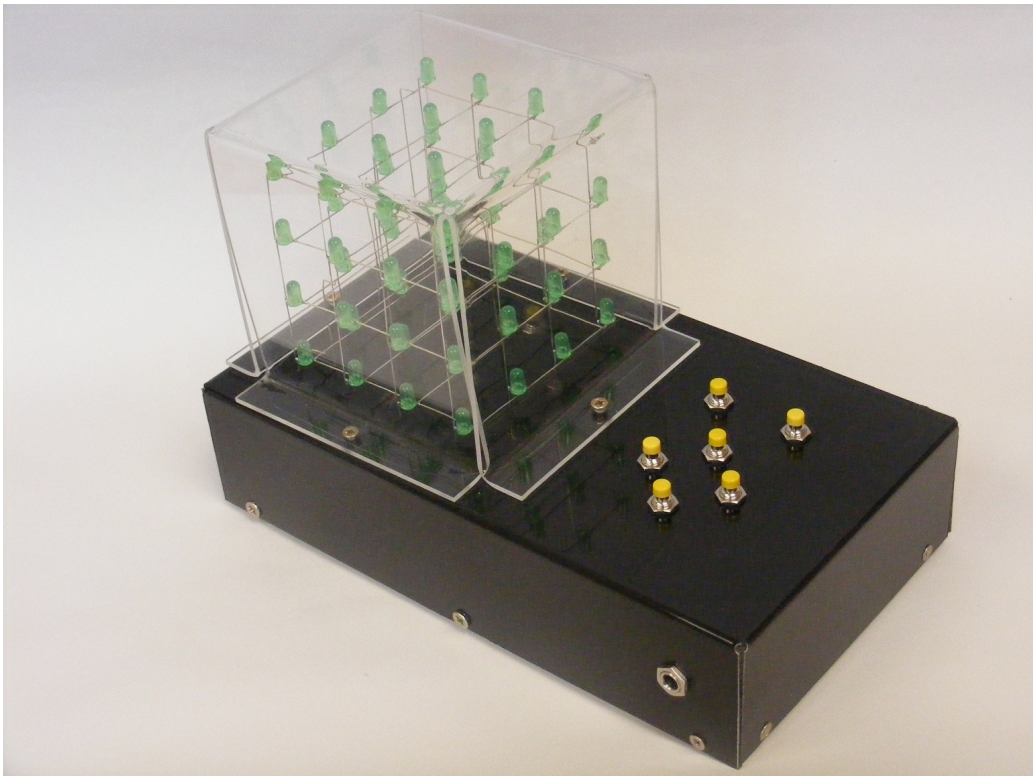
Karta projektu zaliczeniowego
Systemy mikroprocesorowe – 2015

Temat projektu: **Snake 3D**
Imię i nazwisko: **Paulina Wróbel**
Politechnika Poznańska
kierunek: **AiR**, grupa: **A2**, nr albumu: **109555**

1. Opis projektu

Temat projektu („Snake 3D") nawiązuje do popularnej gry „Snake". Zaczerpnięty został pomysł poruszającego się węża, który wydłuża się, gdy trafi na jedzenie. Celem gry jest zapełnienie całej kostki przez węża. Wąż jest wyświetlany na kostce zbudowanej z diod LED, a sterowanie jest zapewnione przez sześć przycisków znajdujących się przed kostką. Wygląd projektu został przedstawiony na rys. 1. Przykładowe działanie układu przedstawia załącznik [Z.1] i [Z.2].



