

# Wprowadzenie do programowania urządzeń Arduino

(Arduino dla Informatyków)



Zbiór zadań dla bloku obieralnego  
**Inteligentne Systemy Autonomiczne**

Autor: **Piotr Duch, Tomasz Jaworski**  
Instytut Informatyki Stosowanej  
Politechnika Łódzka

Kontakt: [pduch@is.p.lodz.pl](mailto:pduch@is.p.lodz.pl), [tjaworski@iis.p.lodz.pl](mailto:tjaworski@iis.p.lodz.pl)

# **Programowanie systemów autonomicznych**

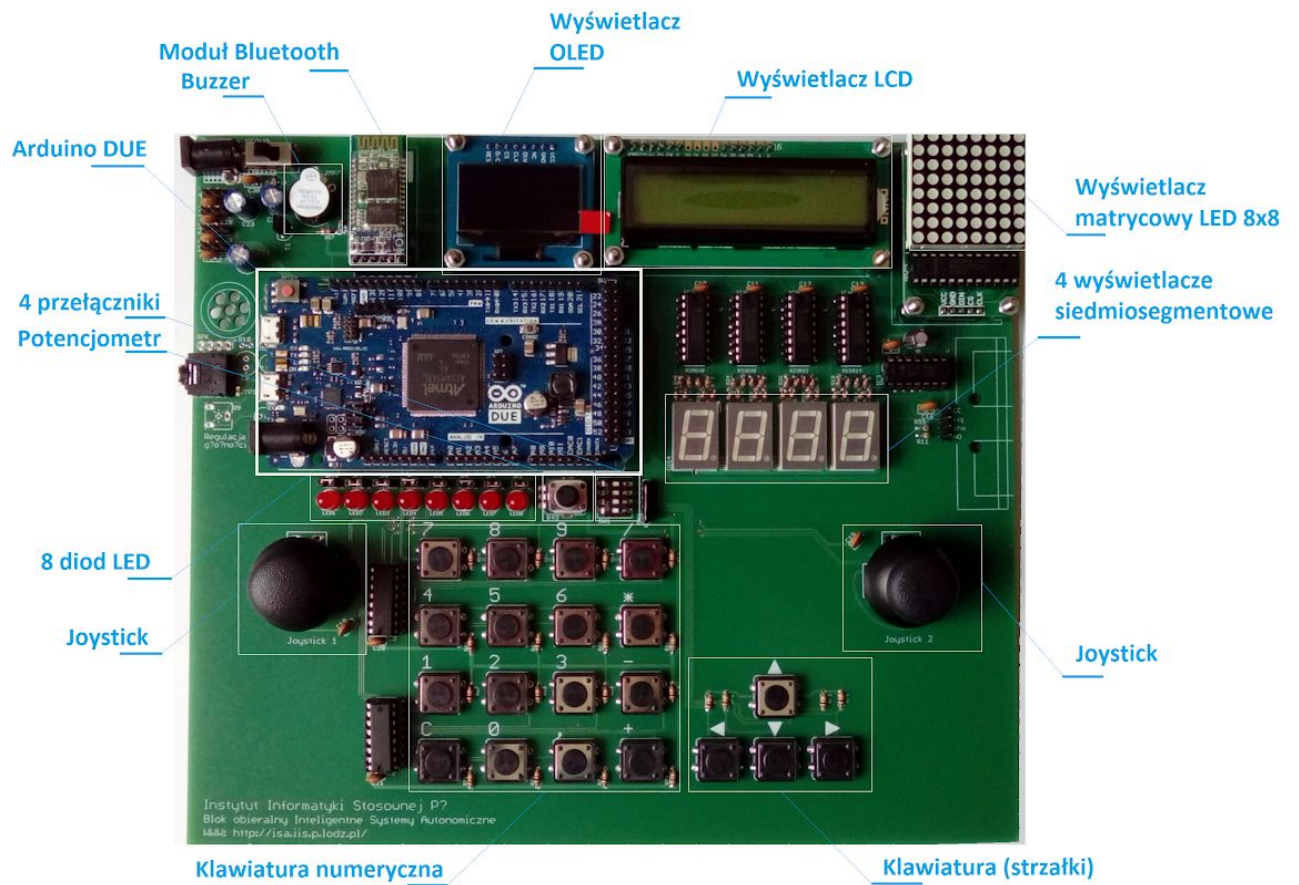
Inteligentne systemy autonomiczne

# Wstęp

Skrypt ten jest przeznaczony dla studentów 3 roku studiów inżynierskich, którzy wybrali blok Inteligentne Systemy Autonomiczne na kierunku Informatyka, na wydziale Elektrotechniki, Elektroniki, Automatyki i Informatyki Politechniki Łódzkiej. Celem zajęć realizowanych w ramach przedmiotu “Programowanie systemów autonomicznych” jest uzyskanie umiejętności programowania mikrokontrolerów, które będą nadzorowały pracę urządzeń autonomicznych, w szczególności, posiadających możliwości samodzielnego poruszania się. Programowanie będzie realizowane w języku C, a wartością dodaną w stosunku do programowania klasycznych komputerów, będzie objęcie nadzorem urządzeń stanowiących źródła informacji dla systemu i urządzeń umożliwiających wykonywanie przez system odpowiednich czynności. Zajęcia są podzielone na trzy części. Początkowe ćwiczenia mają na celu przedstawienie platformy sprzętowej, stanowiącej pierwszy filar obliczeniowy budowanych systemów inteligentnych - mikrokontroler Arduino Due (drugi - Raspberry Pi zostanie wprowadzony w ramach przedmiotu ‘Rozumienie obrazów i sygnałów’). W trakcie tej części Student będzie pracował na stanowisku laboratoryjnym, które zostało wyposażone w różnorodne układy wejścia/wyjścia, takie jak m.in.: diody, przyciski, wyświetlacze LCD oraz OLED. W części drugiej zajęć przedstawione zostaną sensory i efektory, stanowiące wyposażenie autonomicznych pojazdów, niezbędne dla realizacji projektu końcowego. Ostatnia część zajęć zostanie przeznaczona na samodzielną realizację jednego z dwóch (do wyboru) projektów końcowych: pojazdu poruszającego się samodzielnie po określonej ścieżce lub pojazdu samodzielnie jeżdżącego w stałej odległości od ściany.

# Opis stanowiska laboratoryjnego

## Opis stanowiska laboratoryjnego



*Ilustracja: Zdjęcie stanowiska laboratoryjnego.*

# Część 1 - Wprowadzenie do Arduino

## Konfiguracja środowiska Arduino IDE

Arduino IDE jest to rozbudowane środowisko programistyczne, przeznaczone do pisania kodu na mikrokontrolery jednoukładowe otwartego projektu Arduino. Arduino IDE może działać na różnych systemach operacyjnych: Windows, Mac OS oraz Linux.

