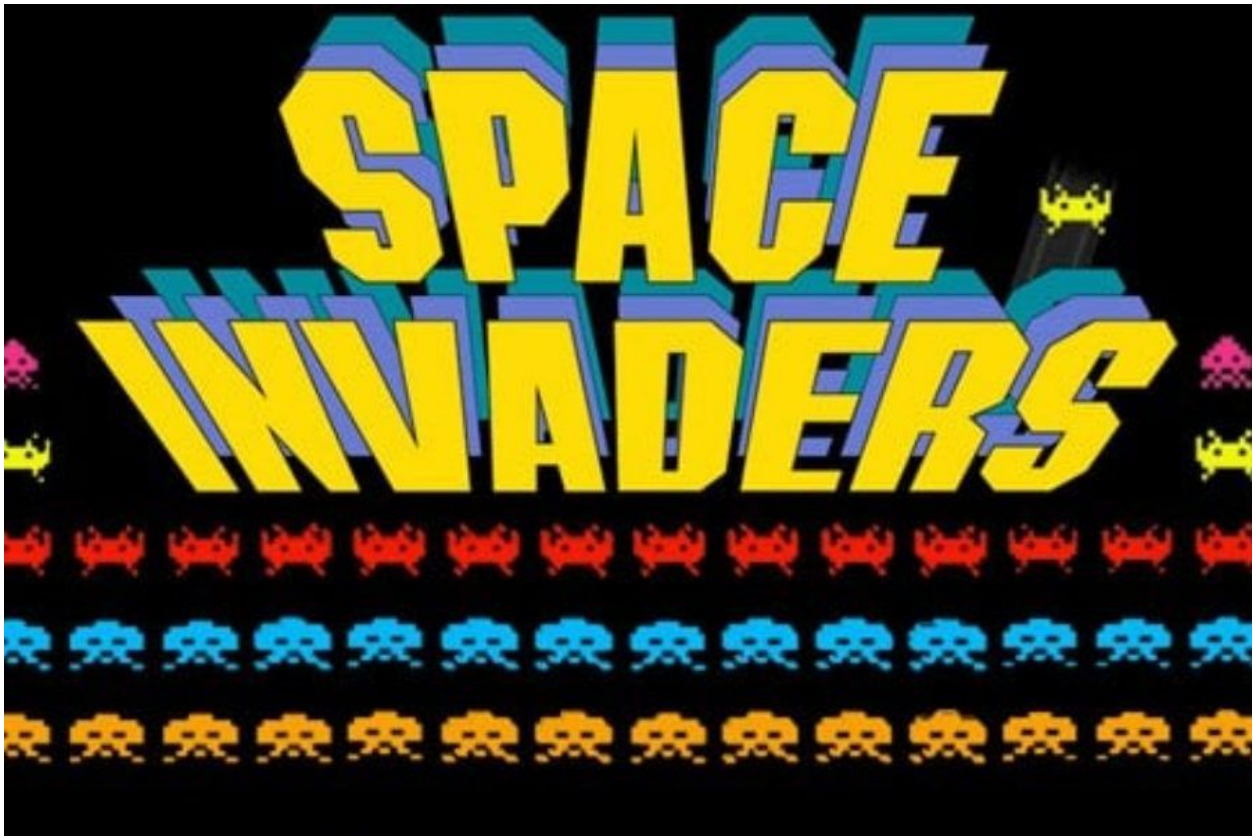


Arduino Space Invaders - proof of concept version based on ATmega 2560

24.11.2019

Author: Szymon Sandura

Studied subject: Techniki mikroprocesorowe i systemy wbudowane

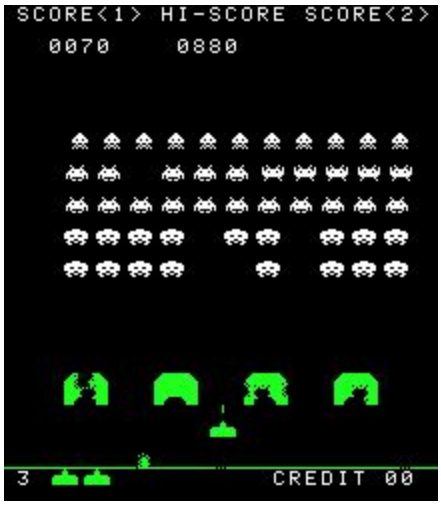
Supervisor: mgr inż. Jan Duda

Introduction

This project is an adaptation of famous Space Invaders game. It was loosely inspired by the project from 1962 named Spacewar and the concept of sci-fi movies. The first release of Space Invaders was created in 1978 and was originally released in form of arcade system named Taito 8080.

