Arduino to Blender (010; 28.07.2009; *arduino; blender*)

W artykule tym opiszę podstawy komunikacji pomiędzy Arduino a środowiskiem Blendera. Komunikacja oparta będzie na transmisji poprzez port szeregowy i wykorzystaniu języka Python do sterowania wirtualnym światem Blendera.

Na początek wideo ukazujące efekt opisanych niżej rozważań: http://www.vimeo.com/5798613

Do projektu niezbędne będą:

Blender 2.49a (http://www.blender.org/download/get-blender/)

Python 2.6.2 (http://www.python.org/download/releases/2.6.2/)

pywin32-214 (pywin32-214.win32-py2.6.exe; http://sourceforge.net/projects/pywin32/files/)

pySerial 2.4 (pyserial-2.4.win32.exe ; http://sourceforge.net/projects/pyserial/files/)

Instalacja Pythona 2.6.2 jest niezbędna do prawidłowej pracy Blendera w wersji 2.49a. Biblioteka pywin32 jest zestawem niezbędnych rozszerzeń funkcji języka Python dla środowiska Windows i daje również dodatkowe środowisko edycji i uruchamiania skryptów Pythona. Biblioteka pySerial umożliwia komunikację szeregową w środowisku Python. Ważne jest, aby biblioteki były dopasowane do posiadanej wersji Pythona i prawidłowo zainstalowane.

Odbieranie danych z portu szeregowego w Blenderze.

Programujemy Arduino do wysyłania danych na port szeregowy np. z odczytu napięcia z potencjometru (artykuł 007). W blenderze przechodzimy do okna Text Editor i umieszczamy tam następujący kod:

```
import serial  #import biblioteki komunikacji szeregowej

serialport = serial.Serial('COM4', 9600)    #deklaracja portu

for i in range(1, 20):   #pętla wykonywana 20-krotnie
     x = serialport.read(size=1)   #odczyt jednego bajtu z portu
     y = ord(x)     #zamiana na liczbę dziesiętną
     print "y=", y  #wyświetlenie wartości zmiennej

else:
     serialport.close()  #zamknięcie portu
```

Skrypt uruchamiamy kombinacją klawiszy alt+p lub z menu *Text/Run Python Script.* Po uzyskaniu połączenia w oknie konsoli Blendera pojawiać powinny się kolejne wartości przesłane z Arduino. Nie polecam wykonywać dużej liczby powtórzeń pętli w programie, gdyż tego programu po uruchomionego nie da się przerwać (można tylko zamknąć 'na siłę' Blendera).

Plik Blendera do pobrania: http://myinventions.pl/010/SerialTest.blend

Edycja obiektu w Blenderze z poziomu skryptów.

Uzależnijmy położenie domyślnego sześcianu od wartości odczytanej z portu następująco:

```
import serial
import Blender #import modułu ogólnego Blendera

serialport = serial.Serial('COM4', 9600)
ob = Blender.Object.Get ('Cube')

for i in range(1, 100):
     x = serialport.read(size=1)
     y = 0.01*ord(x)
     ob.setLocation(0,y,0)    #ustalenie nowego położenia obiektu
     Blender.Redraw()    #odświeżenie wszystkich okien programu

else:
     serialport.close()
```

Nie jest to jedyna możliwość realizacji tego zadania. Dokonanie importu submodułu na początku kodu pozwala uprościć późniejszy kod, np.:

```
import serial
import Blender
from Blender import Object

serialport = serial.Serial('COM4', 9600)
ob = Object.Get ('Cube')      #a było  Blender.Object.Get


for i in range(1, 100):
     x = serialport.read(size=1)
     y = 0.01*ord(x)
     print "y=", y
     ob.setLocation(0,y,0)
     Blender.Redraw()

else:
     serialport.close()
```

Wczytanie obiektu możemy zrealizować również poprzez klasę Scene następująco zmieniając kod:

```
import serial
import Blender

serialport = serial.Serial('COM4', 9600)
ob = Blender.Scene.GetCurrent().getActiveObject()

for i in range(1, 100):
     x = serialport.read(size=1)
     y = 0.01*ord(x)
     print "y=", y
     ob.setLocation(0,y,0)
     Blender.Redraw()
else:
     serialport.close()
```