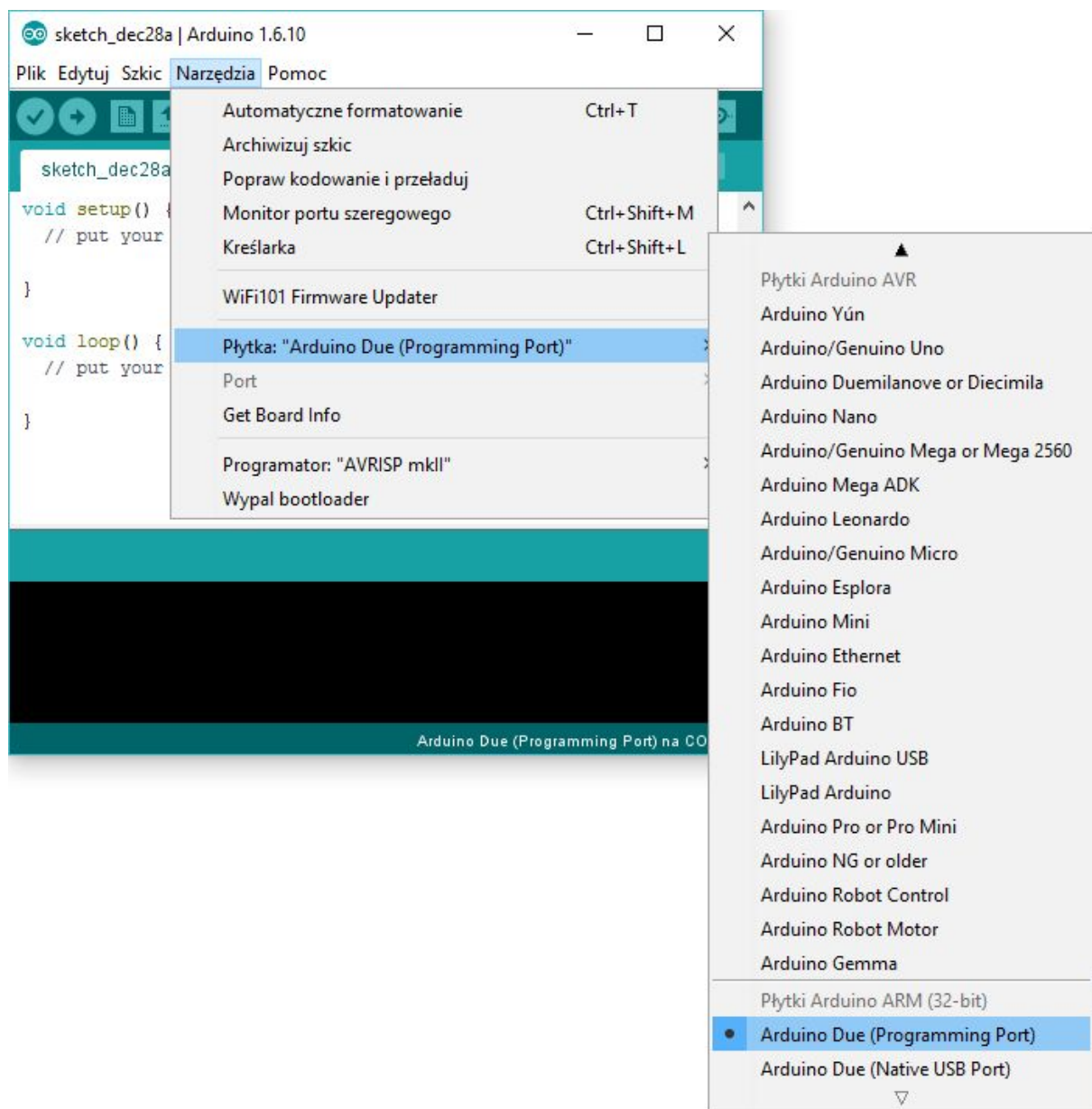
### Konfiguracja środowiska

Pierwszym krokiem jest ściągnięcie najnowszej wersji środowiska Arduino IDE ze strony Arduino. Następnym krokiem jest zainstalowanie środowiska programistycznego dla mikrokontrolera Arduino DUE. W tym celu należy otworzyć Menedżer płytek (Narzędzia -> Płytki -> Menedżer płytek) i następnie wybrać i zainstalować "Arduino SAM Boards".



### Wybór płytki

Po zainstalowaniu środowiska dla Arduino DUE należy ustawić je jako aktywne. W tym celu z podmenu Narzędzia -> Płytki należy wybrać Arduino Due (Programming Port).

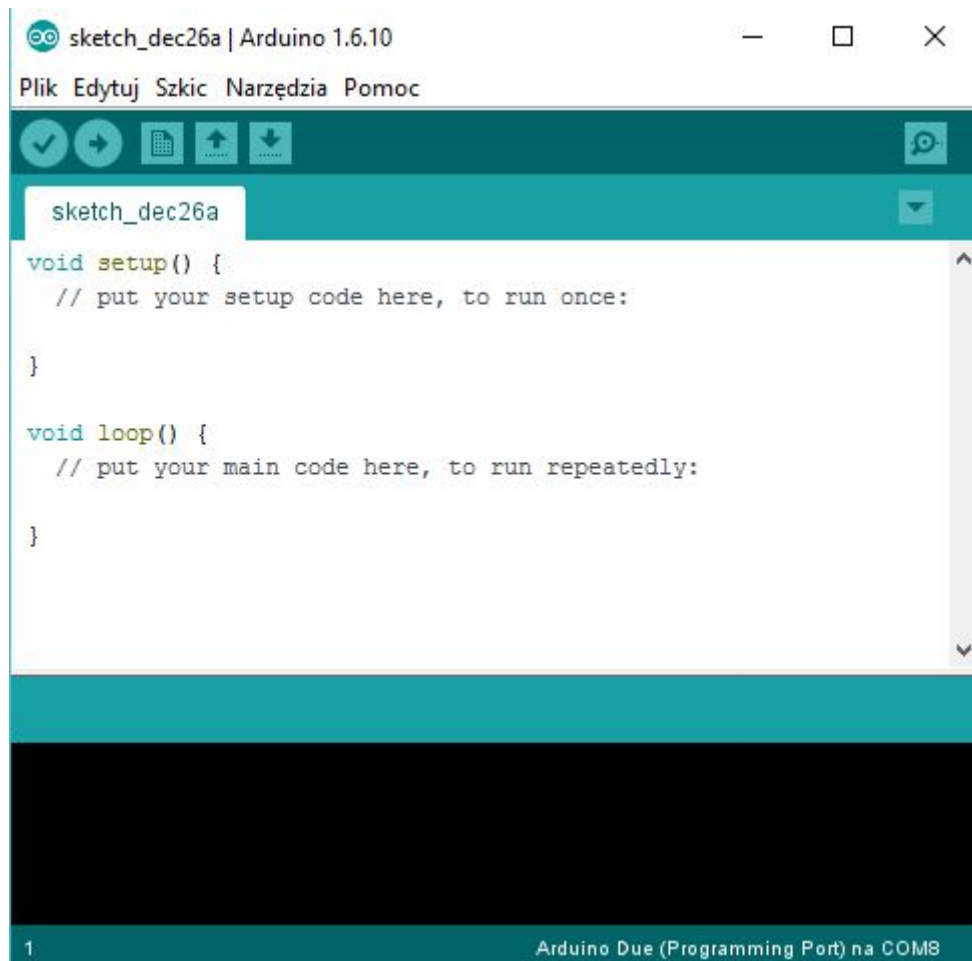


## Wybór portu

Z podmenu Narzędzia -> Port należy wybrać port USB, do którego podłączona jest płytki Arduino DUE.

## Pierwsze uruchomienie



Po uruchomieniu środowiska otwarty zostanie poprzednio zamknięty szkic projektu, a w przypadku gdy jest to pierwsze uruchomienie zostanie automatycznie utworzony nowy, pusty szkic projektu.



*Ilustracja 1: Pusty szkic programu.*

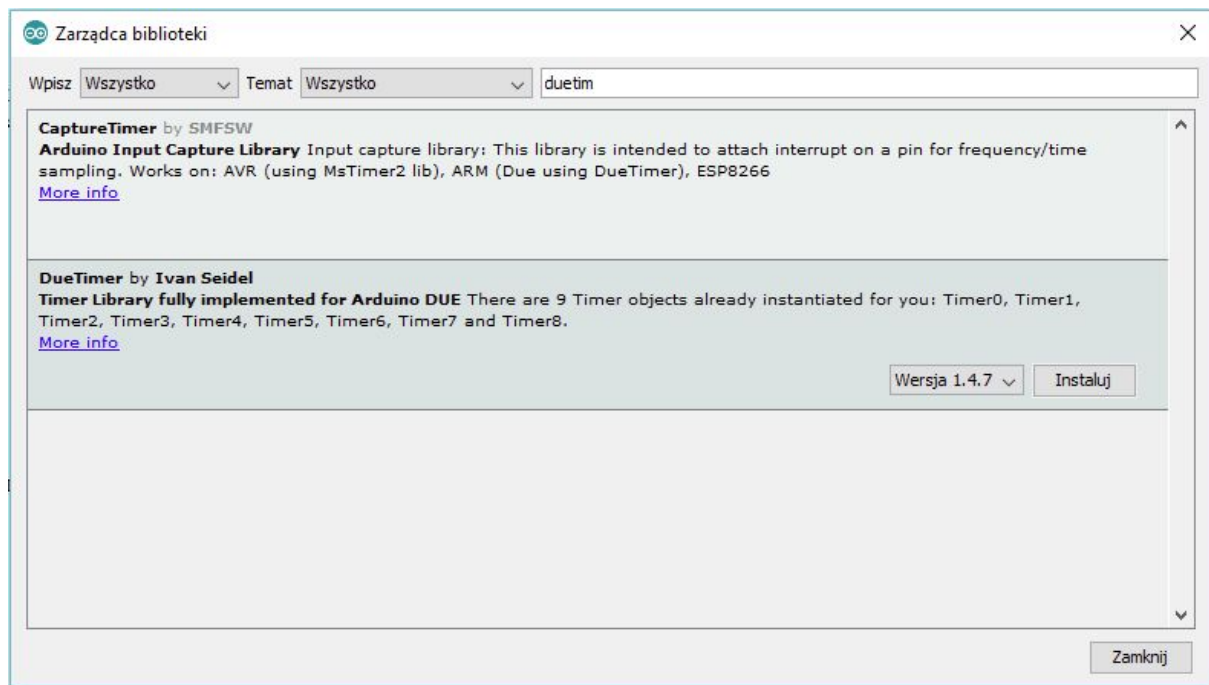
W każdym programie napisanym na Arduino musi być funkcja *setup()*. Funkcja ta jest wywoływana w momencie uruchomienia się programu. Celem tej funkcji jest zainicjowanie środowiska, w którym będzie uruchamiany program. Drugą funkcją, która musi być zawsze w programie jest funkcja *loop()*. Funkcja ta jest wywoływana raz za razem przez cały czas działania programu, w nieskończoność.

Kompilacja i uruchomienie programu:

W celu skompilowania programu należy nacisnąć przycisk zweryfikuj () , a wgrania programu na płytkę Arduino przycisk wgraj ().

Instalacja bibliotek Arduino:

W celu zainstalowania biblioteki wybieramy Szkic -> Dołącz bibliotekę -> Zarządzaj bibliotekami:



Następnie wybieramy odpowiednią bibliotekę i naciskamy przycisk **Instaluj**.



## Zadanie 1.1 Wstęp do Arduino

Cel ćwiczenia: Zapoznanie się z podstawowymi funkcjami Arduino oraz diodami LED.

Przydatne funkcje:

*delay(unsigned long ms)*<sup>1</sup> - umożliwia zatrzymanie aktualnie wykonywanego programu na określony czas. *ms* - czas w milisekundach, na jaki ma zostać zatrzymany program.