Concept

The Space Invaders game has very basic rules. Gameplay revolves around shooting incoming enemies coming from the top of the screen. The player is controlling a ship placed on the bottom of the screen and is able to steer it horizontally. Additionally, his ship is equipped with laser gun.



Przykład implementacji gry Space Invaders

Main objectives

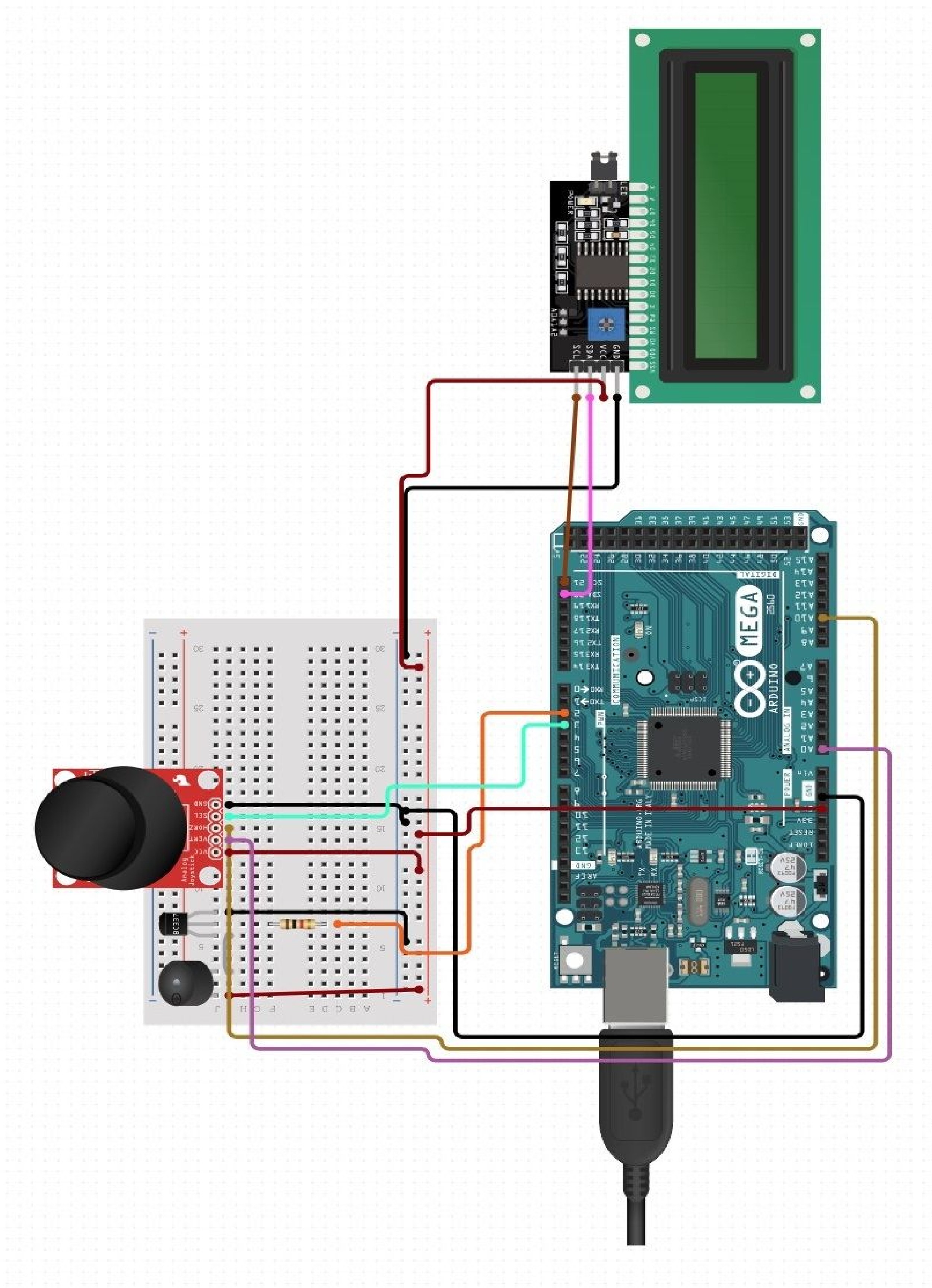
1. Reproduction of essential game mechanics from the original game, namely: movement of bullets, enemies, moving player's ship, bullet-ship collision.
2. Building game maps using random number generator.
3. Implementing analog stick controls (also utilizing built-in button)
4. Presentation of game's content on LCD screen.
5. Handling asynchronous threads in order to decouple velocity of different objects in the game.
6. Composing two simple soundtracks. One for the initialization of the game, one before level reloading.
7. Building a simple amplifier scheme in order to produce louder sound.

