

Magazyn programistów i liderów zespołów IT

programista

5/2017 (60)
czerwiec/lipiec

Cena 23,90 zł (w tym VAT 5%)

PICO-8

PODRÓŻ W KRAINĘ CYFROWEJ FANTAZJI



Chmura obliczeniowa
w Pythonie

C++: Rule of Zero

Jak zbudować robota
z Raspberry Pi

Wywiad z Rasmusem
Lerdorfem

Wdrażanie aplikacji
w środowisku Openshift



Join our team of Makers!

career.cybercom.com

**Change
tomorrow
with us**

www.cybercom.pl



Cybercom Poland Sp. z o.o.

ul. Hrubieszowska 2, 01-209 Warszawa / ul. Dowborczyków 25, 90-019 Łódź

Technologiczne „kombinatorstwo”

Współczesne technologie komputerowe umożliwiają programistom realizowanie wytwórzów wyobraźni pod postacią różnorodnych gier w sposób niemalże nieograniczony. Niektóre osoby powiązane z tzw. „starą szkołą” twierdzą jednak, że zaawansowanie technologiczne częściowo odbiera frajdę związaną z tworzeniem. Nad tym problemem skupili się autorzy „zmyślonej” konsoli PICO-8, którzy zdefiniowali własną platformę sprzętową przypominającą niegdysiejsze konsole. Głównym założeniem projektowym było wprowadzenie celowych, stosunkowo restrykcyjnych ograniczeń co do zasobów udostępnionych twórcom gier. Tytułowy artykuł na temat programowania, a raczej „kombinowania” w PICO-8, kierujemy do wszystkich osób, które wytwarzanie oprogramowania traktują nie tylko profesjonalnie, ale również hobbyściecznie.

Na łamach tego wydania prezentujemy również wywiad z Rasmusem Lerdorfem, twórcą PHP. Wysoki procentowy udział tego języka programowania w sektorze aplikacji webowych powoduje, że nieustannie jest o nim głośno, a teraz nawet jeszcze głosniej – głównie w związku z ukazaniem się stabilnej wersji siódmej oraz problemami skutkującymi porzuceniem prac nad wersją 6.0. O kwestie związane z rozwojem PHP pytaliśmy naszego rozmówca podczas konferencji We Are Developers 2017, która odbyła się niedawno w stolicy Austrii.

PS. Autorem projektu okładki do tego numeru jest Sebastian Rosik.

Zapraszam do lektury!
Michał Leszczyński

BIBLIOTEKI I NARZĘDZIA

- [Konsola PICO-8: podróz w krainę cyfrowej fantazji](#) 4
Rafał Kocisz

JĘZYKI PROGRAMOWANIA

- [Rule of zero – pisz więcej, pisząc mniej](#) 16
Paweł "KrzaQ" Zakrzewski

PROGRAMOWANIE SYSTEMÓW OSADZONYCH

- [Budujemy robota. Sterowanie serwami z poziomu Raspberry Pi](#) 24
Wojciech Sura

PROGRAMOWANIE APLIKACJI WEBOWYCH

- [Wdrażanie aplikacji w środowisku Openshift](#) 34
Michał Pawlik

LABORATORIUM BOTTEGA

- [Chmura obliczeniowa w Pythonie](#) 42
Krzesztof Mędrela

TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ

- [Generowanie danych za pomocą języka SQL](#) 50
Marek Żukowicz

WYWIAD

- [Dokład zmierza PHP. Wywiad z Rasmusem Lerdorfem, twórcą języka PHP](#) 54
Michał Leszczyński, Rafał Kocisz

KLUB LIDERA IT

- [Kryteria wartości biznesowej](#) 58
Mariusz Sieraczkiewicz, Paweł Wrzeszcz

- [Sztuka udzielania feedbacku](#) 62
Katarzyna Suska

STREFA CTF

- [CONFidence 2017 – C64 & Encryption Service](#) 66
Krzesztof "sasza" Stopczyński, Robert "rodbert" Tomkowski

PLANETA IT

- [Przygoda z testami UX nie tylko dla stron WWW i aplikacji](#) 72
Will Parker

- [Kurs angielskiego dla programistów. Lekcja 5](#) 74
Łukasz Piwko

reklama



BOOTCAMP

Front-end



BOOTCAMP

Full-stack

PIERWSZY
BOOTCAMP
FULL-STACK
W POLSCE



BOOTCAMP

Data
Science

Zdobądź
zawód
przyszłości

Dowiedz się więcej na
odołamacz.pl

sages

Konsola PICO-8: podróż w krainę cyfrowej fantazji

Kiedy w 2012 roku Joseph White (pseudonim „zep”) rozpoczął prace nad „wyimaginowaną” konsolą PICO-8, mało kto chyba spodziewał się, że ten malutki, niezależny projekt zdobędzie tak dużą popularność i rozbudzi wyobraźnię tak wielkiej rzeszy ludzi – poczynając od osób rozpoczynających swoją przygodę z komputerem, a kończąc na doświadczonych programistach. W ramach niniejszego artykułu postaram się znaleźć odpowiedź na pytanie, co kryje się za fenomenem PICO-8, licząc jednocześnie, że uda mi się zarazić Cię, drogi Czytelniku, fascynacją tym ciekawym narzędziem.

ZAMIAST WSTĘPU...

Co jak co, ale programowanie konsol gier od zawsze kojarzyło mi się z czymś niedostępny, odległym i elitarnym. Większość konsol to tzw. platformy zamknięte – tworzenie oprogramowania na takie systemy wymaga zazwyczaj dołączenia do specjalnego programu developerskiego (tylko po spełnieniu określonych, często drakońskich wymagań), zakupu drogiego devkitu, zapoznania się z tonami trudnej do zrozumienia dokumentacji i wreszcie – spełnienia szeregu wymagań dotyczących kontroli jakości tworzonego produktu. Nic dziwnego, że wytwarzaniem produkcji konsolowych zajmują się zazwyczaj duże firmy z branży gamedev dysponujące sporą ilością wiedzy i pieniędzy – nawet pomimo tego, że ostatni czas bariera wejścia na ten rynek została i tak poważnie obniżona dla niezależnych twórców gier.

Spróbujmy przenieść się na chwilę do krainy fantazji, gdzie istnieje wyimaginowana konsola, do której programowania nie potrzeba devkitu (co więcej, nie wymaga ona żadnego dedykowanego urządzenia poza PC-tem), z dostępem do kompletnej i czytelnej dokumentacji (a ponadto z możliwością podejrzenia kodu źródłowego każdej dostępnej na nią aplikacji), z setkami darmowych gier, z systemem dystrybucji, który nie narzuca żadnych ograniczeń czy wymagań. I na dodatek wspierana przez potężną, przyjazną społeczność, gdzie zawsze znajdziesz się ktoś chętny do pomocy.

Eeee, nie ma przecież takiej krainy i takiego urządzenia... – po myślisz sobie zapewne. A jednak! Poznaj PICO-8: całkiem rzeczywiście konsolę rodem z fantazji, którą możesz sobie zafundować za niecałe piętnaście dolarów USD :)

PICO-8... CO TO WŁAŚCIWIE JEST?

Zaczniemy od udzielenia odpowiedzi na pytanie fundamentalne: czym jest PICO-8? Odpowiedź, która w mojej opinii jest najbardziej trafna, brzmi: PICO-8 to minimalistyczna, aczkolwiek pod wieloma względami innowacyjna platforma programowo-sprzętowa, pozwalająca w łatwy sposób pobierać, używać, tworzyć i dystrybuuwać niewielkie gry oraz aplikacje. PICO-8, pomimo swojej prostoty i ograniczeń (nałożonych celowo), posiada wszystkie kluczowe składniki nowoczesnych platform (takich jak PlayStation, Xbox, Android czy iOS). Są to:

- » środowisko uruchomieniowe,

- » narzędzia developerskie oraz dokumentacja (wsparcie dla twórców aplikacji),
- » narzędzia do cyfrowej dystrybucji aplikacji,
- » duża ilość dedykowanych aplikacji,
- » preżna społeczność użytkowników i twórców oprogramowania.

W ujęciu czysto praktycznym: PICO-8 to program komputerowy, za pomocą którego będziesz mógł łatwo, szybko i przyjemnie uruchamiać, tworzyć i współdzielić z innymi jego użytkownikami minimalistyczne (ale wcale nie banalne) gry i aplikacje.

Skoro PICO-8 jest konsolą, to jakie ma parametry techniczne? Celne pytanie! Już spieszę z odpowiedzią. Specyfikacja techniczna PICO-8 jest następująca:

- » wyświetlacz o rozdzielcości 128x128 pikseli,
- » stała paleta 16 kolorów,
- » kontroler wyposażony w 6 przycisków,
- » 4 kanały dźwięku plus 64 konfigurowalne efekty dźwiękowe,
- » obsługiwane nośniki danych: kartridże o rozmiarze 32 kB,
- » język programowania: Lua (maksymalnie 8192 symboli kodu per kartridż),
- » bank 128 duszków o rozmiarach 8x8 pikseli oraz kolejnych 128 współdzielonych,
- » mapa kafelków o rozmiarach 128x32 (drugi, podobny obszar – współdzielony).

Tak, tak... to nie żart. Ósemka w nazwie PICO-8 znalazła się nie bez kozery. Mamy tu do czynienia z konsolą jak najbardziej ośmioramienną. Jednakże jest to nowoczesna konsola ośmioramienna, o czym przekonać się możesz, kontynuując lekturę tego artykułu. To, co jest znamienne w przypadku PICO-8, to fakt, że jego autor celowo nałożył na swoje dzieło opisane powyżej ograniczenia, licząc na to, że pobudzą one kreatywność potencjalnych twórców oprogramowania na ten system. I – o dziwo – tak się właśnie stało!

W kolejnych podpunktach niniejszego tekstu omówię po kolei elementy składowe platformy PICO-8, przedstawię szczegółowe omówienie API tej „zmyślonej” konsoli. W drugiej części artykułu (która będziesz mógł przeczytać na łamach następnego numeru „Programisty”) opiszę, jak przy pomocy PICO-8 zaprogramować prostą (ale pełną) grę. Liczę, że lektura moich artykułów na temat PICO-8 zachęci Cię do pełnej przygód podróży w krainę cyfrowej fantazji w towarzystwie tej małej, ale oferującej ocean możliwości „zmyślonej” konsoli.

TKH Development Center



praca z pasją

kadry@cctechnology.pl

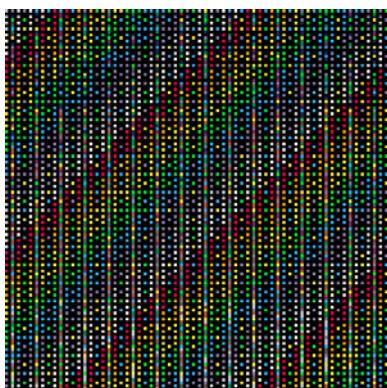
ŚRODOWISKO URUCHOMIENIOWE

Aby rozpocząć przygodę z PICO-8, należy skierować się na stronę <https://www.lexaloffle.com/pico-8.php>. W pierwszej kolejności trzeba założyć konto (potrzebny do tego będzie adres mailowy), a następnie zakupić aplikację PICO-8. Aplikacja ta jest dostępna na systemy operacyjne Windows, Linux, Mac oraz Raspberry Pi. Kupując licencję, uzyskujemy dostęp do plików wykonywalnych aplikacji PICO-8 na wszystkie wspomniane wyżej platformy (w wersji DRM-free), wraz z prawem do wszystkich przyszłych wersji tego programu. Cena PICO-8 to 14.99 dolarów. Warto też rozważyć alternatywną opcję zakupu w postaci drugiej „wirtualnej” konsoli oferowanej przez Lexaloffle: Voxatron (<https://www.lexaloffle.com/voxatron.php>). Przy takim zakupie (cena: 19.99 dolarów) otrzymujemy PICO-8 jako darmowy bonus (oferta ta jest ważna przez czas, w którym Voxatron pozostaje w fazie Alfa). Sam Voxatron jest – podobnie jak PICO-8 – wyimaginowaną konsolą, tyle że bardziej rozbudowaną i operującą na wokselach: pozwalającą budować aplikacje 3D.

Wracając do aplikacji PICO-8: jest ona niewątpliwie kluczowym składnikiem platformy: oferuje ona środowisko uruchomieniowe, narzędzia developerskie oraz mechanizm do zdalnego pobierania aplikacji (tzw. cartów), dodanych przez użytkowników PICO-8 za pośrednictwem serwisu <https://www.lexaloffle.com>. Przekonajmy się, jak wygląda pierwsze uruchomienie tej „zmyślonej” konsoli.

HELLO, PICO-8!

Uruchomienie PICO-8 objawia się charakterystycznym efektem wizualnym (patrz Rysunek 1) oraz dźwiękowym. Para ta stanowi rodzaj wizytówki, tudzież znaku rozpoznawczego tej małej konsolki.



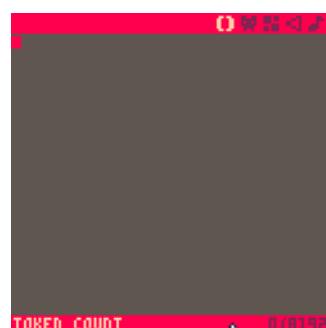
Rysunek 1. Efekt wizualny towarzyszący uruchomieniu PICO-8

W dalszej kolejności użytkownika wita ekran powłoki z klasycznym znakiem zachęty (>) i przyjaźnie migającym, czerwonym kursem (patrz Rysunek 2).



Rysunek 2. PICO-8: ekran powłoki PICO-8 tuż po uruchomieniu

Powłoka stanowi istotny element PICO-8, w dalszej części artykułu opowiem szerzej na jej temat, teraz jednak chciałbym, nie wchodząc zbytnio w szczegóły, przedstawić proces zbudowania pierwszej działającej aplikacji PICO-8. W tym celu należy wcisnąć klawisz Esc, co przestawi konsolę w tryb edycji kodu (patrz Rysunek 3).



Rysunek 3. PICO-8: ekran edycji kodu

W edytorze wpisujemy następujący wiersz kodu:

```
print("hello, pico-8!")
```

Ponownie wciskając klawisz Esc, powracamy do trybu powłoki, wpisujemy komendę run i... możemy cieszyć się naszym pierwszym działającym programem na konsoli PICO-8 (patrz Rysunek 4).



Rysunek 4. Hello, PICO-8!

PICO-8: POWŁOKA

W tym podpunkcie opowiem o tym, jakie możliwości oferuje powłoka PICO-8 i jak za jej pomocą zarządzać zasobami naszej wirtualnej konsoli.

Po instalacji PICO-8 w systemie gospodarza tworzony jest katalog, w którym przechowywane będą jej zasoby. W moim przypadku (korzystam z systemu Windows 10) katalog ten znajduje się w następującej lokalizacji: c:\Users\rkocisz\AppData\Roaming\pico-8

Jest to standardowy katalog instalacyjny PICO-8. Jeśli korzystasz z Windows i podmienisz swoją nazwę użytkownika w tej ścieżce (rkocisz) na Twoją – to bez trudu znajdziesz ten katalog. Wewnątrz odkryjesz sporo plików konfiguracyjnych oraz roboczych podkatalogów PICO-8. Na tym etapie zwrócić proszę uwagę na podkatalog carts, w którym umieszczone są kartidze z aplikacjami PICO-8. Katalog ten jest bardzo istotny: w systemie plików PICO-8 postrzegany jest on jako katalog główny (/). Tuż po instalacji PICO katalog ten jest pusty (można się o tym przekonać, wpisując w powłoce komendę dir lub ls). Po uruchomieniu komendy install_demos w katalogu głównym utworzony zostanie podkatalog demos, w którym znajdziesz zestaw cartów prezentujących

możliwości PICO-8. Przejdz do tego katalogu z poziomu powłoki PICO-8 (`cd demos`), wypisz jego zawartość (`dir`) i wczytaj przykładową wybraną aplikację za pomocą komendy `load`, np.: `load jelpi.p8`. Teraz wywołaj komendę `run` (jej zadaniem jest uruchomienie karty załadowanej do pamięci PICO-8) i podziwiaj możliwości naszej wirtualnej konsolki (patrz Rysunek 5).



Rysunek 5. Jelpi: mini-gra platformowa demonstrująca możliwości PICO-8

PICO-8: WBUDOWANE NARZĘDZIA

Jak już być może zauważyłeś, siła i urok PICO-8 polega (między innymi) na tym, że mamy tu do czynienia z bardzo prostym i wygodnym środowiskiem. Wszystkie elementy PICO-8 są ze sobą świetnie zintegrowane, przez co otrzymujemy do swojej dyspozycji bardzo samowystarczalne narzędzie, z którym po prostu przyjemnie się pracuje. Wiele osób zajmujących się uczeniem programowania czy też projektowania gier decyduje się na korzystanie z PICO-8, rezygnując z potężniejszych, lecz też bardziej skomplikowanych narzędzi (takich jak np. Unity). Pracę z PICO-8 można rozpoczęć w przeciągu kilku minut, zaś po kolejnych kilkunastu – widać bardzo konkretne efekty. Prostota jest niewątpliwie jedną z najmocniejszych stron PICO-8. W tym podpunkcie przyjrzymy się po kolei wbudowanym narzędziom PICO-8: poznamy ich możliwości i ograniczenia.

Na początek wczytaj ponownie kartridż z grą Jelpi. Wciśnij teraz klawisz Esc, co przełączy PICO-8 w tryb edycji kodu (patrz Rysunek 6).

```
THE ADVENTURES OF JELPI
BY ZEP

TO DO:
-- LEVELS AND MONSTERS
-- TITLE / RESTART LOGIC
-- BLOCK LOOT
-- TOP-SOLID GROUND
-- BETTER DUMPING

-- CONFIG: NUM_PLAYERS 1 OR 2
NUM_PLAYERS = 1
CORRUPT_MODE = FALSE
MAX_ACTORS = 128
MUSIC(0, 0, 3)

CHUNK_TAP MOVE ACTORS V 0.01
TAP 17124 240478168
```

Rysunek 6. PICO-8: edytor kodu (Jelpi)

Przeglądając Rysunek 6, zwróciły uwagę na 5 ikon umieszczonych na górnym pasku z prawej strony (zaznaczyłem je na rysunku białym prostokątem). Klikając na te ikonki, można przełączać się pomiędzy narzędziami oferowanymi przez PICO-8. Pierwsza ikonka (nawiasy) reprezentuje edytor kodu (pokazany na Rysunku 6). Przyznać trzeba, że autor PICO-8 musiał się nieźle nagimnastykować, aby stworzyć w miarę funkcjonalny i wygodny edytor działający w rozdzielcości 128x128 pikseli. Kluczową rolę spełnia tutaj zapewne świetnie zaprojektowana, charakterystyczna czcionka o stałej wielkości зна-

ków (4x8 pikseli). Edytor naszej wirtualnej konsolki wyposażony jest w zestaw podstawowych opcji dostępnych w „poważnych” narzędziach służących do pisania kodu: zaznaczanie, kopianie, nawigacja po tekście, obsługa wcięć, wyszukiwanie, kopianie linii, a także cofanie oraz odtwarzanie operacji.

Druga ikonka (duszek) reprezentuje edytor duszków (ang. *sprites*). Na Rysunku 7 przedstawiony jest ten edytor, wypełniony elementami gry Jelpi.



Rysunek 7. PICO-8: edytor duszków (Jelpi)

Zanim przejdziemy do szczegółowego omówienia elementów przedstawionego ekranu, wypada powiedzieć kilka słów na temat możliwości graficznych PICO-8. Zaczniemy od tego, że nasza „wyimaginowana” konsola posiada stałą paletę barw w liczbie 16. Paleta ta pokazana jest na Rysunku 8.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Rysunek 8. PICO-8: paleta kolorów (źródło: <https://neko250.github.io/pico8-api/>)

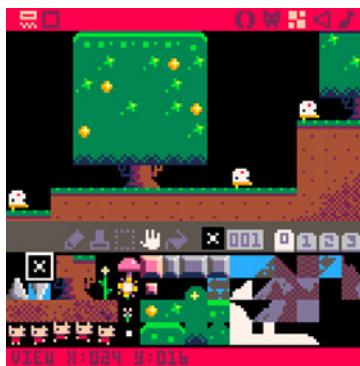
Każdy kolor z palety ma przypisany konkretny numer, który przekazywany jest do funkcji rysujących PICO-8. W efekcie tej decyzji wszystkie gry na tą konsolę mają bardzo charakterystyczną, łatwo rozpoznawalną kolorystykę. Dzięki temu, podobnie jak w przypadku starych gier na 8-bitowe platformy pokroju ZX Spectrum czy Commodore 64, zazwyczaj wystarczy jeden rzut oka na zrzut ekranu aplikacji, żeby stwierdzić: to jest gra na PICO-8. Ten celowy zabieg jest kolejnym elementem budowania unikalnego charakteru wirtualnej konsolki.

Kolejna cecha silnika graficznego PICO-8 to stały rozmiar duszków (8x8 pikseli) oraz ograniczona ich liczba (128 slotów podstawowych oraz 128 slotów współdzielonych z mapą kafelków).Więcej na temat tych ograniczeń opowie w sekcji poświęconej graficznemu API konsoli.

Konstrukcja edytora duszków zdeterminowana jest przez opisane wyżej ograniczenia. W lewym górnym rogu znajduje się obszar edycji konkretnego duszka; duszek ten oznaczony jest w dolnej części ekranu reprezentującą jedną z czterech części arkusza duszków. W obszarze edycji duszka możemy wykonywać jedną z pięciu operacji: rysowanie kolorem (oznaczonym na palecie umieszczo-

nej w prawym górnym rogu ekranu), stempłowanie (co w praktyce oznacza wklejanie skopiowanego fragmentu duszka), zaznaczanie, przeciąganie oraz wypełnianie kolorem. Pod obszarem palety kolorów znajdują się dwa suwaki: jeden z nich (górnny) służy do definiowania rozmiaru pędzla, drugi – pozwala ustawiać rozmiar obszaru arkusza duszków widocznego w obszarze edycji. Pod suwakami znajduje się osiem ikonek służących do ustawiania flag duszków (o flagach napiszę więcej w dalszej części artykułu). Jeszcze niżej znajduje się indykatorka bieżącego duszka oraz zakładki służące do przełączania aktywnych obszarów arkusza duszków. Na samym dole ekranu umieszczony jest pasek statusu, gdzie PICO-8 wyświetla bieżące, użyteczne informacje (np. współrzędne piksela edytowanego na arkuszu duszków, numer wybranego koloru itp.). Być może brzmi to wszystko dość skomplikowanie, jednak w praktyce edytor duszków jest bardzo wygodnym i intuicyjnym narzędziem (wypróbuj go, a przekonasz się!).

Kolejna ikonka na górnym pasku (mapa) reprezentuje edytor mapy kafelków (patrz Rysunek 9). Dolna część edytora jest identyczna jak w przypadku edytora duszków: mamy tutaj widok fragmentu arkusza duszków, na którym możemy wybrać aktywnego duszka. W górnej części ekranu znajduje się obszar mapy, na której możemy ustawiać poszczególne kafelki (duszki). Obszar ten można oczywiście przesuwać, powiększać i pomniejszać, tak aby edycja była wygodna.



Rysunek 9. Jelpi: edytor mapy kafelków

Czwarta ikonka (trąbka) reprezentuje edytor efektów dźwiękowych (patrz Rysunek 10), a piąta (nutka) – edytor muzyki (patrz Rysunek 11).

Te dwa elementy PICO-8 omówię bardziej szczegółowo w drugim odcinku artykułu, przy okazji projektowania efektów dźwiękowych oraz komponowania muzyki dla omawianej tam, przykładowej gry.



Rysunek 10. Jelpi: edytor efektów dźwiękowych



Rysunek 11. Jelpi: edytor muzyki

PICO-8: DYSTRYBUCJA APLIKACJI

Mechanizm cyfrowej dystrybucji aplikacji jest dziś nieodłącznym elementem każdej szanującej się platformy sprzętowo-programowej. PICO-8 nie jest tu wyjątkiem. W niniejszym podpunkcie opiszę, jak w jego przypadku wygląda to rozwiązanie.

Pierwszy element układanki to BBS: prosty system współdzialeń kartów dostępny na stronie producenta PICO-8: <https://www.lexaloffle.com/bbs>. W sekcji BBS poświęconej PICO-8 mamy dostęp do olbrzymiego (i z dnia na dzień powiększającego się) zbioru kart, które dodają użytkownicy PICO-8.

Proces dodawania nowego carta jest uproszczony do granic możliwości. Na początek trzeba zalogować się na stronie Lexaloffle (każdy zarejestrowany użytkownik PICO-8 tak czy inaczej posiada konto na tej stronie, więc nie powinien być to żaden problem). Następnie przechodzimy do sekcji *Submit* (<https://www.lexaloffle.com/pico-8.php?page=submit>), gdzie umieszczona jest prosta i klarowna instrukcja dotycząca submisji. Cały ten proces trwa naprawdę kilka minut. Dystrybuowane w ten sposób carty dostępne są dla każdego użytkownika serwisu (logowanie nie jest wymagane). W związku z tym każdy odwiedzający stronę BBS może przeglądać dostępne aplikacje, zagrać w nie (dzięki temu, że PICO-8 oferuje opcję eksportu gry w wersji HTML5) tudzież skomentować (każda gra posiada swój własny wątek na forum BBS).

Jeśli z kolei chciałbyś pobrać i uruchomić aplikacje w natywnym środowisku, aplikacja PICO-8 oferuje narzędzie o nazwie Splore, za którego pośrednictwem można przeglądać zasoby kartów dostępnych w BBS i pobierać je. Przeglądarkę Splore uruchamiamy z poziomu powłoki, wpisując komendę *splore*; jej ekran powitalny przedstawiony jest na Rysunku 12.



Rysunek 12. PICO-8: Ekran powitalny Splore

Splore, jak wszystkie elementy PICO-8, działa prosto i intuicyjne. Wciskając odpowiednio klawisze lewo/prawo, wybieramy jedną z kategorii przeglądania: lokalne, ulubione, nowe, wyróżnione, praca w toku, współpraca, jam, szukaj. Wciskając Enter, wchodzimy

w konkretny tryb przeglądania. W przypadku trybu lokalnego możemy skorzystać z prostej przeglądarki katalogów (to dobra opcja dla osób, które nie lubią pracować z wierszem poleceń); w przypadku trybu wyszukiwania – wpisujemy żądane słowo kluczowe (np. tytuł carta); w pozostałych przypadkach PICO-8 pobiera z BBS'a listę kartów z konkretnej kategorii i wyświetla je na liście. Po wybraniu interesującego nas carta wystarczy raz jeszcze wcisnąć Enter i już po chwili jego zawartość załadowana jest do pamięci PICO-8, po czym carta zostaje automatycznie uruchomiony. Oczywiście w każdej chwili możemy wrócić do Splore, przejść do trybu edycji, przejrzeć kod itd. Załadowany do pamięci carta możemy za pomocą komendy save zapisać w lokalnym systemie plików. Na Rysunku 13 przedstawiona jest przeglądarka Splore wyświetlająca listę wyróżnionych gier.



Rysunek 13. Splore: lista wyróżnionych gier

Istotnym elementem systemu dystrybucji gier tworzonych za pomocą PICO-8 jest wspomniana wcześniej opcja eksportu aplikacji do formatu HTML5 (komenda *export* z poziomu powłoki). Dzięki magii transkompilatora Emscripten (<https://github.com/kripken/emscripten>) otrzymujemy w pełni niezależną, działającą w każdej nowoczesnej przeglądarce wersję eksportowanej aplikacji i co najważniejsze – możemy zrobić z nią, co nam się żywnie podoba (np. umieścić w zewnętrznych serwisach sprzedaży czy dystrybucji gier takich jak itch.io, Newgrounds czy Kongregate).