*pinMode(pin, mode)*<sup>2</sup> - umożliwia określenie trybu pracy poszczególnych pinów. *pin* - numer pinu, którego tryb pracy ma być ustawiony, *mode* - tryb pracy pinu, może przyjąć jedną z dwóch wartości: **OUTPUT** lub **INPUT**:

- Stała **INPUT** oznacza, że pin będzie pracował w trybie wejścia, co pozwoli uruchomionemu programowi wczytywać informacje od dołączonego do pinu *pin* urządzenia (np. przycisk strzałki w górę).
- Stała **OUTPUT** oznacza, że pin będzie pracował w trybie wyjścia, co pozwoli uruchomionemu programowi na sterowanie urządzeniem podłączonym do danego pinu, np. włączaniem/wyłączaniem brzęczyka lub diody LED.

*digitalWrite(pin, value)*<sup>3</sup> - umożliwia ustawienie stanu pinu. *pin* - numer pinu, którego stan ma zostać ustawiony, *value* - stan pinu, może przyjąć jedną z dwóch wartości: **HIGH** lub **LOW**.

- Stała **HIGH** odpowiada logicznej wartości prawdy (1) w języku C (np. brzęczyk włączony).
- Stała **LOW** odpowiada logicznej wartości fałszu (0) w języku C (np. brzęczyk wyciszony).

Stanowisko laboratoryjne:

Płytkę edukacyjną posiada osiem diod LED, oznaczonych symbolami **LED1**, **LED2**, ..., **LED7**, **LED8**. Dostępna jest również tablica **LEDS[]**, pozwalająca na odwoływanie się do odpowiedniej diody za pomocą wyrażenia indeksującego, np. kod *int i = 4; LEDS[i]* oznacza diodę **LED5**. Wszystkie te symbole dostępne są po dołączeniu pliku nagłówkowego **ISADefinitions.h**.

---

<sup>1</sup> <https://www.arduino.cc/en/Reference/Delay>

<sup>2</sup> <https://www.arduino.cc/en/Reference/pinMode>

<sup>3</sup> <https://www.arduino.cc/en/Reference/DigitalWrite>



*Ilustracja: Położenie diod na stanowisku laboratoryjnym oraz numery pinów, do których diody są podłączone.*

Przykład - program włączający diodę LED1:

```
#include <ISADefinitions.h>

void setup(){
    pinMode(LED1, OUTPUT);
    digitalWrite(LED1, HIGH);
}

void loop(){}
```

### Zadanie 1.1.1 Funkcja setup

Opis ćwiczenia:

Napisz program, który zapali, a następnie po upływie 5 sekund zgasi diodę LED. W funkcji *setup()* ustaw tryb pracy jednego z pinów, do których podłączona jest dioda na **OUTPUT**. Następnie ustaw stan tego pinu na **HIGH**, a po upływie 5 sekund (wykorzystaj funkcję *delay()*) na **LOW**.

### Zadanie 1.1.2 - Funkcja loop

Opis ćwiczenia:

Napisz program, który będzie naprzemiennie zapalał i gasił diodę LED. W funkcji *setup()* ustaw tryb pracy jednego z pinów, do których podłączona jest dioda na **OUTPUT**. Następnie w funkcji *loop()* ustaw stan tego pinu na **HIGH**, a po upływie 1 sekundy na **LOW**. Zmodyfikuj program w taki sposób, żeby czas, w którym dioda się pali był dwa razy dłuższy niż czas kiedy jest wyłączona.

### Zadanie 1.1.3 - Wizualizacja bajtu

Opis ćwiczenia:

Napisz program umożliwiający wyświetlanie zawartości jednego bajtu (typ unsigned char) za pomocą przygotowanych ośmiu diod LED. Napisz funkcję, która będzie przyjmowała liczbę, a następnie wyświetlała jej wartość w systemie binarnym na diodach. Wykorzystaj fakt, że diod jest osiem tak samo, jak bitów w bajcie. Dowolnie wybierz, która dioda odpowiada za bit najstarszy, a która za najmłodszy.

### Zadanie 1.1.4 - Stoper

Opis ćwiczenia:

Napisz programu, który będzie wyświetlał w systemie binarnym, co sekundę, czas jaki minął od uruchomienia programu za pomocą diod LED.

### Zadanie 1.1.5 - Efekty świetlne

Opis ćwiczenia:

Podziel diody LED na pary, a następnie napisz program, który będzie migał każdą z par diod ze stałą losowo wybraną częstotliwością oraz wypełnieniem.

## Zadanie 1.2 - Timery i przerwania

Cel ćwiczenia: Zapoznanie się z wykorzystaniem timerów i ich przerwań.

### Trochę teorii:

Wykorzystując funkcję *delay()* realizujesz tzw. aktywne oczekiwanie – mikroprocesor jest zajęty wytracaniem czasu, np. poprzez wykonywanie w pętli pewnego fragmentu kodu, co do którego dokładnie znany jest czas potrzebny na jego wykonanie.

Lepszym podejściem jest wykorzystanie timerów generujących przerwania. Timer jest formą stopera, który po zaprogramowaniu odlicza czas od pewnej zadanej wartości do zera. W chwili osiągnięcia zera, timer wysyła do mikroprocesora żądanie przerwania.

Przerwanie to sytuacja, w której mikroprocesor zawiesza (wstrzymuje) wykonywanie danego kodu A po to, aby wykonać inny kod B. Po zakończeniu wykonywania kodu B mikroprocesor wznawia wykonywanie kodu A od miejsca, w którym przerwał. Na czas przerwania stan mikroprocesora jest zapamiętywany. Z każdym przerwaniem skojarzony jest kod obsługi tego przerwania, zwany kodem ISR (ang. *Interrupt Service Routine*). Ponieważ przerwania mogą być generowane nie tylko przez timery ale i inne zdarzenia, pojawiające się bardzo często, kody tych przerwań muszą być możliwie krótkie. W funkcjach wywoływanych przez przerwania nie można stosować funkcji *delay()*.

### Przydatne funkcje:

Biblioteka **DueTimer**<sup>4</sup> daje dostęp do obsługi wszystkich dziewięciu timerów z mikrokontrolera AT91SAM3X8E, na którym oparta jest płyta Arduino Due. Ze względów praktycznych zalecane jest wykorzystywanie jedynie timerów 4 oraz 5 (*Timer4* i *Timer5*). *Timer3* jest zarezerwowany do obsługi przycisków (biblioteka *Buttons.h*).

*Timer4.start(us)* - metoda włączająca timer, *us* - parametr opcjonalny, okres (w mikrosekundach) co jaki ma się wywoływać funkcja przypisana do timera.

*Timer4.stop()* - metoda zatrzymująca działanie timera.

*Timer4.attachInterrupt(handler)* - metoda przypisująca funkcję do danego timera. *handler* - nazwa procedury, która ma zostać przypisana do timera (procedura taka nie może przyjmować żadnych argumentów).

*Timer4.setFrequency(long frequency)* - metoda ustawiająca częstotliwość pracy timera. *frequency* - częstotliwość działania timera (to znaczy ile razy na sekundę ma zostać wywołana procedura przypisana do timer).

*Timer4.setPeriod(long microsecond)* - metoda ustawiająca okres pracy timer. *microseconds* - okres (w mikrosekundach) co jaki ma się wywoływać procedura przypisana do timera.

---

<sup>4</sup> <https://github.com/ivanseidel/DueTimer>

### Przykład - wykorzystanie timera:

```
#include <ISADefinitions.h>
#include <DueTimer.h>
bool state = false;
void dioda(){
    digitalWrite(LED1, OUTPUT);
    state = !state;
}
void setup(){
    pinMode(LED1, OUTPUT);
    Timer4.attachInterrupt(dioda);
    Timer4.start(1000000);
}
void loop(){}

```

### Zadanie 1.2.1- Miganie diodą

#### Opis ćwiczenia:

Zmodyfikuj program z zadania **1.1.2** tak, aby zmiana stanu wyjść odpowiadających za pracę diod LED była zawarta jedynie w kodzie obsługi przerwania. Funkcja *loop()* Twojego programu ma być pusta.

### Zadanie 1.2.2- Wędrująca dioda

#### Opis ćwiczenia:

Napisz program, który będzie zapalał kolejne diody (co 250 milisekund) gasząc aktualną, najpierw od strony lewej do prawej, a następnie od prawej do lewej.

### Zadanie 1.2.3- Efekty świetlne

#### Opis ćwiczenia:

Zmodyfikuj program z zadania **1.1.5** tak, aby zmiana stanu wyjść odpowiadających za pracę poszczególnych par diod LED odbywała się w osobnych procedurach wywoływanych w przerwaniach. Skorzystaj tylko z trzech timerów. Funkcja *loop()* Twojego programu ma być pusta.

## Zadanie 1.2.4 - Stoper (przerwania)

Opis ćwiczenia:

Zmodyfikuj program z zadania [1.1.4](#) tak, aby działał na przerwaniach. Funkcja *loop()* Twojego programu ma być pusta.

## Zadanie 1.3 - Pierwsza interakcja

Cel ćwiczenia: Zapoznanie się z działaniem oraz obsługą przycisków.