Aby dokonać obrotu obiektu wokół osi należy użyć polecenia *setEuler(x,y,z), a* do skalowania obiektu polecenia *setSize(x,y,z)*, gdzie za *x,y* i *z* podstawiamy wartości lub zmienne liczbowe. Polecenia *WaitCursor(1)* i *WaitCursor(0)* submodułu *Window* modułu *Blender*, powodują odpowiednio włączenie i wyłączenie kursora w postaci klepsydry. Kod łączący te wszystkie polecenia:

```
import serial
import Blender

serialport = serial.Serial('COM4', 9600)
ob = Blender.Object.Get ('Cube')
Blender.Window.WaitCursor(1)

for i in range(1, 100):
     x = serialport.read(size=1)
     y = 0.01*ord(x)
     #print "y=", y
     ob.setLocation(0,2*y,2.55)
     ob.setEuler(0,0,y)
     ob.setSize(0.5+y,0.5+y,0.01+y)
     Blender.Redraw()
else:
     serialport.close()
     Blender.Window.WaitCursor(0)
```

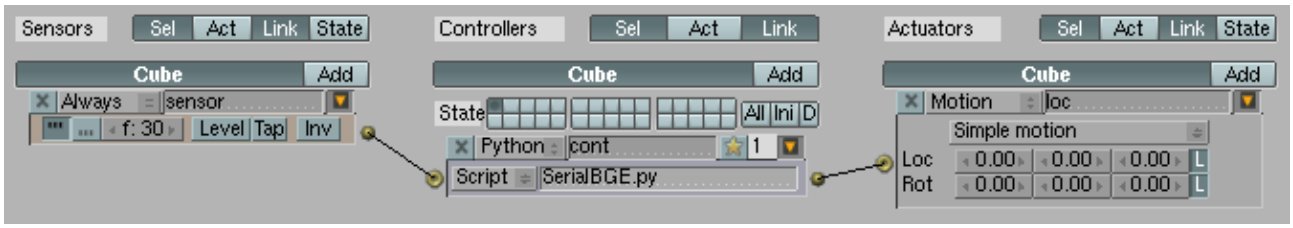
Plik Blendera z powyższym kodem do pobrania: http://myinventions.pl/010/SerialEditObject.blend

Dla tego przykładu zadowalająca częstotliwość wysyłania danych przez Arduino wynosi 30Hz. Powyżej około 65Hz skrypt w Blenderze nie nadąża odbierać danych.

Komunikacja szeregowa w Blender Game Engine.

Niesłychanie przydatna byłaby możliwość wykorzystania danych przesyłanych przez port szeregowy w silniku gier Blendera. Obsługa klawiatury, myszy i joysticka jest już standardowo wbudowana za pomocą odpowiednich sensorów.

Do obiektu, którym chcemy sterować przyporządkowujemy sensor Always, kontroler Python oraz aktuator Motion o nazwie *loc*. Obiekt pozostawiamy typu Static. Dla sensora włączamy Pulse Mode – True Level Triggering i ustawiamy *f:30* (sensor wysyłać będzie impuls True co 30 jednostek czasu – domyślnie w Blenderze na sekundę przypada 60 jednostek czasu). W kontrolerze wpisujemy nazwę skryptu pythona *SerialBGE.py.*



W oknie Text Editor Bledera wpisujemy następujący kod (zmieniając domyślną nazwę na *SerialBGE.py):*

```
import serial
import GameLogic
ser = serial.Serial('COM4', 9600)
contr = GameLogic.getCurrentController()
location=contr.actuators["loc"]
x = ser.read(size=1)
y = 0.010*(ord(x)-128)
print "y=", y
location.dLoc=[y,0,0]
contr.activate(location)
ser.close()
```

Programujemy Arduino tak, aby wysyłało wartości jednobajtowe na port szeregowy z częstotliwością maksymalną 2Hz (np. odczytane z wejścia analogowego przy zastosowaniu potencjometrycznego dzielnika napięcia – patrz artykuł 007).