Hardware Requirements

- Arduino ATmega 2560 or similar
- LCD 16x2 characters screen oriented vertically using I2C connection
- Analog stick with built-in button (alternatively a standalone switch)
- Passive DC-powered buzzer
- Resistor 1k Ohm
- Transistor BC337 of NPN type

Scheme

On the next page there is a sample scheme of built system.


MEGA
ARDUINO

Code concept documentation

Full code available below:
https://github.com/HerringTheCoder/Arduino-Space-Invaders/blob/master/sketch_SpaceInvaders/sketch_SpaceInvaders.ino

I. Initiating application

```
void setup()
{
  randomSeed(analogRead(5)); // using unconnected A5 pin to generate random numbers
  pinMode(JOYS_SW_DIO, INPUT_PULLUP);
  pinMode(BUZZER_DIO, OUTPUT);
  digitalWrite(JOYS_SW_DIO, HIGH);
  lcd.init();       // initialize the lcd
  lcd.backlight(); //turn on backlight
  loadLevel();
  playMusic();
}
```

First line of the setup method contains the definition of RNG (Random Number Generator) based on specifically picked, unconnected analog pin. Noise generated on it ensures the unpredictability of generated numbers. Next comes the configuration of pin used for handling analog stick button, which is configured as an input using built-in pull up resistor and the pin responsible for sending a signal to buzzer.

In the next step there is an initialization of LCD screen along with activating its backlight.
After these procedures comes level loading and playing the initial ‘welcome’ music.

II. Handling screen

Do obsługi wyświetlacza wykorzystano bibliotekę LiquidCrystal_I2C dostępną pod adresem: https://github.com/johnrickman/LiquidCrystal_I2C

Jest to specjalna wersja znanej biblioteki LiquidCrystal zapewniająca kompatybilność z interfejsem I2C, ale poza drobnymi różnicami we wstępnej konfiguracji, obsługa programowa tej biblioteki jest prawie identyczna. Stworzenie obiektu reprezentującego wyświetlacz zostało zdefiniowane w następujący sposób:

```
/* global variables */
LiquidCrystal_I2C lcd(0x3f,16,2);  // set the LCD address to 0x3f for a 16 chars and 2 line display
```

Uwaga! Przypisany adres może zależeć od producenta wyświetlacza LCD oraz od zastosowanego układu. Celem znalezienia adresu zaleca się użycie programu I2cScanner dostępnego pod adresem: https://playground.arduino.cc/Main/I2cScanner/

III. Handling sound

Stworzona ścieżka dźwiękowa składa się z wybranych wysokości dźwięku oraz rytmu. W tym celu zadeklarowano częstotliwości odpowiadające wybranym nutom w systemie równomiernie temperowanym przy założeniu wzorcowego A=440Hz :

```
 /*sound section*/
#define NOTE_G4 391 //G4
#define NOTE_A4 440 //A4
#define NOTE_C5 523 //C5
#define NOTE_D5 587 //D5
#define NOTE_E5_FLAT 622 //E flat (Eb)
#define NOTE_E5 659  //E5
```

Dźwięki występujące w wyższych oktawach generuje się przez podwojenie częstotliwości dźwięku z oktawy niższej (np. A4= 440Hz, A5= 880Hz, A6= 1760Hz itd.)