SPOŁECZNOŚĆ PICO-8

Poziom sukcesu danej platformy jest wprost proporcjonalny do rozmiaru społeczności, która powstaje wokół niej. Jeśli przyjąć taki punkt widzenia, to można śmiało powiedzieć, że PICO-8 – jako platforma – miewa się bardzo dobrze.

Jaka jest społeczność PICO-8? Pierwsze skojarzenia, które przychodzą do głowy, to: otwarta i przyjacielska. Otwartość w pewnym sensie wbudowana jest w PICO-8. Wszystkie carty opublikowane za pośrednictwem BBS'a z definicji są otwarte, każdy może obejrzeć sobie ich zawartość, włącznie z przejrzeniem kodu źródłowego. Jaka inną platformą oferuje aż taką transparentność?

Każda społeczność potrzebuje medium służącego do wymiany informacji. Forum Lexaloffle (BBS), Twitter, Facebook, wiki, blogi, serwisy techniczne... Wszędzie tam znajdziesz masę przydatnych informacji o PICO-8. Ostatnimi czasy wśród hackerów (rozumianych w pozytywnym sensie) popularne stało się przenoszenie przeróżnych aplikacji na PICO-8 tudzież udowadnianie, że coś, co wydaje się niemożliwe do zrealizowania na tą malutką konsolkę, właśnie jest możliwe. W ten sposób na PICO-8 powstał re-make silnika SCUMM, interpreter języków Lisp oraz Brainfuck i cała masa innych niesamowitych projektów. Przy czym autorzy tychże projektów chętnie dzielą się informacjami na temat swoich „przy-

gód” z PICO-8, dokumentując je w mediach społecznościowych, na swoich blogach itd. PICO-8 pomaga w tym, jak może, oferując wygodne narzędzia do robienia zrzutów ekranu, a nawet filmików (w postaci animowanych gifów).

PROGRAMOWANIE PICO-8

Mam nadzieję, że na tym etapie czujesz już mniej więcej, czym jest PICO-8 jako platforma. Teraz czas przejść do bardziej konkretnych zagadnień (w końcu nazwa magazynu, który właśnie czytasz, do czegoś zobowiązuje!): opowiem teraz pokrótce o tym, jak programować aplikacje dla PICO-8.

First things first, jak mawiają Amerykanie: aplikacje na platformę PICO-8 pisane są w języku programowania Lua.Lua ma niewątpliwie ugruntowaną pozycję w środowisku *gamedev*, głównie jako język skryptowy, osadzany w grach lub narzędziach wspomagających tworzenie gier. Implementacja interpretera Lua jest stabilna (bieżąca wersja to 5.3.4, PICO-8 opiera się na wersji 5.2), wydajna i wyprbowana w wielu bojach. Sam język jest dobrze udokumentowany i sprawdza się bardzo dobrze w szeregu zastosowań. Jego składnia jest prosta i przejrzysta, zaś dynamiczny charakter daje wiele ciekawych możliwości i sprawia, że kod tworzony w Lua jest bardzo ekspresyjny. Krótko mówiąc – Lua dobrze wpasowuje się w charakter PICO-8, tak więc wybór autora „wyimaginowanej” konsoli w tym zakresie wydaje się oczywisty. Co ciekawe, autor wprowadził kilka drobnych, aczkolwiek przydatnych modyfikacji języka (np. operator typu `+=`, `=itp.`, bądź skróconą składnię instrukcji warunkowej).

Drugim kluczowym elementem związanym z programowaniem aplikacji dla PICO-8 jest zbiór funkcji wbudowanych (rodzaj biblioteki standardowej) służących do manipulacji zasobami konsoli. Zbiór ten (PICO-8 API) dzieli się na następujące kategorie:

- » rysowanie,
- » obsługa urządzeń wejścia,
- » rysowanie i manipulacja mapą,
- » operacje matematyczne,
- » operacje na pamięci,
- » operacje na tablicach,
- » obsługa dźwięku,
- » obsługa danych kartów,
- » obsługa współprogramów (ang. *coroutines*),
- » różne.

PICO-8 API cechuje się podobnymi właściwościami jak inne elementy składowe platformy: jest proste, spójne i celowe. Patrząc z programistycznego punktu widzenia, za projekt tego API autorowi należą się wielkie brawa. Każda funkcja wchodząca w jego skład ma konkretny, określony cel. Nie znajdziesz tu redundancji czy dwuznaczności. W kolejnych podpunktach omówię najistotniejsze elementy tego API.

PICO-8: PĘTLA GRY

Pętla gry to mechanizm wbudowany w PICO-8, który steruje procesem wyświetlania zawartości ekranu oraz aktualizacji logiki aplikacji. Pętla ta 30 razy na sekundę wywołuje dostarczone przez użytkownika funkcje `zwrotne_update()` oraz `_draw()`, po czym kopiuje zawartość bufora graficznego na wyświetlacz. W funkcji `_update()` umieszczać możemy „popychający” logikę aplikacji, z kolei funkcja `_draw()` renderuje bieżący stan aplikacji, korzystając z dedykowanych ku temu funkcji PICO-8. PICO-8 obsługuje też funkcję `_init()`, która wywoływana jest po uruchomieniu carta – jak

łatwo się domyśleć, warto umieścić tam kod inicjujący aplikację. W Listingu 1 przedstawiony jest krótki program, który wyświetla na środku ekranu skaczącego duszka o indeksie 1. Program ten korzysta z funkcji `_update()`, `_draw()` oraz `_init()`. W ciele funkcji `_init()` definiowane i inicjalizowane są globalne zmienne kontrolujące animację skoku. Wewnątrz funkcji `_update()` umieszczona jest logika animacji, zaś `_draw()` zawiera kod odrysowujący ekran. Na bazie tego prostego szkieletu budowana jest cała aplikacja.

Listing 1. Przykład zastosowania funkcji zwrotnych pętli gry w PICO-8

```
function _init()
    x=60 y=60
    floor=64
    up=5
    g=0.5
end

function _update()
    y=y-up
    up=up-g
    if y>floor then
        up=5
    end
end

function _draw()
    cls()
    spr(1,x,y)
end
```

W tym miejscu warto nadmienić, że PICO-8 pozwala programistie zdefiniować swoją własną pętlę gry, jeśli z jakichś przyczyn tak wbudowana mu nie odpowiada.Więcej na ten temat napiszę w kolejnym podpunkcie.

PICO-8 API: RYSOWANIE

Zestaw funkcji odpowiedzialnych za rysowanie jest niewątpliwie jednym z kluczowych elementów API PICO-8. Zanim przejdę do przeglądu funkcji wchodzących w skład tego zestawu, warto napisać kilka słów na temat tzw. stanu rysowania (ang. *draw state*). Stan ten jest zbiorem zmiennych przechowywanych w pamięci PICO-8, wpływających na zachowanie funkcji rysujących. Zrozumienie roli tych zmiennych jest kluczem do efektywnego korzystania z ww. funkcji, przyjrzymy się im zatem:

- » domyślny kolor: wykorzystywany jest przez funkcje rysujące, które przyjmują kolor jako argument opcjonalny,
- » pozycja kamery: para wartości (x, y) określająca globalne przesunięcie pozycji przekazywanych do funkcji rysujących,
- » pozycja kurSORA: domyślna pozycja przy rysowaniu tekstu za pomocą funkcji `print()` bez określenia argumentów x oraz y,
- » prostokąt obcinający: określa obszar ekranu PICO-8, w którym operacje rysujące są aktywne; rysowanie poza tym obszarem nie przynosi efektu,
- » modyfikacje palet: określają rodzaj filtra, który podmienia wybrane kolory w trakcie rysowania. PICO-8 obsługuje dwa rodzaje modyfikacji (tudzież mapowania) palet: pierwszy z nich stosuje się w trakcie modyfikacji bufora graficznego, drugi – kiedy zawartość bufora graficznego kopiwana jest finalnie na ekran.

PICO-8 oferuje zestaw funkcji służących do manipulacji zmiennych stanu rysowania. I tak za pomocą funkcji `color()` możemy ustawić domyślny kolor. Funkcja `camera()` pozwala kontrolować pozycję kamery, zaś funkcja `cursor()` – pozycję kurSORA. Prostokąt

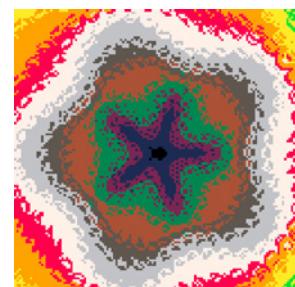
obcinający można ustawić za pomocą funkcji `clip()`, z kolei funkcje `pal()` i `palt()` pozwalają modyfikować mapowanie palet.

Ekran PICO-8 ma rozmiary 128x128 pikseli. Każdy piksel reprezentowany jest jako wartość z przedziału 0-15. Funkcje graficzne PICO-8 operują na obszarze pamięci konsoli zwanej buforem graficznym (lub buforem ekranu). Jeden bajt w tym buforze reprezentuje dwa piksele (przy czym starsze 4 bity w bajcie reprezentują prawy piksel, zaś 4 młodsze bajty – determinują wartość lewego piksela w parze). Cały bufor ekranu zajmuje 8 kilobajtów. Funkcje rysujące (takie jak `rect()` czy `circfill()`) modyfikują zawartość tego bufora, która finalnie kopiwana jest na ekran. Ten model rysowania pozwala przeprowadzać ciekawe efekty związane z modyfikacją palet. Bezpośredni dostęp do bufora ekranu za pomocą funkcji operujących na surowej pamięci również daje spore możliwości w zakresie implementacji niebanalnych efektów graficznych, z czego skwapiwie korzystają użytkownicy PICO-8. Bardzo popularne jest na przykład pisanie krótkich programów, tzw. tweetcartów (nazwa ta wzięła się z tego, że długość programu nie może przekroczyć ilości znaków dozwolonych w tweecie), generujących przeróżne efekty graficzne. PICO-8 posiada wbudowane mechanizmy pozwalające robić zrzuty ekranu i generować krótkie filmiki (w postaci animowanych gifów), dzięki czemu bardzo łatwo można dzielić się efektami swoich dokonań ze społecznością konsolki. Przypomina to bardzo czasy komputerów ośmiobitowych i konkursy polegające na pisaniu programów o określonej liczbie linii (np. 5 lub 10). W dawnych czasach konkursy takie odbywały się na zlotach scenowych lub na łamach czasopism. Programiści prześcigali się w pomysłach, co da się wycisnąć z ich 8-bitowych maszynek. Dziś mogą wrócić do tych czasów dzięki PICO-8. Być może dlatego ta wirtualna konsolka zaskarbiła sobie sympatię wielu byłych programistów demosceny i zyskała sobie nawet status samodzielnej platformy w serwisie pouet.net!

W Listingu 2 pokazany jest kod przykładowego efektu autorstwa Jose Guerra (<https://twitter.com/guerragames>). Na Rysunku 14 zaprezentowano jedną z klatek animacji tego efektu. Warto zwrócić tutaj uwagę na zastosowanie niestandardowej pętli gry, zaimplementowanej przy użyciu instrukcji `goto` oraz wbudowanej funkcji `flip()`, która kopiuje bufor graficzny na ekran, po czym wykonuje synchronizację ramki przy 30 FPS.

Listing 2. Kod źródłowy przykładowego tweetcarta (źródło: <https://twitter.com/guerragames>)

```
t=0:::_:::
t+=.002
for r=0,90,3 do
    for a=0,1,.02 do
        z=r+8*sin(r/64+t*9)*cos(5*a)
        circ(64+z*cos(a+t),64+z*sin(a+t),3,r/8)
    end
end
flip()
goto _
```



Rysunek 14. Przykładowy tweetcart w działaniu

Zestaw funkcji rysujących oferowanych przez PICO-8 dzieli się na kilka grup. Mamy tam grupę funkcji rysujących prymitywy: linię (`line()`), prostokąt (`rect()`), wypełniony prostokąt (`rectfill()`), okrąg (`circ()`) i koło (`circfill()`). Następna grupa funkcji służy do rysowania duszków. Za pomocą funkcji `spr()` można łatwo rysować „regularne” duszki, przekazując jako argument ich indeks. Funkcja pozwala również narysować większy fragment arkusza duszków oraz rysować duszki odwrócone w pionie i w poziomie. Druga funkcja z omawianej grupy: `sspr()` pozwala z kolei narysować dowolny obszar z arkusza duszków z możliwością zastosowania skalowania.

Do rysowania tekstu służy tylko jedna funkcja: `print()`. PICO-8 oferuje też funkcje gwarantujące bezpośredni dostęp do bufora graficznego (`pget()` i `pset()`) oraz do arkusza duszków (`sget()` i `sset()`).

Kolejna grupa funkcji PICO-8 pozwala modyfikować i rysować zawartość mapy. Funkcje te to: `mget()` (pobieranie wartości kafelka), `mset()` (ustawianie wartości kafelka), `map()` (rysowanie określonego zakresu mapy).

W Listingu 3 przedstawiłem kod źródłowy testu API dołączonego do PICO-8 (wraz z odpowiednimi komentarzami); zapraszam do zapoznania się z zawartością tego kodu w celu zaobserwowania praktycznego zastosowania opisanych wyżej funkcji. Efekt działania tego kodu przedstawiony jest na Rysunku 15. Z kolei na Rysunku 16 przedstawiony jest arkusz duszków i mapa dołączone do tego przykładu.

Listing 3. Kod źródłowy demonstracyjnej aplikacji prezentującej możliwości API PICO-8

```
cls() -- czyszc ekran czarnym kolorem
rect(0,0,127,127,1)

-- odgrywaj muzyke od paternu 02
music(0)

-- odgrywaj dźwięk 0 na kanale 3
sfx(0, 3)

-- rysuj paletę PICO-8 wraz
-- z indeksami kolorów
x=3
rectfill(1,1,7,7,5)
for i=0,15 do
    print(i,x,2,i)
    x = x + 6 + flr(i/10)*4
end

-- testuj rysowanie prymitywów graficznych
camera(-10,0) -- przesun kamere o 10 pikseli w prawo
rectfill(10,10,18,20,8)
rect(20,10,28,20,9)
circfill(35,15,5,10)
circ(45,15,5,11)
line(52,10,58,20,12)
pset(60,10,13)
pset(62,10,pget(60,10)+1)
clip(72,10,8,10)
rectfill(0,0,127,127,6)
clip()

-- testuj rysowanie duszków
rectfill(60,24,85,33,15)
spr(1,10,25)
spr(1,20,25,1,1,true)
pal(15,12) -- podmien kolor 15 w palecie na 12
sspr(8,0,8,8, 30,25,24,8)
pal(15,0)
pal(1,8)
sspr(8,0,8,8, 60,25,24,8)
pal() -- zresetuj mapowanie palety

-- testuj rysowanie mapy
mset(3,3,mget(1,3)+2)
map(0,3,10,35,4,3,1)
fset(6,1,true)
-- rysuj tylko kafelki z ustawiona flaga 2
map(0,3,50,35,4,3,2)

-- ustaw pozycje kursora oraz domyslny kolor
```

```
cursor(0,90)
color(7)

-- testuj funkcje matematyczne
cursor(0,70)
x = cos(0.125) + mid(1,2,3)
x = (x%1) + abs(-1)
print("x: "..x)

-- testuj obsługę tablic i napisów
a={}
add(a, 11) add(a, 12)
add(a, 13) add(a, 14)
del(a, 13)
str="a: "
for i in all(a) do
    str=str..i.." "
end
print(str)

-- testuj konstrukcję foreach
-- i funkcje anonimowe
total=0
foreach(a,
    function(i)
        total=total+i
    end
)
print("total: "..total)

-- funkcje zwrotne petli gry
-- pico-8 wywołuje je automatycznie
-- co ramkę (o ile sa zdefiniowane)

t=0
function _update()
    t=t+1
end

function _draw()
    -- pokaz bieżąca wartość zmiennej t
    rectfill(0,90,30,96,5)
    print("t: "..t,1,91,7)

    -- pokaz bieżący stan klawiszy
    print("buttons: ", 1,101,7)

    for p=0,7 do
        for i=0,5 do
            col=5
            if(btn(i,p)) then col=8+i end
            rectfill(40+i*10,100+p*3,48+i*10,101+p*3, col)
        end
    end
end
```



Rysunek 15. Demonstracyjna aplikacja prezentująca możliwości API PICO-8



Rysunek 16. Demonstracyjna aplikacja prezentująca możliwości API PICO-8: arkusz duszków i mapa

PICO-8 API: OBSŁUGA KONTROLERÓW

Jak wcześniej było wspomniane, PICO-8 obsługuje wirtualny kontroler wyposażony w przyciski: *góra, dół, lewo, prawo, kółko oraz krzyżk*. W aplikacji PICO-8 przyciski dla dwóch graczy zamapowane są odpowiednio na klawiaturę (patrz Rysunek 17).



Rysunek 17. Kontroler PICO-8: mapowanie klawiszy (źródło: <https://neko250.github.io/pico8-api/>)

Do obsługi kontrolerów służą dwie funkcje z API PICO-8: `btn()` i `btnp()`. Obydwie z nich zwracają wartość logiczną określającą stan wybranego klawisza (jego indeks wraz z numerem gracza musimy podać jako argument). W wersji bezargumentowej funkcje te zwracają pole bitowe, w którym kolejne bity oznaczają stan poszczególnych klawiszy obydwu kontrolerów. `btnp()` wprowadza dodatkowo specjalne opóźnienia, które ułatwiają obsługę kontrolera w przypadku programowania logiki menu lub przycisków.

W Listingu 4 przedstawiony jest prosty program pozwalający przemieszczać duszka na ekranie za pomocą klawiszy kierunkowych, korzystający z funkcji `btn()`.

Listing 4. Przemieszczanie duszka za pomocą klawiszy kierunkowych

```
function _init()
x=60
y=60
end

function _update()
if (btn(0)) x-=2
if (btn(1)) x+=2
if (btn(2)) y-=2
if (btn(3)) y+=2
end

function _draw()
cls()
spr(1,x,y)
end
```

PICO-8: FUNKCJE OPERUJĄCE NA SUROWEJ PAMIĘCI

PICO-8 posiada 32 kilabajty adresowalnej pamięci, która stanowi serce tego systemu. Poznanie mapy tej pamięci jest kluczowe przy programowaniu konsoli, szczególnie jeśli chodzi o stosowanie bardziej zaawansowanych technik. Jednakże nawet przy tworzeniu prostych programów znajomość układu pamięci PICO-8 bywa przydatna. Układ ten wygląda następująco:

- » bajty 0x0000 – 0x0fff: arkusz duszków (indeksy: 0-127)
- » bajty 0x1000 – 0x1fff: arkusz duszków (indeksy: 128-255) / mapa (wiersze 32-63) (współdzielone)
- » bajty 0x2000 – 0x2fff: mapa (wiersze 0-31)
- » bajty 0x3000 – 0x30ff: flagi duszków
- » bajty 0x3100 – 0x31ff: muzyka
- » bajty 0x3200 – 0x42ff: efekty dźwiękowe
- » bajty 0x4300 – 0x5dff: pamięć ogólnego użytku

- » bajty 0x5e00 – 0x5eff: trwałe dane kartridża (przydatne do zapisywania stanu gry)
- » bajty 0x5f00 – 0x5f3f: stan rysowania
- » bajty 0x5f40 – 0x5f7f: stan sprzętu (Raspberry Pi)
- » bajty 0x5f80 – 0x5fff: piny GPIO (Raspberry Pi)
- » bajty 0x6000 – 0x7fff: bufor ekranu

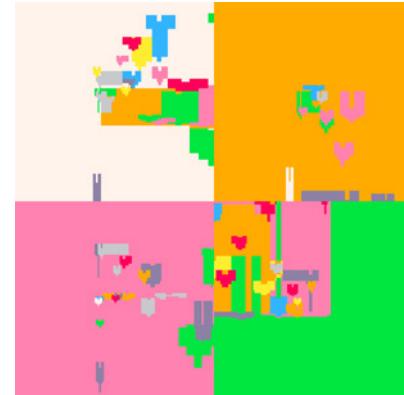
PICO-8 oferuje następujące funkcje do operowania na pamięci:

- » `peek()`: pozwala pobrać wartość bajtu spod wskazanego adresu,
- » `poke()`: pozwala ustawić wartość bajtu pod wskazanym adresem,
- » `memcpy()`: pozwala kopować zadany obszar pamięci,
- » `memset()`: pozwala wypełnić określony obszar pamięci zadaną wartością,
- » `reload()`: pozwala kopować zadany fragment pamięci kartridża do pamięci konsoli.

Operacje na surowej pamięci dają niesamowite możliwości w zakresie tworzenia samo-modyfikujących się programów. W Listingu 5 pokazano krótki fragment kodu, który w locie losowo modyfikuje arkusz duszków i za pomocą funkcji `memcpy()` nadpisuje bufor ekranu. W rezultacie uzyskujemy bardzo dynamiczny, pseudo-loowy efekt wizualny (na Rysunku 18 umieściłem 4 klatki tego efektu, jeśli jednak masz pod ręką PICO-8, to polecam przepisanie tego kodu i obejrzenie efektu „na żywo”).

Listing 5. Kod źródłowy efektu wykorzystującego bezpośredni dostęp do pamięci ekranu

```
u=8192
t=0::1::
print("\135",50+rand(20),52+rand(20),rand(9)+6)
memcpy(0,u*3,u)sspr(6+cos(t)*2,5+sin(t)*2,118,118,0,0,128,128)
flip()t+=0.01
goto 1
```



Rysunek 18. Efekt wykorzystujący bezpośredni dostęp do pamięci ekranu

PICO-8 API: POZOSTAŁE FUNKCJE

PICO-8 oferuje wachlarz przydatnych funkcji matematycznych, dobrany głównie pod kątem przydatności w programowaniu gier. Ciekawe jest, że PICO-8 korzysta ze stałoprzecinkowej reprezentacji liczb rzeczywistych i wszystkie jego funkcje matematyczne operują na liczbach w takim formacie. W API PICO-8 nie może oczywiście zabraknąć funkcji do generowania liczb pseudolosowych (`rnd()` i `srand()`). Mamy też oczywiście funkcje do obsługi dźwięku (`music()` służąca do odgrywania muzyki oraz `sfx()`), dzięki którym możemy poprosić PICO-8 o wydanie określonego efektu dźwiękowego). PICO-8 oferuje też garść funkcji do obsługi tablic, współpraco-

Zintegruj własną aplikację z SMSAPI

Skorzystaj z gotowych bibliotek w językach:



Załącz konto firmowe z kodem polecenia i odbierz pakiet SMS-ów na przetestowanie naszych usług:

FORMULARZ REJESTRACYJNY:

www.smsapi.pl/rejestracja

KOD POLECENIA:

BONUS:

PR56 500 SMS-ÓW

SMSAPI



gramów (ang. *coroutines*) i do zapisywania stanu gry. I w zasadzie to wszystko... API PICO-8 można poznać bardzo szybko, jednakże aby osiągnąć w nim biegłość, potrzeba sporo czasu i praktyki. Pomimo pozornej prostoty, za jego pomocą można wyczarować prawdziwe cuda, co prowadzi nas do kolejnego podpunktu...

PICO-8: PRZYKŁADOWE PROJEKTY

Tutaj w zasadzie mógłbym pisać i pisać i pisać... Aż zabrakłoby papieru :). Aby zademonstrować, co da się uzyskać za pomocą PICO-8, wybrałem osiem przykładowych projektów zrealizowanych za pomocą tego narzędzia. Każdy z tych projektów jest wyjątkowy i w pewnym sensie przełomowy. Przyjrzyjmy się im po kolej.

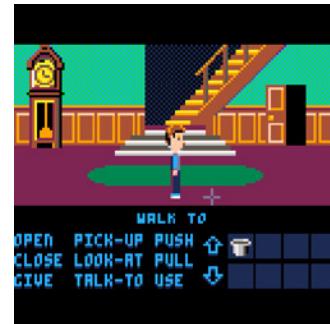
- » Dank Tomb: to w zasadzie nie jest jeszcze finalna gra, a póki co demo techniczne, które pokazuje, jak w PICO-8 zaimplementować grę 2D z dynamicznym oświetleniem. O *Dank Tomb* zrobiło się głośno, kiedy artykuł autora gry, Jakuba Wasilewskiego (twitter.com/krajzeg), opisujący implementację tego efektu trafił na główną stronę serwisu Hackernoon.com. Warto przestudiować ten 4-częściowy techniczny tekst – to, co autorowi udało się wycisnąć z PICO-8, robi wielkie wrażenie (patrz Rysunek 19).
- » SCUMM-8: próba odtworzenia w PICO-8 SCUMM, czyli klasycznego silnika do tworzenia gier przygodowych rodem z LucasArts. Warto nadmienić, że próba całkiem udana, o czym można przekonać się, grając w demo do złudzenia przypominające pewną klasyczną grę przygodową. Warto też poczytać wątek dotyczący tego silnika na forum Lexaloffle: można tam znaleźć szereg cennych wskazówek związanych z optymalizacją kodu aplikacji PICO-8 (patrz Rysunek 20).
- » The Green Legion: na PICO-8 powstało wiele strzelanek, ale ta to absolutne arcydzieło, zarówno pod kątem efektów wizualnych, jak i grywalności (patrz Rysunek 21).
- » Kidd Bludd's Treacherous Tower oraz Advanced Micro Platformer - Starter Kit: rewelacyjna gra platformowa oraz oparta na niej zestaw startowy, który można wykorzystać do tworzenia własnych gier. Jeśli chcesz nauczyć się robić dobre platformówki dla PICO-8, nie możesz przegapić tych dwóch cartów (patrz Rysunki 22 i 23).
- » Witch n'Wiz: świetnie zrealizowana i dopracowana w najdrobniejszych szczegółach gra logiczna bazująca na idei Catrap'a (patrz Rysunek 24).
- » ZEPTON: strzelanka 3D (!) dla PICO-8 oparta na silniku voxelowym, zaimplementowana przez REZ'a: jednego z koderów grupy demoscenowej RAZOR 1911. Ta gra udowadnia, że limity w przypadku PICO-8 nie istnieją (patrz Rysunek 25).
- » Pico Racer: dowód na to, że na PICO-8 da się zrobić rewelacyjną grę wyścigową (patrz Rysunek 26).



Rysunek 19. Demo techniczne gry Dank Tomb



Rysunek 20. Silnik SCUMM-8



Rysunek 21. Gra The Green Legion



Rysunek 22. Gra Kidd Bludd's Treacherous Tower



Rysunek 23. Advanced Micro Platformer - Starter Kit



Rysunek 24. Gra Witch n' Wiz



Rysunek 25. Gra ZEPTON



Rysunek 26. Gra Pico Racer

ZAMIAST PODSUMOWANIA...

Liczę, że lista gier, którą przedstawiłem w poprzednim punkcie, rozbudziła Twój apetyt. Kolejnym naturalnym krokiem jest podjęcie próby implementacji własnej gry na PICO-8. Próbę taką opiszę w drugiej części mojego artykułu (będzie ona dostępna w kolejnym numerze „Programisty”), gdzie krok po kroku opiszę proces stworzenia prostej (aczkolwiek niebanalnej), kompletnej gry logicznej z pewnym gadem w roli głównej. A w międzyczasie zachęcam do eksperymentów z „wyimaginowaną” konsolą PICO-8. Do zobaczenia za miesiąc!