Po uruchomieniu symulacji (klawisz p) obserwujemy kolejno:

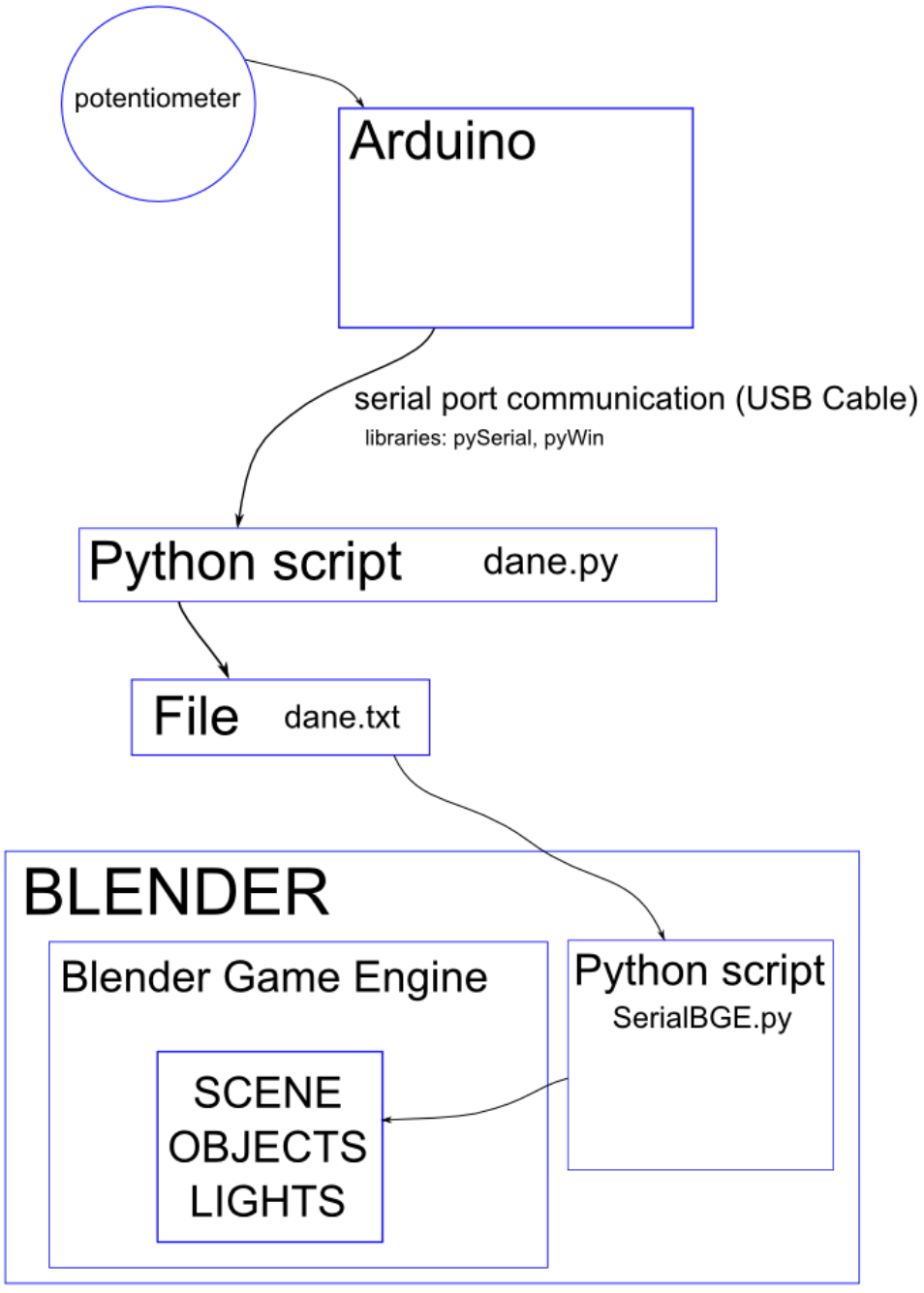
- zadziałanie aktuatora Always – uruchomienie skryptu: otwarcie portu, odczytanie wartości, obliczenia, wyświetlenie wartości w konsoli, ustalenie prędkości obiektu, aktualizacja aktuatora, zamknięcie portu (całość trwa około 0,5s);
- uruchomienie symulacji (widoczny ruch obiektu);
- zadziałanie aktuatora Always po przerwie f=30 jednostek czasu – zatrzymanie symulacji i uruchomienie skryptu (znów trwa około 0,5s, widoczne jest ponowne łączenie się skryptu z Arduino, które restartuje się i przesyła dane);
- ponowne uruchomienie symulacji (widoczny ruch obiektu z nową prędkością);
- itd...

