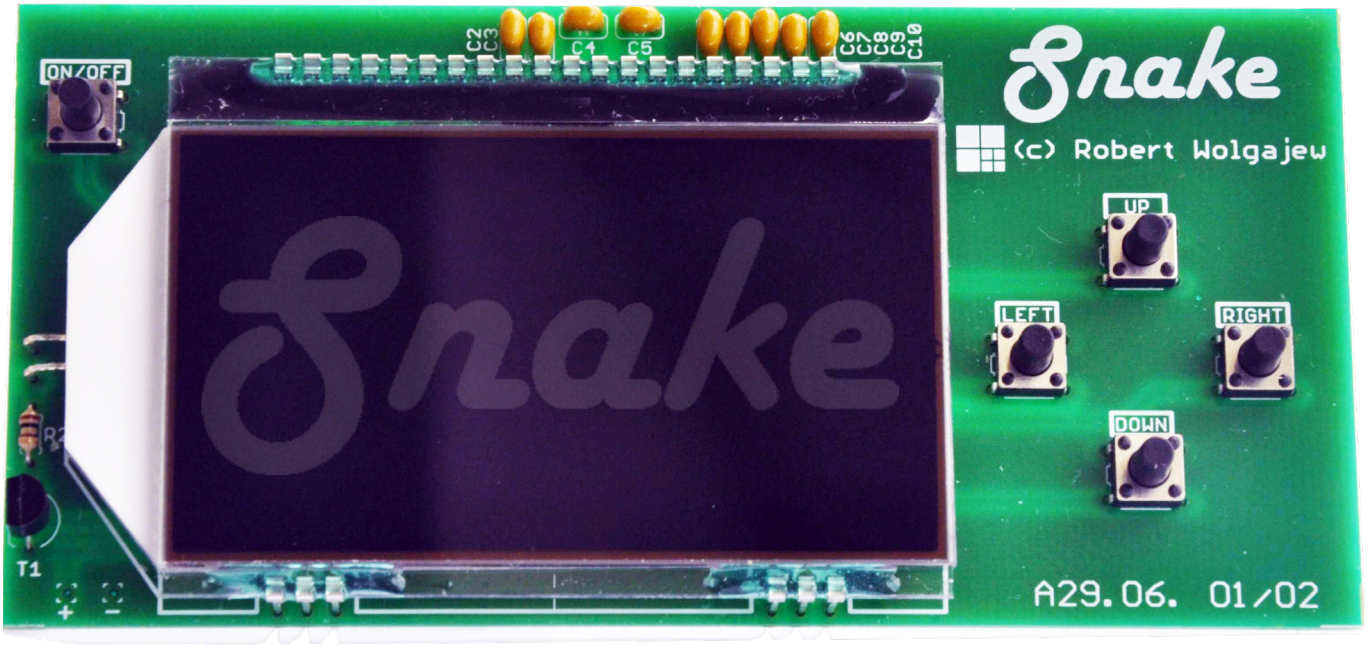


Gra elektroniczna Snake

Wielu z nas, zwłaszcza tych po przysłowiowej czterdziestce, pamięta czasy początków telefonii komórkowej GSM i co się z nimi wiąże – pierwsze nowoczesne telefony. Mnie osobiście te czasy kojarzą się z niezmiernie wtedy popularnymi słuchawkami spod znaku fińskiej Nokii, które oprócz oczywistej funkcjonalności telefonu bezprzewodowego wyposażone były w gry, w tym w kultowego Snake'a. I właśnie to wspomnienie było przyczyną do powstania niniejszego projektu, jakim jest znana chyba wszystkim, niezależnie od wieku, gra Snake.