RAFAŁ KOCISZ

rafal.kocisz@gmail.com

Od ponad piętnastu lat pracuje w branży związanej z produkcją oprogramowania. Jego zawodowe zainteresowania skupiają się przede wszystkim na nowoczesnych technologiiach mobilnych oraz na programowaniu gier. Rafał pracuje aktualnie jako Starszy Techniczny Kierownik Projektów w firmie BLStream.

Informacja prasowa

.NET DeveloperDays 2017 – poznaj agendę i dołącz do międzynarodowej społeczności programistów!

Trzy równoległe ścieżki tematyczne, całodzienne prelekcje poświęcone konkretnemu zagadnieniu (pre-cons) oraz impreza integracyjna – to czeka na uczestników kolejnej edycji .NET DeveloperDays, która odbędzie się między 18. a 20. października.

Kto wystąpi podczas konferencji?

Organizatorzy ujawnili właśnie nazwiska wszystkich prelegentów oraz agendę konferencji. Pierwszego dnia konferencji sesja otwarcia zostanie wygłoszona przez Michele Leroux Bustamante (Surviving Microservices), zaś wydarzenie zostanie zamknięte przemówieniem Neala Forda (Stories Every Developer Should Know).

W ciągu dwóch dni wystąpią m.in.:

- » Alex Mang (How Far Can ‘Serverless’ Actually Go?)
- » Alon Fliesen (Past, Present & Future of C# Debugging)
- » Daniel Marbach (Microservices with Service Fabric. Easy... or is it?)
- » Dino Esposito (The Same Software Just the Other Way Around)
- » Dror Helper (Secret unit testing tools no one ever told you about)
- » Eran Stiller (Bot-Tender: A .NET Chatbot Walks into a Bar)
- » Gill Cleeren (Practical Xamarin.Forms)
- » Szymon Kulec (Performance that pays off)

Jest to tylko część z przygotowanych prelekcji. Skróty poszczególnych sesji i bio prelegentów dostępne są na stronie wydarzenia.

Poznaj innych ekspertów!

Pierwszego dnia po zakończeniu wszystkich sesji przewidziana jest impreza integracyjna dla wszystkich uczestników oraz prelegentów. W praktyce przez cały czas trwania wydarzenia uczestnicy

będą mieli okazję na dyskusje oraz wymienianie się wiedzą, jednak plany na imprezę integracyjną pozwalają sądzić, iż to wówczas najłatwiej będzie poznać innych programistów z całego świata.

» Branża IT jest jedną z najsilniej rozwijających się dziedzin, dlatego ogromny nacisk kładziemy na aktualność tematów, które poruszane będą podczas Konferencji – mówią Organizatorzy. Co roku przyjeżdżają do nas specjaliści z różnych krajów. Najwięcej uczestników pochodzi z krajów sąsiadzkich, ale zdarzają się też ludzie z dalej położonych zakątków świata, takich jak Bałkany, Portugalia, a nawet Stany Zjednoczone i Australia! To pozwala nam nie tylko przekazać wiedzę merytoryczną, ale też współtworzyć międzynarodową społeczność specjalistów .NET – dodają.

Nie może Cię tam zabraknąć!

Konferencja oraz prelekcje odbywać się będą w języku angielskim. System rejestracji i wszystkie informacje na temat wydarzenia dostępne są na stronie internetowej:

- » <http://net.developerdays.pl/>

oraz na facebookowym profilu Konferencji:

- » <https://www.facebook.com/DeveloperDays/>.

Organizatorem wydarzenia jest firma DATA MASTER, która organizuje także Join! Database Conference.

Wśród sponsorów znajdują się między innymi: KMD Poland, JCommerce, RavenDb, IT KONTRAKT, DevExpress.

Rule of zero

– pisz więcej, pisząc mniej

Jedną z cech charakterystycznych większości języków programowania jest nadmierna elastyczność w akceptowanych konstrukcjach składniowych. Dzięki temu życie użytkownika takiego języka – programisty – nie jest przesadnie utrudnione. Drugą stroną medalu jest to, że kod nieidiomatyczny, zawierający oczywiste defekty, może być formalnie poprawny.

Aby temu zapobiec, kodyfikowane są różne idiomy i reguły. W artykule opisany jest jeden z nich, w mniemaniu autora jeden z najważniejszych dla programistów C++: *Rule of Zero*. Jest to zaproponowane przez R. Martinho Fernandesa w 2012 roku [1] uzupełnienie zasady *Rule of Three*. Scott Meyers rozszerzył tę regułę w 2014 roku w swoim artykule *A Concern about the Rule of Zero* [2].

RYS HISTORYCZNY

W 1991 r., czyli, z perspektywy C++, u zarania dziejów, Marshall Cline sformułował termin *The Rule of Three* [3]. Jest to dość prosta reguła, która stanowi: *jeśli definiujesz w klasie dowolny z poniższych, musisz zdefiniować pozostałe dwa:*

- » destruktor,
- » konstruktor kopiący,
- » operator przypisania.

Jest to powiązane bezpośrednio z odziedziczoną po języku C zarządzaniem zasobami i domylnym generowaniem tych funkcji dla struktur (w C++ – klas) – domylny konstruktor kopiący lub operator przypisania dokonują kopii każdego elementu klasy. W efekcie dla typów wbudowanych, agregatów i klas z niepoprawną implementacją tych funkcji sprowadza się to do płytkej kopii (ang. *shallow copy*). W przypadku klas zarządzających zasobami płytka kopia elementów jednoznacznie prowadzi do problemów.

Listing 1. Przykładowy kod nie stosujący Rule of Three

```
class my_string
{
public:
    my_string(char const* other):
        ptr{new char[strlen(other)+1]}
    {
        strcpy(ptr, other);
    }

    ~my_string() {
        delete[] ptr;
    }

private:
    char* ptr;
};
```

Zaimplementowana w Listingu 1 klasa `my_string` przedstawia typowy błąd początkującego programisty C++. Przy testach z jednym obiektem wszystko wydaje się działać poprawnie, ale wystarczy tylko wykonać kopię tego obiektu, aby sprokurować niezdefiniowane zachowanie (ang. *undefined behavior*). Przedstawiono to w Listingu 2.

Listing 2. Wykorzystanie błędu w kodzie z Listingu 1 [4]

```
my_string a("42");
my_string b = a;
```

Dzieje się tak, ponieważ domylnie utworzony konstruktor kopiący dokonuje wspomnianej wyżej płytkiej kopii. W tym przypadku oznacza to skopiowanie wartości wskaźnika `ptr` – a nie ciągu znaków, na który wskazuje. W efekcie tego destruktory obu obiektów wywołują `delete[] ptr`, co jest niedozwolone.

Po poprawieniu klasy w taki sposób, aby jej kopianie, bądź przypisywanie, było zgodne z logiką (Listing 3), problem ustępuje. Przy tym przykładzie autor nadmienia, że `new` i `delete` zostały użyte na potrzeby egzemplifikacji problemu, a miejsca ich akceptowalnego użycia zostały przedstawione w jego innych artykułach [5], [6], [7].

Listing 3. Poprawiona definicja klasy `my_string`, spełniająca Rule of Three

```
class my_string
{
public:
    my_string(char const* other):
        ptr{new char[strlen(other)+1]}
    {
        strcpy(ptr, other);
    }

    my_string(my_string const& other):
        ptr{new char[strlen(other.ptr)+1]}
    {
        strcpy(ptr, other.ptr);
    }

    my_string& operator=(my_string const& other) {
        if(this == &other)
            return *this;

        delete[] ptr;
        ptr = new char[strlen(other.ptr)+1];
        strcpy(ptr, other.ptr);
    }

    ~my_string() {
        delete[] ptr;
    }

private:
    char* ptr;
};
```

The Rule of Three to jedna z najbardziej popularnych zasad w C++03; posiada nawet własne hasło w Wikipedii [8]. Poprawnie użyta pozwala uniknąć całej kategorii błędów, a kod do niej się stosujący będzie poprawny nawet dziś, ze względu na bardzo wysoką kompatybilność wsteczną kolejnych standardów C++.

DISCOVER YOUR ERICSSON

Praca w ICT

Jeśli masz głowę pełną pomysłów i jesteś ambitny – w Ericsson stoi przed Tobą perspektywa pracy nad innowacyjnymi technologiami, które zmieniają nasze życie. We współczesnym świecie, technologia napędza biznes i zmiany w społeczeństwie. Pracownicy Ericsson to inicjatorzy zmian, łączący inwencję, nieszablonowe myślenie i wiedzę o nowoczesnych technologiach, by kształtać swoje otoczenie. To oni upowszechniają technologię łączności w przemyśle, społeczeństwie i lokalnych społecznościach.



ROZWÓJ
ZAWODOWY



UMOWA
O PRACĘ



MIĘDZYNARODOWE
ŚRODOWISKO



SZKOLENIA
WEWNĘTRZNE



BOGATY PAKIET
SOCJALNY



KLUBY
ZAINTERESOWAŃ



SOFTWARE

DEVELOPER (C/C++)

SOFTWARE

TESTER/DEVELOPER



WORK-LIFE
BALANCE



PRZYJAZNA
ATMOSFERA

C++11 – TRZĘSIENIE ZIEMI

Wprowadzony w 2011 r. standard zaoferował bardzo dużo zmian. Najbardziej istotną z punktu widzenia tego artykułu jest wprowadzenie semantyki przenoszenia (ang. *move semantics*), a w konsekwencji *rvalue-references* (ang. *rvalue references*). Oznaczane symbolem `&&`¹ symbolizują referencję do wartości *rvalue*, czyli takiej, która może wystąpić tylko po prawej stronie operacji przypisania.

Od tego czasu twórcy klas mogą też zdefiniować konstruktor przenoszący oraz przenoszący operator przypisania, których parametrami są właśnie *rvalue-references*. Często jest to konieczne ze względu na reguły wybierania przeładowań, które musiały być respektowane przez nowy standard (Rysunek 1). W efekcie *Rule of Three* stało się de facto *Rule of Five*. Przykładowy kod z Listingu 3 uzupełniony o nowe funkcje znajduje się w Listingu 4.

Listing 4. Klasa z Listingu 3 uzupełniona o funkcje przyjmujące rvalue-referencje

```
class my_string
{
public:
    my_string(char const* other):
        ptr{new char[strlen(other)+1]}
    {
        strcpy(ptr, other);
    }

    my_string(my_string const& other):
        ptr{new char[strlen(other.ptr)+1]}
    {
        strcpy(ptr, other.ptr);
    }

    my_string(my_string&& other):
        ptr{other.ptr}
    {
        other.ptr = nullptr;
    }

    my_string& operator=(my_string const& other) {
        if(this == &other)
            return *this;

        delete[] ptr;
        ptr = new char[strlen(other.ptr)+1];
        strcpy(ptr, other.ptr);
    }

    my_string& operator=(my_string&& other) {
        if(this == &other)
            return *this;

        ptr = other.ptr;
        other.ptr = nullptr;
    }

    ~my_string() {
        delete[] ptr;
    }

private:
    char* ptr;
};
```

RULE OF ZERO

Nietrudno zauważyc, że kod z Listingu 4 jest pełen powtarzalnych konstrukcji, niejako narzuconych przez język. Ponadto można spostrzec, że tego typu klasa, która sama definiuje swoją politykę zarządzania zasobami, obok funkcji, którą z założenia sprawuje, narusza zasadę jednej odpowiedzialności [B] (ang. *single responsibility principle*).

1. Nie mylić z referencjami uniwersalnymi/forwardującymi (ang. *universal references/forwarding references*), które występują tylko w kontekście dedukcji typów i mogą, ale nie muszą, ostatecznie okazać się *rvalue-references*. Szerzej opisał to Scott Meyers [9].

Special Members						
compiler implicitly declares						
user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Rysunek 1. Informacja, kiedy kompilator generuje specjalne funkcje klas.
Autor: Howard Hinnant [A]

Cytując ten ostatni powód, w szczególności R. Martinho Fernandes zaproponował [1] wydzielenie odpowiedzialności zarządzania zasobami do osobnych klas, jako przykłady podając kontenery standardowe, takie jak `std::vector`, oraz wprowadzone w C++11 inteligentne wskaźniki: `std::unique_ptr` i `std::shared_ptr`.

Dzięki użyciu klas implementujących zgodne z potrzebami zarządzanie zasobami zbędne staje się tworzenie dowolnej z pięciu funkcji objętych regułą *Rule of Five*. Jest tak, ponieważ ich implementacje domyślnie wygenerowane przez kompilator to wywołanie tej samej funkcji na wszystkich elementach klasy.

Rule of Zero stanowi: *nie deklaruj destruktora, ani konstruktorów i operatorów przypisania – ani przenoszących, ani kopiących. Wszystkie powinny zostać automatycznie wygenerowane na podstawie elementów klasy.*

Koncepcyjnie jest to przedstawione w Listingach 5 i 6.

Listing 5. Klasa spełniająca Rule of Zero

```
class zero
{
private:
    foo a;
    bar b;
    baz c;
};
```

Listing 6. Koncepcyjny przykład wygenerowanych przez kompilator implementacji operatorów i konstruktorów z Listingu 5 (pseudokod)

```
class five
{
public:
    five(five const& o) {
        a = o.a;
        b = o.b;
        c = o.c;
    }

    five(five&& o) {
        a = std::move(o.a);
        b = std::move(o.b);
        c = std::move(o.c);
    }

    five& operator=(five const& o) {
        a = o.a;
        b = o.b;
        c = o.c;
        return *this;
    }
}
```

```

five& operator=(five&& o) {
    a = std::move(o.a);
    b = std::move(o.b);
    c = std::move(o.c);
    return *this;
}

private:
    foo a;
    bar b;
    baz c;
};

```

Dla elementów od razu zachowują się zgodnie z oczekiwaniami. Inaczej mówiąc, można pominąć tworzenie wszystkich z nich – stąd nazwa *Rule of Zero*.

Jeśli klasa realizująca politykę zarządzania zasobami zgodną z wymaganiami nie znajduje się w bibliotece standardowej, ani żadnej innej użytej w projekcie, należy taką stworzyć. Modyfikując zgodnie z tą zasadą przykład z Listingu 4, klasa `my_string` może zostać zaimplementowana na kilka sposobów, przedstawionych w Listingach 7, 8, 9 i 10.

Listing 7. Implementacja klasy `my_string` z użyciem `std::vector`

```

class my_string
{
public:
    my_string(char const* other):
        data(other, other + strlen(other) + 1)
    {
    }

private:
    std::vector<char> data;
};

```

Listing 8. Implementacja klasy `my_string` z użyciem `std::string`

```

class my_string
{
public:
    my_string(char const* other):
        data(other)
    {
    }

private:
    std::string data;
};

```

Listing 9. Implementacja klasy `my_string` z użyciem fikcyjnego intelligentnego wskaźnika `clone_ptr`

```

class my_string
{
public:
    my_string(char const* other):
        data{new char[strlen(other)+1]}
    {
        strcpy(data.get(), other);
    }

private:
    clone_ptr<char[]> data;
};

```

Listing 10. Implementacja klasy `my_string` z użyciem `std::unique_ptr`.

```

class my_string
{
public:
    my_string(char const* other):
        data{new char[strlen(other)+1]}
    {
        strcpy(data.get(), other);
    }

private:
    std::unique_ptr<char[]> data;
};

```

Pierwsze, co rzuca się w oczy, to fakt, że cztery implementacje (Listingi 7-10) klasy `my_string` mają razem mniej więcej tyle samo linii kodu co implementacja z Listingu 4. Nietrudno jest też stwierdzić, jak klasa zachowuje się przy operacjach kopii, przenoszenia bądź destrukcji – wystarczy posiadać podstawową znajomość biblioteki standardowej C++.

STD::UNIQUE_PTR – ZASTOSOWANIE I OPTYMALIZACJA WIELKOŚCI

Należy tu zauważyć, że wersja przedstawiona w Listingu 10 (używająca `std::unique_ptr`) ma inną semantykę niż oryginał i pozostała – nie pozwala na wykonywanie kopii. O ile w przypadku klasy reprezentującej stringa nie ma to sensu, to sama idea unikalnego dostępu do zasobu jest jak najbardziej zasadna. Może być przydatna np. do obsługi połączeń sieciowych, uchwytów do plików bądź połączeń z peryferiami.

R. Martinho Fernandes za przykład takiego użycia podaje klasę trzymającą załadowaną dynamicznie bibliotekę w systemie Windows:

Listing 11. Przykład użycia `std::unique_ptr` do zarządzania zasobem: biblioteką dynamiczną. Źródło: [1], zmodyfikowane

```

void free_library(void* h)
{
    ::FreeLibrary(static_cast<HINSTANCE>(h));
}

class module {
public:
    explicit module(std::wstring const& name)
        : handle {
            ::LoadLibrary(name.c_str()),
            &::free_library
        }
    {
    }

private:
    using module_handle =
        std::unique_ptr<void, decltype(&::free_library)>;
    module_handle handle;
};

```

Kod przedstawiony w Listingu 11 jest formalnie poprawny, ale jest nieoptymalny pod względem zajmowanej pamięci: `module_handle` to tak naprawdę `std::unique_ptr<void, BOOL (*) (HMODULE)>`, czyli `unique_ptr` z niestandardowym deleterem będącym wskaźnikiem na funkcję o zadany typie. Jednak sam wskaźnik podawany jest w czasie wykonania programu i teoretycznie może też być różny dla różnych instancji klasy, chociaż w tym przykładzie wyraźnie jest podawany na sztywno. Ponieważ wskaźnik na funkcję nie jest bezstanowy, jego stan – czyli adres konkretnej funkcji – musi być trzymany wewnątrz klasy. Na wszystkich znanych autorowi architekturach powoduje to podwojenie wielkości struktury. Kod przedstawiony w Listingu 12 komplituje się na linuksach na x86, x86-64 i armv7, co oznacza, że warunek `static_assert` został spełniony.

Listing 12. Dodanie stanowego deletera powoduje podwojenie wielkości `std::unique_ptr` [C]

```

using T1 = std::unique_ptr<int>;
using T2 = std::unique_ptr<int, void(*)(int*)>;
static_assert(sizeof(T1) * 2 == sizeof(T2), "");

```

Można tego uniknąć, stosując deleter bezstanowy – taki właśnie jest stosowany jako domyślny argument szablonu `std::unique_ptr`. Dzięki wykorzystaniu *empty base optimization* [D] bezstanowy deleter nie zwiększa ostatecznego rozmiaru `std::unique_ptr`. Przykładowy bezstanowy deleter dla klasy z Listingu 11 przedstawiony jest w Listingu 13.

Listing 13. Bezstanowy deleter zastosowany w przykładzie z Listingu 11

```
struct free_library
{
    void operator()(void* h) const
    {
        ::FreeLibrary(static_cast<HINSTANCE>(h));
    }
};

class module {
public:
    explicit module(std::wstring const& name)
        : handle { ::LoadLibrary(name.c_str()) }
    {
    }

private:
    using module_handle =
        std::unique_ptr<void, free_library>;
    module_handle handle;
};
```

Ponieważ wywoływana funkcja zakodowana jest w typie, który jest wykorzystywany jako deleter, nie musi on trzymać żadnego stanu, co w konsekwencji pozostawia identyczną wielkość inteligentnego wskaźnika. Ponadto nie trzeba podawać adresu funkcji do konstruktora `handle`.

Niestety, tworzenie nowego typu dla każdej użytej klasy jest nadmierną komplikacją. Na szczęście można jej się pozbyć, stosując szablony. Przykładowy bezstanowy szablonowy deleter przedstawiony jest w Listingu 14, a jego użycie w Listingu 15.

Listing 14. Bezstanowy deleter – szablon

```
template<typename T, T* func>
struct function_caller
{
    template<typename... Us>
    void operator()(Us&&... us) const {
        func(std::forward<Us>(us)...);
    }
};
```

Listing 15. Wykorzystanie szablonowego deletera z Listingu 14

```
void free_library(void* h)
{
    ::FreeLibrary(static_cast<HINSTANCE>(h));
}

class module {
public:
    explicit module(std::wstring const& name)
        : handle { ::LoadLibrary(name.c_str()) }
    {
    }

private:
    using module_handle = std::unique_ptr<
        void,
        function_caller<
            free_library
        >
    >;
    module_handle handle;
};
```

W C++17 kod ten można jeszcze bardziej uprościć, wykorzystując dedukcję typu argumentu szablonu [E]. Przykładowy kod przedstawiony został w Listingach 16 (szablon) i 17 (użycie).

Listing 16. Szablon wykorzystujący dedukcję parametru szablonu klasy nie będącego typem

```
template<auto func>
struct function_caller
{
    template<typename... Us>
    void operator()(Us&&... us) const {
        func(std::forward<Us>(us)...);
    }
};
```

Listing 17. Wykorzystanie szablonu z Listingu 16. Kompilator online: [F]

```
void free_library(void* h)
{
    ::FreeLibrary(static_cast<HINSTANCE>(h));
}

class module {
public:
    explicit module(std::wstring const& name)
        : handle { ::LoadLibrary(name.c_str()) }
    {
    }

private:
    using module_handle = std::unique_ptr<
        void,
        function_caller<
            free_library
        >
    >;
    module_handle handle;
};
```

PUŁAPKI

Chciałoby się w tym momencie przejść do podsumowania i szczerze polecić bezwarunkowe stosowanie *Rule of Zero*. Niestety, Scott Meyers w artykule *A concern about the Rule of Zero* [2] przedstawił pewne rozszerzenie tej reguły, niezbędne do zachowania pełnej wydajności.

W celu zachowania zgodności wykonania kodu napisanego w starszych standardach C++ reguły automatycznego generowania funkcji specjalnych klasy nie są w pełni intuicyjne. Jak widać na Rysunku 1, jeśli użytkownik zadeklaruje destruktor, to kompilator wygeneruje wyłącznie konstruktor kopiący i kopiący operator przypisania. Wersje przenoszące (z *rvalue-referencjami*) nie zostaną wygenerowane.

Oznacza to, że dodanie destruktorów, np. logującego destrukcji, może znaczco wpływać na wydajność kodu. W Listingu 18 przedstawiono klasę stosującą *Rule of Zero*. Odkomentowanie destruktora `rule_of_zero` spowoduje wypisanie `copy` zamiast wcześniejszego `move`.

Listing 18. Przedstawienie pułapki w naiwnym użyciu Rule of Zero [10]

```
struct foo
{
    foo(){}
    foo(foo const&) { cout << "copy\n"; }
    foo(foo&&) { cout << "move\n"; }
};

struct rule_of_zero
{
```

```
// ~rule_of_zero() { cout << "bye\n"; }
foo f;
};

int main()
{
    rule_of_zero a;
    rule_of_zero b = std::move(a);
}
```

Podobnie problematyczne może się okazać zadeklarowanie pozostały funkcji specjalnych (ponownie: patrz Rysunek 1). Na szczęście w tym samym artykule Scott Meyers proponuje nową regułę, która ma temu zaradzić: *Rule of Five Defaults*. Programista może jawnie zażądać od kompilatora wygenerowania domyślnych implementacji funkcji specjalnych. *Rule of Five Defaults* stanowi: zawsze *żądaj wygenerowania wszystkich funkcji specjalnych, których klasa będzie używać*. Przedstawiono to na przykładzie w Listingu 19.

Listing 19. Rule of Five Defaults w praktyce, z użyciem przykładu z Listingu 17 [11]

```
struct foo
{
    foo(){}
    foo(foo const&) { cout << "copy\n"; }
    foo(foo&&) { cout << "move\n"; }
};

struct rule_of_zero
{
    rule_of_zero() = default;
    ~rule_of_zero(){ cout << "bye\n"; }

    rule_of_zero(rule_of_zero const&) = default;
    rule_of_zero(rule_of_zero&&) = default;
    rule_of_zero& operator=(rule_of_zero const&) = default;
    rule_of_zero& operator=(rule_of_zero&&) = default;

    foo f;
};
```

```
int main()
{
    rule_of_zero a;
    rule_of_zero b = std::move(a);
}
```

Warto zauważyć, że poza dopisaniem czterech nowych domyślnych funkcji specjalnych konieczne było również dodanie konstruktora domyślnego, nieobjętego tą regułą.

PODSUMOWANIE

Rule of Zero jest stosunkowo młodą regułą w C++ i chociaż jej powstanie to częściowo skutek zmian w języku wraz z wprowadzeniem C++11, to można ją stosować także w starszych wersjach języka. Pozwala ona na tworzenie czytelnego i zwięzłego kodu, w niektórych przypadkach wręcz deklaratywnego.

W opinii autora, pomijając specjalne przypadki, jak np. klasy implementujące zarządzanie zasobami, *Rule of Zero* powinna być stosowana jako podstawa w tworzonym kodzie. Funkcje specjalne powinny być pozostawione do wygenerowania kompilatorowi na podstawie zadeklarowanych elementów klasy.

Jeśli jednak – z jakiegokolwiek powodu – konieczna jest jawnia deklaracja jednej z funkcji specjalnych, zadeklarowane powinny zostać wszystkie – zgodnie z *Rule of Five Defaults*. Pozwoli to stosunkowo niewielkim narzutem uzyskać pełną wydajność oraz odpowiednio elastyczne zachowanie.

Efektem programowania z uwzględnieniem tych reguł będzie kod bezpieczniejszy, bardziej czytelny, łatwiejszy w utrzymaniu i nie mniej wydajny, niż implementujący jawnie wszystkie funkcje specjalne wymienione przez *Rule of Five*.

Bibliografia:

- [1]: <https://rmf.io/cxx11/rule-of-zero>
- [2]: <http://scottmeyers.blogspot.com/2014/03/a-concern-about-rule-of-zero.html>
- [3]: <http://www.drdobbs.com/c-made-easier-the-rule-of-three/184401400#1>
- [4]: <https://wandbox.org/permlink/v6MQx9Z9iiruXU0L>
- [5]: Paweł "KrzaQ" Zakrzewski, *Czytelny i wydajny – C++ XXI wieku, „Programista”*, 12, 2016 s. 22-28
- [6]: Paweł "KrzaQ" Zakrzewski, *Piękny kod, „Programista”*, 3, 2017 s. 10-14
- [7]: <https://dsp.krzaq.cc/post/176/ucze-sie-cxx-kiedy-uzywac-new-i-delete/>
- [8]: [https://en.wikipedia.org/wiki/Rule_of_three_\(C%2B%2B_programming\)](https://en.wikipedia.org/wiki/Rule_of_three_(C%2B%2B_programming))
- [9]: <https://isocpp.org/blog/2012/11/universal-references-in-c11-scott-meyers>
- [A]: <https://youtu.be/vLinb2fgkHk?t=1420>
- [B]: https://en.wikipedia.org/wiki/Single_responsibility_principle
- [C]: <https://wandbox.org/permlink/WcB0AZa2LhNbIPCy>
- [D]: <http://en.cppreference.com/w/cpp/language/ebo>
- [E]: <https://goo.gl/Mw4Cgj>
- [F]: <https://wandbox.org/permalink/sGPuZWSl3RuKDhR>
- [10]: <https://wandbox.org/permalink/tBoMzPZO3aALKZYx>
- [11]: <https://wandbox.org/permalink/z38uwJtcKEtLIWnv>

PAWEŁ "KRZAQ" ZAKRZEWSKI

<https://dev.krzaq.cc>

Absolwent Automatyki i Robotyki na Zachodniopomorskim Uniwersytecie Technologicznym. Pracuje jako programista w firmie Logzact S.A. Programowaniem interesuje się od dzieciństwa, jego ostatnie zainteresowania to C++ i metaprogramowanie.