Ogromną wadą tego sposobu komunikacji, jest długie wykonywanie skryptu, który powoduje niemożliwe do przyjęcia przerwy w symulacji. Czy istnieje na to jakiś sposób? Po przekopaniu się przez dokumentację poleceń Pythona dla Blendera i BGE postanowiłem szukać innego sposobu niż obsługa portu szeregowego w Blenderze.

Plik Blendera z powyższym kodem do pobrania: http://myinventions.pl/010/SerialBGE.blend

Komunikacja poprzez plik.

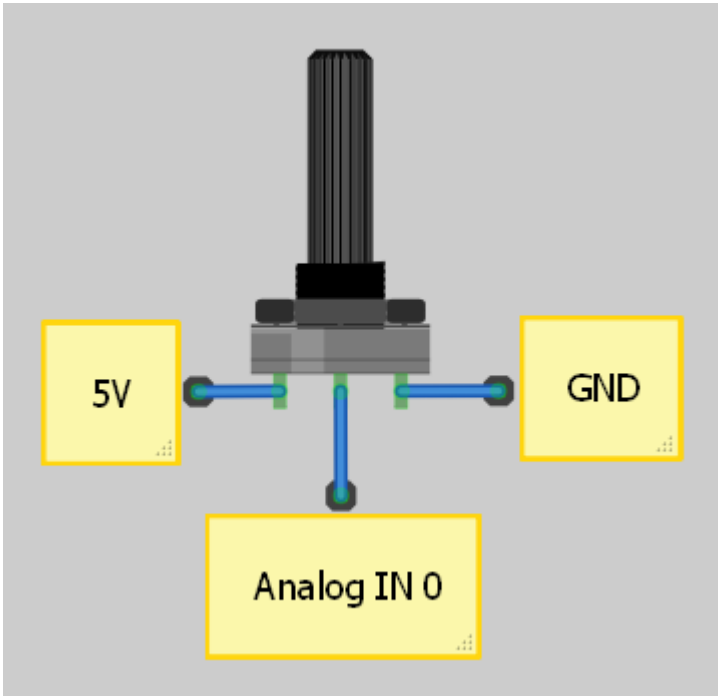
Schemat działania metody będzie następujący: skrypt pythona inicjuje połączenie szeregowe i w pętli zapisuje wartości odczytane z portu do pliku. Skrypt pythona uruchamiany przez Blendera z poziomu kontrolera logiki odczytuje wartości z pliku i dokonuje pożądanych operacji w BGE. Zarówno częstotliwość wysyłania sygnału przez Arduino, częstotliwość wykonywania pętli programu zapisującego plik, jak i częstotliwość wywoływania kontrolera BGE muszą być odpowiednio dobrane.



1. Zaprogramujmy Arduino do wysyłania danych z częstotliwością 25Hz

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print(analogRead(0)/4, BYTE);
  delay(40);
}
```



2. Utwórzmy dwa plik o nazwach dane.txt oraz dane.py. Do edycji tego drugiego możemy użyć notatnika, ale zalecane jest użycie edytora języka python (w języku tym ważne są tabulatury i spacje). Do edycji użyć możemy zatem zainstalowanego wcześniej PythonWin (prawy przycisk myszy: *edit with PythonWin*). W menu widnieć może również polecenie *edit with IDLE*, otwierające równie dobry edytor. Treść kodu:

```
import serial

port = serial.Serial('COM4', 9600) #otwarcie połączenia
     #szeregowego równoznaczne z restartem Arduino oraz
     #wyczyszczeniem bufora portu

for i in range(0, 10):
    dane=open('dane.txt', 'r+b') #otwarcie pliku w trybie odczytu
          # i zapisu binarnego
    x = port.read(size=1)     #odczytanie jednego bajtu
    print x          #wyświetlenie wartości zmiennej w konsoli
    dane.write(x)    #zapis danej do pliku - skasowanie poprzedniej
    dane.close()     #zamknięcie pliku

port.close()   #zamknięcie portu
```

Składnia języka Python wymaga użycia wcięcia bloku poleceń pętli for tabulaturą. Polecenie odczytu jednego bajtu z portu nie zostanie wykonane, dopóki tenże nie znajdzie się w buforze portu, zatem następuje niejako automatyczna synchronizacja częstotliwości pobierania sygnału z częstotliwością ich pojawiania się na porcie (nie jest wymagane wstawianie przerwy w pętli).

W tym miejscu można przetestować poprawność działania układu (po podłączeniu Arduino i uruchomieniu skryptu w konsoli powinno wyświetlić się 10 kolejnych znaków).

Do dalszych zastosowań można usunąć linię 'print x',oraz należy zwiększyć liczbę wykonywanych pętli do np. 200, co da nam 8 sekund działania programu (200/25).

3. Tworzymy nową scenę Blendera. Do obiektu, którym chcemy sterować przyporządkowujemy sensor *Always*, kontroler *Python* oraz aktuator *Motion* o nazwie *loc*. Obiekt pozostawiamy typu Static. Dla sensora włączamy *Pulse Mode – True Level Triggering* i ustawiamy *f:3*. W kontrolerze wpisujemy nazwę skryptu pythona *SerialBGE.py*. Treść skryptu jest następująca:

```
import GameLogic

dane=open('dane.txt', 'rb') #otworzenie pliku do odczytu binernego
x=dane.read()  #odczytanie wartości z pliku
dane.close()   #zamknięcie pliku
print x

contr = GameLogic.getCurrentController()
location=contr.actuators["loc"]
y = 0.001*(ord(x)-128)
location.dLoc=[y,0,0]
contr.activate(location)
```

Ponieważ dla sensora ustawiliśmy *f:3* pobieranie danych z pliku następować będzie z częstotliwością około 20Hz. Okazuje się, że metoda z przesyłaniem danych poprzez plik jest wydajna w tym zastosowaniu.

4. Pliki dane.txt, dane.py oraz scena blendera umieścić należy w jednym folderze. Uruchamiamy plik blendera, podłączamy Arduino, uruchamiamy skrypt dane.py (można po prostu dwa razy kliknąć) i uruchamiamy silnik gry Blendera (klawisz P). W konsolach pythona i blendera obserwować możemy kolejne wartości, a w oknie Blendera poruszający się obiekt z prędkością zależną od położenia potencjometru.

Pliki do pobrania:

http://myinventions.pl/010/dane.py

http://myinventions.pl/010/SerialBGE2.blend

Co dalej?

Jak widać, komunikacja szeregowa daje duże możliwości zastosowania skryptów Pythona do sterowania Blenderem. Wspomnę, że Pythona możemy rozszerzyć np. o bibliotekę analizy obrazu video z kamery na żywo i tenże obraz może oddziaływać na środowisko Blendera.

Zwracam uwagę, że jak dotąd wykorzystywałem w artykułach komunikację szeregową do przesyłania pojedynczych bajtów danych – bo takie najłatwiej odebrać. Jednak gdy zajdzie potrzeba wysyłać większe wartości liczbowe niezbędne będzie ich odpowiednie odbieranie, gdyż zawarte będą one w większej liczbie ramek danych.

Pełna dokumentacja poleceń Pythona dla Blendera 2.49:
http://www.blender.org/documentation/249PythonDoc/

Dokumentacja poleceń Pythona dla Blender Game Engine:
http://www.blender.org/documentation/249PythonDoc/GE/
http://www.tutorialsforblender3d.com/GameDoc/index_GameDoc.html

Podstawy pythona:
http://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie
http://www.python.rk.edu.pl/
http://www.python.org.pl/
http://troman.pl/?p=510

Dokumentacja pySerial zawierająca opis poleceń możliwych do zastosowania dzięki tej bibliotece:
http://pyserial.sourceforge.net/pyserial.html; http://pyserial.wiki.sourceforge.net/pySerial

Strona pywin32: http://python.net/crew/mhammond/win32/