Przydatne funkcje:

Biblioteka **ISAButtons** odpowiada za obsługę przycisków (obecnie 16 przycisków klawiatury numerycznej, przyciski numerowane są od 0 do 15), dostępna po dołączeniu do programu pliku **ISAButtons.h**. W swoim programie możesz mieć tylko **jeden** obiekt klasy **ISAButtons**.

**Metody dostępne w ramach klasy ISAButtons:**

- *buttons.init()* - metoda inicjalizująca przyciski. Należy ją wywołać w funkcji *setup()*. *buttons* - obiekt klasy **ISAButtons**.
- *buttons.buttonPressed(buttonId)* - metoda sprawdza, czy przycisk o podanym indeksie został przyciśnięty i zwraca **true** jeżeli przycisk był wciśnięty, a **false** w przeciwnym wypadku. *buttons* - obiekt klasy **ISAButtons**, *buttonId* - id przycisku, którego stan ma zostać sprawdzony. Stan przycisku odświeżany jest co 25 milisekund.
- *buttons.buttonReleased(buttonId)* - metoda sprawdza czy przycisk o podanym indeksie został zwolniony, zwraca **true** jeżeli przycisk był zwolniony, a **false** w przeciwnym wypadku. *buttons* - obiekt klasy **ISAButtons**, *buttonId* - id przycisku, którego stan ma zostać sprawdzony. Stan przycisku odświeżany jest co 25 milisekund.
- *buttons.buttonState(buttonId)* - metoda sprawdza stan przycisku o podanym indeksie, zwraca **true** jeżeli przycisk jest wciśnięty, a **false** w przeciwnym wypadku. *buttons* - obiekt klasy **ISAButtons**, *buttonId* - id przycisku, którego stan ma zostać sprawdzony. Stan przycisku odświeżany jest co 25 milisekund.

### Klawiatura strzałek

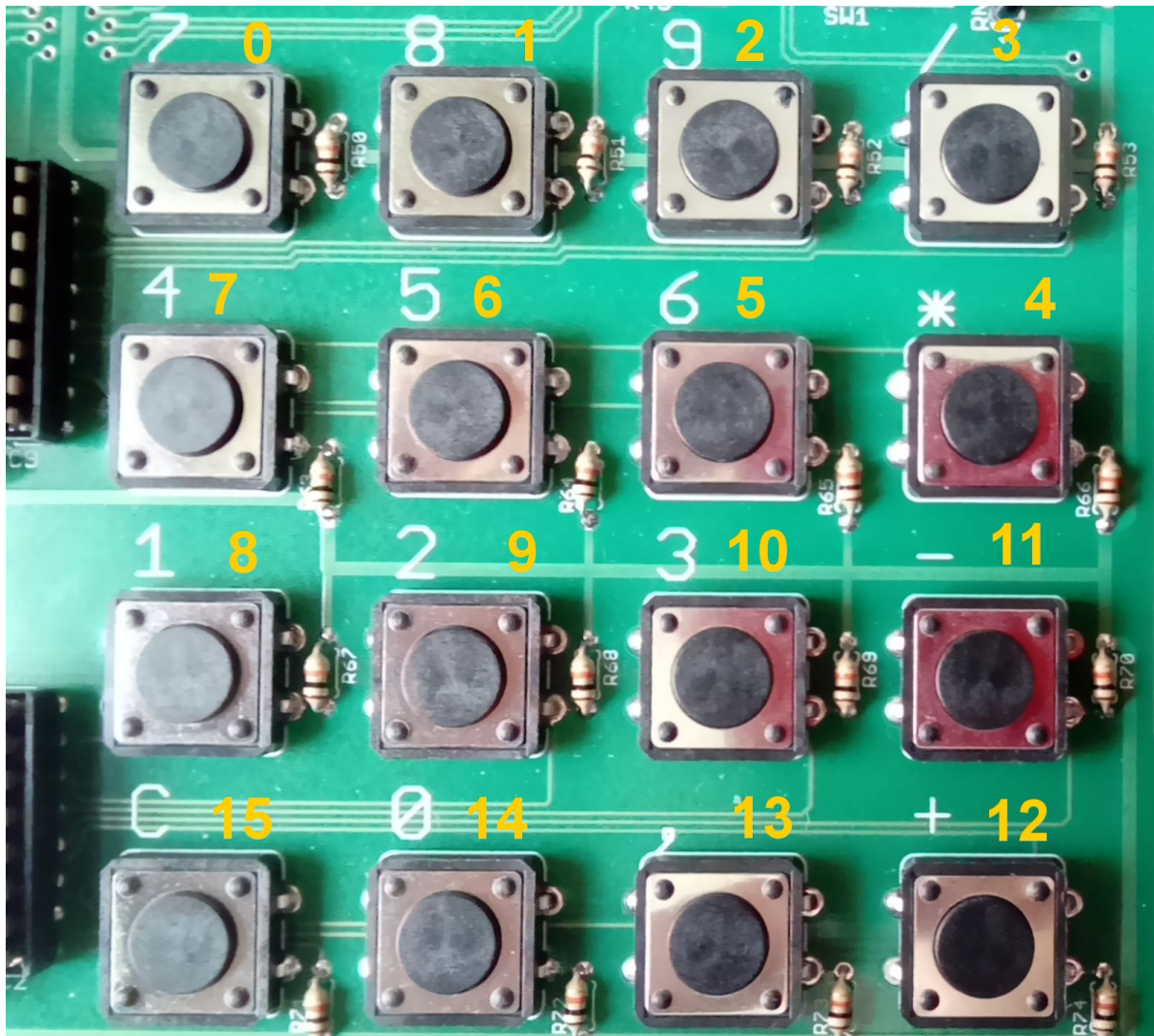
Dodatkowo dostępne są klawisze strzałek. Nie są one jednak obsługiwane przez klasę **ISAButtons** (która służy do odczytu klawiszy numerycznych). Aby odczytywać stany tych klawiszy, należy skorzystać z funkcji *pinMode* oraz *digitalRead*. Same klawisze dostępne są pod symbolami **KEY\_UP**, **KEY\_DOWN**, **KEY\_RIGHT** oraz **KEY\_LEFT**. Istnieje również tablica klawiszy strzałek - **KEY\_ARROWS[]**. Symbole dostępne są w pliku nagłówkowym **ISADefinitions.h**.

Uwaga! Klawisze strzałek działają w logice odwrotnej - funkcja *digitalRead* dla klawisza wciśniętego zwraca *false* a dla puszczanego zwraca *true*. Należy skorzystać z negacji (!).

### Stanowisko laboratoryjne:

Na stanowisku laboratoryjnym dostępne jest 20 przycisków - 16 na klawiaturze numerycznej + 4 strzałki.





*Ilustracja: Numeracja przycisków na klawiaturze numerycznej.*

Przykład - sprawdzenie stanu przycisku z biblioteką ISAButtons:

```
#include <ISADefinitions.h>
#include <ISAButtons.h>
ISAButtons button;
bool state = false;
void setup(){
    pinMode(LED1, OUTPUT);
    button.init();
}
void loop(){
    if (button.buttonPressed(0)) {
        state = !state;
        digitalWrite(LED1, state);
    }
}
```



```
    }  
    delay(30);  
}
```

Przykład - sprawdzenie stanu klawisza “w górę”:

```
#include <ISADefinitions.h>  
  
void setup(){  
    pinMode(KEY_UP, INPUT);  
    pinMode(LED1, OUTPUT);  
}  
void loop(){  
    bool up = !digitalRead(KEY_UP);  
    digitalWrite(LED1, up);  
}
```

### Zadanie 1.3.1 - Przyciski i dioda

Opis ćwiczenia:

Wykorzystując zestaw dwóch przycisków przesuwać zapaloną diodę w lewo lub w prawo.

### Zadanie 1.3.2- Wędrująca dioda 2

Opis ćwiczenia:

Zmodyfikuj program z zadania **1.2.2** w taki sposób, żeby przyciskami można było zmieniać kierunek poruszania się zapalanej diody.

### Zadanie 1.3.3 - System binarny

Opis ćwiczenia:

Wykorzystując zestaw czterech przycisków zaimplementuj następujące funkcjonalności: zwiększenie liczby binarnej o 1, zmniejszenie liczby binarnej o 1, zwiększenie liczby binarnej dwukrotnie, zmniejszenie liczby binarnej dwukrotnie. Aktualna wartość powinna być zawsze wyświetlana za pomocą diod LED.

### Zadanie 1.3.4 - Stoper

Opis ćwiczenia:

Rozbuduj program z zadania **1.2.4** o następujące funkcjonalności: wstrzymanie działania zegar, wznowienie działania zegara, zresetowanie zegara.

### Zadanie 1.3.5 - Klawiatura numeryczna

Opis ćwiczenia:

Wykorzystując zestaw 12 przycisków zaimplementuj funkcjonalności prostej klawiatury numerycznej (wprowadzanie cyfr z zakresu 0 - 9, przycisk clear czyszczący wprowadzoną liczbę i przycisk enter). Po naciśnięciu przycisku enter wprowadzona liczba ma zostać wyświetlona (o ile to możliwe) w systemie binarnym za pomocą 8 diod LED.

## Zadanie 1.4 - Sterowanie wyjściem PWM<sup>5</sup> (Pulse Width Modulation)

Cel ćwiczenia: Zapoznanie się z działaniem wyjść analogowych.