Zmień swoje życie i zostań programistą w 6 tygodni!

Wszyscy koderzy stali kiedyś tam, gdzie Ty teraz – na linii startu... Nie ma dróg na skróty, ale nauczenie się programowania jest zupełnie realne. Sprawdź, jak kurs Reaktor PWN może pomóc Ci mądrze rozpocząć karierę w najszybciej rozwijającej się branży na świecie!

DLACZEGO WARTO NAUCZYĆ SIĘ KODOWANIA?

Czy wiesz, że według raportów z rynku pracy programista to obecnie jeden z **najbardziej poszukiwanych zawodów** w Polsce (i na świecie)? W raporcie płacowym firmy rekrutacyjnej HAYS zarobki programisty wynoszą znacznie powyżej 6000 zł brutto. To jeden z argumentów, dla których wiele osób myśli o przeniesieniu swojej ścieżki kariery do branży IT.

Jednocześnie w Polsce, według OECD, 0,2% osób jest programistami – to niewiele, dlatego firmy konkurują między sobą i oferują coraz lepsze warunki pracy. Ten rynek jest jeszcze nienasycony, więc to doskonały czas na przekwalifikowanie się.

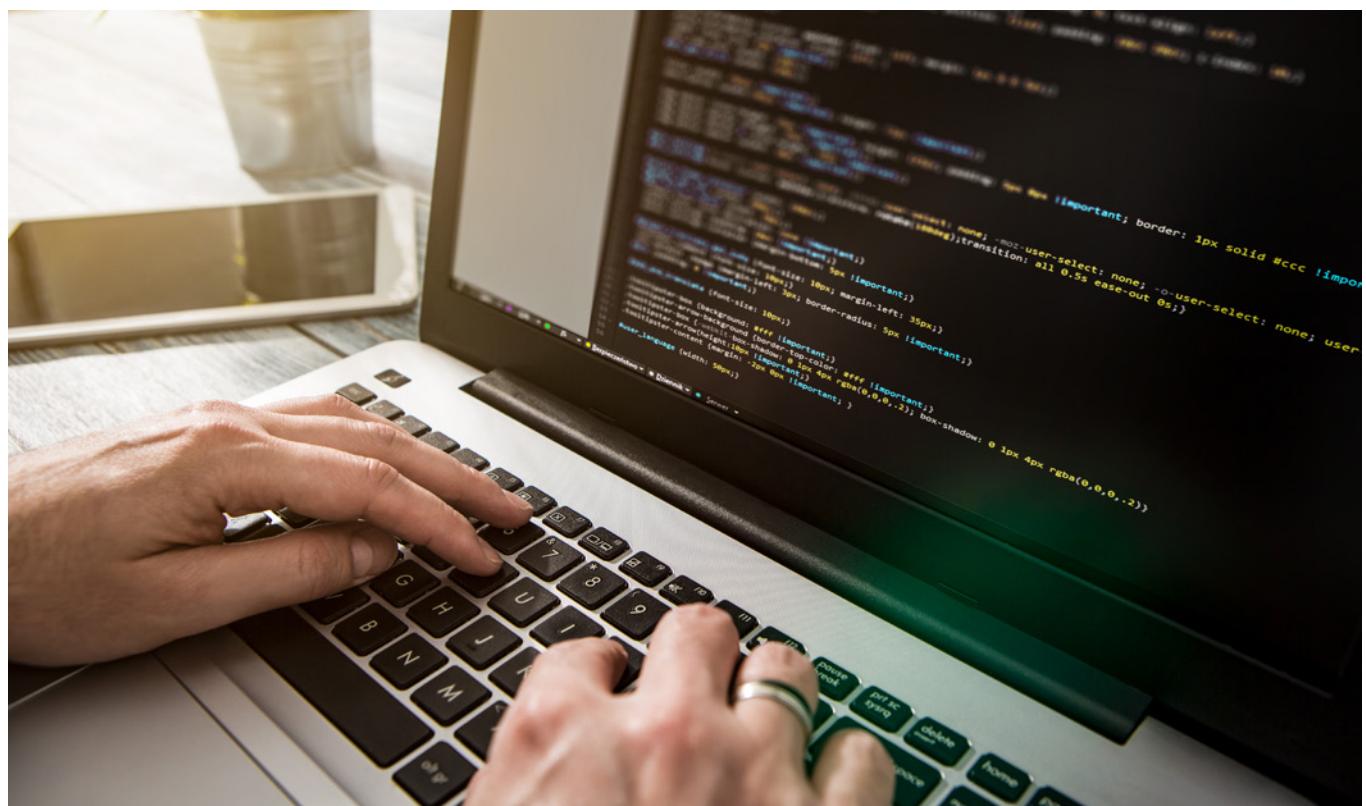
JAK ROZPOCZĄĆ NAUKĘ?

Kodowania może nauczyć się każdy, ale nie jest to zawód dla wszystkich. Jeśli oczekujesz pracy od 8 -16 i nie lubisz długo siedzieć przed komputerem, odpuść. Szybko się spalisz, a praca nie będzie dawała Ci satysfakcji. Jeżeli jednak lubisz zagadki logicz-

ne i masz zdolność analitycznego myślenia, to ten zawód jest dla Ciebie. Najszybszym sposobem na naukę jest boot camp programistyczny.

CO TO TAKIEGO BOOT CAMP?

Boot camp (z języka angielskiego obóz szkoleniowy dla rekrutów) to metodologia oparta o intensywną naukę przez praktykę, dzięki której w krótkim czasie przygotujesz się do pracy w określonym zawodzie. Boot camp to świetna alternatywa dla studiów podyplomowych, bo nauka trwa 6 tygodni, a nie rok... Dodatkowo trenerzy skupią się na pracy praktycznej. Takie ćwiczenia pozwolą Ci stworzyć swoje portfolio już na starcie kariery. Jeśli potrzebujesz jeszcze więcej argumentów, żeby podjąć wyzwanie bootcampowe, obejrzyj animację pokazującą, na czym polegają takie kursy. Znajdziesz ją na stronie szkoły Reaktor PWN (<https://reaktor.pwn.pl/praca-it-dla-kobiet/>) i na kanale youtube: <https://www.youtube.com/watch?v=bq9fmwl2Sxc>. Jeśli zdecydujesz się na naukę metodą boot camp, rozejrzyj się i wybierz odpowiedni kurs. Przeglądając oferty, sprawdź koniecznie, ile godzin trwa dany kurs.





Minimum to 150 h kursu, ale optymalna liczba to aż 240 h nauki pod okiem wyspecjalizowanych trenerów. Bożena Jamka-Czyż, manager projektu Reaktor (<https://reaktor.pwn.pl/praca-it-dla-kobiet/reaktor.pwn.pl>) w szkole PWN, jest przekonana, że taki kurs to dobry wybór, nawet dla osób, które nigdy nie miały styczności z programowaniem. – *Mamy przygotowane autorskie programy kursów i korzystając z wieloletniego doświadczenia w branży edukacji, wiemy, jak nauczać programowania. Współpracujemy również z pracodawcami i firmami rekrutacyjnymi, z którymi podczas zakończenia pierwszych kursów Reaktor spotkali się nasi świeżo upieczeni absolwenci. To buduje i sprawia, że wiemy, że nasze kursy są potrzebne zarówno pracodawcom, jak i potencjalnym pracownikom* – dodaje.

DLACZEGO KURS REAKTOR PWN?

- » Reaktor jest częścią Grupy PWN, która od początku swojego istnienia jest źródłem **wiarygodnej, rzetelnej wiedzy**.
- » Mamy przygotowane **autorskie programy kursów** i korzystając z wieloletniego doświadczenia PWN, wiemy, jak skutecznie uczyć.
- » Podczas kursu będziesz programować, tworzyć aplikacje i strony internetowe, dzięki czemu stworzysz swoje **portfolio** niezbędne przy szukaniu pracy.
- » Nasze kursy trwają 240 godzin, dzięki czemu nie musisz opisywać połowy materiału samodzielnie.

- » Nasi trenerzy to nie tylko specjaliści IT, ale także osoby o szerokiej **sieci kontaktów**, dzięki czemu są w stanie doradzić Ci, jaki kierunek rozwoju będzie dla Ciebie najlepszy. Dodatkowo masz gwarancję, że wykładowca na kursie nie będzie się zmieniać co tydzień, co jest niestety częstą praktyką na kursach programowania.
- » Decydując się na kurs w Reaktor PWN, otrzymujesz **GRATIS** półroczny dostęp do biblioteki e-booków na temat programowania, wydanych przez PWN.

CO PO KURSIE?

Po każdym kursie typu boot camp otrzymujesz Certyfikat ukończenia, ale to nie wszystko. W niektórych szkołach, jak np. w Reaktor PWN (<https://reaktor.pwn.pl/praca-it-dla-kobiet/>), nie zostawiają Cię samego, będziesz mieć wsparcie partnerów rekrutacyjnych projektów, a jeśli będziesz jednym z dwóch najlepszych kursantów, to organizatorzy zapewnią Ci staż. Na co więc czekać? Jeśli masz pytania, napisz: reaktor@pwn.pl.



Budujemy robota

Sterowanie serwami z poziomu Raspberry Pi

Czy mieliście kiedyś ochotę zbudować robota? Ja tak – i to chyba od samego początku mojej przygody z programowaniem. Jednak cały czas powstrzymywało mnie przekonanie, że zrealizowanie takiego projektu wymaga solidnie ugruntowanej wiedzy z zakresu fizyki i elektroniki, biegłości w pisaniu w asemblerze oraz odpowiednio zasobnego portfela na wszystkie potrzebne części. Dopiero kiedy przełamałem się i zacząłem nieśmiało szukać informacji na ten temat w Internecie, okazało się, że sprawa wygląda znacznie bardziej optymistycznie niż mi się początkowo wydawało...

Odpowiedzialność

Pamiętajmy o tym, że prąd elektryczny – choć daje bardzo dużo możliwości – jest też bardzo niebezpieczny. Postarałem się ująć w początkowej części artykułu wszystkie ważniejsze aspekty pracy z elektroniką, a ponieważ komponenty elektroniczne pracują na niskich napięciach i płynie w nich stosunkowo niski prąd, ewentualne pomyłki skończą się prawdopodobnie tylko uszkodzeniem danego układu. Artykuł ten nie jest jednak kompletnym kompendium pracy z urządzeniami elektrycznymi, dlatego autor tego artykułu nie ponosi odpowiedzialności za ewentualne szkody powstałe w wyniku opisywanych działań.

PRĄD ELEKTRYCZNY – PODSTAWY FIZYKI

Na samym wstępnie muszę z niejakim zakłopotaniem przyznać, że choć programowaniem interesuję się bodaj od 8. roku życia (pierwsze programy pisałem w BASICu na Atari 800 XL), to z fizyką bardzo długo byłem na bakier. Po skończeniu liceum odetchnałem z ulgą, nieświadom kompletnie, że za jakiś czas wróć do starannie unikanych tematów całkowicie z własnej woli.

Kiedy jednak zacząłem zgłębiać logikę rządzącą przepływem prądu elektrycznego, okazało się, że to wszystko nie jest wcale aż tak straszne i niezrozumiałe. Ponieważ budowa robota wymaga choć podstawowej wiedzy na ten temat, postaram się w dużym skrócie przybliżyć te zagadnienia, które będą nam potrzebne.

Rura z wodą

Przepływ prądu elektrycznego polega na przemieszczaniu się w przewodniku częstek obdarzonych ładunkiem elektrycznym; w przypadku metali częstek tymi są elektryny. Bardzo często spotykałem się z analogią – porównaniem prądu płynącego w przewodniku do wody płynącej w rurze. Posłużę się nią również i w tym artykule, ale z zastrzeżeniem, iż ma ona zastosowanie tylko do pewnej części zagadnień związanych z prądem.

Przepływ wody w rurze charakteryzuje się trzema ważnymi czynnikami:

- » Ciśnieniem, które wywołane w taki czy inny sposób powoduje ruch wody;
- » Przekrojem rury, który reguluje maksymalną ilość wody możliwą do przetransportowania w czasie;
- » Kubaturą transportowanej wody.

Zwróćmy uwagę, że wszystkie trzy czynniki są ze sobą ściśle powiązane. Zmniejszenie przekroju rury przy stałym ciśnieniu skutkuje przyspieszeniem jej przepływu. Przy stałym przekroju zmniejszenie ciśnienia spowoduje, że w tym samym czasie przepłynie mniej wody – i tak dalej. Okazuje się, że bardzo podobnie jest w przypadku prądu elektrycznego, a odpowiadającymi czynnikami są tutaj:

- » Napięcie elektryczne, które jest odpowiednikiem ciśnienia, a jego zaistnienie powoduje przepływ prądu. Aby napięcie mogło się pojawić, musi wystąpić różnica potencjałów, czyli nadmiar elektronów po jednej stronie przewodnika i niedobór po drugiej. Dodatkowym, ważnym, warunkiem jest też zamknięcie obwodu: prąd płynie bowiem tylko w obwodzie zamkniętym (to jest bardzo ważny fakt, o czym dowiemy się za chwilę).
- » Oporność (rezystancja) przewodnika, która jest odpowiednikiem przekroju rury. Im większy jest opór, tym (przy stałym napięciu) będzie płynęło mniej prądu.
- » Wreszcie natężenie prądu, które jest odpowiednikiem kubatury płynącej wody. Natężenie obliczane jest jako ilość przemieszczonych przez przewodnik ładunków w czasie.

Podobnie jak w przypadku rury z wodą, wszystkie trzy czynniki są ze sobą powiązane i wyrażone w postaci proporcji zwanej prawem Ohma:

$$I = \frac{U}{R}$$

gdzie:

- » U – napięcie elektryczne, jego jednostką jest wolt (V),
- » R – opór (inaczej rezystancja), jego jednostką jest ohm (Ω),
- » I – natężenie, jego jednostką jest amper (A).

Zauważmy na przykład: podwojenie napięcia przy stałym oporze spowoduje w efekcie podwojenie natężenia prądu. Jeżeli z kolei przy stałym napięciu podwoimy opór, uzyskamy dwukrotny spadek natężenia prądu.

Konsekwencje prawa Ohma

Prawo Ohma pozwala nam ustrzec się przed wieloma błędami kończącymi się zwykle katastrofalnie. Wróćmy bowiem do naszej rury z wodą: każda rura ma swoją wytrzymałość, która pozwala

bezpiecznie transportować wodę tylko o określonym maksymalnym ciśnieniu. Jeżeli zwiększymy ciśnienie ponad możliwości rury, woda rozsadzi ją.

W przypadku prądu jest tak samo – efektem ubocznym jego przepływu jest bowiem wydzielające się ciepło. Jeżeli przez dany przewodnik (lub element elektroniczny) przepuścimy zbyt dużo prądu, mocno się nagrzeje – aż do momentu, w którym uszkodzi się, a w skrajnym przypadku nawet zapali albo stopi. Aby tego uniknąć, trzeba starannie monitorować napięcie na wejściu każdego zasilanego komponentu. Na szczęście każdy układ elektroniczny ma zwykle podane maksymalne lub nominalne napięcie, które można do niego bezpiecznie podłączyć.

Wydajność prądowa

Drugim kluczowym aspektem związanym z przepływem prądu jest wydajność prądowa. Każde źródło zasilania dostarcza napięcia o konkretnej wartości (umożliwiającego przepływ prądu), ale ma również górne ograniczenie dotyczące ilości prądu (natężenia), którego jest w stanie dostarczyć.

Przypuśćmy na przykład, że mamy źródło zasilania o napięciu 5V będące w stanie naraz oddać prąd o natężeniu 1A. Co stanie się, gdy spróbujemy pobrać z niego więcej prądu?

Istnieje kilka możliwości, w zależności od tego, o jakim źródle zasilania mówimy. Jeżeli są to zwykłe baterie, nie będą one w stanie dostarczyć wymaganego prądu, co w efekcie spowoduje spadek napięcia. Spadek taki jest niepożądany, ponieważ może zakłócić działanie urządzeń elektronicznych. Sam doświadczylem tego przy budowie mojego robota – silniki, które natrafiliły na duży mechaniczny opór, spowodowały krótki spadek napięcia, w efekcie którego płytka Arduino kompletnie „zgłębiała” – zaczęła przesyłać losowe wartości na losowe piny wyjściowe, a sytuacja wróciła do normy dopiero po jej restarcie.

Podobnie zachowa się zasilacz sieciowy, który na wejściu zasilany jest napięciem przemiennym 230V, a na wyjściu dostarcza stałego napięcia (np. 5V). Większość takich zasilaczy wyposażona jest w zabezpieczenie nadprądowe, które obniży lub nawet odetnie napięcie, jeżeli spróbujemy pobrać z zasilacza więcej prądu niż pozwala na to jego konstrukcja. Jeżeli zasilacz takiego zabezpieczenia nie posiada, możemy go zniszczyć.

Druga możliwość jest bardziej niebezpieczna. Przykładem mogą być akumulatory litowo-polimerowe (LiPo), które z jednej strony są w stanie oddawać znacznie więcej prądu niż akumulatory innej konstrukcji, ale z drugiej – są dosyć niebezpieczne w eksploatacji. Próba pobrania z nich więcej prądu niż pozwala na to ich konstrukcja spowoduje, że zacząć się gwałtownie grzać – do tego stopnia, że w skrajnym przypadku mogą nawet wybuchnąć.

Zwarcia

Rozważmy jeszcze jeden przypadek – zwarcie. Występuje ono wówczas, gdy połączymy bezpośrednio przewodem dwa biegunki źródła zasilania. Zwykły przewód ma radykalnie mniejszy opór niż jakiekolwiek komponenty elektroniczne, więc – zgodnie z prawem Ohma – spowoduje przepływ bardzo wysokiego prądu. Jak już wiemy, powoduje to wydzielanie dużych ilości ciepła, co w efekcie może doprowadzić do zapłonu; co więcej, przepływ dużej ilości prądu przez akumulator może również trwale go uszkodzić. Akumulatory litowo-polimerowe zareagują na zwarcie praktycznie w taki sam sposób, jak na próbę pobrania z nich zbyt dużego prądu.

Ogólne zasady

Zbierzmy więc podstawowe zasady, których należy się trzymać podczas konstruowania robota:

- » Starannie dbamy o to, żeby napięcie przykładane do danego komponentu nie przekraczało jego napięcia znamionowego (zalecanego). Napięcie możemy bezpiecznie zmierzyć przy pomocy prostego miernika, który tanio kupimy np. w marketie budowlanym. Jeżeli źródło zasilania dostarcza większego napięcia, korzystamy z układu obniżającego je (BEC).
- » Podczas projektowania modelu sumujemy maksymalne prądy wszystkich komponentów i sprawdzamy, czy źródło zasilania będzie w stanie taki prąd dostarczyć. Jeżeli tak nie jest, musimy wybrać inne komponenty albo zmienić źródło zasilania na bardziej wydajne.
- » Starannie izolujemy wszystkie miejsca, w których przewody są na wierzchu. Łączenia (lutowania) przewodów izolujemy najprościej, korzystając z koszulek termokurczliwych – po założeniu na miejsce lutowania podgrzewamy je za pomocą zapalniczki, aby skurczyły się i ścisnęły przewody, zabezpieczając przed przypadkowym zwarciem. Uważamy też na to, żeby nie dotknąć niczym metalowym do układów scalonych (Arduino, Raspberry i innych), bo w ten sposób też możemy bardzo łatwo spowodować zwarcie. Najbezpieczniej jest umieścić je w dedykowanych, najlepiej zamkniętych obudowach.
- » Bardzo ostrożnie obchodzimy się z układami podłączonymi do napięcia. Pechowe dotknięcie śrubokrętem takiego układu może w najgorszym przypadku nieodwracalnie go uszkodzić. Dobrze też wyposażyć robota w mechaniczny wyłącznik, który pozwoli w razie potrzeby szybko odciąć zasilanie.

Potrzebujemy planu

Bogatsi o wiedzę dotyczącą reguł rządzących przepływem prądu możemy zacząć powoli myśleć o projekcie robota. Aby go zbudować, będziemy potrzebować kilku kluczowych komponentów:

- » Jednostki centralnej, którą będziemy mogli oprogramować, a która będzie sterowała jego pracą;
- » Silników elektrycznych, które umożliwią robotowi ruch;
- » Czujników, za pomocą których będziemy mogli obserwować otaczający robota świat;
- » Źródła zasilania;
- » Wreszcie ramy, na której całość zamocujemy.

Kiedy zdecydujemy się na konkretne części, musimy połączyć wszystko w taki sposób, by komponenty ze sobą prawidłowo współpracowały (i żeby niczego przy okazji nie spalić). Na koniec pozostań nam oprogramowanie robota i przetestowanie jego działania. Postaram się teraz opisać, jaki mamy wybór w zakresie komponentów, z których będziemy budować naszego robota.

JEDNOSTKA CENTRALNA

W zakresie jednostki centralnej wybór mamy bardzo szeroki, dlatego ustalmy najpierw kryteria:

- » Jednostka musi być relatywnie nieduża, aby można było ją łatwo zamocować wewnętrz robotu. Na dobrą sprawę jednostką taką mógłby być nawet komputer PC lub notebook, ale taki robot nie byłby zbyt mobilny.
- » Jednostka powinna od razu wspierać podłączanie zewnętrznych urządzeń: silników i czujników.

- » Z uwagi na fakt, iż dopiero zaczynamy przygodę z budowaniem robotów, dobrze byłoby, gdyby jednostkę taką dało się zaprogramować za pomocą (relatywnie) wysokopoziomowego języka.

Powyższe kryteria spełnia dosyć dużo gotowych rozwiązań, z których przyjrzymy się dwóm bodaj najbardziej popularnym. Pierwszym z nich jest Raspberry Pi – pełnoprawny komputer w architekturze ARM, który ma rozmiar pudełka papierów. Można zainstalować na nim dedykowanego Linuksa, podłączyć klawiaturę, mysz i ekran i obsługiwać jak zwykły komputer. Co odróżnia go jednak od peceta czy notebooka, to szyna GPIO (General Purpose Input-Output), do której możemy podłączać różne urządzenia, a następnie nimi sterować. Odpowiednie biblioteki można łatwo ściągnąć przy użyciu menedżera pakietów apt-get, a programować możemy w praktycznie dowolnym języku; dość powiedzieć, że bardzo mocne wsparcie w postaci wielu gotowych bibliotek ma Python (zainstalowany domyślnie w dedykowanym systemie operacyjnym Raspbian).

Raspberry Pi w wersji trzeciej jest wyposażony w bardzo przyzwyczajone parametry (jak na komputer tak niewielkich rozmiarów): 64-bitowy, czterordzeniowy procesor o taktowaniu 1,2 Ghz, 1 GB pamięci RAM, wbudowany moduł Wifi i Bluetooth Low Energy 4.1. Do tego szyna GPIO z 40 pinami.

Drugim produktem, na który definitywnie warto zwrócić uwagę, jest platforma Arduino. Tym razem schodzimy o poziom niżej – mowa bowiem o układzie scalonym z procesorem i zestawem pinów, który programujemy bezpośrednio. Oznacza to, że na układzie tym nie pracuje żaden system operacyjny – albo inaczej: program, który napiszemy, w pewnym sensie pełni rolę systemu operacyjnego.

Może brzmieć to na początku strasznie, ale wszystko staje się o wiele bardziej przystępne, gdy okaże się, że do Arduino istnieje dedykowane środowisko programistyczne, nasze programy możemy wgrywać na płytę przez zwykły kabel MicroUSB, a pisać je możemy w stosunkowo prostym języku zbliżonym swoją konstrukcją do C lub C++. Przykładowy program dla zdalnie sterowanego robota jeżdżącego na czterokołowej platformie zajmuje około 160 linii kodu. Co więcej, dzięki temu, że platforma jest popularna, w Internecie można znaleźć mnóstwo bibliotek, które znacznie upraszczają pisanie programów.

Seria Arduino zawiera wiele różnych płyt, które różnią się nieco możliwościami – liczbą pinów, do których możemy podłączać urządzenia, oraz wbudowanymi periferiami: niektóre z nich mają na przykład wbudowane czytniki kart microSD, moduły Wifi czy Bluetooth.

Przykładowa, prosta płytko – Arduino Leonardo – ma na pokładzie mikrokontroler Atmega32u4 taktowany zegarem 16 Mhz, 32 kB pamięci programowej oraz 2,5 kB pamięci operacyjnej. Choć wartości te mocno odbiegają od standardów współczesnych komputerów osobistych, pamiętajmy, że na płytce takiej pracuje tylko jeden program – do takich zastosowań parametry te są całkowicie wystarczające.

Dodam na koniec, że współcześnie dostępnych jest znacznie więcej niewielkich komputerów – na przykład LattePanda z procesorem x86 i zainstalowanym na pokładzie pełnym Windowsie 10 (w odróżnieniu od Windowsa 10 IoT, którego można zainstalować na Raspberry), bardzo dobrze wyposażony Asus Tinker Board, którego układ złącz i komponentów praktycznie 1:1 odpowiada Raspberry, Banana Pi z ósmiodzeniowym procesorem i tak dalej. Jednak dla początkującego konstruktora robotów Raspberry i Arduino będą chyba najlepszym wyborem przez to, że są to projekty dojrzałe, obecne na rynku przez długi czas i posiadające dużo kompatybilnego (czasami dedykowanego) sprzętu oraz bardzo duże wsparcie społeczności (głównie w postaci gotowych bibliotek).

SILNIKI

Najbardziej praktycznymi silnikami wykorzystywanymi w robotach są oczywiście silniki elektryczne. Mamy do wyboru ich cztery podstawowe rodzaje.

Silniki szczotkowe

Silniki szczotkowe są stosunkowo tanie, wykorzystywane zwykle jako napęd niewielkich modeli, często dostępne od razu z przekładnią zębatą, która zmniejsza prędkość obrotową, ale za to zwiększa moment obrotowy silnika. Wadą silników szczotkowych są zakłócenia, które powstają w wyniku pracy takiego silnika – dobrze jest je odseparować od układów elektronicznych lub zadbać o tłumienie zakłóceń.

Silniki bezszczotkowe

Silniki bezszczotkowe są nieco droższe, wykorzystywane zwykle jako napęd modelu; stosunkowo wydajne i wysokoobrotowe, dla tego też często wykorzystuje się je do budowy modeli latających. Charakteryzują się też dużą trwałością – są znacznie bardziej bezobsługowe niż silniki szczotkowe.

Zarówno w przypadku silników szczotkowych, jak i bezszczotkowych możemy sterować ich prędkością obrotową. Nie jest jednak możliwe nadanie im konkretnej, zadanej prędkości obrotowej (w szczególności dwa takie same silniki sterowane w taki sam sposób mogą obracać się z różną prędkością) – oznacza to, że nie nadają się do precyzyjnego sterowania robotem. Można jednak używać ich w połączeniu z czujnikami, za pomocą których określmy aktualne położenie lub prędkość robota, lub sterować robotem zdalnie.