Rysunek 1: Zdjęcie przedstawiające wygląd projektu

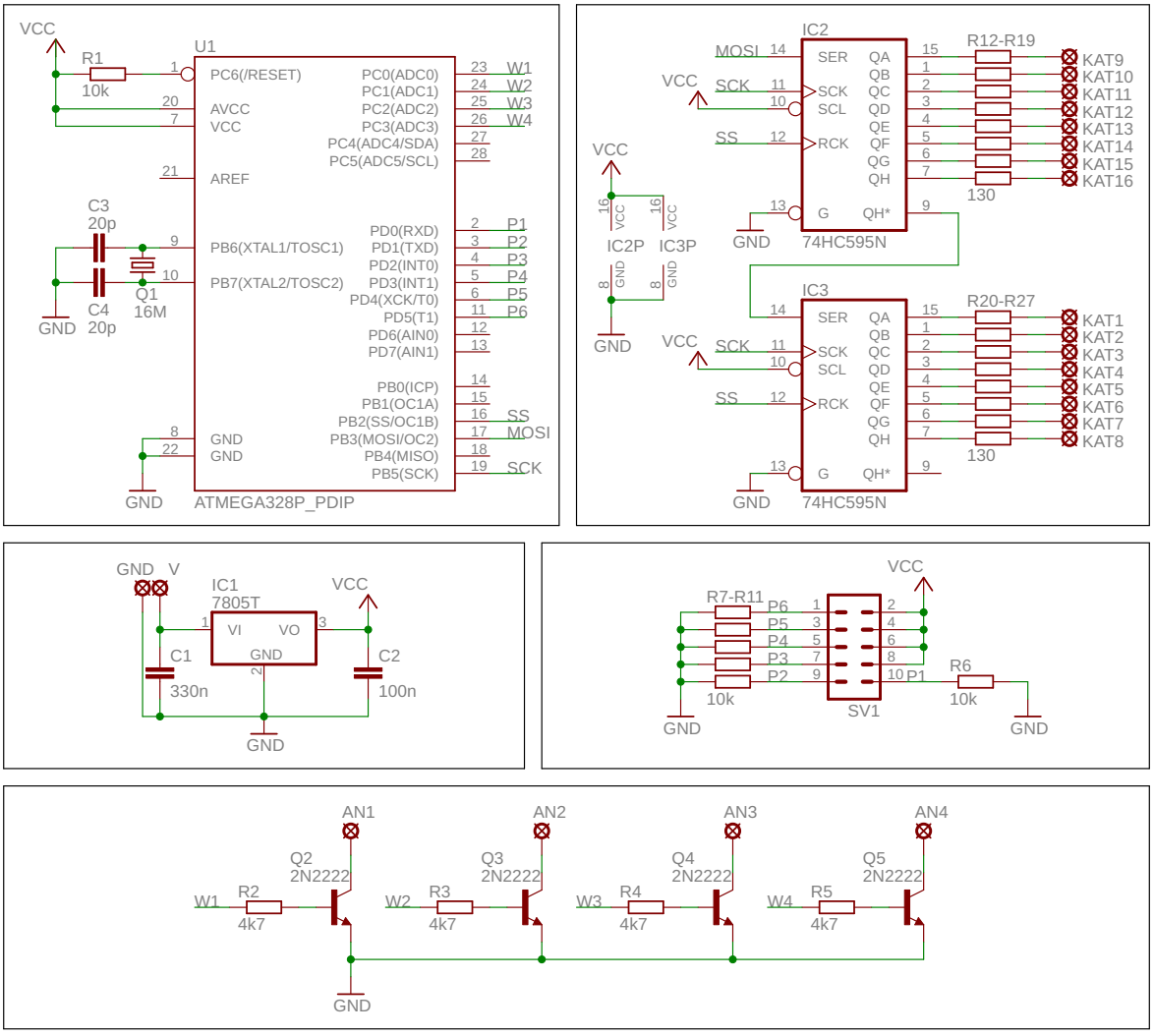
2. Budowa układu

Kostka LED jest sterowana dzięki wykorzystaniu mikrokontrolera ATmega328p. Wyświetlenie pojedynczej diody jest możliwe dzięki rejestrom przesuwnym oraz kluczom tranzystorowym. Listę zastosowanych elementów zestawiono w tabeli 1.