Trochę teorii:

PWM (Pulse Width Modulation) jest techniką pozwalającą uzyskać na wyjściu pinu sygnał analogowy przy wykorzystaniu wyjść cyfrowych. Za pomocą sterowania cyfrowego tworzony jest fala prostokątna poprzez przełączanie pomiędzy wysokimi i niskimi stanami. Tak powstały wzorzec może symulować napięcie z zakresu 5V (cały czas stan wysoki) a 0V (cały czas stan niski) poprzez zmienianie długości czasu, w którym sygnał znajduje się w stanie wysokim. Okres czasu, w którym wystawiany jest sygnał wysoki nazywa się szerokością impulsu (pulse width). W celu uzyskania różnych napięć wyjściowych należy zmieniać szerokość impulsu. Jeżeli taki wzorzec będzie powtarzany bardzo szybko z podłączoną diodą do pinu, wówczas w rezultacie otrzymane zostanie stałe napięcie z zakresu 0V do 5V sterujące jasnością diody LED.

Przydatne funkcje:

*analogWrite(pin, value)*<sup>6</sup> - umożliwia przypisanie wartości analogowej za pomocą sygnału PWM do pinu. Może być wykorzystywana do ustawiania jasności diody lub prędkości obracania się silnika. *pin* - numer pinu, do którego ma być przypisana wartość, *value* - wartość z zakresu 0 – 255, gdzie 0 odpowiada diodzie zgaszzonej, a 255 świecącej z maksymalną jasnością.

*random(max)*<sup>7</sup> - funkcja generująca liczby pseudolosowe, zwraca liczbę z przedziału <0 - *max*). *max* - górna granica przedziału, z którego będą losowane liczby.

*random(min, max)* - funkcja generująca liczby pseudolosowe, zwraca liczbę z przedziału <*min* - *max*). *min* - dolna granic przedziału, z którego będą losowane liczby, *max* - górna granica przedziału, z którego będą losowane liczby.

*randomSeed(seed)*<sup>8</sup> -funkcja inicjalizująca generator liczb pseudolosowych, wywołanie tej funkcji z losową wartością *seed* spowoduje, że za każdym uruchomieniem programu losowana będzie inna liczba. *seed* - jeżeli konieczne jest zagwarantowanie losowania różnych liczb przy każdym kolejnym uruchomieniu programu zaleca się zainicjalizowanie generatora z losową liczbą, na przykład poprzez wykorzystanie funkcji *analogRead()* z numerem dowolnego, niepodłączonego pinu.

---

<sup>5</sup> <https://www.arduino.cc/en/Tutorial/PWM>

<sup>6</sup> <https://www.arduino.cc/en/Reference/analogWrite>

<sup>7</sup> <https://www.arduino.cc/reference/en/language/functions/random-numbers/random/>

<sup>8</sup> <https://www.arduino.cc/en/Reference/RandomSeed>

Przykład - program losujący liczbę z przedziału <0 - 300):

```
long randomNumber;
void setup() {
    long seed = 0;
    for (int i = 0; i < 12; i++)
        seed += analogRead(i);
    randomSeed(seed);
    randomNumber = random(300);
}
void loop() {}
```

### Zadanie 1.4.1 - Niech stanie się jasność

Opis ćwiczenia:

Napisz program, który będzie stopniowo zwiększał jasność świecenia diody od 0 do 255, a po osiągnięciu maksymalnej wartości zmniejszał ją do 0. Sprawdź od jakiej wartości jasności diody nie jesteś w stanie zauważyć kolejnych jej skoków.

### Zadanie 1.4.2 - Sortowanie

Opis ćwiczenia:

Napisz program, który będzie sortował diody w zależności od ich jasności. W tablicy ośmioelementowej do poszczególnych komórek przypisz losowe wartości. Indeks komórki w tablicy będzie odpowiadał numerowi diody. Następnie zaimplementuj wybrany przez siebie algorytm sortowania i zademonstruj sposób jego działania za pomocą diod.

### Zadanie 1.4.3 - Sterowanie jasnością diody

Opis ćwiczenia:

Wykorzystując zestaw 6 przycisków zaimplementuj następujące funkcjonalności: zwiększenie i zmniejszenie jasności diody o 1, o 10 oraz 50.

### Zadanie 1.4.4 - Knight Rider Effect

Opis ćwiczenia:

Napisz program, który będzie imitował efekt taki sam jak na filmiku (<https://www.youtube.com/watch?v=OX6MuI1pFIU>).

## Zadanie 1.5 - Buzzer

Cel ćwiczenia: Zapoznanie się z działaniem wyjść cyfrowych.

Trochę teorii:

Brzęczyk (Buzzer) jest sterowanym elektrycznie sygnalizatorem akustycznym. Wystawienie na pinie, do którego podłączony jest brzęczyk stanu wysokiego powoduje wygenerowanie dźwięku, natomiast stan niski wyłącza dźwięk.

Stanowisko laboratoryjne:

Buzzer jest podłączony do pinu 24, oznaczony jest symbolem **BUZZER**.



*Ilustracja: Podłączenie i umiejscowienie Buzzera na stanowisku laboratoryjnym.*

Przykład - program uruchamiający Buzzer na pół sekundy:

```
#include <ISADefinitions.h>
void setup(){
    pinMode(BUZZER, OUTPUT);
    digitalWrite(BUZZER, HIGH);
    delay(500);
    digitalWrite(BUZZER, LOW);
}
void loop(){}
}
```

## Zadanie 1.5.1 - Czasomierz

Opis ćwiczenia:

Zmodyfikuj program z zadania 1.3.4 w taki sposób, żeby działał jak czasomierz. To znaczy ma być możliwość ustawienia czasu, od którego rozpocznie się odliczanie. Po tym jak zadany czas minie ma być wydany sygnał dźwiękowy, a diody mają zacząć mrugać. Do ustawienia czasu skorzystaj z *Klawiatury numerycznej* z zadania 1.3.5. Czas ma być wyświetlany w postaci binarnej za pomocą diod. Oprogramuj przyciski do rozpoczęcia i wstrzymania odliczania.

## Zadanie 1.5.2 - Kod Morse'a<sup>9</sup>

Opis ćwiczenia:

Korzystając z *Klawiatury Numerycznej* zaimplementowanej w zadaniu 1.3.5 napisz program, który będzie zamieniał wprowadzoną liczbę na sekwencję dźwięków zgodną z alfabetem Morse'a.

---

<sup>9</sup> [https://pl.wikipedia.org/wiki/Kod\\_Morse%E2%80%99a](https://pl.wikipedia.org/wiki/Kod_Morse%E2%80%99a)

## Zadanie 1.6 - Wyświetlacz siedmiosegmentowy

Cel ćwiczenia: Zapoznanie się z działaniem wyświetlacza siedmiosegmentowego.

Trochę teorii:

Z punkty widzenia Arduino wyświetlacz taki to osiem oddzielnych diod LED, każda podłączona pod inne wyjście cyfrowe.

Przydatne funkcje:

Biblioteka **ISA7SegmentDisplay**<sup>10</sup> odpowiada za obsługę wyświetlaczy siedmiosegmentowych, dostępna jest po dołączeniu do programu pliku **ISA7SegmentDisplay.h**.

**Metody dostępne w ramach klasy ISA7SegmentDisplay:**

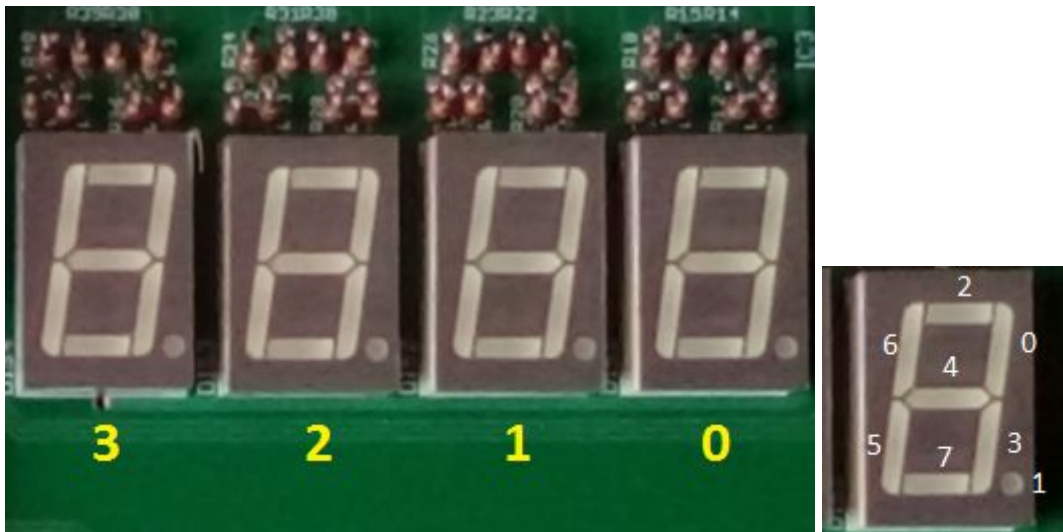
- *seg.init()* - metoda inicjalizująca wyświetlacz siedmiosegmentowy. *seg* - obiekt typu **ISA7SegmentDisplay**. Metodę należy wywołać w funkcji *setup()*;
- *seg.displayDigit(digit, dispID, dot = false)* - metoda pozwalająca wyświetlić cyfrę na wyświetlaczu. *seg* - obiekt typu **ISA7SegmentDisplay**, *digit* - cyfra, która ma zostać wyświetlona, musi być z zakresu <0, 9>, w innym przypadku funkcja nic nie zrobi, *dispID* - id wyświetlacza, na którym ma zostać wyświetlona cyfra, musi być wartością z zakresu <0, 4>, w inny przypadku funkcja nic nie zrobi, *dot* - przekazuje informację czy kropka ma zostać zapalona czy nie, domyślnie wartość jest ustawiona na **false**.
- *seg.setLed(values, dispID)* - metoda do ustawiania stanu diod na zadanym wyświetlaczu. *seg* - obiekt typu **ISA7SegmentDisplay**, *values* - ośmiobitowa wartość, w której każdy bit ustawiony na 1 włączy daną diodę, a ustawiony na 0 wyłącza, *dispID* - id wyświetlacza, na którym mają zostać włączone diody, musi być wartością z zakresu <0, 4>, w inny przypadku funkcja nic nie zrobi.