Silniki krokowe

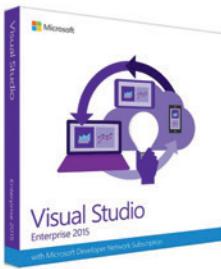
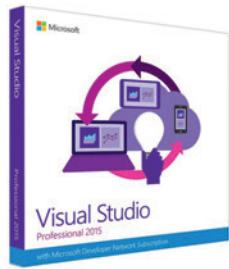
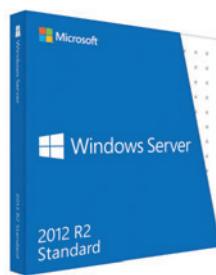
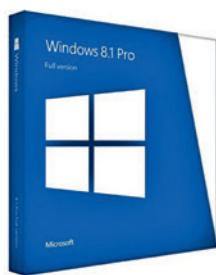
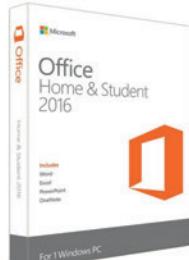
Silniki krokowe to takie silniki, którymi można sterować znacznie bardziej precyzyjnie. Każdy silnik krokowy ma zdefiniowaną liczbę kroków na obrót – na przykład 200 kroków na obrót oznacza, że jeden krok ma 1,8 stopnia. Możemy dzięki temu dosyć dokładnie przesunąć robota o zadaną odległość albo obrócić o zadaną kąt. Nie mamy jednak żadnej kontroli nad tym, w jakim położeniu znajduje się silnik początkowo – tu również musimy wspomóc się czujnikami.

Serwomechanizmy (serwa)

Są to silniki szczotkowe z wbudowaną przekładnią, wzbogacone o mały układ elektroniczny, który umożliwia uzyskanie konkretnego położenia kątowego. Serwa mają zwykle stosunkowo niewielki zakres pracy, na przykład ± 90 lub ± 45 stopni. Pozwalają one jednak na ustawienie w konkretnym położeniu z dosyć dużą dokładnością. Nadają się więc najlepiej do budowy ramion lub układów pan-tilt (obracających się w dwóch osiach – możemy na nich na przykład zamontować kamerę). Serwa charakteryzują się również dużym momentem obrotowym – na przykład na malutkim serwie TowerPro SG-90, które kosztuje 9 PLN i waży zaledwie 9 g, na ramienu o długości 1 cm możemy powiesić aż 1,8 kg. Nieco większe serwo TowerPro MG-995, ważące 55 g i kosztujące 27 PLN, pozwala na centymetrowym ramieniu powiesić aż 13 kg (wartości te maleją oczywiście proporcjonalnie wraz z długością ramienia, ale i tak robią wrażenie).



TTS Company rekomenduje oprogramowanie Microsoft ®



www.OprogramowanieKomputerowe.pl

Microsoft Azure

Office 365

OneDrive

Więcej informacji: ☎ (22) 272 94 94 ✉ sales@tts.com.pl

Sprzedaż



Dystrybucja



Import na zamówienie

CZUJNIKI

Kolejnym istotnym elementem robota są czujniki. Muszę szczerze przyznać, że byłem zszokowany, jak wiele czujników jest obecnie dostępnych i w jak niskich cenach. Jako anegdotę mogę powiedzieć, że pierwsze kontrolery lotu do quadrocopterów zbudowane były w oparciu o układ scalony wymontowywany z kontrolera popularnej konsoli Wii. Kontroler ten miał wbudowany akcelerometr i żyroskop i taniej było kupić taki kontroler i go zdementować niż zdobyć sam układ. W tej chwili stosunkowo dokładny trójosiowy akcelerometr z żyroskopem można kupić za mniej niż 50 PLN. Dodam jeszcze, że choć wspomniany projekt jest już w późnym stadium zaawansowania i korzysta ze zwykłych czujników, wciąż została jego historyczna nazwa – MultiWii.

Wśród dostępnych na rynku czujników możemy więc znaleźć: akcelerometry (mierzące przyśpieszenie), żyroskopy (mierzące przechył), magnetometry (cyfrowe kompasy), czujniki ciśnienia i wysokości, gazów, położen krańcowych (np. do wykrycia zamknięcia lub otwarcia drzwi lub okna), światła i koloru, poziomu cieczy, medyczne (w tym pulsometry, termometry, pomiar aktywności serca, pomiar aktywności mięśni), nacisku, odległości i ruchu, pogodowe (prędkości wiatru, temperatury, wilgotności i ciśnienia), prądu, przepływu cieczy czy wreszcie linii papilarnych. Co ważne, przeważająca większość komunikuje się z kontrolerem na jeden z trzech sposobów: analogowo (wysokością napięcia) lub cyfrowo (PWM albo poprzez standard I²C). Kiedy więc nauczymy się korzystać z tych trzech metod komunikacji (co na przykład w przypadku Arduino często sprowadza się do wywołania pojedynczego polecenia), możemy współpracować z dowolnym czujnikiem.

ZASILANIE

Istnieje kilka sposobów zasilania robota. Użycie zasilacza sieciowego w przypadku robotów stacjonarnych często jest najprostszym i jednocześnie najskuteczniejszym sposobem. Oprócz tego możemy skorzystać z baterii – są one łatwo dostępne, więc możemy je wymienić, gdy zajdzie taka potrzeba, choć wygodniejsze jest chyba podłączenie akumulatora (np. NiMh albo LiPo), tylko wtedy musimy pamiętać też o odpowiedniej ładowarce. Szeroki wybór akumulatorów o różnej konstrukcji, napięciach i pojemności dostępny jest w sklepach modelarskich.

W elektronice amatorskiej dominują napięcia zasilania: 3,3V, 5V i 12V. Miejmy jednak na uwadze, że nie jest to reguła – wiele komponentów (szczególnie silników) ma swoje specyficzne potrzeby, do których trzeba się dopasować.

W razie potrzeby można wyposażyć się w układy służące do obniżania lub podwyższania napięcia, tzw. step-down, step-up albo BEC/UBEC/SBEC – Battery Elimination Circuit (układ eliminowania [dodatkowego] akumulatora). Warto jednak mieć na uwadze napięcia zasilania i w miarę możliwości dobierać komponenty, które zasilane są takim samym napięciem (a jeszcze lepiej: równym napięciu źródła zasilania, czyli np. akumulatora). Regulowanie napięcia przy pomocy układów BEC jest bardzo łatwe, ale problem pojawia się wówczas, gdy przewody i BECe zaczynają zajmować połowę miejsca w naszym robocie.

RAMA

Tu otwiera się przed nami całe spektrum możliwości. Jeżeli planujemy robota jeżdżącego, możemy zaopatryć się w gotowe pod-

wozie – najpopularniejsze są dwukołowe z podpórkami, cztero-kołowe oraz gąsienicowe. Dwukołowe mają dużą zaletę – bardzo łatwo się obracają. Obrót podwozi cztero-kołowych odbywa się na podobnej zasadzie (koła z jednej strony obracają się w jednym kierunku, zaś z drugiej – w przeciwnym), ale zwiększone tarcie powoduje znacznie większy pobór prądu. Równie wygodne w użytkowaniu są podwozia gąsienicowe. Liczmy się jednak z tym, że gotowe podwozie będzie trochę kosztowało.

Ramę robota możemy również zbudować samodzielnie z dobowego materiału: pleksiglasu, drewna, aluminium, laminatu czy włókna węglowego (pamiętajmy tylko o odpowiednim izolowaniu elektroniki montowanej na materiałach przewodzących prąd). Sam chętnie korzystam z profili aluminiowych, które można kupić w sklepach budowlanych – tyle że wymaga to trochę pracy z brzeszczotem, pilnikami i wiertarką.

Do tego dochodzą gotowe zestawy montażowe z aluminium, drukowanie 3D – słowem, do wyboru, do koloru – ograniczeniem jest tylko pomysłowość projektanta i zasobność jego portfela – niektóre rozwiązania są niestety bardziej kosztowne od innych.

PROSTY PROJEKT

Spróbujmy zrealizować teraz prosty projekt polegający na zbudowaniu platformy pan-tilt (obracającej się w dwóch osiach) sterowanej z poziomu Raspberry Pi. Po zamontowaniu na niej kamery otrzymamy mechanizm umożliwiający Raspberry rozglądarkę się po okolicy.

Do zrealizowania takiego projektu będziemy potrzebować:

- » Raspberry Pi 3 (pamiętajmy o zasilaniu, klawiaturze, myszy i ekranie). Zainstalujemy na nim dedykowaną dystrybucję Raspbian Jessie;
- » Platformę pan-tilt – możemy zbudować ją samodzielnie lub skorzystać z gotowego rozwiązania;
- » Dwa serwa modelarskie dopasowane do skonstruowanej lub kupionej platformy;
- » Źródło zasilania do serw (5V).

STEROWANIE SERWEM, CZYLI SYGNAŁ PWM

Na początku spróbujmy podłączyć do Raspberry pojedyncze serwo i ustawić je w pożądanym położeniu.

Zasada działania serwa jest dosyć prosta. Serwo modelarskie jest niewielkim urządzeniem składającym się z silnika szczotkowego, przekładni, potencjometru i niewielkiego układu scalonego. Zadaniem przekładni jest zmniejszenie prędkości obrotowej silnika, ponieważ dzięki temu zwiększa się jego moment obrotowy. Układ scalony porównuje bieżący kąt obrotu serwa (odczytywany z potencjometru) z kątem docelowym i uruchamia silnik, żeby wykonać odpowiednią korektę.

Każde serwo wyprowadza trzy przewody – zwykle czarny, czerwony i biały lub brązowy, czerwony i żółty. Do czerwonego przewodu podłączamy biegun dodatni źródła zasilania (5V), do brązowego lub czarnego – biegun ujemny (masę, GND), zaś przewód biały lub żółty jest przewodem sygnału – poprzez niego sterujemy serwem.

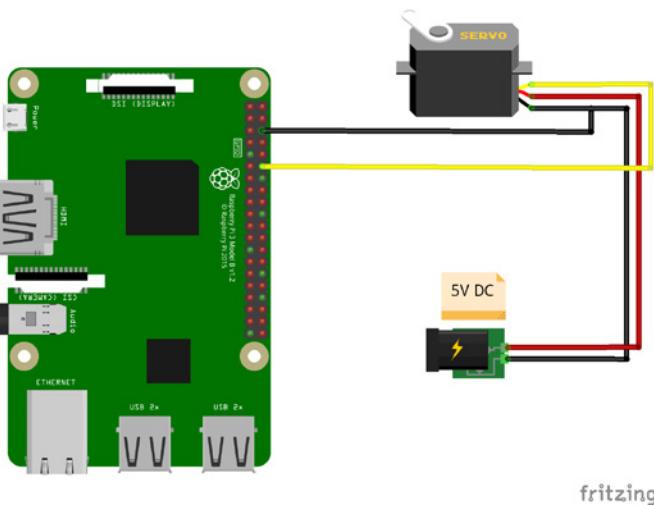
Serwo sterowane jest sygnałem PWM – akronim ten rozwija się do „Pulse-Width Modulation”, czyli modulacja szerokością impulsu. Na przewód sygnałowy około 55 razy na sekundę podawane jest napięcie 5V, zaś czas jego trwania definiuje docelowe położenie serwa. Czas trwania impulsu równy 1000 mikrosekund (czyli 1 milisekundę,

1/1000 sekundy) jest interpretowany jako pozycja „-1”, czyli maksymalne wychylenie w jedną stronę, czas trwania 1500 mikrosekund oznacza pozycję „0”, czyli środkową, zaś czas trwania równy 2000 mikrosekund przekładany jest na pozycję „1”, czyli maksymalne wychylenie w drugą stronę. Aby więc sterować serwem, musimy przy pomocy Raspberry wygenerować odpowiedni sygnał PWM.

Wartości graniczne długości trwania impulsu: 1 i 2 milisekundy są traktowane jako standardowe – na przykład PWM w takim właśnie zakresie generuje wiele odbiorników aparatur modelarskich. Wiele serw pracuje jednak w szerszym zakresie – nawet od 0.5 do 2.5 milisekundy. Należy jednak korzystać z wartości skrajnych ostrożnie – jeżeli wartość będzie poza zakresem obrotu serwa, to może ono próbować osiągnąć ją „na siłę”, co może doprowadzić w skrajnym przypadku do spalenia silnika.

SCHEMAT POŁĄCZENIA

Schemat połączenia Raspberry z serwem znajdziemy na Rysunku 1. Zwrócmy uwagę na fakt, że kabel GND serwa (czarny) połączony jest nie tylko ze źródłem zasilania, ale również z pinem GND komputera. Musimy dodać to połączenie, aby przewodem sygnału mógł płynąć prąd (jak pamiętamy – prąd płynie tylko w zamkniętym obiegu).



Rysunek 1. Połączenie Raspberry Pi z serwem

STEROWANIE BEZPOŚREDNIO Z RASPBERRY

Programy sterujące serwem będziemy pisać w Pythonie – jak wspomniałem, ma on bardzo duże wsparcie ze strony społeczności użytkowników Raspberry. Do sterowania pinami GPIO użyjemy na początku bibliotekę RPi.GPIO, która w najnowszej dystrybucji Raspbiana powinna być zainstalowana od razu.

Kod skryptu sterującego serwem wygląda następująco:

Listing 1. Skrypt Pythona sterujący serwem

```
import RPi.GPIO as GPIO
import time

# Serwo działa z częstotliwością 55 Hz
FREQUENCY = 55

# 55 razy na sekundę, czyli jeden cykl trwa
# 1/55 ~ 0.018 s, czyli 18 ms
CYCLE_MS = (1 / float(FREQUENCY)) * 1000

# Pin 18 Raspberry obsługuje sprzętowe
```

```
# generowanie PWM
PIN = 18

# Funkcja oblicza, jaki procent (0~100) czasu
# trwania cyklu sygnał powinien być wzbudzony
def calcDutyCycleFor(timeMs):
    return (timeMs / (CYCLE_MS)) * 100

# Funkcja oblicza czas trwania wzbudzonego
# sygnału dla zadanego kąta. 0 - 1 ms, 180 - 2 ms
def evalTimeForAngle(angle):
    return (float(angle) / 180.0) + 1.0

# Funkcja ustawia serwo w zadanym położeniu
def setServoAngle(pwm, angle):
    timeMs = evalTimeForAngle(angle)
    duty = calcDutyCycleFor(timeMs)
    pwm.ChangeDutyCycle(duty)

# Numerujemy piny według nazw (18 = GPIO18)
GPIO.setmode(GPIO.BCM)

# Pin 18 ustawiamy jako wyjściowy
GPIO.setup(PIN, GPIO.OUT)

# Ustalamy częstotliwość PWM
pwm = GPIO.PWM(PIN, FREQUENCY)

# Zaczynamy wysyłać PWM z wartością początkową
# 1.5 ms, czyli „zerową”
pwm.start(calcDutyCycleFor(1.5))

# Obracamy serwo od 0 do 180 stopni w 18 krokach
for i in range(0, 19):
    setServoAngle(pwm, i * 10)
    time.sleep(1)

# Zerujemy położenie serwa
setServoAngle(pwm, 90)

# Dajemy mu chwilę czasu na ustawienie
time.sleep(0.5)

# Zatrzymujemy sygnał PWM
pwm.stop()
GPIO.cleanup()
```

Biblioteka GPIO w dosyć specyficzny sposób podchodzi do sygnału PWM. Podczas inicjalizowania obiektu `pwm` podajemy pin, na którym generowany będzie sygnał, oraz częstotliwość, z jaką wzbudzane będzie napięcie. Serwa modelarskie odbierają sygnał PWM z częstotliwością ok. 55 Hz (czyli 55 razy na sekundę) – stąd wartość przekazana do funkcji `GPIO.PWM`. Dochodzi teraz kwestia szerokości impulsu – podajemy ją bibliotece GPIO jako procent czasu trwania pojedynczego cyklu. 55 Hz daje nam:

$$\frac{1}{55} \cdot 1000 \approx 18 \text{ ms} (0.018 \text{ s})$$

Jeżeli więc chcemy osiągnąć szerokość pulsu 1.5 ms, by ustawić serwo w położeniu środkowym, musimy obliczyć:

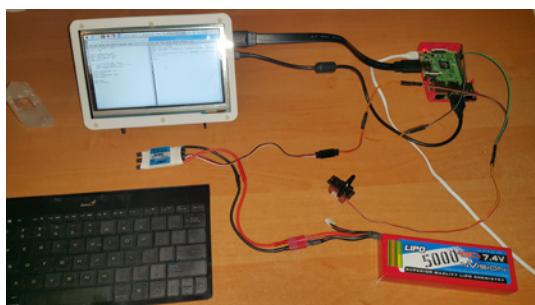
$$\frac{1.5}{(\frac{1}{55}) \cdot 1000} \cdot 100 = 8.25\%$$

Tę wartość przekazujemy do `pwm.start` jako początkową wartość szerokości impulsu lub do `pwm.ChangeDutyCycle`, by wartość tę zmienić w trakcie działania skryptu.

Podłączamy zasilanie do serwa zgodnie z opisany wcześniej schematem (uwaga na polaryzację, czyli plus i minus, oraz na wartość napięcia!), uruchamiamy skrypt – i serwo zaczyna się poruszać. Przykładowe, wypróbowane przeze mnie ustawienie można zobaczyć na Rysunku 2.

KWESTIA ZASILANIA

Raspberry Pi na kilku pinach GPIO wyprowadza również +5V – niektóre przykłady w Internecie sugerują wręcz, by zasilić serwo właśnie z tych pinów. Nie jest to jednak dobry pomysł.



Rysunek 2. Tak wygląda serwo podłączone do Raspberry

Większość serw wyposażona jest w silnik szczotkowy, który – z powodu specyficznej konstrukcji – podczas pracy generuje zakłócenia. Jeżeli podłączymy serwo bezpośrednio do Raspberry, zakłocenia te przeniosą się na układy elektroniczne komputerka. Poza tym serwo podczas zmiany kierunku pobiera chwilowo bardzo dużo prądu, co może spowodować spadek albo zanik napięcia – w efekcie czego Raspberry może zawiesić się albo nawet wyłączyć.

Do tego dochodzi jeszcze jeden aspekt: część pinów dostarczających napięcie +5V jest podłączona bezpośrednio do wtyczki micro USB, z której Raspberry jest zasilane, ale część z nich podłączonych jest do elektroniki komputerka. Jeśli zasilimy z nich zewnętrzne urządzenia, przez przewody te będzie płynął (relatywnie) wysoki prąd, który również może komputer uszkodzić.

W skrócie: zasilanie układów elektronicznych z jednego, a mechanicznych z drugiego źródła prądu pozwoli nam oszczędzić sobie dużo niepotrzebnych problemów.

PIERWSZE PROBLEMY

Jeżeli przyjrzymy się dokładnie pracy serwa, zauważymy, że po osiągnięciu danego położenia w czasie działania skryptu nie stoi ono w miejscu, tylko porusza się prawie niezauważalnie w lewo i prawo. Łatwiej jest to usłyszeć niż zobaczyć, ponieważ poruszenia są na granicy widoczności, ale słyszać pracę silnika i przekładni. Skąd poruszenia owe się biorą?

Problem leży (pośrednio) w bibliotece RPi.GPIO. Linux, na którym pracujemy, nie jest systemem czasu rzeczywistego, co oznacza, że w danym momencie obsługuje od kilkunastu do kilkudziesięciu procesów. Biblioteka RPi.GPIO działa najprawdopodobniej w ten sposób, że wzbudza sygnał na wybranym pinie, usypia wątek obsługujący PWM na czas trwania pulsu, po czym wygasza sygnał – i tak dalej. Z uwagi na fakt, iż Linux jest wielozadaniowy, nie ma żadnej gwarancji, że wątek zostanie obudzony dokładnie po określonym czasie. W momencie wybudzenia wątku mogło więc upływać więcej czasu niż wątek sobie życzył, ale niestety ta odrobiną czasu jest kluczowa do precyzyjnego ustawienia serwa.

Z faktu, że RPi.GPIO nie utrzymuje serwa stabilnie w jednym położeniu, możemy wywnioskować, że PWM generowany jest programowo. Tymczasem Raspberry na GPIO18 (do którego podłączliśmy przewód sygnału) potrafi generaować sygnał sprzętowo, co oznacza, że zadanie to jest zlecone odpowiedniemu układowi, który pracuje niezależnie od procesora i systemu operacyjnego. Aby to zrobić, musimy jednak skorzystać z innej biblioteki – wiringpi.

INSTALACJA WIRINGPI

Wiringpi jest napisaną przez Gordona Hendersona biblioteką do obsługi portów GPIO Raspberry. Została ona przygotowana z my-

szą o wykorzystaniu w języku C, ale pojawiły się również wrappery umożliwiające wykorzystanie jej w Pythonie. Nie jest ona bezpośrednio dostępna w Raspbianie, więc musimy ją najpierw zainstalować.

Na początku aktualizujemy apt-get:

```
sudo apt-get update
```

Następnie instalujemy bibliotekę python-dev, która będzie potrzebna do zbudowania pakietu wiringpi2 dla Pythona:

```
sudo apt-get install python-dev
```

Kolejnym krokiem jest doinstalowanie pip – menadżera pakietów dla Pythona:

```
sudo apt-get install python-pip
```

Możemy go teraz wykorzystać do automatycznego ściągnięcia i zainstalowania biblioteki wiringpi2.

```
sudo pip install wiringpi2
```

Teraz bardzo ważna kwestia: wiringpi działa prawidłowo tylko wtedy, gdy skrypt Pythona uruchomimy z uprawnieniami roota. Możemy to łatwo zrobić, uruchamiając środowisko Idle przy pomocy polecenia sudo:

```
sudo idle
```

STEROWANIE SERWEM PRZY POMOCY WIRINGPI

Możemy teraz napisać drugi skrypt sterujący serwem, tym razem przy użyciu biblioteki wiringpi.

Listing 2. Skrypt Pythona sterujący serwem przy użyciu wiringpi

```
import wiringpi, time
# UWAGA!
#
# Ten skrypt zadziała tylko z uprawnieniami roota.

# Pin GPIO18
PIN = 18
# Wartości dzielników częstotliwości
PWM_CLOCK = 192
PWM_RANGE = 2000

# Docelowa częstotliwość cyklu: 50 Hz
TARGET_FREQ = float(19200000) / (PWM_CLOCK * PWM_RANGE)
# Czas trwania jednego cyklu
CYCLE_TIME = 1 / TARGET_FREQ
# Czas trwania jednego "kroku"
STEP_TIME = CYCLE_TIME / PWM_RANGE
# Liczba kroków dla 1 ms
PULSE_MIN = int(0.001 / STEP_TIME)
# Liczba kroków dla 2 ms
PULSE_MAX = int(0.002 / STEP_TIME)

def SetAngle(angle):
    wiringpi.pwmWrite(PIN, int(PULSE_MIN + (float(angle) / 180.0) * (PULSE_MAX - PULSE_MIN)))

# Uruchamiamy wiringpi - stosujemy nazewnictwo
# GPIO (czyli 18 = GPIO18)
wiringpi.wiringPiSetupGpio()
# Ustalamy pin 18 jako wyjście PWM
wiringpi.pinMode(PIN, wiringpi.GPIO.PWM_OUTPUT)

# Ustawiamy tryb klasyczny generowania PWM
```

```
wiringpi.pwmSetMode(wiringpi.GPIO.PWM_MODE_MS)
# Ustawiamy PWM Clock
wiringpi.pwmSetClock(PWM_CLOCK)
# Ustawiamy PWM Range
wiringpi.pwmSetRange(PWM_RANGE)

# Obracamy serwo
for angle in range(0, 180):
    print "Angle: " + str(angle)
    SetAngle(angle)
    time.sleep(0.1)
```

Wiringpi steruje sygnałem PWM nieco inaczej niż RPi.GPIO. Po pierwsze, musimy przełączyć sposób generowania PWM na tryb MS – mark space mode. Jest to tryb „tradycyjny”, w którym wzbudzanie napięcia jest realizowane ze stałą częstotliwością. Zainteresowanych odsyłam do odpowiedniego wątku na raspberrypi.stackexchange.com – jest tam podane nieco więcej informacji na ten temat (link na końcu artykułu).

Drugim krokiem jest ustawienie wartości `pwmClock` i `pwmRange`.

Zegar Raspberry Pi generujący sygnały PWM pracuje z częstotliwością 19.2 MHz. Częstotliwość ta jest dzielona przez iloczyn `pwmClock` i `pwmRange`, by osiągnąć bazową częstotliwość pojedynczego cyklu. Cykl ten jest następnie dzielony na `pwmRange` kroków – podczas każdego takiego kroku inkrementowany jest wewnętrzny licznik PWM. Gdy osiągnie on wartość równą `pwmRange`, jest resetowany do zera. Szerokość sygnału określa się, podając liczbę kroków, podczas których powinien być podawany na wyjście sygnał wysoki.

Na przykład, jeżeli chcemy osiągnąć 50 Hz, możemy ustawić `pwmClock` na wartość 1920, zaś `pwmRange` na 200 (bo $1920 \times 200 = 50$). Przy częstotliwości 50 Hz i liczbie kroków równej 200 jeden krok trwa $(1/50)/200 = 0.0001 = 0.1$ ms. Aby ustawić serwo w pierwszym położeniu skrajnym, ustawiamy wartość graniczną na 1.0 ms / 0.1 ms = 10. Drugą wartość skrajną osiągniemy po wprowadzeniu wartości 2.0 ms / 0.1 ms = 20.

Zwróciły uwagę, że osiągnięta w ten sposób rozdzielcość pracy jest stosunkowo niska. Możemy zwiększyć ją, zmieniając dzielniki – na przykład `pwmClock` na 192, a `pwmRange` na 2000. Teraz jeden krok wewnętrz cyklu ma $(1/50)/2000 = 0.01$ ms, a pełny zakres serwa osiągniemy, ustawiając wartość graniczną w zakresie 100 do 200. Jeden z użytkowników raspberrypi.stackexchange.com sprawdził doświadczalnie, że `pwmClock` może przyjmować wartości od 2 do 4095, zaś `pwmRange` – od 2 do 4096.

DEDYKOWANY UKŁAD

Po wcześniejszych doświadczeniach wiemy już, że nie uda nam się sterować przewidywalnie dwoma serwami – jednym z nich musielibyśmy sterować programowo, a jak już wiemy – nie daje to zadowalających wyników.

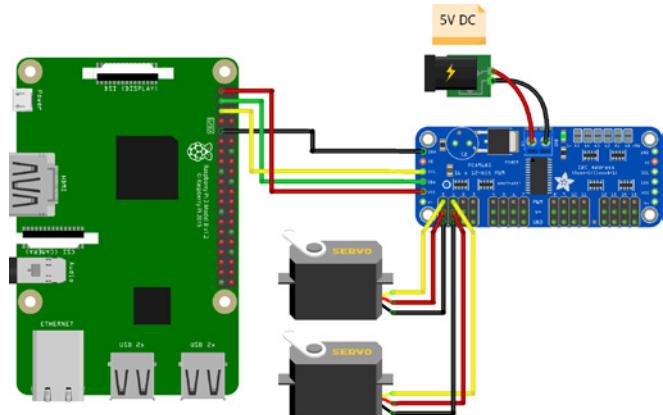
W takiej sytuacji możemy zastosować dedykowany układ, który pomaga rozwiązać ten problem. Mowa o Adafruit PCA9685 – 16-kanałowym sterowniku PWM. Komunikuje się on z Raspberry przez protokół I²C i potrafi sterować 16 serwami z 12-bitową precyją (4096 kroków).