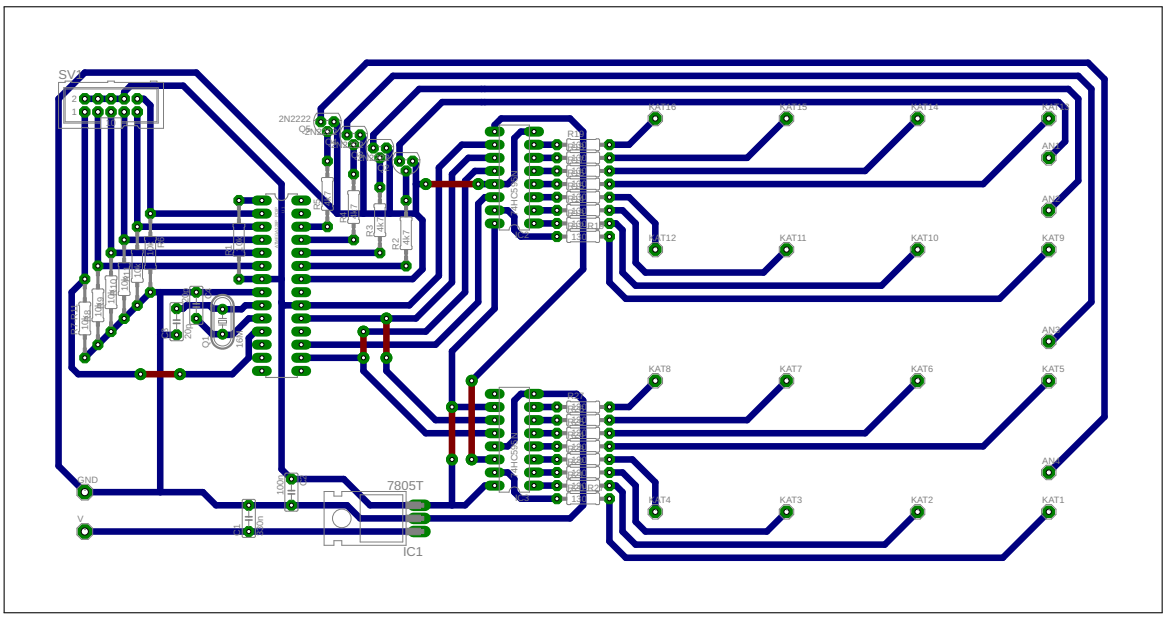
Schemat zrealizowanych połączeń został przedstawiony na rys. 2. Na jego podstawie przygotowano projekt płytki drukowanej (rys. 3). Ze względu na wygodę kostkę umieszczono obok pozostałych elementów, a przyciski połączono z płytką za pomocą wtyku IDC.

element	ilość	wartość/uwagi
mikrokontroler	1	ATmega328p
rejestr przesuwny	2	74HC595
oscylator kwarcowy	1	16 MHz
stabilizator napięcia	1	7805
kondensator	1	330 nF
kondensator	1	100 nF
kondensator	2	20 pF
rezystor	7	10 kΩ
rezystor	16	130 Ω
rezystor	4	4,7 kΩ
diody LED	64	zielone, 5mm, kąt 50°
przyciski chwilowe	6	—
tranzystor npn	4	2N2222

Tabela 1: Zastosowane elementy elektroniczne



Rysunek 2: Schemat układu



Rysunek 3: Projekt płytki drukowanej

3. Elementy oprogramowania

Pliki i biblioteki wchodzące w skład oprogramowania projektu zostały opisane poniżej. Pliki zostały zawarte w załączniku [Z.3].

Plik `main.c`

Plik zawiera podprogram główny. W pliku nie zostały zdefiniowane żadne funkcje – program korzysta z bibliotek opisanych w kolejnych punktach.

Schemat blokowy podprogramu głównego został przedstawiony na rys. 4.