Stanowisko laboratoryjne:

Na stanowisku laboratoryjnym znajdują się cztery wyświetlacze siedmiosegmentowe.

---

<sup>10</sup> <http://isa.iis.p.lodz.pl/#/materials>



*Ilustracja: Położenie oraz numery id wyświetlaczy siedmiosegmentowych na stanowisku laboratoryjnym oraz numery poszczególnych diod.*

### Zadanie 1.6.1 Stoper

Opis ćwiczenia:

Napisz programu, który będzie wyświetlał w systemie dziesiętnym, ósemkowym i szesnastkowym, co sekundę, czas jaki minął od uruchomienia programu na wyświetlaczu siedmiosegmentowym. System w jakim jest wyświetlany aktualny czas ma być zmieniany za pomocą przycisku.

### Zadanie 1.6.1 Czasomierz

Opis ćwiczenia:

Zmodyfikuj program z zadania 1.5.1 w taki sposób, żeby działał wyświetlacz siedmiosegmentowym.

Przykład - program wyświetlający cyfry na wyświetlaczu siedmiosegmentowym:

```
#include <ISADefinitions.h>
#include <ISA7SegmentDisplay.h>
ISA7SegmentDisplay sseg;
void setup(){
    sseg.init();
}
void loop(){
    for (int i = 0; i < 10; ++i) {
```



```
    sseg.displayDigit(i, 0);  
    delay(250);  
  }  
}
```

## Zadanie 1.7 - Wyświetlacz LCD

Cel ćwiczenia: Zapoznanie się z działaniem wyświetlacza LCD.

Trochę teorii:

Przydatne funkcje:

Biblioteka **ISALiquidCrystal**<sup>11</sup> odpowiada za obsługę wyświetlacza LCD zamontowanego na płytce edukacyjnej IIS. Jest to wyświetlacz alfanumeryczny, o rozmiarze 2-óch wierszy i 16-stu kolumn. Klasa **ISALiquidCrystal** odpowiadająca za obsługę wyświetlaczy LCD, dostępna po dołączeniu do programu pliku **LiquidCrystal.h**. W swoim programie możesz mieć tylko **jeden** obiekt klasy **ISALiquidCrystal**.

**Metody dostępne w ramach klasy ISALiquidCrystal:**

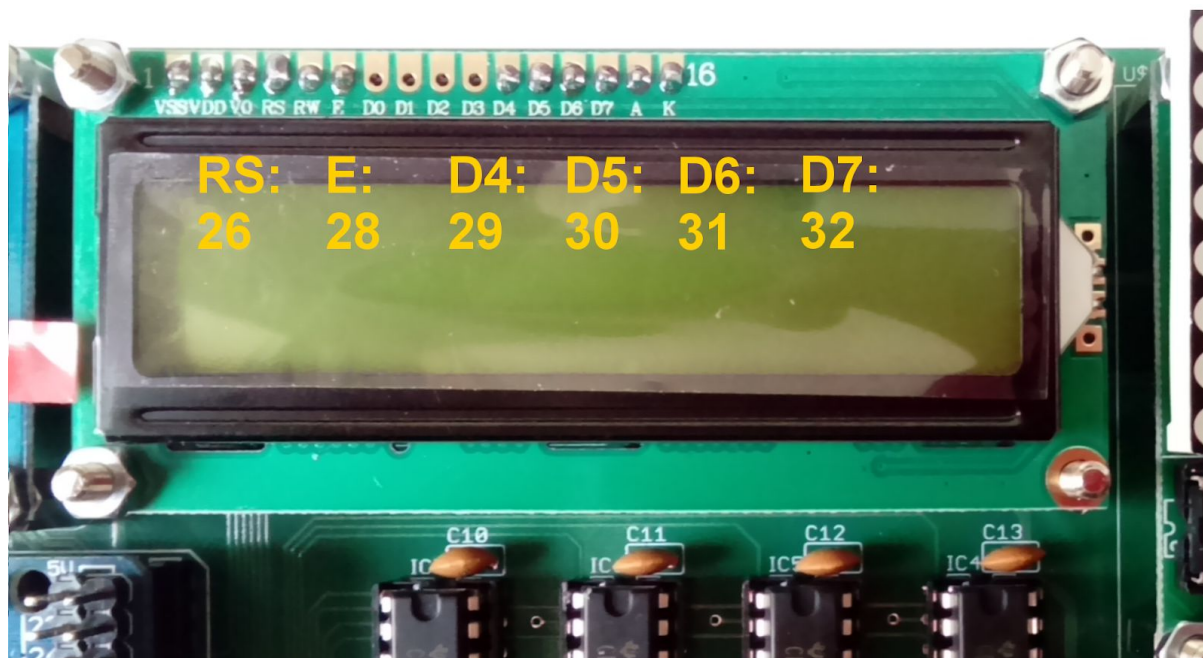
- *lcd.begin()* - uruchamia (inicjuje) sterownik wyświetlacza LCD. Należy ją wywołać w funkcji *setup()*. *lcd* - obiekt klasy **ISALiquidCrystal**.
- *lcd.print(data)* - metoda wyświetla wiadomość przekazaną w parametrze *data* na wyświetlaczu LCD. Dane do wyświetlenia mogą być następujących typów: **char**, **byte**, **int**, **long** lub **String**. *data* - dane, które mają zostać wyświetlone na wyświetlaczu LCD.
- *lcd.setCursor(col, row)* - metoda do ustawiania kursora w zadanej pozycji. Parametr *col* - numer kolumny, w której ma zostać ustawiony kursor, *row* - numer wiersza, w którym ma zostać ustawiony kursor. **Wiersze i kolumny są numerowane od 0.**
- *lcd.clear()* - metoda czyści ekran wyświetlacza LCD i ustawia kursor w lewym górnym rogu.

Stanowisko laboratoryjne:

Wyświetlacz LCD podłączony jest do pinów: 26 (RS), 28 (E), 29 (D4), 30 (D5), 31 (D6), 32 (D7).

---

<sup>11</sup> <https://www.arduino.cc/en/Reference/LiquidCrystal>



*Ilustracja: Umieszczenie i podłączenie wyświetlacza LCD na stanowisku laboratoryjnym.*

Przykład - program wyświetlający napis "hello, world!" na wyświetlaczu LCD:

```
#include <ISALiquidCrystal.h>
ISALiquidCrystal lcd;
void setup()
{
    lcd.begin();
    lcd.print("hello, world!");
}
void loop() {}
```

### Zadanie 1.7.1 Kalkulator

Opis ćwiczenia:

Napisz program, który rozszerzy funkcjonalność programu z zadania **1.3.5 (Klawiatura numeryczna)** o możliwość wykonywania podstawowych operacji matematycznych (dodawanie, odejmowanie, mnożenie, dzielenie). Wprowadzone liczby oraz znaki powinny być wyświetlane w pierwszym wierszu wyświetlacza LCD, po naciśnięciu przycisku *Enter* w drugim wierszu ma wyświetlić się wynik działania.

### Zadanie 1.7.2 Menu

Opis ćwiczenia:

Napisz program, który będzie posiadał menu umożliwiające wybór jednego z czterech trybów migania diod: wszystkie diody zgaszone, diody zmieniają stan co pół sekundy, diody zmieniają stan co sekundę, wszystkie diody wyłączone. Menu ma być wyświetlone na

wyświetlaczu LCD. Strzałka (->) ma wskazywać aktualnie wybraną pozycję w menu. Za pomocą dwóch przycisków (góra, dół) ma się ona przesuwać w dół lub w górę. Naciśnięcie przycisku *Enter* uruchamia wybrany tryb pracy diod.

## Zadanie 1.8 - Potencjometr i Joysticki

Cel ćwiczenia: Zapoznanie się z wejściami analogowymi Arduino na przykładzie potencjometru i joysticków.

