8     if (!bombs[1][currentColumn])
9     {
10        lcd.setCursor(currentColumn, 1);
11        lcd.write(6);
12    }
13    if (!bombs[0][currentColumn - 1] && (currentColumn > 0))
14    {
15        lcd.setCursor(currentColumn - 1, 0);
16        lcd.write(7);
17    }
18    if (!bombs[1][currentColumn - 1] && (currentColumn > 0))
19    {
20        lcd.setCursor(currentColumn - 1, 1);
21        lcd.write(8);
22    }
23 }
```

## Funkcje końcowe gry / poziomu gry:

```
1 //-----
2 // funkcje końcowe gry/poziomu
3 //-----
```

1. Funkcja `speedUP()` odpowiada za zwiększanie prędkości węża wraz z progresywnym przechodzeniem poziomów.

```
1 void speedUP() //zarządzanie progresywnym(ograniczonym) poziomem trudności
2 {
3     if(velocity < 200) velocity += 40;
4 }
```

2. Funkcja `gameend()` pokazuje nam ekran końcowy gry z wynikiem gracza.

```
1 void gameend() //ekran końca gry
2 {
3     lcd.clear();
4     epilepsy();
5     lcd.setCursor(3, 0);
6     lcd.print("YOU LOSE!");
7     lcd.setCursor(3, 1);
8     lcd.print("SCORE: ");
9     lcd.print(points);
10    delay(1000);
11    points = 0;
12    clearSnake();
13    velocity = 0;
14 }
```

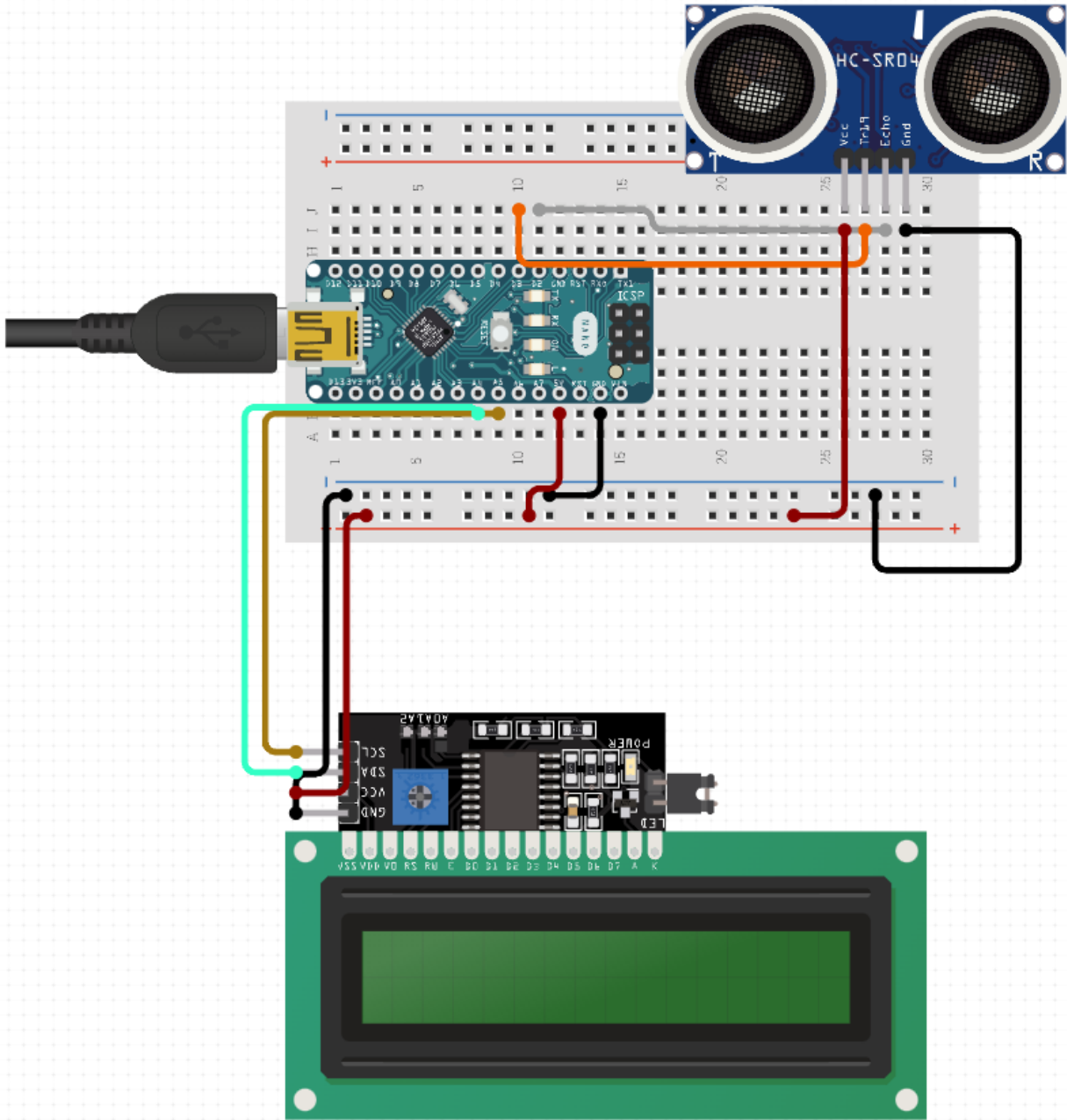
3. Funkcja `epilepsy()` tworzy ciekawy efekt na wyświetlaczu po przegranej.

```
1 void epilepsy() //efekt ekranu końcowego
2 {
3     for (int i = 0; i < 7; ++i)
4     {
5         lcd.backlight();
6         delay(20);
7         lcd.noBacklight();
8         delay(20);
9     }
10    lcd.backlight();
11 }
```

### Pętla programu:

```
1 //-----
2 //pętla programu
3 //-----
4
5 void loop()
6 {
7     reset(); //początek poziomu
8     gameState = 2;
9     while (gameState == 2) //pętla gry
10    {
11        getDirection();
12        delay(200 - velocity);
13        goAnywhere(directionS);
14    }
15    if (gameState == 0) //koniec gry
16    {
17        gameend();
18    }
19    else //przejdźcie na następny poziom
20    {
21        speedUP();
22        points++;
23    }
24 }
```

SCHEMAT PROJEKTU



## PODSUMOWANIE I WNIOSKI

Realizacja rzeczywista projektu gry na zajęciach przebiegła sprawnie, wliczając w to wstawianie odpowiednich poprawek błędów i konfiguracji kontroli.

Gra ta sprawiła nam niezmierną radość jako projekt, a sam rezultat uzyskany poprzez wykorzystane i obmyślane przez nas rozwiązania wobec napotkanych problemów zagwarantował nam usatysfakcjonowanie z jej wykonania