Zmienny rytm uzyskano natomiast dzięki zastosowaniu opóźnień. Długość opóźnienia determinuje tempo odtwarzania muzyki.

```
#define TEMPO 200 /*sets the sustain parameter of each note/pause, higher value = slower */
```

Przykładowy zapis melodii:

```
void playMusicBreak()
{
  tone(BUZZER_DIO, NOTE_A4);
  delay(TEMPO);
  noTone(BUZZER_DIO);
  delay(TEMPO);
  tone(BUZZER_DIO, NOTE_C5);
  delay(TEMPO);
  noTone(BUZZER_DIO);
  delay(TEMPO);
  tone(BUZZER_DIO, NOTE_G4);
  delay(TEMPO);
  tone(BUZZER_DIO, NOTE_A4);
  delay(TEMPO);
  noTone(BUZZER_DIO);
}
```

IV. Generating game map

Mapa gry przechowywana jest w postaci tablicy znaków o domyślnym dwukrotnością wielkości ekranu. Pierwsza połowa tabeli przechowuje obiekty widoczne dla gracza i pokrywa się z tablicą wyświetlacza. Druga połowa tabeli to przestrzeń zarezerwowana dla statków przeciwnika.
Każdy występujący obiekt ma przydzielony stosowny symbol:

```
#define SHIP_ASCII 0b10101010 /* Katakana 工 sign */
#define ALIEN_ASCII 0b01011000 /* Upper Case 'X' */
#define BULLET 0b10100101 /* - sign */
#define EXPLOSION 0b00101010 /*   * character */
```

Poniższa tabela obrazuje powyższy koncept mapy gry (tu dla wyświetlacza o wymiarach 10 x 2):

			X	X		X			X		X								
		X			X		X	X		X		X	*				-	-	工

Przestrzeń niewidoczna Przestrzeń ekranu LCD

V. Handling controls

Sterowanie w grze pozwala na wykonanie trzech akcji: skręt w lewo, skręt w prawo oraz wystrzelenie pocisku. Zmiana pozycji zrealizowana została za pomocą gałki analogowej z przetwornikiem analogowo-cyfrowym o rozdzielczości 10 bitów (zakres wartości od 0 do 1023). Dla zapewnienia responsywności zadeklarowano wartości graniczne (ang. deadzone) przy których odbywa się przemieszczenie statku. Wartości zostały dobrane w sposób heurystyczny.

```
/*analog input secton*/
#define JOYS_VRY_DEADZONE_LEFT 750
#define JOYS_VRY_DEADZONE_RIGHT 150
```

W pętli głównej programu co ok. 20ms odczytywana jest bieżąca pozycja gałki, a następnie porównywana z wartościami granicznymi, co przy spełnieniu warunków skutkuje wywołaniem stosownej metody: turnLeft (linie 79-86), turnRight (linie 90-98) lub w przypadku naciśnięcia przycisku fire (linie 130-146).

VI. Asynchronous events - main loop

Układ ATmega 2560 dysponuje procesorem jednowątkowym. Oznacza to, że w danym momencie można obsłużyć wyłącznie jedną operację.
Kluczowym warunkiem udanej rozgrywki w grze akcji jest responsywność sterowania, co w praktyce realizuje się skracając opóźnienia między operacjami odczytu pozycji kontrolera. Konflikt powstaje jednak w momencie kiedy na tym samym wątku trzeba obsłużyć również przemieszczanie się statków przeciwnika, które powinno odbywać się nawet kilka lub kilkanaście razy wolniej.
W związku z tym należy zaimplementować asynchroniczne odwołania.
Aby uzyskać ten efekt czas gry podzielono na cykle odczytu (nazwa luźno nawiązuje do cykli procesora), których interwał wynosi ustaloną długość opóźnienia. Dodatkowo wprowadzono stałą odnoszącą się do skali czasu, która determinuje po ilu cyklach następuje wyjście z pętli odczytu.
Przykładowo dla domyślnych wartości:

```
/*Speed control*/
#define READ_DELAY 20
#define TIMESCALE 5 /*sets the time scale of objects (enemies, bullets) moving on the map,
```

Teoretyczny czas jednej tury wynosi:

$$READ\_DELAY\ (20ms)\ *\ TIMESCALE\ (5)\ =\ 100ms$$

Po upłynięciu tego czasu wykonują się procedury moveBullets (linie 150-176) oraz moveEnemies (linie 178-198), które aktualizują pozycję pocisków (przesunięcie w stronę statków wroga) oraz pozycję przeciwników (przesunięcie w stronę gracza).

Po zakończeniu 32. tury (czyli po upłynięciu czasu potrzebnego do przebycia drogi między krańcowymi punktami tabeli/mapy gry) następuje ponowne wygenerowanie poziomu, licznik tur resetuje się i pojawia się przerywnik muzyczny.