Plik `definicje.h`

Plik został napisany w celu nadania portom i pinom nazw wygodniejszych w użyciu. Ponadto zadeklarowano zmienne używane przez większość bibliotek.

Bibioteka `timery`

Biblioteka jest odpowiedzialna za zainicjowanie liczników oraz określenie wektorów przerwań. Posiada następujące funkcje:

- `void timery_ustaw()` — ustawienie liczników w tryb CTC, określenie preskalerów i wartości OCnA.
- `ISR(TIMER0_COMPA_vect)` — przerwanie odpowiadające za sprawdzenie, czy użytkownik wcisnął przycisk. Program wykrywa zbocze narastające, zatem zastosowano zmienną pamiętającą stan obecny przycisków oraz zmienną określającą stan poprzedni. Stan danego przycisku jest określony kontkretnym bitem w tych zmiennych.
- `ISR(TIMER1_COMPA_vect)` — przerwanie odpowiadające za inkrementację zmiennej `krok`. Poza tym z każdym wywołaniem zmieniany jest stan diody symbolizującej jedzenie, dzięki czemu odróżnia się ono od węża.
- `ISR(TIMER2_COMPA_vect)` — przerwanie odpowiadające za zapalanie odpowiednich diod. Przez zastosowanie multipleksowania w danym czasie zapalają się diody tylko na jednej warstwie. W celu zminimalizowania efektu migania diod dobrano doświadczalnie wartość OC2A. W każdym wywołaniu przerwania dane o aktualnej warstwie wysyłane są do rejestrów, zapalane są diody na tej warstwie i inkrementowany jest licznik warstw.

Bibioteka `rejestry`

Biblioteka odpowiada za obsługę rejestrów przesuwnych. Posiada następujące funkcje:

- `void rejestry_ustaw()` — zainicjowanie interfejsu SPI.
- `uint8_t rejestry_wyslij(uint8_t dane))` — wysłanie danych do rejestru SPDR i czekanie na koniec transmisji.
- `uint8_t rejestry_16(uint16_t dane)` — umożliwia wysłanie danych 16–bitowych.

Bibioteka `animacja`

Biblioteka definiuje sekwencje animacji i długość ich trwania. Posiada następujące funkcje:

- `void animacja_poczatek()` — animacja wyświetlana przy włączeniu kostki.
- `void animacja_koniec()` — animacja wyświetlana przy śmierci węża.
- `void animacja_wygrana()` — animacja wyświetlana, gdy użytkownik wygra, czyli gdy wąż będzie miał długość 64.

Bibioteka `waz`

Biblioteka definiuje zachowanie weza. Posiada następujące funkcje:

- `void waz_ustaw()` — wyzerowanie wszystkich elementów węża. Potem następuje ustalenie początkowej długości węża i jego początkowe położenie.
- `void waz_przesun()` — wąż uzyskuje nowe położenie (przesunięcie o jedno pole w danym kierunku). Również sprawdzane jest, czy głowa węża trafiła na pole zajęte przez jego ciało, jeżeli tak, to wąż umiera (zmienna `koniec` przyjmuje wartość 1).
- `void waz_kierunek()` — pozycja głowy węża jest przesuwana o jedno pole w odpowiednim kierunku. Jednocześnie jest sprawdzane, czy głowa nie wyjdzie poza obszar kostki, jeżeli tak, to wąż umiera.