Rekomendacje: *świetny gadżet dla wszystkich małych i dużych dzieci.*

Kultową grę „Snake" postanowiłem wykonać wykorzystując niewielki mikrokontroler AVR oraz niedrogi wyświetlacz graficzny o rozdzielczości 128×64 piksele, który jest akceptowalnym kompromisem pomiędzy możliwościami graficznymi a ceną. Jako że gra od strony elektronicznej jest niezwykle nieskomplikowana, bo ogranicza się do mikrokontrolera, wyświetlacza i kilku przycisków. Dlatego tym razem w opisie skupię się przede wszystkim organizacji logicznej stosownego programu obsługi aplikacji.

Po pierwsze, postanowiłem podzielić cały dostępny obszar ekranu na pola o rozdzielczości 8×8 pikseli będące podstawowym elementem graficznym. Zgodnie z powyższymi założeniami powstał obszar roboczy o rozmiarze 16 (szerokość) na 8 (wysokość) pól graficznych, którego wygląd wraz z zasadami numeracji pokazano na **rysunku 1**.

Przejdźmy zatem do szczegółów implementacyjnych programu obsługi aplikacji. Aby ułatwić poruszanie się po wcześniej zdefiniowanym obszarze roboczym, wprowadzimy nowy typ strukturalny o nazwie *TCOORD* integrujący koordynaty X i Y o budowie, jak niżej:

```
typedef struct
{
    int8_t X, Y;
} TCOORD;
```

Ponadto, zdefiniujemy kolejny typ strukturalny o nazwie *TSNAKE*, którego zadaniem będzie przechowywanie wszystkich parametrów węża. Definicja wspomnianego typu wygląda następująco:

```
typedef struct
{
```

Ustawienia fusebitów:
CKSEL3…0: 0010
SUT1…0: 10
CKDIV8: 0
CKOUT: 1
EESAVE: 0

Dodatkowe materiały do pobrania ze strony www.media.avt.pl

W ofercie AVT* AVT-5639

Podstawowe parametry:
- Mikrokontroler ATmega88.
- Wyświetlacz graficzny 128×64 piksele.
- Złożony wyłącznie z elementów THT.
- Zasilanie 3 V (2 baterie paluszki AA).
- Interfejs użytkownika złożony z wyświetlacza i 5 przycisków.

Projekty pokrewne na www.media.avt.pl:

AVT-5592	Gra elektroniczna Sudoku (EP 7/2017)
AVT-5554	Gra elektroniczna „Snake" (EP 11/2016)
AVT-1651	Gra „Kto pierwszy ten lepszy" (EP 11/2011)
AVT-723	Uniwersalna gra zręcznościowa (EP 6/2004)
AVT-5028	Elektroniczna gra w kości (EP 8/2001)
AVT-5014	Gra zręcznościowa (EP 5/2001)

Wykaz elementów:
Rezystory: (1/8 W)
R1: 22 kΩ
R2: 47 Ω
R3: 4,7 kΩ

Kondensatory: (MLCC, R=0,1 cala)
C1: 100 nF
C2…C10: 1 µF

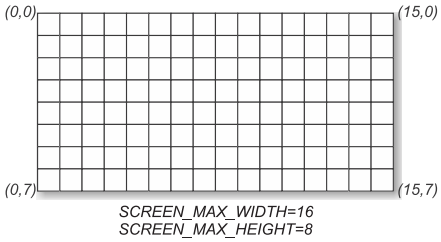
Półprzewodniki:
U1: ATmega88 (DIL28)
T1: BC557 (TO-92)

Inne:
LCD: wyświetlacz graficzny COG typu LCD-AG-C128064CF-DIW W/KK-E6 PBF z podświetleniem (sterownik ST7565R)
UP, DOWN, LEFT, RIGHT, ON/OFF– – mikroprzełącznik TACT (długość osi 6 mm)