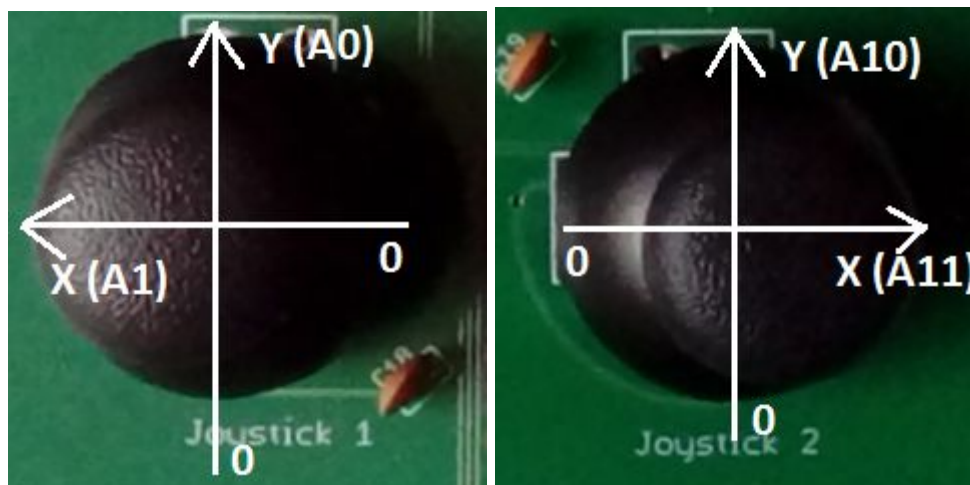
Trochę teorii:

Przydatne funkcje:

`analogRead(pin)`<sup>12</sup> - umożliwia odczytanie wartości z pinów będących wejściem analogowym. Zwraca wartość całkowitą z zakresu 0 - 1023. *pin* - numer pinu, do którego ma być przypisana wartość.

Stanowisko laboratoryjne:

Potencjometr oraz joysticki dostępne są za pomocą symboli z pliku nagłówkowego **ISADefinitions.h**. Symbole te to: **POT** (potencjometr), **JOY1X**, **JOY1Y** (joystick 1, osie X oraz Y), **JOY2X**, **JOY2Y** (joystick 2, osie X oraz Y).



a) joystick lewy

b) joystick prawy

*Ilustracja: Umieszczenie i podłączenie joysticków na stanowisku laboratoryjnym.*

Przykład - program odczytujący aktualną wartość potencjometru i wyświetlający ją na wyświetlaczu LCD:

```
#include <ISADefinitions.h>
#include <ISALiquidCrystal.h>
ISALiquidCrystal lcd;
```

---

<sup>12</sup> <https://www.arduino.cc/en/Reference/analogWrite>

```
void setup() {  
    lcd.begin();  
}  
void loop()  
{  
    lcd.clear();  
    int pot = analogRead(POT);  
    lcd.print(pot);  
    delay(250);  
}
```

### Zadanie 1.8.1 - Sterowanie jasnością diody

Opis ćwiczenia:

Napisz program, który umożliwi sterowanie jasnością wybranej diody za pomocą potencjometru.

### Zadanie 1.8.2 - Wędrująca dioda 3

Opis ćwiczenia:

Zmodyfikuj program z zadania [1.2.2](#) w taki sposób, żeby czas po jakim ma się zapalić kolejna dioda był regulowany za pomocą potencjometru.

## Zadanie 1.9 - Wyświetlacz matrycowy LED 8x8

Cel ćwiczenia: Zapoznanie się z działaniem wyświetlacza matrycowego LED.

Trochę teorii:

Moduł matrycy złożonej z 64 diod LED ułożonych w kwadrat 8 x 8.

Przydatne funkcje:

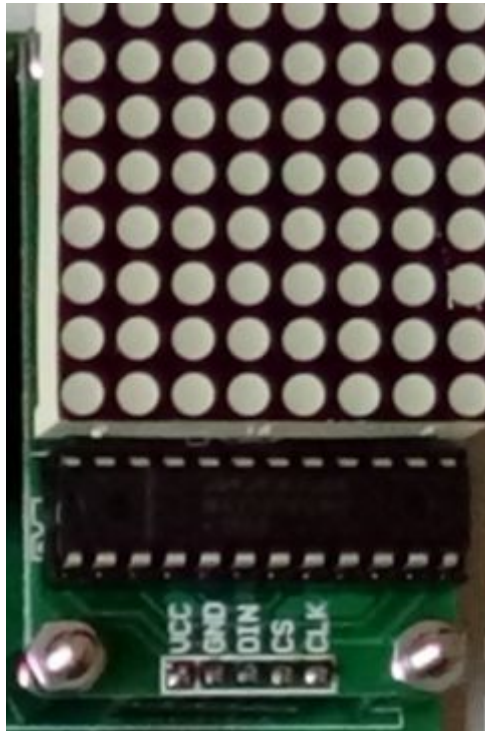
Biblioteka **ISALedControl**<sup>13</sup> odpowiada za obsługę wyświetlacza matrycowego LED 8x8, dostępna jest po dołączeniu do programu pliku **ISALedControl.h**.

- *lc.init()* - metoda inicjalizująca wyświetlacz matrycowy LED 8x8. *lc* - obiekt typu **ISALedControl**. Metodę należy uruchomić w funkcji *setup()*.
- *lc.clearDisplay()* - metoda wyłączająca wszystkie diody na wyświetlaczu. *lc* - obiekt typu **ISALedControl**.
- *lc.setRow(row, value)* - metoda do ustawiania stanu diod w danym wierszu. *lc* - obiekt typu **ISALedControl**, *row* - numer wiersza, *value* - wartość ośmiobitowa, w której każdy bit ustawiony na 1 włączy daną diodę, a ustawiony na 0 wyłączy.
- *lc.setColumn(col, value)* - metoda do ustawiania stanu diod w danej kolumnie. *lc* - obiekt typu **ISALedControl**, *col* - numer kolumny, *value* - wartość ośmiobitowa, w której każdy bit ustawiony na 1 włączy daną diodę, a ustawiony na 0 wyłączy.
- *lc.setLed(row, col, state)* - metoda włącza lub wyłącza diodę. *lc* - obiekt typu **LedControl**, *row* - numer wiersza (z zakresu <0; 7>), *col* - numer kolumny (z zakresu <0; 7>), *state* - stan diody (**true** - włączona, **false** - wyłączona).

---

<sup>13</sup> <http://isa.iis.p.lodz.pl/#/materials>

Stanowisko laboratoryjne:



*Ilustracja: Wyświetlacz matrycowy LED 8x8.*

Przykład - program zapalający diodę w lewym górnym rogu wyświetlacza matrycowego LED 8x8:

```
#include <ISALedControl.h>
ISALedControl led;
void setup()
{
    led.init();
    led.setLed(0, 0, true);
}
void loop() {}
```

## Zadanie 1.9.1 X

Opis ćwiczenia:

Napisz program, który zapali wszystkie diody znajdujące się na przekątnych wyświetlacza matrycowego LED 8x8.

## Zadanie 1.9.2 Szachownica



Opis ćwiczenia:

Napisz program, który zapali diody we wzorze szachownicy (co druga dioda zapalona).

### Zadanie 1.9.3 Spirala

Opis ćwiczenia:

Napisz program, który będzie zapalał kolejne diody po spirali na wyświetlaczu matrycowego LED 8x8 dopóki nie będą się paliły wszystkie diody.

### Zadanie 1.9.4 – Spadająca kropla

Opis ćwiczenia:

Napisz program, który będzie symulował spadającą kroplę na wyświetlaczu matrycowym 8x8 LED. Program ma najpierw wylosować kolumnę, w której ma spadać kropla a następnie w odstępie 100 milisekund zmniejszać / zwiększać indeks wiersza, w którym się znajduje. Po dotarciu na koniec wyświetlacza kropla ma zniknąć, a w jej miejsce pojawić się następna.

### Zadanie 1.9.5 – Mały deszcz

Opis ćwiczenia:

Zmodyfikuj program z zadania **1.9.4**, w taki sposób, żeby możliwe było pojawienie się więcej niż jednej kropli naraz. Oznacza to, że pojawienie nowej kropli już nie będzie zależało od położenia poprzedniej, a będzie niezależnym zdarzeniem.

### Zadanie 1.9.6 – Kałuże

Opis ćwiczenia:

Zmodyfikuj program z zadania **1.9.5**, w taki sposób, żeby krople nie znikwały, a po znalezieniu się w ostatnim wolnym wierszu w danej kolumnie pozostawały w tym miejscu na stałe. W ten sposób wyświetlacz powinien się powoli wypełniać włączonymi diodami.

### Zadanie 1.9.7 – Pierwszy ruch

Opis ćwiczenia:

Zmodyfikuj program z zadania **1.9.6**, w taki sposób, żeby na jednocześnie mogła być tylko jedna spadająca kropla. Dodaj oprogramowanie dwóch przycisków pozwalających przesuwac opadającą kroplę w lewo lub w prawo.

## Zadanie 1.9.8 – Znikające wiersze

Opis ćwiczenia:

Zmodyfikuj program z zadania [1.9.7](#), w taki sposób, żeby wiersze, w których wszystkie diody są zapalone, zostały wygaszone, a wszystkie włączone diody znajdujące się na wierszach wyższych przesunęły się o jeden w dół. Sprawdzanie czy usunąć wiersz ma się odbywać dopiero po osiągnięciu przez zapaloną diodę ostatniego możliwego wiersza.

## Zadanie 1.9.9 – Mini tetris

Opis ćwiczenia:

Rozbuduj program z zadania [1.9.8](#) o dodatkowy kształt. Tym razem nie tylko pojedyncza dioda będzie mogła opadać, ale również dwie zapalone obok siebie. Oprogramuj dodatkowy przycisk umożliwiający obracanie opadającego kształtu o 90 stopni.