Bibioteka `jedzenie`

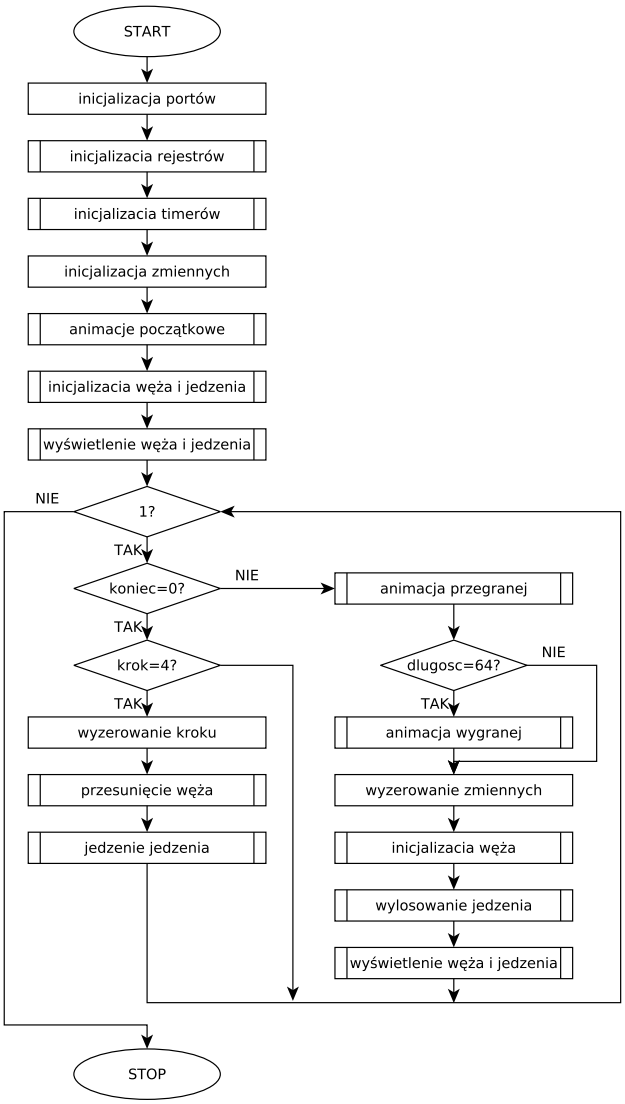
Biblioteka losuje pozycję jedzenia i sprawdza, czy wąż na nie trafia. Pozycje jedzenia nie musiały być losowane w sposób dokładny, zatem zastosowano uproszczony liniowy generator kongruencyjny. Dodatkową losowość uzystano przez modyfikowanie stanu w zależności od wybranego kierunku (następuje to w przerwaniu licznika TIMER0). Biblioteka posiada następujące funkcje:

- `void jedzenie_ustaw()` — zainicjowanie stanu, na podstawie którego będzie ustalana pozycja jedzenia, następnie ustalana jest ta pozycja.
- `void jedzenie_losuj()` — w tej funkcji jest generowana liczba pseudolosowa, następnie na jej podstawie określa się współrzędne jedzenia.
- `uint8_t jedzenie_blad()` — sprawdzenie, czy pozycja wylosowanego jedzenia nie pokrywa się z położeniem węża oraz czy ta pozycja nie wychodzi poza granice kostki.
- `void jedzenie_ustal()` — wykonywanie funkcji `jedzenie_losuj()` tak długo, aż funkcja `jedzenie_blad()` nie określi, że jedzenie znajduje się w akceptowalnej pozycji.
- `void jedzenie_jedzenia()` — sprawdzanie, czy głowa węża trafiła na jedzenie. Jeżeli tak, to długość węża zwiększa się o 1, losowane jest kolejne jedzenie oraz zwiększa się szybkość poruszania węża.

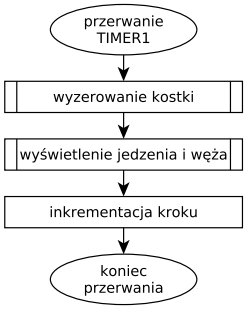
Bibioteka `kostka`

Biblioteka odpowiada za przypisanie do struktury `kostka` takich wartości, aby zapalały się odpowiednie diody. Posiada następujące funkcje:

- `void kostka_zeruj()` — zerowanie wsztystkich elementów struktury `kostka`.
- `void kostka_z_weza()` — dodanie do struktury wartości położenia elementów węża.
- `void kostka_z_jedzenia(uint8_t x)` — w zależności od zmiennej `x` następuje zapalenie lub zgaszenie diody oznaczającej pozycję węża.