Aby skorzystać z tego sterownika, musimy wcześniej trochę się przygotować. Zaczytamy od zainstalowania dwóch pakietów:

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

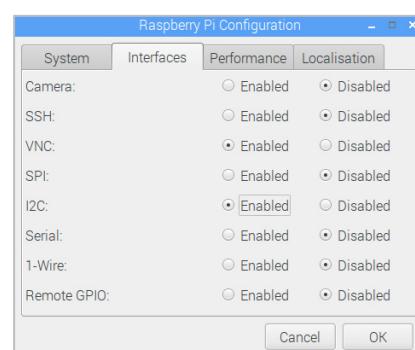
Drugi z pakietów nie jest konieczny, ale zawiera przydatne narzędzia, które za chwilę będą pomocne. W najnowszej dystrybucji Raspbiana oba pakiety powinny być zainstalowane domyślnie.

Na Rysunku 3 pokazano, w jaki sposób podłączyć układ PCA9685 do Raspberry. Należy zwrócić szczególną uwagę, aby pin 3v3 Raspberry podłączyć do pinu VCC układu, a nie do V+. Układ na pinie V+ podaje 5V pobrane z zasilania serw – pomyłka w najgorszym wypadku może nas kosztować spalenie Raspberry.



Rysunek 3. Podłączenie układu PCA9685 do Raspberry

Teraz musimy włączyć w Raspberry obsługę I²C. W tym celu otwieramy w środowisku graficznym główne menu i przechodzimy do *Preferences | Raspberry Pi Configuration*. Tam upewniamy się, że interfejs I²C jest włączony – jak na Rysunku 4. W starszych wersjach Raspberry opcje te znajdują się w konsolowym narzędziu `raspi-config`.

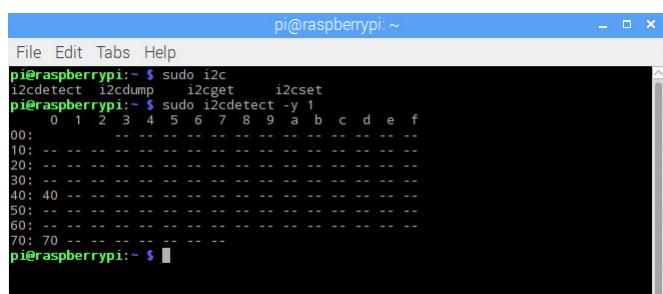


Rysunek 4. Włączanie obsługi I²C

Jeżeli mamy już podłączony moduł PCA9685, możemy z konsoli wywołać polecenie:

```
sudo i2cdetect -y 1
```

Jeżeli wszystko jest w porządku, powinniśmy zobaczyć wynik jak na Rysunku 5. Pozycja 40 oznacza podłączony moduł.



Rysunek 5. Weryfikacja modułów podłączonych przez I²C

Teraz możemy przystąpić do pisania skryptu sterującego serwami.

Listing 3. Skrypt sterujący dwoma serwami za pomocą modułu PCA9685

```
import time
import Adafruit_PCA9685

# Tworzymy obiekt pwm (przyjmie domyślnie adres 0x40)
pwm = Adafruit_PCA9685.PCA9685()

# Alternatywnie możemy podać samodzielnie odpowiednie parametry
# pwm = Adafruit_PCA9685.PCA9685(address=0x41, busnum=2)

# Częstotliwość sterowania PWM
FREQUENCY = 50
# Czas jednego cyklu
CYCLE_TIME = 1 / float(FREQUENCY)
# Procentowy czas cyklu dla położenia minimalnego
SERVO_MIN_PERCENT = 0.001 / CYCLE_TIME
# Procentowy czas cyklu dla położenia maksymalnego
SERVO_MAX_PERCENT = 0.002 / CYCLE_TIME
# Wartość minimalna dla 4096 kroków
SERVO_MIN = int(4096 * SERVO_MIN_PERCENT)
# Wartość maksymalna dla 4096 kroków
SERVO_MAX = int(4096 * SERVO_MAX_PERCENT)

# Funkcja ustawia konkretnemu serwu zadany kat
def SetAngle(channel, angle):
    pwm.set_pwm(channel, 0, int(SERVO_MIN + (float(angle) / 180.0) * (SERVO_MAX - SERVO_MIN)))

# Ustawiamy częstotliwość generowania PWM
pwm.set_pwm_freq(FREQUENCY)

# Obracamy dwoma serwami
for angle in range(0, 180, 10):
    SetAngle(14, angle)
    SetAngle(15, angle)
    time.sleep(0.2)
```

Pozostaje tylko uruchomić skrypt i przekonać się, że serwa obracają się zgodnie z przewidywaniami.

W tym przypadku nie ma też zbyt wiele do komentowania – skrypt korzystający z modułu PCA9685 jest chyba najprostszy ze wszystkich dotąd zaprezentowanych.

Wspomniałem na początku, że budowa robota nie jest aż tak kosztowna, jak mogłoby się wydawać. Zobaczmy teraz, jak kształtują się ceny komponentów, z których korzystaliśmy w niniejszym artykule.

Na początek musimy zaopatrzeć się w sam komputerek Raspberry Pi 3. Będzie nas to kosztowało około 190 PLN i to jest chyba najdroższy element całego montażu. Jeżeli cena taka nie jest akceptowalna, możemy kupić Raspberry Pi Zero W (wersja z wbudowanym Wifi i Bluetooth) za 52 PLN lub Raspberry Pi Zero za 26 PLN – na obu z nich powinno nam się udało zrealizować wszystkie opisane projekty, tylko będzie to wymagało nieco więcej przygotowań (na przykład oba minikomputerki nie mają przylutowanych goldpinów GPIO, trzeba to zrobić we własnym zakresie lub lutować przewody bezpośrednio).

Pamiętajmy, że dla wygody pracy z Raspberry dobrze jest mieć dla niego ekran, klawiaturę i mysz. Ja wyposażałem się w 7" dotykowy ekran, ale Raspberry powinno udało się podłączyć do większości monitorów przez kabel HDMI. Na komputerku jest też zainstalowane

wany serwer VNC, więc wystarczy ściągnąć program VNC Viewer, by pracować na Raspberry zdalnie z komputera PC lub notebooka (upewnijmy się, że VNC jest na Raspberry włączone – możemy to zrobić w tym samym miejscu, w którym włączaliśmy I²C).

Drugim krokiem jest zakup serw modelarskich. Jeżeli planujemy zrobić tylko proof-of-concept, swobodnie wystarczy malutkie TowerPro SG-90 micro (9-12 PLN). Większe serwa – na przykład TowerPro SG-5010 – będą nas kosztować 30 PLN. Zestaw pan-tilt na mikro serwa kupimy za 27 PLN. Możemy też od razu kupić zestaw pan-tilt z serwami: w wersji mikro będzie to koszt 75 PLN, zaś z serwami o rozmiarze standard – 85 PLN. Dodajmy, że gdy kupujemy serwo, dostajemy do niego od razu kilka orczyków oraz wkręty do plastiku, którymi możemy przykręcić do nich większe elementy. Do samej osi serwa możemy też przykręcać elementy – najczęściej śrubami M3 (choć nie jest to reguła).

Do łączenia komponentów będą nam potrzebne przewody połączeniowe – są one standardowe i dostępne w większości sklepów elektronicznych, również tych stacjonarnych. Ja kupiłem od razu dla wygody pakiet: 3x40 szt: męsko-męskie, męsko-żeńskie i żeńsko-żeńskie – jest to koszt 30 PLN. Pojedynczy zestaw 40 przewodów jednego rodzaju będzie nas kosztował 10 PLN.

Jeżeli chodzi o elektronikę, to mamy jeszcze moduł PCA9685 – oryginalny Adafruit będzie nas kosztował 72 PLN, a zamiennik o identycznej funkcjonalności – 24 PLN. Serwa musimy zasilić 5V – zwykle korzystam tu z akumulatora LiPo 2S lub 3S i układu BEC. Układ taki kosztuje 15 PLN. Możemy do niego podłączyć też np. koszyk z 4 bateriami AA (4x1,5V = 6V). Taniej będzie zastosować akumulatorki, ale pamiętajmy, że mają one napięcie 1.2V, a nie 1.5V, jak baterie.

PODSUMOWANIE

Budowanie robotów wcale nie jest tak trudne, jak mogliby się wydawać! Nie ukrywajmy: możliwość pisania oprogramowania dla robota w tak wysokopoziomowym języku, jakim jest Python, oraz korzystania z bogatego zasobu dokładnych czujników, kamer i silników – nie wspominając już o tym, że programy możemy piisać na samym kontrolerze, na którym pracuje pełnoprawny Linux – pozwala naprawdę niewielkim kosztom wejść w świat robotyki. W jednym z kolejnych artykułów postaram się opisać, z jakimi problemami można spotkać się podczas budowy jeżdżącego robota sterowanego przez płytę Arduino.

WOJCIECH SURA

wojciechsura@gmail.com



Programuje od przeszło dziesięciu lat w Delphi, C++ i C#, prowadząc również prywatne projekty. Obecnie pracuje w polskiej firmie PGS Software S.A., zajmującej się tworzeniem oprogramowania i aplikacji mobilnych dla klientów z całego świata.

W sieci

- ▶ http://www.multiwii.com/wiki/?title>Main_Page – Strona wiki kontrolera quadrocopterów MultiWii.
- ▶ <https://botland.com.pl/> – Bardzo dobrze wyposażony sklep internetowy – można tu kupić Raspberry, Arduino, silniki, serwa, czujniki i mnóstwo innych komponentów służących do budowy robotów.
- ▶ <https://nettigo.pl> – Kolejny bardzo dobrze wyposażony sklep internetowy – pierwszy dystrybutor Raspberry Pi w Polsce.
- ▶ <http://abc-rc.pl> – Sklep modelarski – zaopatrzymy się tu między innymi w układy BEC czy zamiennik PCA9685.
- ▶ <https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/> – Dokumentacja biblioteki RPi.GPIO.
- ▶ <http://wiringpi.com> – Strona internetowa biblioteki wiringpi.
- ▶ <https://raspberrypi.stackexchange.com/questions/4906> – Wątek dotyczący sprzętowego sposobu generowania sygnału PWM w Raspberry.
- ▶ <http://www.multiwii.com> – Strona projektu MultiWii.

18-20 PAŹDZIERNIKA 2017

HALA EXPO XXI, WARSZAWA

NAJWIĘKSZA W EUROPIE ŚRODKOWO-WSCHODNIEJ
KONFERENCJA POŚWIĘCONA PLATFORMIE .NET



PRE-CONS - 18.10.2017

Alex Mang	Dino Esposito	Gill Cleeren	Sasha Goldshtain
Breaking Apps Apart Intentionally - Visual Studio + Docker + Sprinkles of Azure = Modern Microservices	Programming ASP.NET MVC Core	Developing cross-platform apps with C# using Xamarin	Production Performance and Troubleshooting of .NET Applications

DAY 1 - 19.10.2017

Hall A	Hall A	Hall B	Hall C
09:00 - 09:20 Conference opening 09:20 - 10:20 Surviving Microservices (Opening Keynote) Michele Leroux Bustamante 10:40 - 11:40 5 unit testing facts I wish I know 10 years ago (200) Dror Helper 12:00 - 13:00 Async/Await and the Task Parallel Library: await headexplosion (400) Daniel Marbach 14:00 - 15:00 Sponsor Session	 15:20 - 16:35 Adding History to CRUD (400) Dino Esposito 16:55 - 17:55 Software Architecture That Every Developer Should Know (300) Alon Fließ 18:15 - 19:00 The Same Software Just the Other Way Around (200) Dino Esposito 19:15 - 20:00 Q&A with speakers	 10:40 - 11:40 Deep Dive into Machine Learning with Azure Cloud (400) Lino Tadros 12:00 - 13:00 How Far Can 'Serverless' Actually Go? (300) Alex Mang 15:20 - 16:35 Building a Complete IoT Massive Multiplayer Game with C# (200) Eran Stiller 16:55 - 17:55 Launching patterns for containers - it's more than just scheduling (200) Michele Leroux Bustamante	 10:40 - 11:40 Building your first real-world Android app using Xamarin in just 60 minutes (300) Gill Cleeren 12:00 - 13:00 Bot-Tender: A .NET Chatbot Walks into a Bar (200) Eran Stiller 15:20 - 16:35 Debugging and Profiling .NET Core Apps on Linux (300) Sasha Goldshtain 16:55 - 17:55 Practical Xamarin.Forms (400) Gill Cleeren

DAY 2 - 20.10.2017

Hall A	Hall A	Hall B	Hall C
09:00 - 10:00 Performance that pays off (300) Szymon Kulec 10:20 - 11:20 I Have a Microservices Architecture and I Didn't Know (200) Dino Esposito 11:40 - 12:55 Building Evolutionary Architectures (300) Neal Ford 14:00 - 15:00 Sponsor Session: Securing .NET applications in Azure (300) Sebastian Solnica	 15:15 - 16:15 Unit testing patterns for concurrent code (300) Dror Helper 16:30 - 17:30 Stories Every Developer Should Know (Closing Keynote) Neal Ford 17:30 - 18:00 Closing of the Conference	 09:00 - 10:00 Building a real life IoT Project for the cloud (300) Lino Tadros 10:20 - 11:20 Past, Present & Future of C# Debugging (300) Alon Fließ 11:40 - 12:55 Secret unit testing tools no one ever told you about (200) Dror Helper 15:15 - 16:15 How I Built An Open-Source Debugger (300) Sasha Goldshtain	 09:00 - 10:00 Microservices with Service Fabric. Easy... or is it? (300) Daniel Marbach 10:20 - 11:20 The Performance Investigator's Field Guide (300) Sasha Goldshtain 11:40 - 12:55 Creating a real-world Xamarin application architecture using MVVM (400) Gill Cleeren 15:15 - 16:15 So many Docker platforms, so little time... (200) Michele Leroux Bustamante

EVENT SPONSORS:



Wdrażanie aplikacji w środowisku Openshift

Konteneryzacja staje się standardem w dziedzinie wytwarzania i dostarczania oprogramowania. Na rynku istnieje wiele rozwiązań związanych z wydawaniem oprogramowania w takim środowisku. W tym artykule przybliżymy temat otwartej platformy Openshift dostarczanej przez firmę RedHat.

DOCKERYZACJA APLIKACJI – PRZYGOTOWANIE DOCKERFILE

Przykładowa aplikacja

Na potrzeby artykułu napiszemy prostą aplikację, którą później spróbujemy wydać w środowisku Kubernetes/Openshift. Niech nasza aplikacja zlicza odwiedziny, przechowując licznik w zewnętrznym serwisie. Dzięki zastosowaniu zależności w postaci Redis cache nasz przykład będzie bardziej miarodajny.

Przykładowa aplikacja napisana w Python 3 z wykorzystaniem framework'a Tornado [1] wygląda następująco:

Listing 1. Przykładowa aplikacja

```
#!/usr/bin/env python
import tornado.ioloop
import tornado.web
import redis

class VisitHandler(tornado.web.RequestHandler):
    def get(self):
        r = redis.StrictRedis(host='redis', port=6379, db=0)
        cnt = r.get('count')
        if cnt is not None:
            visit = int(cnt, 10)+1
        else:
            visit = 1
        r.set('count', visit)
        self.write("Visit no. {}".format(visit))

def make_app():
    return tornado.web.Application([
        ('/visit', VisitHandler),
    ])

if __name__ == "__main__":
    app = make_app()
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

Zależności uruchomieniowe aplikacji, zawarte w pliku requirements.txt, to:

Listing 2. Plik zależności

```
appdirs==1.4.3
packaging==16.8
pyparsing==2.2.0
redis==2.10.5
six==1.10.0
tornado==4.2
```

Aplikację możemy uruchomić w wirtualnym środowisku Pythona z zainstalowanymi zależnościami:

Listing 3. Uruchamianie przykładowej aplikacji

```
virtualenv -p python3 .venv
source .venv/bin/activate
pip install -r requirements.txt
./main.py
```

Prawidłowo uruchomiona aplikacja wyświetli w przeglądarce ilość dotychczasowych wizyt po odwiedzeniu <http://localhost:8888>.



Rysunek 1. Przykładowa aplikacja

Dockerfile

Aplikacja jest już gotowa. Kolejnym krokiem jest przygotowanie jej do dystrybucji. W tym celu utworzymy Dockerfile. Na łamach „Programisty” temat Dockera był już poruszany, więc przypomnijmy tylko, że jest to plik opisujący obraz dockerowy. Jego rolą jest opisanie sposobu instalacji i uruchomienia aplikacji. Dockerfile dla opisanego powyżej przykładu może wyglądać następująco:

Listing 4. Dockerfile przykładowej aplikacji

```
FROM python:3.5
EXPOSE 8888
WORKDIR /visitor
COPY main.py main.py
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
ENTRYPOINT ./main.py
```

Nie jest zaskoczeniem, że obraz bazowy to python:3.5. Z kodu aplikacji wynika, że pracuje ona na porcie 8888, stąd kontener z naszą aplikacją powinien nasłuchiwać połączeń na tym porcie. Do katalogu /visitor stworzonego poleceniem WORKDIR dodajemy poleceniem COPY kod aplikacji oraz listę zależności. Przed uruchomieniem aplikacji należy spełnić zależności, zatem zostaną one zainstalowane na etapie budowania obrazu w wyniku wykonania polecenia RUN pip install -r requirements.txt. Ostatnim krokiem jest uruchomienie aplikacji poleceniem ENTRYPOINT.

Docker compose

Dotarliśmy do etapu, w którym posiadamy kod aplikacji oraz opisaliśmy sposób jej instalacji i wykonania kodu. Ostatnim potrzebnym

SZUKASZ PRACY MARZEŃ?



ZRÓB PIERWSZY KROK!

PRZYJDŹ DO SZKOŁY JĘZYKA ANGIELSKIEGO SPEAK UP

- 🕒 NAUCZYMY CIĘ ANGIELSKIEGO SZYBKO I SKUTECZNIE
- ▶ STAWIAMY NA TWÓJ ROZWÓJ POPRZEZ INNOWACYJNE METODY NAUCZANIA
- ⌚ GWARANTUJEMY ELASTYCZNOŚĆ I INDYWIDUALNE PODĘJŚCIE
- 🏆 Z NAMI ZDOBĘDZIESZ CERTYFIKAT JĘZYKOWY

The background image shows a panoramic view of the Kuala Lumpur city skyline at dusk or night. The iconic Petronas Twin Towers are brightly lit, standing prominently against the dark sky. Other skyscrapers and buildings are visible, illuminated by their own lights. In the foreground, there's a large, modern building complex with a glass facade and some greenery.

THE ENGLISH SCHOOL
**SPEAK
UP**

WWW.SPEAK-UP.PL

elementem układanki jest opis uruchomienia aplikacji wraz z jej zależnościami. W realizacji tego etapu pomoże nam plik docker-compose.yml. Podobnie jak w poprzednim punkcie, jest to temat, który był już poruszany na łamach magazynu, więc przejdźmy od razu do przykładu:

Listing 5. Docker compose przykładowej aplikacji

```
version: "2"
services:
  redis:
    image: redis:3.0

  visitor:
    build: .
    ports:
      - 8888:8888
    links:
      - redis:redis
```

Opis nie jest skomplikowany, wynikają z niego trzy istotne informacje, a mianowicie:

- » Kontener o nazwie redis należy uruchomić, używając obrazu redis:3.0.
- » Do uruchomienia kontenera visitor konieczna jest gotowa do linkowania instancja kontenera redis.
- » Kontener redis uruchamiamy z obrazu, który należy najpierw zbudować z katalogu lokalnego.

Po bardziej szczegółowy opis struktury pliku docker-file.yml odsyłam do dokumentacji [2].

DEPLOYMENT DOCKEROWY

Docker registry

Nasza aplikacja jest gotowa do dystrybucji. Na tym etapie musimy odpowiedzieć sobie na pytanie: jak należy ją dystrybuować? Autorzy Dockera oraz społeczność open source dostarczają nam narzędzie, dzięki któremu możemy na nie odpowiedzieć. Tym narzędziem jest docker registry.

Docker registry to repozytorium, w którym przechowujemy obrazy dockerowe. Istnieje wiele rozwiązań, które możemy uruchomić wewnętrznej sieci firmowej, takich jak podstawowy obraz registry [3] czy bardziej zaawansowany Portus [4] od Suse. Na potrzeby artykułu zastosujemy jednak publicznie dostępny Docker Hub [5], do korzystania z którego wystarczy założyć darmowe konto.

Praca z rejestrzem dockerowym wygląda zawsze tak samo, niezależnie od tego, jakie oprogramowanie jest używane na jego serwerze. Docker dostarcza polecenie login o następujących parametrach:

Listing 6. Logowanie do docker registry

```
Usage: docker login [OPTIONS] [SERVER]
Log in to a Docker registry.
Options:
  --help                  Print usage
  -p, --password string   Password
  -u, --username string   Username
```

Wystarczy zatem posiadać konto w docelowym rejestrze, aby publikować obrazy.

Mając tą wiedzę, możemy opublikować obraz naszej aplikacji. Docker Hub wymaga, aby dla obrazu utworzyć repozytorium.

W katalogu ze źródłami naszej aplikacji należy wykonać następujące polecenia:

Listing 7. Publikowanie obrazu dockerowego

```
docker login docker.io
docker build -t <LOGIN>/visitor .
docker push LOGIN/visitor
```

Do wykonania komend należy mieć uprawnienia super-użytkownika w systemach Linux/Mac. W powyższym listingu przyjęto, że login użytkownika w serwisie docker.com to <LOGIN>.

Pierwsza komenda jest jasna – otwieramy sesję użytkownika na serwerze rejestru; po wpisaniu komendy zostaniemy poproszeni o login i hasło.

Komenda build buduje obraz, który następnie metodą push wysyłamy do rejestru zdalnego.

To wszystko – nasz obraz jest od teraz dostępny pod adresem: <https://hub.docker.com/r/<LOGIN>/visitor/>. Na innych komputerach możemy pobrać go za pomocą docker pull <LOGIN>/visitor – ta komenda nie wymaga logowania do rejestru, ponieważ domyślnie obraz jest publicznie dostępny.

Uruchomienie aplikacji

Aplikacja jest już przygotowana do dystrybucji. Teraz gdy obraz aplikacji dostępny jest z centralnego repozytorium, możemy uniknąć budowania go na serwerze produkcyjnym. Dokonajmy zatem drobnej zmiany w pliku docker-compose:

Listing 8. Zaktualizowany docker-compose.yaml

```
version: "2"
services:
  redis:
    image: redis:3.0

  visitor:
    image: LOGIN/visitor
    ports:
      - 8888:8888
    links:
      - redis:redis
```

Jest to jedyny plik, którego potrzebujemy do wydania aplikacji. Teraz na dowolnej maszynie z zainstalowanym docker-engine oraz docker-compose możemy uruchomić naszą aplikację komendą docker-compose up -d.

Na tym etapie Docker zapewnia nam jednolity, prosty i szybki sposób na dystrybucję aplikacji. Pozostaje jednak kilka pytań, na które wciąż nie mamy odpowiedzi:

- » Jak uniknąć niedostępności aplikacji w przypadku awarii pojedynczej maszyny?
- » Jak skalować aplikację horyzontalnie?
- » Jak zapewnić w takim przypadku load balancing?

Problemy tej klasy pomagają nam rozwiązać Kubernetes oraz Openshift.

KUBERNETES I OPENSHIFT

Czym jest Kubernetes?

Kubernetes to manager klastra kontenerów dostarczany jako oprogramowanie open source. Dostarcza on platformę automatyzacji

wydawania i skalowania aplikacji oraz ułatwia zarządzanie nimi. Do uruchamiania kontenerów wykorzystuje technologię Docker.

Platforma Kubernetes dostarcza kilka podstawowych mechanizmów, takich jak:

- » Pod – elementarna jednostka wykonująca zadania na platformie. Reprezentuje ona jeden lub wiele kontenerów pracujących na tej samej maszynie. Podobnie jak kontener dockero-wy – pod otrzymuje własny adres IP oraz może deklarować wolumeny.
- » Serwis – jednostka automatyzacji zadań. Zrzesza jeden lub więcej podów realizujących dane zadanie, takie jak pojedyncza aplikacja czy baza danych. Serwis otrzymuje adres IP oraz nazwę domenową na platformie, a jego głównym zadaniem jest równomierne rozprowadzanie zadań pomiędzy podległe mu pody.
- » Controller – realizuje zadania administracyjne klastra. Rozróżniamy kilka rodzajów kontrolerów:
- » Replica Controller – dba o skalowanie i podtrzymywanie odpowiedniej ilości podów. Jeśli z powodu błędów logicznych lub awarii sprzętu część kontenerów zostanie wyłączena, Replica Controller zadba o utworzenie nowych podów.
- » DaemonSet Controller – odpowiada za uruchamianie co najmniej jednego podu na węzłach klastra.
- » Job Controller – dostarcza możliwości ustawiania zadań. Zapewnia, że do realizacji zadania zostanie przydzielona zadana ilość podów i wszystkie ukończą zadanie.

Czym jest Openshift?

Openshift jest platformą służącą do wydawania i zarządzania skonteneryzowanym oprogramowaniem. Stanowi ona rozszerzenie Kubernetesa. Oprogramowanie Openshift podzielone jest na cztery gałęzie:

- » Openshift Origin – główny projekt o otwartym kodzie źródłowym dostępnym na licencji Apache w wersji 2.0 pod adresem [6]. Jest rdzeniem wszystkich produktów Openshift.
- » Openshift Container Platform – platforma budowana przez RedHat w infrastrukturze klienta.
- » Openshift Dedicated – rozwiązanie budowane na własnej infrastrukturze RedHat dostępne w formie Platform as a Service.
- » Openshift Online – platforma dostępna dla pojedynczych developerów i zespołów. W momencie pisania artykułu platforma posiada kolejkę oczekujących na rejestrację, ponieważ działa w fazie developer preview.

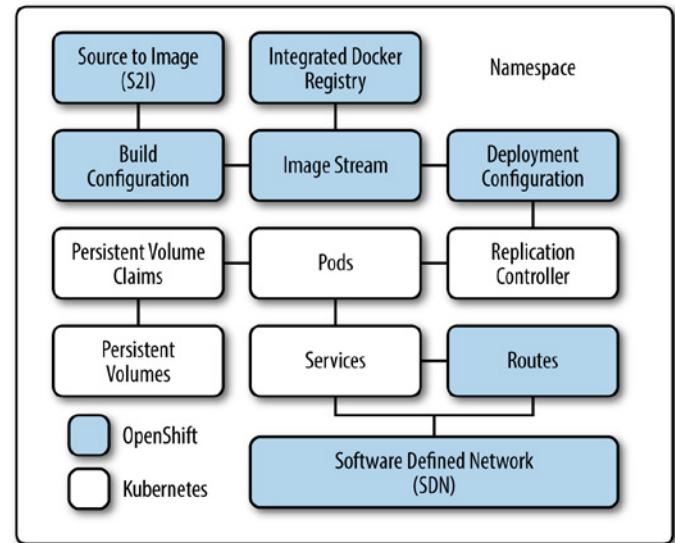
Openshift a Kubernetes

Zależność między Openshift a Kubernetes doskonale przedstawiła diagram (Rysunek 2) zaczerpnięty z książki *OpenShift for Developers* by Grant Shiplej and Graham Dumpleton (O'Reilly). Copyright 2016 RadHat, Inc., 978-1-491-94300-7:

Przewaga nad samodzielnym Dockerem

Platforma Openshift prezentuje swoje atuty w sytuacji, kiedy potrzebujemy wydać wiele aplikacji w jednolitej infrastrukturze. Dzięki dobrodziejstwom Kubernetesa pozwala na zachowanie wysokiej dostępności, sama natomiast wprowadza dodatkową warstwę konfiguracyjną dla sieci i automatyzacji wydań oprogramowania.

Dobrym przykładem przewagi platformy Openshift nad opisymi poprzednio technologiami jest sytuacja, kiedy dwa zespoły w



Rysunek 2. Diagram architektury Openshift

dziele potrzebują wydać aplikację internetową. Najczęściej aplikacja taka potrzebuje pracować na portach 80 oraz 443. Wiąże się to naturalnie z ekspozycją portów.

Rozważmy, jak powinno wyglądać wydanie dwóch takich aplikacji w kolejnych środowiskach:

- » Docker – konieczne jest uruchomienie dwóch maszyn wirtualnych, z których każda posiada docker-engine, własny adres IP lub domenę. Sytuacja staje się problematyczna, kiedy wydajemy kolejne aplikacje internetowe, ponieważ wzrasta ilość maszyn wirtualnych. Utylizacja staje się coraz mniej optymalna.
- » Docker Swarm – docker-engine w najnowszych wersjach dostarcza natywne wsparcie dla trybu klastra. Dzięki niemu możemy zbudować klasztro złożony z wielu maszyn wyposażonych w docker-engine. Niestety, jeden klasztro pozwala na zajęcie adresu przez jeden serwis, jest więc konieczne stworzenie własnego load-balancera, aby skalować ilość aplikacji na klastrze. Uruchomienie wielu klastrów replikuje problem opisany przy samym Dockerze.
- » Kubernetes – umożliwia uruchomienie wielu przestrzeni nazw, w każdej z nich możemy uruchomić usługę działającą na dowolnym porcie, co rozwiązuje problem występujący przy samym Dockerze. Pojawia się jednak problem natury administracyjnej – uprawnień użytkowników nie można dzielić pomiędzy przestrzeń nazw, więc potrzebny jest jeden odpowiedzialny zespół administratorów klastra.
- » Openshift – dzięki wbudowanemu mechanizmowi projektów każdy z zespołów developerskich posiada własny przydział zasobów wyrażony w ilości procesorów, pamięci oraz przydziałach dyskowych w postaci woluminów. Dodatkowo Openshift rozszerza Kubernetesa o mechanizm subdomen. Odpowiednio konfigurując wydanie, możemy automatycznie przydzielić subdomenę dla naszego oprogramowania, a zintegrowany load-balancer sam zadba o wysoką dostępność aplikacji.

URUCHAMIANIE APLIKACJI NA PLATFORMIE OPENSHIFT

Znając zalety klastrów konteneryzacyjnych, spróbujmy uruchomić przykładową aplikację w środowisku OpenShift.

Własna instalacja Openshift

Na potrzeby artykułu zastosujemy minimalną instalację Openshifta w postaci kontenera dockerowego. Posłuży nam do tego komenda:

Listing 9. Uruchomienie lokalnego serwera Openshift