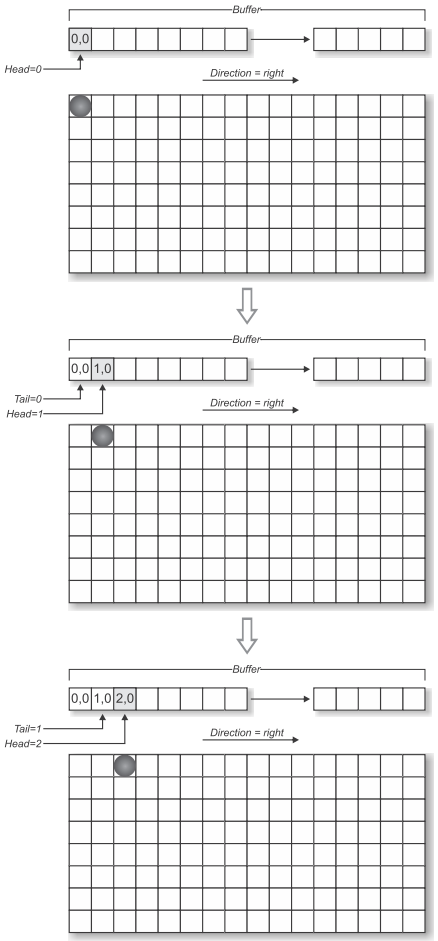
*** Uwaga!** **Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania!**
Podstawową wersją zestawu jest wersja [B] nazywana potocznie KITem (z ang. zestaw). Zestaw w wersji [B] zawiera elementy elektroniczne (w tym [UK] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu.
Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- wersja [C] zmontowany, uruchomiony i przetestowany zestaw [B] (elementy wlutowane w płytkę PCB)
- wersja [A] płytka drukowana bez elementów i dokumentacja

Kity w których występuje układ scalony wymagający zaprogramowania, posiadają następujące dodatkowe wersje:
- wersja [A+] płytka drukowana [A] + zaprogramowany układ [UK] i dokumentacja
- wersja [UK] zaprogramowany układ

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz!
http://sklep.avt.pl



Rysunek 1. Wygląd obszaru roboczego gry Snake z zasadami numeracji wierszy i kolumn



Rysunek 2. Zasady działania mechanizmu obsługi ruchu węża

```
    TCOORD Body[64];  //Bu-
for ciała węża (każdego
jego elementu)
    uint8_t Head;  //Pozycja
głowy węża w buforze
    uint8_t Length;  //Długość
ciała węża
    uint8_t Direction;  //
Kierunek poruszania się węża
    uint8_t Points;  //Liczba
zdobytych punktów gracza
} TSNAKE;
```

Warto w tym miejscu napisać kilka słów komentarza na temat pól nowego typu strukturalnego, gdyż mają one fundamentalne znaczenie w mechanizmie obsługi poruszania się węża po ekranie wyświetlacza. Znaczenie poszczególnych pól przedstawia się następująco:

- **Body[]** jest tablicą zorganizowaną na wzór bufora cyklicznego a przechowującą koordynaty (X i Y) wszystkich elementów ciała węża.
- **Head** jest indeksem wskazującym bieżące położenie głowy węża (w tablicy Body[]).
- **Length** jest długością ciała węża (dzięki czemu możemy określić położenie jego końca).
- **Direction** jest kierunkiem poruszania się węża (predefiniowany typ wyliczeniowy).
- **Points** jest liczbą zdobytych punktów w trakcie gry.

Aby zobrazować mechanizm odpowiedzialny za obsługę ruchu węża i stosowne zmiany parametrów pól struktury *TSNAKE*, posłużę się rysunkiem, na którym pokazano kilka kolejnych kroków ruchu węża. Mechanizm ten ilustruje **rysunek 2**. Jak widać, każdemu ruchowi węża towarzyszy ustalenie nowego położenia „głowy” węża (indeksu *Head*) oraz zapisanie jego parametrów (X, Y) do tablicy *Body[]*. Co oczywiste, nowe położenie (parametry *X, Y*) jest zależne od wartości pola *Direction*, które określa kierunek ruchu węża. Z kolei, dzięki parametrowi *Length* określającemu długość węża możemy określić położenie jego „ogona” (*Tail*), czyli miejsca gdzie kończy się ciało węża. Myślę, że to dość oczywiste i logiczne rozwiązanie mechanizmu tego typu, a jeśli jeszcze nie wszystko jest jasne, to z pewnością stanie się bardziej klarowne po przedstawieniu stosownych funkcji obsługi. Zatem do dzieła!