Rysunek 4: Schemat blokowy podprogramu głównego



(a) TIMER1

(b) TIMER2

Rysunek 5: Schematy blokowe wybranych liczników

4. Wykorzystane narzędzia projektowe

Działając w systemie Windows 7, skorzystano w programu Eagle do określenia schematu połączeń elektrycznych i zaprojektowania płytki drukowanej. Dołączono dodatkowe biblioteki, które były dostępne na stronie www.sparkfun.com. Reszta wykorzystanych programów była uruchamiana w systemie operacyjnym Arch Linux:

- `gedit, vim` – edycja programów pisanych w języku C.
- `gcc-avr` – kompilacja projektu zapisanego w języku C.
- `avr-objcopy` – utworzenie pliku `hex`.
- `avrdude` – wgranie pliku `hex` do pamięci mikrokontrolera, wykorzystano programator USBasp.

5. Weryfikacja poprawności działania układu

Weryfikacja poprawności działania układu polegała na testowaniu gry przez kilka różnych osób. Ponadto sprawdzono, czy układ zachowuje się prawidłowo w ważnych fragmentach. Numery załączników, w których przedstawiono przykładowe porządane działanie podano w nawiasach. Sprawdzono poniższe zachowania układu:

- śmierć węża przy próbie wejścia w ścianę – gdy pozycja głowy węża znajdzie się poza polem diod, odtwarza się animacja śmierci węża (kilkukrotne zapalanie się i gaszenie diod).
- śmierć węża przy próbie wejścia w ciało – gdy pozycja głowy węża znajdzie się na pozycju zajętej przez którykolwiek segment ciała, odtwarza się animacja śmierci węża.
- jedzenie jedzenia – gdzy pozycja głowy węża znajdzie się na pozycji wylosowanego jedzenia, to długość węża zwiększa się o 1 oraz zmniejsza się wartość `OCR1A` (zatem wąż porusza się szybciej).
- przeciwstawne kierunki – na przykład, gdy wąż porusza się w prawo, to nie można zmienić kierunku na lewo.
- początek – przy włączeniu układu odtwarzana jest animacja początkowa.
- wygrana – gdzy wąż osiągnie długość 64 (czyli zajmie wszystkie diody), uznawana jest wygrana i odtwarzana jest specjalna animacja.

6. Obsługa układu

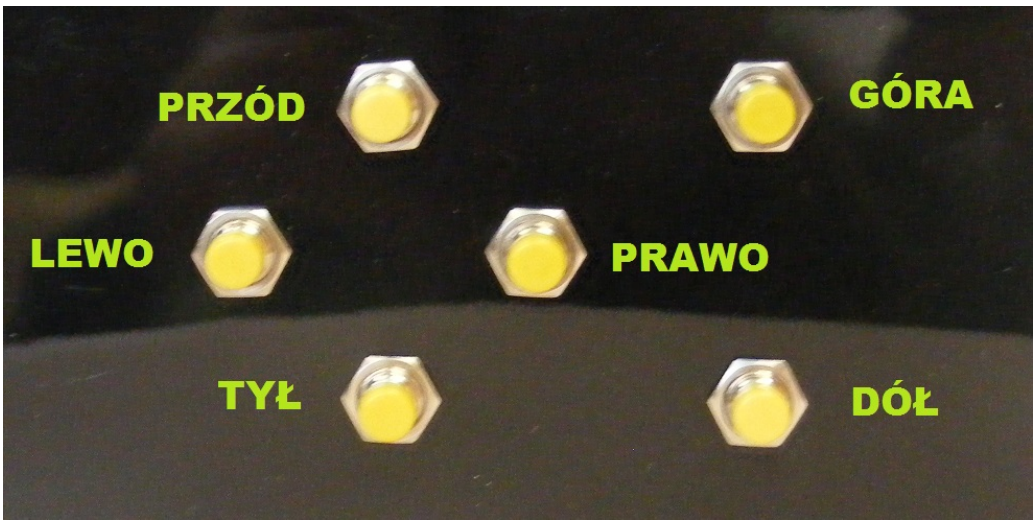
Do poprawnego działania układu potrzebne jest zapewnienie zasilania przez użytkownika. Z lewej strony obudowy znajduje się gniazzdo zasilania, preferowanym napięcie 5V.