## Zadanie 1.9.10 – I jaki wynik?

Opis ćwiczenia:

Do programu z zadanie [1.9.9](#) dodaj wyświetlanie wyniku na wyświetlaczu siedmiosegmentowym. Za każdy skasowany wiersz wynik powinien się zwiększać o 1.

## Część 2 - Sensory i efektory

## Zadanie 2.1 Ultradźwiękowy czujnik odległości

Cel ćwiczenia: Zapoznanie się z działaniem ultradźwiękowych czujników odległości.

Przydatne funkcje:

### Zadanie 2.1.1 - Jak daleko jesteś?

Opis ćwiczenia:

Napisz program, który na jednym z dostępnych wyświetlaczy będzie wyświetlał odległość od najbliższej przeszkody odczytaną z czujnika odległości..

### Zadanie 2.1.2 - Blisko, coraz bliżej

Opis ćwiczenia:

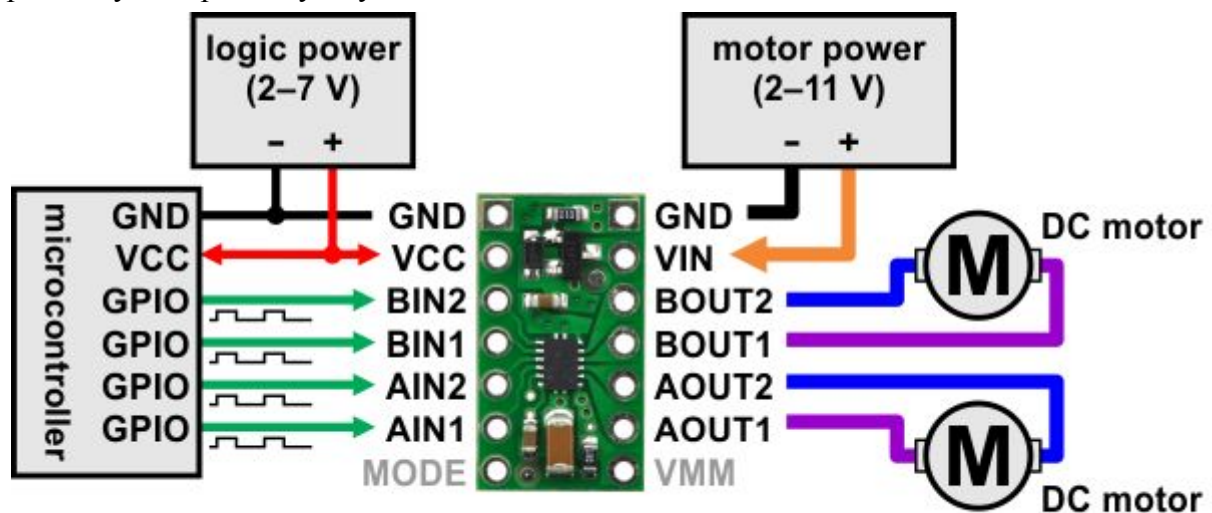
Napisz program, który za pomocą buzzera będzie informował o odległości przeszkody od czujnika. Wraz ze zbliżaniem się przeszkody do czujnika częstotliwość wydawania dźwięków przez buzzer powinna rosnać..

## Zadanie 2.2 Silniki

Cel ćwiczenia: Uruchomienie i pierwsze sterowanie zespołem napędowym TT D65.

Trochę teorii:

Sterowanie silnikiem odbywa się za pomocą sterownika, będącego pełnym mostkiem H (ang. *full H bridge*). Do wykorzystania jest moduł POLOLU 2135<sup>14</sup>, który wyposażony jest w sterownik silnika DRV8835. Silnik do sterownika należy podłączyć zgodnie z oznaczeniem pokazanym na poniższym rysunku.

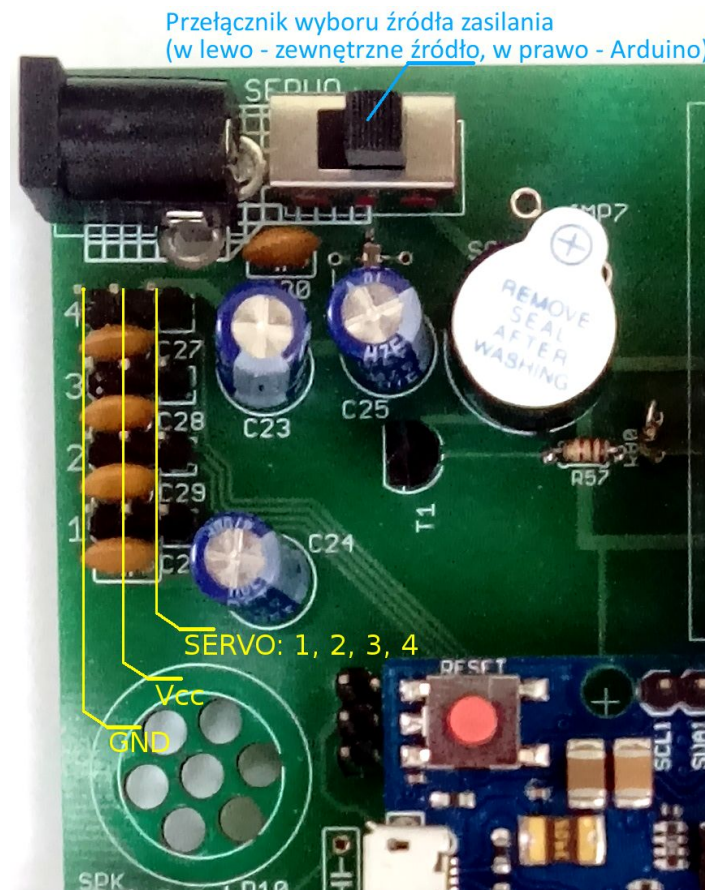


*Ilustracja: Podłączenie silnika do sterownika.*

<sup>14</sup> <http://www.tme.eu/en/Document/55000906f39a0cf512d37491e45bd99b/POLOLU-2135.pdf>

## Stanowisko laboratoryjne:

Na początek, do celów testowych sterownik silnika należy podłączyć do modułu serwonapędów, znajdującego się na płycie laboratoryjnej. **W takim wypadku należy podłączyć zewnętrzne zasilanie! Proszę nie korzystać z zasilania silników przez Arduino!**



*Ilustracja: Moduł serwonapędów.*

### Zadanie 2.2.1 Czas ruszyć do boju

Opis ćwiczenia:

Napisz program, który będzie rozpędzał silnik w jedną stronę do maksymalnej prędkości, następnie zmniejszał ją do 0 i ponownie rozpędzał, tylko w przeciwną stronę.

## Zadanie 2.3 Enkoder

Cel ćwiczenia: Uruchomienie i odczytanie danych z enkodera RS030<sup>15</sup>.

Przydatne funkcje:

*attachInterrupt(intno, ISR, mode)*<sup>16</sup> - funkcja dodaje funkcję obsługi przerwania do wektora przerwań. *intno* - numer przerwania, najlepiej podać korzystając z funkcji *digitalPinToInterrupt*, *ISR* - nazwa funkcji, która ma zostać dodana do przerwania, *mode* - definiuje, kiedy przerwanie ma być wywoływane, może przyjąć jedną z następujących wartości: **LOW**, **CHANGE**, **RISING**, **FALLING** oraz **HIGH**.

*digitalPinToInterrupt(pin)* - generuje wektor przerwania na podstawie numeru pinu. *pin* - numer pinu, dla którego ma zostać wygenerowany wektor przerwań.

Trochę teorii:

Stanowisko laboratoryjne:

Do dyspozycji studentów są dwa enkodery RS030, po jednym enkoderze na koło.

### Zadanie 2.3.1 Ile obrotów zrobiłem?

Opis ćwiczenia:

Napisz program, który będzie wyświetlał na wyświetlaczu LCD liczbę obrotów wykonaną przez silnik oraz aktualną prędkość w jednostkach obrotu na minutę.

### Zadanie 2.3.2 Odpowiedzialne sterowanie silnikiem

Opis ćwiczenia:

Napisz program, który będzie regulował prędkość obrotów silnika na podstawie odczytu z potencjometru.

---

<sup>15</sup> <https://botland.com.pl/enkodery/2182-zestaw-enkoderow-magnetycznych-dagu-rs030.html>

<sup>16</sup> <https://www.arduino.cc/en/Reference/attachInterrupt>

## Zadanie 2.4 Serwo

Cel ćwiczenia: Zapoznanie się z działaniem ultradźwiękowych czujników odległości.

Przydatne funkcje:



## Część 3 - Projekt końcowy

## Zadanie 3.1 Jazda po prostokącie

Oprogramuj autko w taki sposób, żeby przejechało trasę po prostokącie. Długość boków prostokąta ma być wprowadzana przez użytkownika (jednostki są dowolne). Ocena końcowa za wykonanie zadania będzie od tego, jak daleko od punktu startowego, autko znajdzie się po wykonaniu manewru.

## Zadanie 3.2 Jazda do celu

Oprogramuj autko w taki sposób, żeby zatrzymało się w zadanej odległości od ściany (przeszkody). Odległość, w której ma się autko zatrzymać, ma być wprowadzana przez użytkownika (jednostki są dowolne). Ocena końcowa za wykonanie zadania będzie od tego, jak daleko od ściany, autko znajdzie się po wykonaniu manewru.