```
sudo docker run -d --name "origin" \
--privileged --pid=host --net=host \
-v /:/rootfs:ro -v /var/run:/var/run:rw -v /sys:/sys -v /
sys/fs/cgroup:/sys/fs/cgroup:rw \
-v /var/lib/docker:/var/lib/docker:rw \
-v /var/lib/origin/openshift.local.volumes:/var/lib/origin/
openshift.local.volumes:rslave \
origin start --public-master=localhost
```

Dokładny opis instalacji dostępny jest w dokumentacji [7].

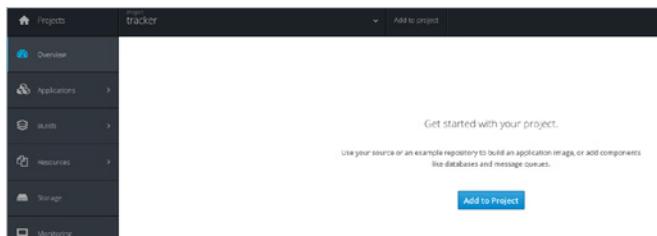
Uwaga: do prawidłowego działania lokalnej instancji Openshift konieczne jest dodanie przełącznika `--public-master=localhost`, jak przedstawiono w komendzie uruchamiania. Ta informacja nie została zawarta na stronie dokumentacji.

Po uruchomieniu Openshift logujemy się na <https://localhost:8443> na konto administratora z domyślnym hasłem admin.



Rysunek 3. Logowanie do Openshift

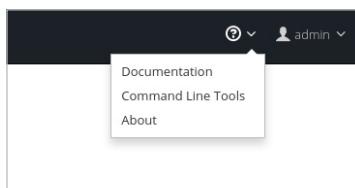
Następnie stworzymy nowy projekt o nazwie tracker:



Rysunek 4. Openshift – widok projektu

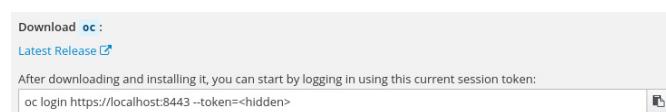
OC tool

W celu używania platformy Openshift z linii komend potrzebujemy narzędzia oc. Opis instalacji oraz dane logowania znajdziemy, klikając w znak zapytania w prawym górnym rogu interfejsu, a następnie wybierając *command line tools*.



Rysunek 5. Openshift – command line tools

W konsoli należy wykonać komendę logowania, którą znajdziemy na stronie:

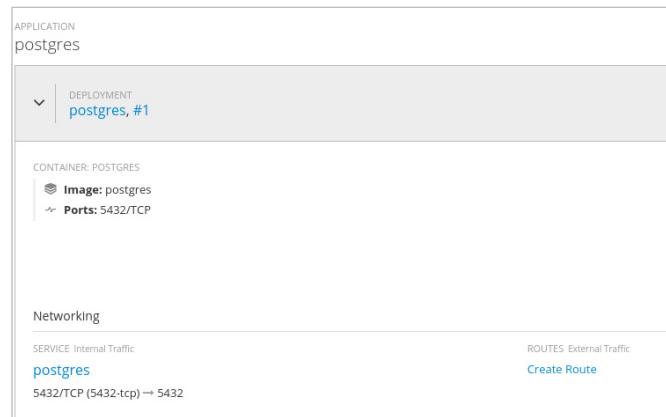


Rysunek 6. Openshift – command line tools

Jesteśmy gotowi do uruchamiania aplikacji na platformie Openshift, co sprowadza się do komendy:

```
oc new-app <nazwa-obrazu-dockerowego>
```

Uruchomienie aplikacji z obrazu postgres automatycznie spowoduje jego pojawienie się w interfejsie użytkownika:



Rysunek 7. Openshift – widok projektu

Szablon aplikacji wraz z zależnościami

Najwygodniejszym sposobem wydawania aplikacji jest przygotowanie opisu wydania podobnego do formatu docker-compose. Openshift dostarcza możliwości tworzenia szablonów aplikacji. Przygotujmy szablon dla naszej aplikacji:

Listing 10. Szablon aplikacji

```
apiVersion: v1
kind: Template
metadata:
  name: visitor-redis-template
  annotations:
    description: "Service logging visitor metadata"
    iconClass: "icon-go-gopher"
    tags: "visit,visitor"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master-pod
  labels:
    name: redis-master-pod
  spec:
    containers:
      image: redis
      name: redis-master-pod
      ports:
        - containerPort: 6379
          protocol: TCP
- kind: Service
  apiVersion: v1
  metadata:
    name: redis
  annotations:
```

```

description: "visitor redis service"
iconClass: "icon-go-gopher"
tags: "visit,visitor"
spec:
  selector:
    name: redis-master-pod
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
- apiVersion: v1
kind: Pod
metadata:
  name: visitor-master
  labels:
    name: visit-visitor-pod
spec:
  volumes:
    - name: 'visitor-data'
  containers:
    - image: LOGIN/visitor
      name: visitor
      ports:
        - containerPort: 8080
          protocol: TCP
- kind: Service
apiVersion: v1
metadata:
  name: visitor-service
annotations:
  description: "Visit visitor service logging visitor"
  metadata"
  iconClass: "icon-go-gopher"
  tags: "visit,visitor"
  "service.alpha.openshift.io/dependencies": "[{\\"name\\": \"redis\", \\"kind\\": \"Service\"}]"
spec:
  selector:
    name: visit-visitor-pod
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 8080

- apiVersion: v1
kind: Route
metadata:
  name: visitor-route
  annotations:
    description: "visitor"
    tags: "visitor"
  labels:
    app: visitor-template
spec:
  host: visitor-service.example.org
  to:
    kind: Service
    name: visitor-service
    weight: 100
  port:
    targetPort: http
  tls:
    termination: passthrough

```

Zapiszmy plik pod nazwą `visitor.yml`. Uruchomienie aplikacji wciąż sprowadza się do prostego polecenia `oc new-app visitor.yml`.

Szablon aplikacji sprowadza się do wykorzystania trzech rodzajów elementów:

- » Pod
- » Service
- » Route

Wszystkie te elementy zostały już omówione na początku artykułu. Plik opisu aplikacji pokazuje nam, w jaki sposób połączyć je ze sobą w działającą całość.

Pod

Listing 11. Fragment szablonu aplikacji opisujący Pod

```

- apiVersion: v1
kind: Pod
metadata:
  name: redis-master-pod
  labels:
    name: redis-master-pod
spec:
  containers:
    image: redis
    name: redis-master-pod
    ports:
      - containerPort: 6379
        protocol: TCP

```

Opis podu, podobnie jak większości elementów, składa się z dwóch części. Pierwsza to metadane, identyfikujące pod w obrębie dokumentu. Druga to specyfikacja dostarczająca informacji o sposobie tworzenia kontenerów dla podu.

Service

Listing 12. Fragment szablonu aplikacji opisujący Service

```

- kind: Service
apiVersion: v1
metadata:
  name: redis
  annotations:
    description: "visitor redis service"
    iconClass: "icon-go-gopher"
    tags: "visit,visitor"
spec:
  selector:
    name: redis-master-pod
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379

```

Serwisy w Openshift służą za load-balancery. W związku z tym poza sekcją opisu zawierają informację o portach, na których działa aplikacja, oraz selektor, czyli wskazanie, z jakimi podami współpracuje dany serwis.

Drugi serwis opisany w ramach pliku yaml zawiera specyficzne pole:

```
"service.alpha.openshift.io/dependencies": "[{\\"name\\": \"redis\", \\"kind\\": \"Service\"}]"
```

Należy pamiętać, że elementy typu service są dostępne na poziomie platformy Kubernetes, którą Openshift obudowuje. Pole `service.alpha.openshift.io/dependencies` jest specyficzne dla Openshift i pozwala na wprowadzenie zależności między serwisami. Jest to szczególnie przydatne, gdy usługi są od siebie zależne oraz istotna jest kolejność ich uruchamiania.

Route

Listing 13. Fragment szablonu aplikacji opisujący Route

```

- apiVersion: v1
kind: Route
metadata:
  name: visitor-route
  annotations:
    description: "visitor"
    tags: "visitor"
  labels:

```

```
app: visitor-template
spec:
  host: visitor-service.example.org
  to:
    kind: Service
    name: visitor-service
    weight: 100
  port:
    targetPort: http
  tls:
    termination: passthrough
```

Ostatnim znaczącym elementem występującym w opisie jest route. Jest to element zapewniający domenę dla naszej aplikacji. Nie jest to natywnie możliwe w pozostałych rozwiązaniach opisanych w tym artykule. Jeśli instancja Openshift ma podpiętą domenę, wewnętrzny serwer DNS jest w stanie dostarczać sub-domeny dla każdej aplikacji. Dzięki temu wiele aplikacji funkcjonujących na tym samym porcie (na przykład web-aplikacji zajmujących 80 i 443) może pracować w ramach jednego klastra Openshift. Warto również zwrócić uwagę na ustawienie termination w sekcji tls. Openshift dostarcza trzy sposoby zarządzania przekazywaniem ruchu szyfrowanego. Ustawienie passthrough informuje platformę, że wszystkie informacje na temat kluczy i certyfikatów posiada aplikacja. Więcej na ten temat dowiemy się w dokumentacji [9].

Jest to element zapewniający domenę dla naszej aplikacji.

URUCHAMIANIE APLIKACJI NA PLATFORMIE OPENSHIFT

Podczas prac integracyjnych nad środowiskiem Openshift napotkaliśmy kilka niestandardowych problemów. Pierwsze instalacje środowiska do celów testowych wykonaliśmy w środowisku Docker. Pomimo tego, że instalacja przebiegła pomyślnie, interfejs webowy nie funkcjonował prawidłowo.

Analiza zachowania interfejsu wykazała, że wykonywał on nieskuteczne próby połączenia do adresu karty sieciowej, która

nie miała zewnętrznego adresu IP. Rozwiązaniem tego problemu było dodanie do linii uruchomienia przełącznika --public-master=localhost. Przełącznik ten decyduje o adresie, do którego będzie łączył się interfejs użytkownika. Obejście tego problemu zostało uwzględnione wyżej w artykule na etapie uruchamiania lokalnej instalacji platformy Openshift.

Rozwiązań tego problemu nie poruszono (w chwili pisania artykułu) w dokumentacji, wymagało ono analizy kodu źródłowego platformy.

Kolejnym problemem, który wystąpił dopiero w momencie powstawania artykułu, było uruchomienie środowiska na najnowszych wersjach docker-engine. Ze względu na błąd #13219 opisany na GitHubie [8], Openshift nie mógł prawidłowo rozpoznać wersji Dockera. Wynika to z faktu zmiany sposobu numerowania wersji dockera. Rozwiązaniem problemu było uruchomienie platformy Openshift na silniku w starszej wersji. Błąd został już naprawiony i nowa wersja obrazu dockerowego Openshift powinna być wolna od usterek.

Ostatni niestandardowy problem, na który się natknęliśmy, to integracja Openshift z zewnętrznym rejestrrem obrazów dockerowych. W ramach istniejącej infrastruktury posiadałyśmy rejestr oparty o oprogramowanie Portus. Niestety integracja nie była możliwa ze względu na nierozwiążany problem po stronie omawianej platformy. Na portalu Github istnieje otwarte zadanie związane z tym zagadnieniem [10].

Jedynym rozwiązaniem tego problemu było wykorzystanie wewnętrznego rejestraru dockerowego dostarczanego wraz z Openshift.

Ten artykuł stanowi tylko wprowadzenie do bardzo obszernego tematu budowy i utrzymania platformy konteneryzacyjnej. Po więcej informacji na temat Openshift odsyłam do darmowych książek udostępnionych przez RedHat na stronie openshift.com: *OpenShift for Developers* [11] *DevOps with OpenShift* [12] *Microservices vs. Service-Oriented Architecture* [13].

W sieci

- ▶ [1] <http://www.tornadoweb.org/en/stable/webframework.html>
- ▶ [2] <https://docs.docker.com/compose/compose-file/compose-file-v2/>
- ▶ [3] https://hub.docker.com/_/registry/
- ▶ [4] <https://github.com/SUSE/Portus>
- ▶ [5] <https://hub.docker.com/>
- ▶ [6] <https://github.com/openshift/origin>
- ▶ [7] https://docs.openshift.org/latest/getting_started/administrators.html#running-in-a-docker-container
- ▶ [8] <https://github.com/openshift/origin/issues/13219>
- ▶ [9] https://docs.openshift.org/latest/architecture/core_concepts/routes.html#passthrough-termination
- ▶ [10] <https://github.com/openshift/origin/issues/9584#issuecomment-282276919>
- ▶ [11] <https://www.openshift.com/promotions/for-developers.html>
- ▶ [12] <https://www.openshift.com/promotions/devops-with-openshift.html>
- ▶ [13] <https://www.openshift.com/promotions/microservices.html>



MICHAŁ PAWLICKI

Architekt oprogramowania w firmie Nokia, entuzjasta programowania funkcyjnego i języka Scala. Absolwent informatyki Politechniki Wrocławskiej.

NOKIA

Are you
proficient in IT?

So are we.

Join us.

Check what we offer:

<http://nokiawroclaw.pl/are-you-proficient-in-IT/>



Chmura obliczeniowa w Pythonie

W pierwszej części artykułu (Programista 4/2017) mogliśmy zaobserwować, jak Python sprawdza się w przetwarzaniu języka naturalnego i uczeniu maszynowym (ang. machine learning). Przekonaliśmy się, że Python świetnie sprawdza się w analizie i przetwarzaniu danych ze względu na prostotę języka i liczne dojrzałe biblioteki i narzędzia. Przy okazji „odczarowaliśmy” uczenie maszynowe i poznaliśmy podstawowe koncepcje i pomysły stojące za tą dziedziną sztucznej inteligencji.

Dzisiaj przyjrzymy się temu, jak Python skaluje się i co ma do zaoferowania w przetwarzaniu dużej ilości danych. Korzystając z darmowych instancji Amazon EC2 i biblioteki dask.distributed, stworzymy chmurę obliczeniową składającą się z dziewięciu węzłów. Przeanalizujemy wszystkie zdarzenia, jakie miały miejsce na GitHubie w ciągu jednego tygodnia. Wystarczy nam tylko 80 sekund, aby nasz klaster pobrał i przeanalizował 20 GB danych!

JAK DZIAŁA CHMURA OBLCZENIOWA?

dask.distributed jest lekką biblioteką Pythona do obliczeń rozproszonych. Pozwala na łatwe stworzenie chmury obliczeniowej, zwanej klastrem (ang. cluster). Klaster składa się z pewnej ilości maszyn, nazywanych inaczej węzłami. dask.distributed skaluje się do kilku tysięcy maszyn.

Na jednej z maszyn uruchomiony jest proces schedulera, który rozdziela zadania pomiędzy pozostałe maszyny i zarządza całą chmurą. Natomiast na pozostałych maszynach uruchomione są procesy wykonujące obliczenia (ang. workers). Oprócz tego potrzebujemy jeszcze lokalnej maszyny, z której będziemy wysyłać do schedulera obliczenia. Może to być np. Twój laptop.

KONFIGURACJA LOKALNEJ MASZYNY

Będziemy pracować na Pythonie 3.5. Python 3 jest domyślnie zainstalowany na Linuxie oraz Macu. Pod Windowsem konieczna jest ręczna instalacja. Instalator można pobrać ze strony: <https://www.python.org/downloads/>.

Potrzebujemy doinstalować kilka bibliotek, które nie są częścią standardowej biblioteki Pythona. W tym celu wykonujemy poniższe polecenie z uprawnieniami administratora:

```
pip3 install jupyter dask dask-ec2 distributed
```

Wszystkie fragmenty kodu w Pythonie będziemy wykonywać w Jupyter Notebooku. Jest to interaktywna konsola (REPL) działająca w przeglądarce. Ponadto posiada dużo dodatkowych funkcjonalności, które przydadzą nam się później. Aby uruchomić Jupyter Notebooka, wykonujemy poniższe polecenie:

```
jupyter notebook
```

Po kilku sekundach powinniśmy zobaczyć w naszej przeglądarce ekran startowy:



Rysunek 1. Menedżer plików w Jupyter Notebook

Aby stworzyć nowy notebook, klikamy po prawej stronie New i wybieramy Python 3.

W nowym notebooku importujemy wszystkie moduły i pakiety, których będziemy potrzebować. Kod wykonujemy, używając kombinacji klawiszy Alt + Enter.

```
# moduły ze standardowej biblioteki Pythona
import sys, subprocess, io, json, urllib, gzip

# niestandardowe moduły
from dask.distributed import Client
import dask.bag, dask_ec2.salt
```

Powyższy kod powinien wykonać się bez żadnego błędu.

CHMURA NA JEDNEJ MASZYNIE?

Scheduler i worker to tylko procesy i nic nie stoi na przeszkodzie, aby uruchomić je na jednym węźle, np. na Twoim laptopie. Oczywiście, obliczenia wysypane do takiej lokalnej chmury nie będą wykonywały się szybciej, ale dzięki temu nie potrzebujemy prawdziwego klastra, aby przetestować aplikację przystosowaną do działania w chmurze. Poza tym konfiguracja jest wówczas dziecinnie prosta:

```
client = Client()
client
```

Client	Cluster
<ul style="list-style-type: none"> Scheduler: tcp://127.0.0.1:46655 	<ul style="list-style-type: none"> Workers: 4 Cores: 4 Memory: 2.42 GB

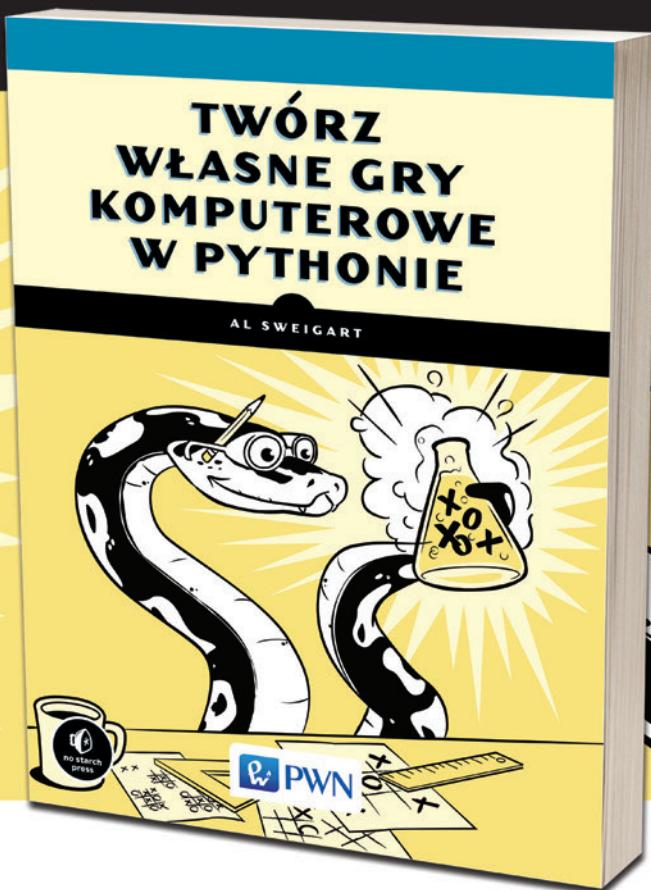
Rysunek 2. Lokalna chmura obliczeniowa

Zlikwidowanie lokalnej chmury jest również banalnie proste:

```
client.shutdown()
```

Spróbujmy teraz stworzyć chmurę, korzystając z Amazon Web Service.

WYKORZYSTAJ ogromne możliwości PYTHONA i napisz swoją oryginalną grę!



 PWN

Odwiedź nas na:
 IT.PWN.PL

Książka dostępna na: www.ksiegarnia.pwn.pl

PATRONI MEDIALNI:

























TWORZENIE CHMURY NA AMAZON EC2

Amazon Web Service to platforma do tworzenia chmur obliczeniowych na żądanie. Najważniejszą usługą jest Amazon Elastic Compute Cloud (EC2), czyli wirtualne maszyny, nazywaneinstancjami. Uruchomimy dziewięć darmowych węzłów, na których będą działać scheduler i workery.

Założenie konta na Amazon Web Services

Potrzebujemy założyć nowe konto na Amazon Web Services. Procedura jest prosta, ale składa się z wielu kroków i zajmuje chwilę czasu. Wykonaj następujące kroki:

1. Upewnij się, że masz przy sobie swój telefon oraz ważną kartę kredytową lub debetową.
2. Wejdź na stronę <https://aws.amazon.com/>
3. Kliknij w prawym górnym rogu żółty przycisk „create an AWS Account”.
4. W polu „e-mail or mobile number” wpisz swój adres mailowy.
5. Poniżej tego pola wybierz opcję „I am a new user”.

E-mail or mobile number:
jan@kowalski.com

I am a new user.

I am a returning user and my password is:
[redacted]

Sign in using our secure server

Rysunek 3. Zakładanie konta na Amazon Web Services

6. Kliknij „Sign in using our secure server”.
7. Wypełnij kolejny formularz i kliknij „create account”:

My name is: Jan Kowalski

My e-mail address is: jan@kowalski.com

Type it again: jan@kowalski.com

note: this is the e-mail address that we will use to contact you about your account

Enter a new password: [redacted]

Type it again: [redacted]

Create account

Rysunek 4. Zakładanie konta na Amazon Web Services c.d

8. Teraz czeka Cię najbardziej czasochłonna część procedury. Wypełnij pięć kolejnych formularzy:
 - » Contact Information (informacje kontaktowe)
 - » Payment Information (dane Twojej karty kredytowej lub debetowej)
 - » Identity Verification (weryfikacja tożsamości)
 - » Support Plan (wsparcie)
 - » Confirmation (potwierdzenie)

Amazon będzie chciał zweryfikować Twoją kartę poprzez zablokowanie na niej niewielkiej ilości środków (np. jedno euro). Udana

weryfikacja jest konieczna do tego, aby korzystać z Amazon EC2. Dlatego upewnij się, że dysponujesz środkami na swojej karcie i że możesz nią płacić w Internecie.

Podczas weryfikacji tożsamości na Twój telefon zostanie wysłany SMS z kodem PIN, który należy wprowadzić w formularzu. Z kolei w oknie „Support Plan” trzeba wybrać opcję „Basic (Free)”.

Uwierzytelnienie

Abyśmy mogli zdalnie tworzyć instancje Amazon EC2 i zarządzać nimi, potrzebujemy uwierzytelić się.

Utwórz plik `~/.aws/credentials` i umieść w nim swoje dane do logowania:

```
[default]
aws_access_key_id = TWOJ_KLUCZ_DOSTEPU
aws_secret_access_key = TWOJ_KLUCZ
```

Obie wartości możesz pobrać w następujący sposób:

1. Kliknij w menu nawigacyjnym swoje imię i nazwisko (na prawo od symbolu dzwonu).
2. Z menu, które się pojawiło, wybierz „My Security Credentials”.
3. W oknie modalnym, które się pojawiło, wybierz „Continue to Security Credentials”.
4. Rozwiń „Access Keys (Access Key ID and Secret Access Key)”.
5. Kliknij „Create New Access Key”.
6. Kliknij „Show Access Key”.
7. Skopiuj wartości do pliku `~/.aws/credentials`.

Ponadto musimy wygenerować klucz kryptograficzny. W tym celu:

1. Wejdź pod adres <https://aws.amazon.com/>.
2. Na górnym panelu nawigacyjnym kliknij „Services” i wybierz „EC2”.
3. Z panelu nawigacyjnego na górze, po prawej stronie wybierz region „EU (Frankfurt)”. Właśnie w tym regionie zostaną uruchomione nasze instancje.
4. Po lewej wybierz „Key Pairs” z grupy „Network & Security”.
5. Kliknij „Create Key Pair”.
6. W oknie modalnym wprowadź nazwę klucza: primary.
7. Kliknij „Create”. W tym momencie automatycznie powinien zostać pobrany plik „primary.pem”.
8. Przenieś ten plik do katalogu `~/.ssh/`.
9. Pod Linuxem i MacOS zmień uprawnienia do pliku: `chmod 400 ~/.ssh/primary.pem`.

Tworzenie instancji Amazon EC2

W tym momencie jesteśmy gotowi do utworzenia instancji EC2, na których uruchomimy procesy schedulera i workery. Na szczęście ten krok jest stosunkowo prosty dzięki narzędziu dask-ec2, które automatyzuje cały proces konfiguracji chmury. dask-ec2 instaluje na każdym węźle Anacondę, czyli dystrybucję Pythona z preinstalowanymi wieloma przydatnymi bibliotekami i narzędziami do przetwarzania danych.

Niestety, jest to dosyć niedojrzałe narzędzie i posiada kilka błędów. Musimy ręcznie zmodyfikować plik `dask_ec2/salt.py`. Ścieżkę do tego pliku łatwo znajdziemy, dokonując introspekcji:

```
dask_ec2.salt.__file__
'/usr/local/lib/python3.5/dist-packages/dask_ec2/salt.py'
```

W tym pliku w metodzie `aggregate_by` należy zmniejszyć wcięcie o cztery spacje linii `return ret` (tuż przed `except TypeError`). Poziom wcięcia powinien być ten sam co pętli `for`.

Aby skonfigurować chmurę, wystarczy wykonać poniższe polecenie i poczekać około 15 minut (niezależnie od liczby węzłów):

```
dask-ec2 up \
--keyname primary \
--keypair ~/ssh/primary.pem \
--region-name eu-central-1 \
--type t2.micro \
--volume-size 8 \
--count 9 \
--ami ami-ac1524b1 \
--tags KEY:VALUE
```

- » Parametr `type` określa, na jakim sprzęcie zostaną uruchomione węzły. `t2.micro` to jedyna darmowa konfiguracja sprzętowa. Każda taka instancja posiada 1 GB RAMu i jeden procesor jednordzeniowy. W ramach Amazon Free Tier możemy przez pierwsze 12 miesięcy wykorzystać do 750 godzin pracy tego rodzaju instancji. Odpowiada to mniej więcej jednej instancji działającej przez cały miesiąc lub dziesięciuinstancjom działającym przez 3 dni. Po przekroczeniu tego limitu Amazon zacznie naliczać opłaty i obciąży naszą kartę.
- » Parametr `count` określa, z ilu węzłów ma składać się nasza chmura. Amazon posiada ograniczenie liczby instancji, które możemy odpalić jednocześnie. W przypadku instancji `t2.micro` limit wynosi 10 maszyn.
- » `ami` określa Amazon Machine Image, czyli system operacyjny. W naszym przypadku to Ubuntu Trusty 14.04 Server 64bit.
- » `tags` jest potrzebne ze względu na błąd w `dask-ec2` i służy tagowaniu instancji, czyli przypisaniu im metadanych.

Po wykonaniu tej komendy zostaną wyświetlane instrukcje, jak połączyć się z chmurą. Najważniejsze, aby zapisać adres IP i port, na którym działa scheduler.

Łączenie z chmurą na Amazon EC2

Wreszcie możemy połączyć się z naszym klastrem!