Interakcja z użytkownikiem zachodzi przez diody tworzące kostkę oraz przez przyciski znajdujące się przed kostką. Przyciski odpowiadają za nadawanie odpowiedniego kierunku poruszania się węża. Można wybrać sześć kierunków, zatem na obudowie znajduje się sześć przycisków. Rozmieszczenie przycisków i odpowiadające im kierunki zostały przedstawione na rys. 6. Efekty działania użytkownika mogą być obserwowane na kostce (zdjęcie kostki przedstawiono na rys 7). Zapalane są diody przedstawiające węża. Ponadto widoczna jest dioda pokazująca położenie jedzenia, można ją odróżnić od reprezentacji węża tym, że okresowo zapala się i gaśnie.

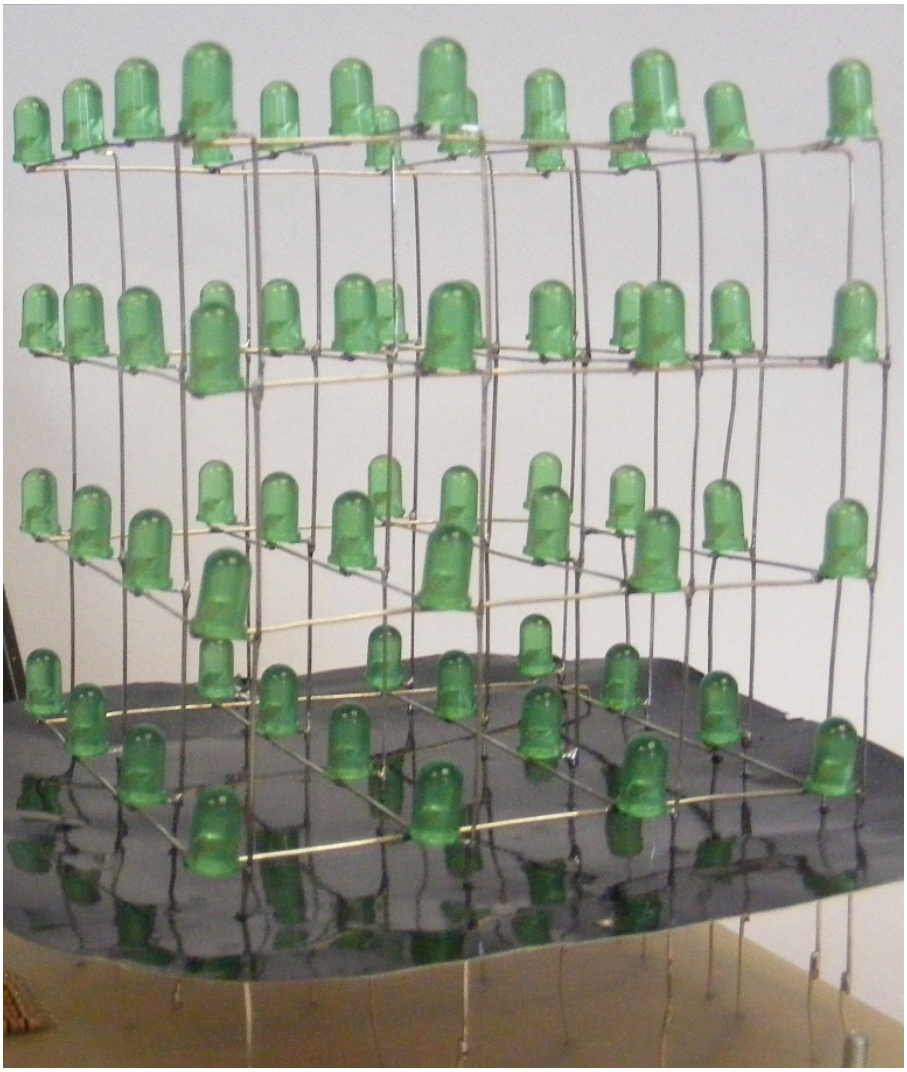
Z punktu widzenia użytkownika przebieg gry składa się z kilku etapów:

1. Obserwacja animacji początkowej.
2. Kierowanie wężam dopóki nie nastąpi jego śmierć.
3. Obserwacja animacji symbolizującej śmierć węża.
4. Jeżeli wąż zapełnił całą kostkę, to następuje wygrana (obserwacja odpowiedniej animacji).
5. Powrót do etapu 2.

Przykładowy przebieg gry został przedstawiony w załączniku [Z.1] i[Z.2].



Rysunek 6: Przyciski do sterowania kierunkiem ruchu węża



Rysunek 7: Kostka zbudowana z diod LED

7. Załączniki

[Z.1] `Snake3D_dzialanie1.avi` – film przedstawiający przykładowy przebieg gry.

[Z.2] `Snake3D_dzialanie2.avi` – film przedstawiający przykładowy przebieg gry.

[Z.3] `Snake3D_oprogramowanie.zip` – oprogramowanie ATmegi napisane w języku C.

8. Literatura

[L.1] 8–bit shift registers with 3–state output registers [online] [dostęp 20 listopada 2014]. Dostępny w Internecie: <http://www.ti.com/lit/ds/scls041h/scls041h.pdf>.

[L.2] Amplifier Transistors NPN Silicon [online] [dostęp 21 listopada 2014]. Dostępny w Internecie: <http://www.solarbotics.net/library/datasheets/2N2222.pdf>.

[L.3] Atmel 8–bit microcontroller wtth 4/8/16/32kbytes in–system programmable flash datasheet [online] [dostęp 17 listopada 2014]. Dostępny w Internecie: <http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf>.

[L.4] Bity konfiguracyjne (FUSEBITY) [online] [dostęp 20 listopada 2014]. Dostępny w Internecie: <http://www.voytek.evbox.pl/programy/fusebity/fusebity.html>.

[L.5] Chmielowiec, Robert. LED Cube - 4x4x4, Karta projektu zaliczeniowego, Systemy mikroprocesorowe – 2014. Poznań, 2014.

[L.6] EPP: Generator liczb losowych by miszcz310 [online] [dostęp 25 grudnia 2014]. Dostępny w Internecie: <http://mikrokontrolery.blogspot.com/2011/03/epp-generator-losowy-program.html>.

[L.7] LED Cube 4x4x4 by chr [online] [dostęp 4 listopada 2014]. Dostępny w Internecie: <http://www.instructables.com/id/LED-Cube-4x4x4/>.

[L.8] LM78XX / LM78XXA 3–Terminal 1 A Positive Voltage Regulator [online] [dostęp 20 listopada 2014]. Dostępny w Internecie: <https://www.fairchildsemi.com/datasheets/LM/LM7805.pdf>.

[L.9] Programowanie mikrokontrolerów AVR w Linuksie [online] [dostęp 14 listopada 2014]. Dostępny w Internecie: <http://www.nibyblog.pl/programowanie-mikrokontrolerow-avr-w-linuksie-3445.html>.

[L.10] QKT – HC49S quartz crystal [online] [dostęp 20 listopada 2014]. Dostępny w Internecie: <http://www.kingtronics.com/pdf/QKT-HC49S.pdf>.