Zacznę od funkcji narzędziowej, której zadaniem jest sprawdzenie położenia punktu *Point(X, Y)* będącego argumentem wywołania, a którą pokazano na **listingu 1**. Funkcja ta zwraca wartość w postaci jednej z trzech predefiniowanych stałych:

- **MEAL**, gdy punkt *Point(X, Y)* pokrywa się z „jedzeniem” węża.
- **SNAKE_BODY**, gdy punkt *Point(X, Y)* pokrywa się z ciałem węża.
- **UNUSED**, gdy punkt *Point(X, Y)* jest nowym punktem na obszarze roboczym.

Warto w tym miejscu zaznaczyć, że funkcja *checkPoint()* korzysta z dwóch zmiennych globalnych: *Snake* typu *TSNAKE* przechowującej wszystkie parametry węża oraz *Meal* typu *TCOORD*, przechowującej bieżące położenie „jedzenia” węża.

Pora na najważniejszą funkcję będącą „silnikiem” całego mechanizmu (i zarazem urządzenia), czyli funkcję odpowiedzialną za obsługę ruchu węża na ekranie, którą pokazano na **listingu 3**.

Zobaczmy, jak działa „silnik” naszego urządzenia. W pierwszej kolejności ustalamy bieżące położenie „głowy” węża (parametry *X* i *Y*). Dalej, w zależności od kierunku jego poruszania się (pole *Direction*), ustalamy nowe położenie „głowy” węża. W tym momencie wywołujemy funkcję narzędziową *checkPoint()*, by sprawdzić jak się ma to nowe położenie do zawartości naszego obszaru roboczego. Możliwe są 3 scenariusze, którym odpowiada zwrócenie stosownej stałej przez funkcję *checkPoint()*:

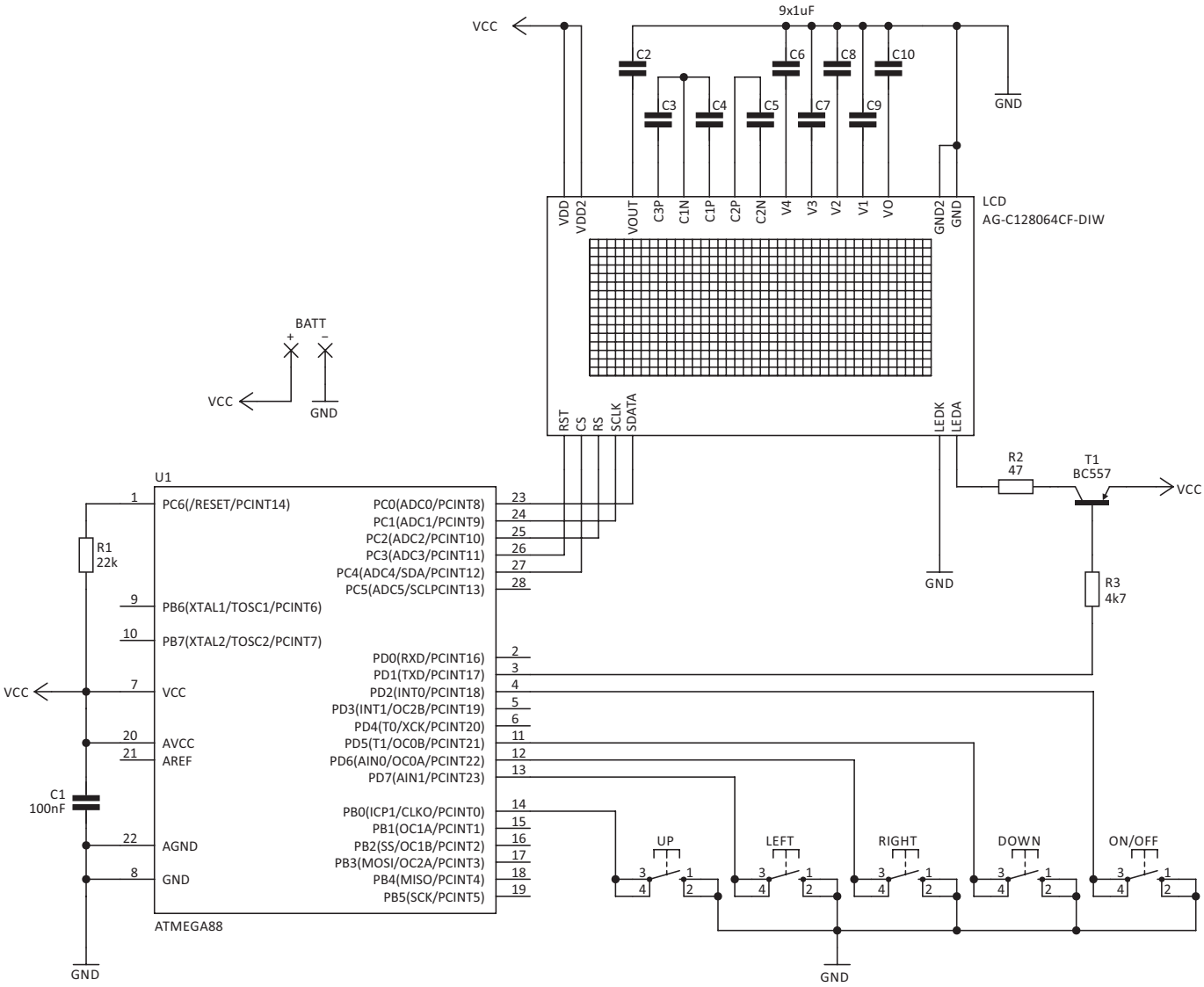
- **SNAKE_BODY**, gdy „głowa” węża uderzyła w „ciało” węża, co powoduje zakończenie bieżącej gry.
- **UNUSED**, gdy „głowa” węża znalazła się w nowym miejscu obszaru roboczego, co powoduje narysowanie jej w tymże miejscu i skasowanie elementu ciała znajdującego się w miejscu „ogona” węża.
- **MEAL**, gdy „głowa” węża znalazła się w miejscu położenia „jedzenia” węża, co powoduje wykonanie czynności, jak dla wartości UNUSED, plus zwiększenie długości „ciała” węża oraz liczby zdobytych punktów.

Prawda, że łatwe? Dla porządku, na **listingu 2** zamieszczono funkcję inicjalizującą zmienną *Snake* typu *TSNAKE*, wywoływaną przed każdym rozpoczęciem gry.

Tyle w kwestiach programowych. Przejdźmy zatem do schematu ideowego naszego urządzenia, który pokazano na **rysunku3**. Zaprojektowano nieskomplikowany system mikroprocesorowy, którego „sercem” jest mikrokontroler ATmega88 odpowiedzialny za realizację całej założonej funkcjonalności, czyli obsługę wyświetlacza graficznego o rozdzielczości 128×64 piksele, wyposażonego w sterownik ekranu ST7565R oraz obsługę pięciu przycisków funkcyjnych: UP, DOWN, LEFT, RIGHT i ON/OFF stanowiących element interfejsu użytkownika. Warto zauważyć, że w układzie, jako wyłącznik zasilania zastosowano zwykły mikroprzełącznik monostabilny, w związku z czym sterowanie zasilaniem odbywa się wyłącznie na drodze programowej. W wypadku wyłączania urządzenia wykonywane są następujące kroki:

Listing 1. Funkcja, której zadaniem jest sprawdzenie położenia punktu Point(X, Y)

```
uint8_t checkPoint(TCOORD Point)
{
    uint8_t Head = Snake.Head;
    //W pierwszej kolejności sprawdzamy, czy punkt (X, Y) nie jest przypadkiem jedzeniem węża
    if(Point.X == Meal.X && Point.Y == Meal.Y) return MEAL;
    //Następnie sprawdzamy, czy punkt (X, Y) należy do ciała węża lub jest nieużywanym punktem
    for(uint8_t i=0; i<Snake.Length; ++i)
    {
        if(Point.X == Snake.Body[Head].X && Point.Y == Snake.Body[Head].Y) return SNAKE_BODY;
        Head = (Head-1) & (SNAKE_MAX_LENGTH-1);  //Kolejny element węża w kierunku jego "ogona"
    };
    return UNUSED;
}
```



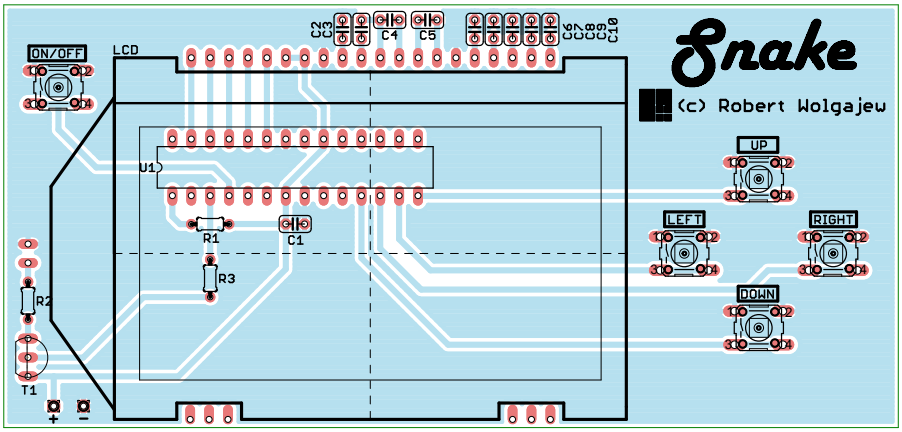
Rysunek 3. Schemat ideowy gry Snake

- Zostaje wyłączony wyświetlacz graficzny poleceniem *CMD_DISPLAY_OFF* przesłanym za pomocą interfejsu SPI.
- Zostaje wyłączone podświetlenie wyświetlacza graficznego sterowane tranzystorem T1.
- Mikrokontroler przechodzi w tryb PowerDown, z którego może zostać wybudzony między innymi dzięki przerwaniu zewnętrznemu INT0, przypisanemu do wyprowadzenia, do którego podłączono przycisk ON/OFF (PD2).

W wypadku włączania urządzenia wykonywana jest sekwencja odwrotna. Ten mechanizm pozwala na zastosowanie software'owego sterowania załączaniem/wyłączaniem urządzenia. Jako źródło zasilania posłużyć może zestaw 2 baterii AA, tzw. „paluszków", gdyż wyświetlacz graficzny akceptuje napięcia zasilania do 3,3 V.

Montaż

Schemat montażowy gry „Snake" pokazano na **rysunku 4**. Zaprojektowano obwód



Rysunek 4. Schemat montażowy gry Snake

```
Listing 2. Funkcja inicjalizująca węża
void initSnake(void)
{
    //Parametry startowe węża
    Snake.Head = 0;
    Snake.Body[0].X = 0;
    Snake.Body[0].Y = 0;
    Snake.Length = 1;
    Snake.Direction = right;
    Snake.Points = 0;
    //Losowanie położenia "jedzenia" węża oraz jego narysowanie
    Meal.X = random() % SCREEN_MAX_WIDTH;
    Meal.Y = random() % SCREEN_MAX_HEIGHT;
    drawElement(Meal, MEAL_ELEMENT);  //Narysowanie jedzenia węża
}
```

Listing 3. Funkcja odpowiedzialna za obsługę ruchu węża na ekranie

```
void drawSnake(void)
{
    uint8_t newHead, Result;
    TCOORD Point, newMeal;
    //Ustalenie bieżącego położenia (X, Y) głowy węża
    Point = Snake.Body[Snake.Head];
    //Zmiana położenia głowy węża w zależności od kierunku jego poruszania się
    switch(Snake.Direction)
   {
        case left: if(--Point.X < 0) Point.X = SCREEN_MAX_WIDTH-1; break;
        case right: if(++Point.X > (SCREEN_MAX_WIDTH-1)) Point.X = 0; break;
        case up: if(--Point.Y < 0) Point.Y = SCREEN_MAX_HEIGHT-1; break;
        case down: if(++Point.Y > (SCREEN_MAX_HEIGHT-1)) Point.Y = 0; break;
    }
    //Sprawdzamy nowe położenie głowy węża i w zależności od jego lokalizacji podejmujemy stosowne akcje
    Result = checkPoint(Point);
    //Trafiliśmy w ciało węża
    if(Result == SNAKE_BODY)
    {
        //Zakończenie gry
        endScreen();
        return;
    }
    //Jeśli trafiliśmy na jedzenie to musimy wylosować nowe nie położone na bieżącym ciele
    //węża po czym je narysować oraz zwiększyć długość węża jak i liczbę punktów gracza
    if(Result == MEAL)
    {
        do
        {
            newMeal.X = random() % SCREEN_MAX_WIDTH;
            newMeal.Y = random() % SCREEN_MAX_HEIGHT;
        } while(checkPoint(newMeal) != UNUSED);
        Snake.Points++;
        Snake.Length++;
        Meal = newMeal;
    }
    drawElement(Meal, MEAL_ELEMENT);  //Narysowanie jedzenia
    //Obliczenie nowej pozycji głowy węża oraz zapisanie jej parametrów X i Y
    Snake.Head = newHead = (Snake.Head + 1) & (SNAKE_MAX_LENGTH-1);
    Snake.Body[newHead] = Point;
    //Narysowanie całego ciała węża począwszy od głowy
    for(uint8_t i=0; i<Snake.Length; ++i)
    {
        drawElement(Snake.Body[newHead], i==0? HEAD_ELEMENT:BODY_ELEMENT);
        newHead = (newHead-1) & (SNAKE_MAX_LENGTH-1);  //Kolejny element węża w kierunku jego "ogona"
    };
    //Wyczyszczenie ekranu w miejscu ogona węża
    drawElement(Snake.Body[newHead], EMPTY_ELEMENT);
}
```

drukowany z wyłącznym wykorzystaniem elementów do montażu przewlekanego. Montaż urządzenia rozpoczynamy od przylutowania mikrokontrolera, następnie lutujemy tranzystor T1, kolejno elementy bierne, zaś na samym końcu mikroprzełączniki. Ostatnim krokiem jest montaż podświetlenia wyświetlacza LCD, które to należy umiejscowić w taki sposób by znalazło się w prawidłowym położeniu w stosunku do bryły modułu tegoż wyświetlacza (jeśli występuje jako oddzielny element). Na samym końcu przylutowujemy sam moduł wyświetlacza, mocując go w taki sposób, by górna płaszczyzna jego obudowy (szkła) znalazła się w odległości 10 mm od płaszczyzny obwodu drukowanego. Poprawnie zmontowany układ nie wymaga żadnych regulacji i powinien działać tuż po włączeniu zasilania.

Robert Wołgajew, EP