```
client = Client('52.59.222.178:8786')
client
```

Client	Cluster
• Scheduler: tcp://52.59.222.178:8786	• Workers: 8
• Dashboard: http://52.59.222.178:8787	• Cores: 8
	• Memory: 5.00 GB

Rysunek 5. Chmura obliczeniowa na Amazon EC2

ZARZĄDZANIE CHMURĄ

Wykonanie kodu na każdym węźle

Sprawdźmy, czy chmura działa. W tym celu wyświetlimy wersję Pythona uruchomioną na poszczególnych węzłach (z wyjątkiem schedulera).

```
def check_version():
    return sys.version
client.run(check_version)
```

```
{'tcp://172.31.17.79:44269': '3.5.2 |Continuum Analytics, Inc.| (default, Jul 2 2016, 17:53:06) \n[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]',
'tcp://172.31.19.209:52009': '3.5.2 |Continuum Analytics, Inc.| (default, Jul 2 2016, 17:53:06) \n[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]',
...
}
```

Restartowanie chmury

Czasami zdarza się, że chmura przestaje wykonywać nowe obliczenia i nie odpowiada. Tak dzieje się, gdy wyślemy do chmury czasochłonne obliczenia, które wykonują się zbyt długo i blokują nowe zadania. Jeżeli chmura przestanie odpowiadać, możemy ją zrestartować za pomocą poniższego polecenia:

```
client.restart()
```

WCZYTANIE DANYCH

Format danych

Na stronie <https://www.githubarchive.org/> dostępne są pliki z danymi o wszystkich zdarzeniach, jakie miały miejsce na GitHubie. Rejestrowane są takie zdarzenia, jak dodanie komentarza, push, release, stworzenie fork'a itd. Pełna lista zdarzeń dostępna jest pod adresem <https://developer.github.com/v3/activity/events/types/>.

Każde zdarzenie jest zapisane w JSONie. Wszystkie zdarzenia z danego dnia i z danej godziny można pobrać z adresu <http://data.githubarchive.org/YYYY-MM-DD-HH.json.gz>, przy czym YYYY-MM-DD oznacza datę (miesiąc i dzień są zawsze dwucyfrowe), natomiast HH to liczba od 0 do 23 reprezentująca godzinę (jedna lub dwie cyfry). Pod tymi adresami dostępne są skompresowane pliki, których każdy wiersz to jedno zdarzenie zakodowane w JSONie. Każdy z tych skompresowanych plików waży około 10 MB.

Wygenerowanie adresów URL

Na początku wygenerujemy listę wszystkich adresów URL plików, które chcemy pobrać. Interesuje nas zakres dat od 5 do 11 czerwca. W tym celu skorzystamy z wyrażeń listowych (ang. *list comprehension*).

W poniższym kodzie `range(x, y)` zwraca wszystkie liczby całkowite od `x` do `y`, bez `y`. Na przykład `range(6, 10) == [6, 7, 8, 9]`. Używamy także wbudowanej metody `format`, która zastąpi `{:02d}` numerem dnia miesiąca (np. 05, 08 lub 11), natomiast `{}` zostanie zastąpione godziną (np. 0, 7 lub 23).

```
urls = [
    'http://data.githubarchive.org/2017-06-{:02d}-{}.{json.gz}'.format(
        day, hour)
    for day in range(5, 12)
    for hour in range(0, 24)
]
```

`urls` stanowi listę, która jest przechowywana na lokalnej maszynie. Teraz musimy wrzucić ją w chmurę, aby była ona dostępna na poszczególnych węzłach. Na szczęście z pomocą przychodzi nam biblioteka `dask`, która udostępnia `dask.bag`, czyli rozproszoną listę.

Każdy `dask.bag` składa się z partycji (ang. *partition*). Każda partycja znajduje się w całości na jednym węźle, ale poszczególne partycje mogą znajdować się na różnych węzłach. W ten sposób

możemy kontrolować, w jak duże porcje danych będą przetwarzać węzły. Parametr `partition_size` określa, ile elementów znajdzie się w każdej partycji. Ustawimy tę wartość na jeden, ponieważ przetworzenie jednego pliku naraz jest wystarczająco dużym zadaniem.

```
urls_bag = dask.bag.from_sequence(urls, partition_size=1)
```

Aby przesyłać dane w drugą stronę, czyli z chmury na lokalną maszynę, wywołujemy funkcję `compute`. Sprawdźmy, czy w `urls_bag` mamy poprawne adresy URL:

```
urls_bag.compute()[22:27]
```

```
['http://data.githubarchive.org/2017-06-05-22.json.gz',
 'http://data.githubarchive.org/2017-06-05-23.json.gz',
 'http://data.githubarchive.org/2017-06-06-0.json.gz',
 'http://data.githubarchive.org/2017-06-06-1.json.gz',
 'http://data.githubarchive.org/2017-06-06-2.json.gz']
```

Pobranie i wczytanie danych

Dla każdego adresu musimy wykonać pewien ciąg operacji:

1. Po pierwsze, musimy wczytać skompresowany plik. Do tego posłuży nam funkcja `urllib.request.urlopen` ze standardowej biblioteki Pythona. Zwraca ona strumień do pliku dostępnego pod tym adresem.
2. Następnie musimy zdekompresować plik. Ponownie użyjemy tutaj narzędzi ze standardowej biblioteki, tym razem `gzip.GzipFile`. Każdy tego rodzaju obiekt to, w uproszczeniu, lista wierszy.
3. Na tym etapie dysponujemy listą plików, z których każdy reprezentowany jest jako lista wierszy. To oznacza, że dysponujemy listą list wierszy, zamiast po prostu płaską listą wierszy. Używamy funkcji `flatten` do dokonania tej transformacji.
4. Cały czas operujemy na poziomie bajtów, a nie unicodu. Musimy zdekodować każdą z linii za pomocą `s.decode()`.
5. Na koniec deserializujemy JSON, aby otrzymać Pythonowy słownik. Używamy wbudowanej biblioteki `json`.

Zauważmy, że na każdym etapie korzystaliśmy tylko i wyłącznie ze standardowej biblioteki Pythona.

Ten ciąg operacji reprezentujemy w następujący sposób:

```
records = urls_bag \
    .map(urllib.request.urlopen) \
    .map(lambda s: gzip.GzipFile(fileobj=s)) \
    .flatten() \
    .map(lambda s: s.decode()) \
    .map(json.loads)
```

Na razie nie są wykonywane żadne obliczenia. Jedynie scheduler buduje drzewo operacji, które mają zostać wykonane. Obliczenia zostaną wykonane dopiero gdy wywołamy metodę `.compute()`.

PRZETWARZANIE DANYCH

Liczba rekordów

Policzmy, ile zdarzeń wydarzyło się w ciągu jednego tygodnia:

```
%time records.count().compute()

CPU times: user 428 ms, sys: 20 ms, total: 448 ms
Wall time: 1min 17s
7602498
```

Ponad 7.5 miliona zdarzeń!

Komenda `%time` służy do wykonania jednej linii Pythona i zmierzenia czasu, który zajmuje jej wykonanie. Jest to funkcjonalność Jupyter Notebooka, a nie poprawna składnia Pythona. Widzimy, że przetworzenie 7.5 miliona rekordów zajęło nam 77 sekund.

Ignorujemy „CPU times”, ponieważ reprezentują one zasoby zużyte na lokalnej maszynie, a nie na węzłach chmury.

Rozmiar danych skompresowanych

Sprawdźmy, jaka ilość danych jest pobierana przez klastera:

```
GB = 1024 * 1024 * 1024

%time urls_bag \
    .map(urllib.request.urlopen) \
    .map(lambda s: len(s.read())) \
    .sum() \
    .compute() / GB

Wall time: 14.3 s
2.6661874800920486
```

Jest to około 2.7 GB. Samo pobranie wszystkich danych zajmuje tylko niecałe 15 sekund. To wskazówka dla nas, że przepustowość połączenia internetowego jest wystarczająco duża i że wąskim gardłem jest procesor, który musi zdekompresować i zdeserializować dane. To oznacza, że nie osiągniemy znaczającej poprawy czasu wykonania poprzez pobranie i zapisanie danych. Dlatego będziemy pobierać dane przy każdych obliczeniach od nowa.

Rozmiar danych nieskompresowanych

Co się stanie, jeśli dodatkowo zdekompresujemy pliki?

```
%time urls_bag \
    .map(urllib.request.urlopen) \
    .map(lambda s: gzip.GzipFile(fileobj=s)) \
    .map(lambda s: len(s.read())) \
    .sum() \
    .compute() / GB

Wall time: 22.9 s
19.673632417805493
```

Widzimy, że czas wykonywania wzrósł z niecałych 15 sekund do prawie 23 sekund. To oznacza, że na razie wąskim gardłem jest deserializacja.

Ponadto widzimy już, że mamy do czynienia z prawie 20 GB danych w JSONie! Sprawdźmy teraz, jak wygląda pojedynczy rekord:

Przykładowy rekord

```
%time records.take(1)[0]

{
  "actor": {
    "avatar_url": "https://avatars.githubusercontent.com/u/20188899?",
    "display_login": "jviriato",
    "gravatar_id": "",
    "id": 20188899,
    "login": "jviriato",
    "url": "https://api.github.com/users/jviriato"
  },
  "created_at": "2017-06-05T00:00:00Z",
  "id": "5999726536",
  "payload": {
    "before": "ae90a356c5eea74763e3d8b749910cc5b7ee2e08",
```

```

"commits": [
  {
    "author": {
      "email": "2d773a7c58a84cdc5f6f61665ae9d292fb2e0870@inf.ufsm.br",
      "name": "Jos\u00e3o Victor Viraio"
    },
    "distinct": true,
    "message": "t7",
    "sha": "d3210f4a7d00439e0034320d901bfe1b258c61e2",
    "url": "https://api.github.com/repos/jviriato/elc117/commits/d3210f4a7d00439e0034320d901bfe1b258c61e2"
  },
  {
    "author": {
      "email": "2d773a7c58a84cdc5f6f61665ae9d292fb2e0870@inf.ufsm.br",
      "name": "Jos\u00e3o Victor Viraio"
    },
    "distinct": true,
    "message": "Merge branch 'master' of https://github.com/jviriato/elc117",
    "sha": "d0e376abf0e3744f5d9fb2e61cb635da9f3143e4",
    "url": "https://api.github.com/repos/jviriato/elc117/commits/d0e376abf0e3744f5d9fb2e61cb635da9f3143e4"
  }
],
"distinct_size": 2,
"head": "d0e376abf0e3744f5d9fb2e61cb635da9f3143e4",
"push_id": 1779765709,
"ref": "refs/heads/master",
"size": 2
},
"public": true,
"repo": {
  "id": 84136609,
  "name": "jviriato/elc117",
  "url": "https://api.github.com/repos/jviriato/elc117"
},
"type": "PushEvent"
}
]
Wall time: 102 ms

```

Metoda `take(n)` powoduje przesłanie danych z workerów do Twojego laptopa i dlatego wymusza uruchomienie obliczeń, podobnie jak `compute()`.

Każdy rekord to obiekt JSON. Najważniejsze pole każdego zdarzenia to `type` określający jego typ. W przypadku tego rekordu zdarzenie jest typu `PushEvent`. Rzut oka na wartość pola `created_at` pozwala zweryfikować, że zdarzenia są przetwarzane w

odpowiedniej kolejności, to znaczy od najwcześniejszych. Niezależnie od rodzaju zdarzenia każdy rekord posiada pole `repo.name` z nazwą repozytorium, którego zdarzenie dotyczy.

Agregacja danych

Wykonajmy teraz bardziej zaawansowane obliczenia, wykorzystujące wszystkie rekordy. Ile jest zdarzeń poszczególnego rodzaju?

```
%time records.pluck('type').frequencies().compute()

CPU times: user 752 ms, sys: 20 ms, total: 772 ms
Wall time: 1min 12s
[('IssuesEvent', 275786),
 ('PullRequestReviewCommentEvent', 152637),
 ('PullRequestEvent', 421264),
 ('MemberEvent', 35294),
 ('PushEvent', 4025987),
 ('PublicEvent', 7717),
 ('WatchEvent', 598653),
 ('DeleteEvent', 176562),
 ('IssueCommentEvent', 549787),
 ('GollumEvent', 46882),
 ('CreateEvent', 1049655),
 ('ForkEvent', 216520),
 ('CommitCommentEvent', 16675),
 ('ReleaseEvent', 29079)]
```

Najczęściej wydawane projekty

Sprawdźmy, które projekty mają najwięcej wydań (ang. *release*). Najpierw użyjemy `filter` i wybierzemy tylko zdarzenia odpowiedniego typu:

```
releases = records \
  .filter(lambda r: r['type'] == 'ReleaseEvent') \
  .persist()
```

Użyliśmy metody `persist()`, która wymusiła rozpoczęcie wykonywania obliczeń i zapamiętanie rezultatu na workerach. Możemy sobie pozwolić na zapamiętanie wszystkich zdarzeń typu `ReleaseEvent`, ponieważ jest ich tylko 29079 i taka ilość rekordów zmieści się w pamięci RAM w chmurze.

Zapamiętanie danych oznacza, że wykonując dalsze obliczenia bazujące na zmiennej `releases`, pomijamy cały narzut związany z pobraniem, dekompresją i deserializacją danych.

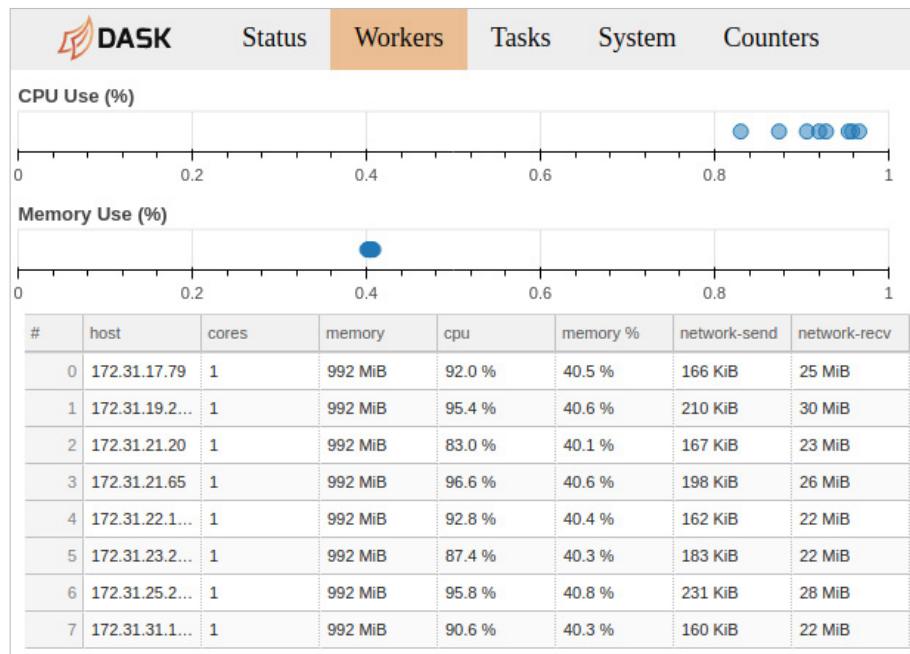
reklama

Szkolenie dla Ciebie lub Twojego zespołu

Warsztat analityka danych w języku Python

Skorzystaj z 10% zniżki na wszystkie szkolenie otwarte z autorskiej oferty Sages ważnej przy zamówieniach złożonych do końca września 2017 r. Hasło: PROGRAMISTAMAG





Rysunek 6. Panel kontrolny służący do debugowania chmury

```
%time releases \
    .pluck('repo') \
    .pluck('name') \
    .frequencies() \
    .topk(10, lambda r: r[1]) \
    .compute()
CPU times: user 172 ms, sys: 4 ms, total: 176 ms
Wall time: 766 ms
[('design4pro/release-me', 1860),
 ('oscar-brito-github/chat-client', 67),
 ('triplea-game/triplea', 49),
 ('blackducksoftware/test-bds-repo2', 44),
 ('github-release-bot/github-release-test-py3', 41),
 ('github-release-bot/github-release-test-py2', 40),
 ('nicholasdille/PowerShell-Statistics', 38),
 ('OpenGreekAndLatin/First1KGreek', 37),
 ('ColinDuquesnoy/MellowPlayer', 37),
 ('galexrt/srcds_exporter', 35)]
```

Dzięki użyciu `persist()` wykonanie tych operacji zajęło mniej niż sekundę!

DEBUGOWANIE CHMURY

Scheduler regularnie odpytuje workery o ich zajętość, obłożenie obliczeniami i inne informacje. Są one następnie publikowane na panelu kontrolnym dostępnym pod adresem:

» <http://52.59.222.178:8787/workers>.

Pamiętaj, aby zmienić adres IP na adres Twojego schedulera. Adres IP Twojego schedulera znajdziesz na końcu logów polecenia `dask-ec2 up`.

ZAMYKANIE CHMURY

Jeżeli przekroczymy darmowe 750 godzin pracy wszystkich instancji, Amazon zacznie naliczać opłaty. Dlatego na sam koniec zamykamy chmurę i usuwamy wszystkie instancje Amazon EC2:

`dask-ec2 destroy`

Powyższe polecenie uruchamiamy z tego samego katalogu, z którego uruchomiliśmy `dask-ec2 up`. Jest to istotne, ponieważ w tym katalogu znajduje się plik `cluster.yaml` z danymi o węzłach klastra.

PODSUMOWANIE

Zobaczyliśmy, jak w Pythonie można stworzyć chmurę obliczeniową z użyciem `distributed.dask`. Przekonaliśmy się, że skonfigurowanie klastra jest stosunkowo proste. Jednocześnie rozproszone struktury danych (takie jak `dask.bag`) powodują, że możemy piisać kod bardzo podobny do kodu działającego na jednej maszynie. Wszystko to powoduje, że Python świetnie sprawdza się w przetwarzaniu i analizie danych.

KRZYSZTOF MĘDRELA

Programista i konsultant specjalizujący się przede wszystkim w aplikacjach webowych tworzonych w Django (Python). Trener w firmie Bottega IT Minds i InfoTraining. Działa w lokalnej społeczności Pythona w Krakowie (Pykonik). Prowadzi bloga pod adresem <http://medrela.com>.

W sieci

- ▶ Instalator Pythona: <https://www.python.org/downloads/>
- ▶ Amazon Web Services: <https://aws.amazon.com/>
- ▶ Pomoc dotycząca zakładania konta na AWS i konfiguracji Amazon EC2: <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html#sign-up-for-aws>
- ▶ Historia zdarzeń na GitHubie: <https://www.githubarchive.org/>
- ▶ Lista wszystkich typów zdarzeń na `githubarchives.org`: <https://developer.github.com/v3/activity/events/types/>



UMBRACO POLAND FESTIVAL | 2017

15 WRZESIEŃ 2017 KRAKÓW



JEDYNE TAKIE WYDARZENIE W POLSCE!

NASI PIERWSI PRELEGENCI



NIELS HARTVIG

twórca Umbraco, zaprezentuje nam swój festiwalowy keynote



SØREN SPELLING LUND

Przyspiesz z Umbraco



ONDREJ PIALEK

Wdrożenie skutecznego testu A/B w
Umbraco z uSplit



WOJCIECH TENGLER

Memcached - alternatywny cache w
Umbraco Brief Description

... I INNI, KTÓRZY BĘDĄ OGŁOSZENI NIEDŁUGO! KUP BILET I BĄDŹ Z NAMI PODCZAS DRUGIEJ EDYCJI UMBRACO
POLAND FESTIVAL!

umbracofestival.pl

Generowanie danych za pomocą języka SQL

Prezentacja danych na gridzie to badanie przeprowadzane w celu określenia stopnia użyteczności serwisu lub aplikacji. W badaniu biorą udział obecni lub potencjalni ich użytkownicy. Wyniki badań, zarówno jakościowe, jak i ilościowe, pozwalają zidentyfikować te elementy, które wpływają pozytywnie na użyteczność i satysfakcję użytkownika, a także takie, które należy zmodyfikować. Użyteczność to nie tylko sposób, w jaki można poruszać się w aplikacji, ale również wygląd danych, sposób ich stronicowania, sposób przewijania i czytelność.

PREZENTACJA DANYCH

Ważną częścią użyteczności systemu informatycznego (aplikacja stacjonarna, aplikacja mobilna czy webowa) jest sama prezentacja danych, na co wpływa m.in.:

- » Rodzaj i wielkość czcionki,
- » Wyrównanie tekstu w polach w tabeli,
- » Kolorystyka strony,
- » Ilość danych.

W artykule skupimy się na przedstawieniu sposobu generowania danych testowych w bazie danych, które mogą być prezentowane na gridach w oprogramowaniu, które poddane zostanie testom akceptacyjnym. Nie jest możliwe stwierdzenie, czy prezentacja danych jest zadawalająca, jeśli ich nie ma w aplikacji, dlatego warto mieć przygotowane pewne strategie testowe lub generatory danych, które w szybki sposób wypełnią bazę danych, a przez to wypełnią również listy na formularzach w aplikacji. W tym artykule skupimy się na zaprojektowaniu generatora danych w języku T-SQL z poziomu bazy danych.

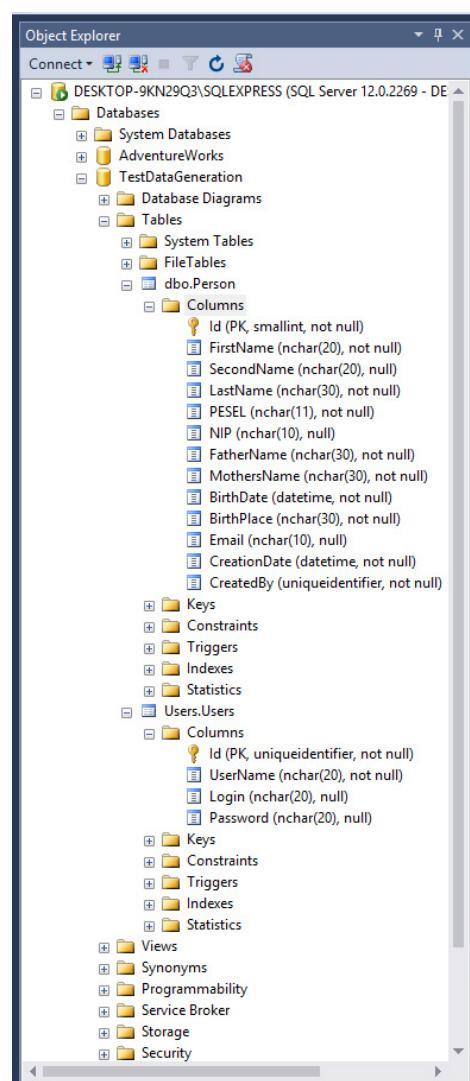
WYPEŁNIANIE TABEL W BAZIE DANYCH

W celu prezentacji generowania strategii wypełniania tabel została wykorzystana baza danych wykonana w technologii MS SQL o nazwie **TestDataGeneration**. Pierwsza tabela będzie dotyczyć danych osobowych osób zarejestrowanych w dowolnym systemie. Najczęściej prezentowane są takie atrybuty osoby jak:

- » Imię,
- » Drugie imię,
- » Nazwisko,
- » PESEL,
- » NIP,
- » Imię ojca,
- » Imię matki,
- » Data urodzenia,
- » Miejsce urodzenia,
- » Mail.

Baza danych może jednak wymagać dodatkowych kolumn, które należy wypełnić, np. data utworzenia, oraz Id osoby, która utworzyła rekord w bazie. W tym celu do tabeli z osobami zostaną dodane kolumny:

- » CreationDate,
- » CreatedBy.



Rysunek 1. Widok tabel i kolumn w bazie **TestDataGeneration**

W bazie znajduje się również tabela o nazwie **Users.Users** zawierająca najbardziej podstawowe informacje o użytkowniku, który tworzy wpisy do bazy **TestDataGeneration**:

- » Id,
- » UserName,
- » Login,
- » Password.

Naszym zadaniem jest wypełnienie tabeli dbo.Person ilością 2000 rekordów w taki sposób, aby dane miały sens, czyli nie zostały wygenerowane jako ciągi znaków. Na Rysunku 1 przedstawiono wcześniej opisane tabele.

Zakładamy, że mamy użytkownika w tabeli Users.Users. W tym celu zostanie wykonany następujący ciąg czynności:

- » Deklaracja zmiennej przechowującej wartość id użytkownika, która będzie wstawiana do kolumny dbo.Person.CreatedBy,
- » Utworzenie tabel tymczasowych zawierających dane, które zostaną wprowadzone do tabeli dbo.Person,
- » Wypełnienie tymczasowych tabel,
- » Napisanie procedury, która wypełni tabelę dbo.Person,
- » Napisanie zapytania wywołującego procedurę,
- » Napisanie zapytań, które usuną tymczasowe tabele oraz procedury.

Ad 1) Id użytkownika, który „będzie generował” wpisy do bazy Test-DataGeneration, otrzymamy za pomocą zapytania instrukcji:

```
declare @adminId uniqueidentifier
select @adminId = Id from Users.Users where UserName = 'Admin'
```

Ad 2) Tymczasowe tabele utworzone zostaną za pomocą zapytań:

```
CREATE TABLE TEMP_FirstName
(
    ID int NOT NULL PRIMARY KEY,
    Name nvarchar(20) NOT NULL,
);
CREATE TABLE TEMP_SecondName
(
    ID int NOT NULL PRIMARY KEY,
    Name nvarchar(20) NOT NULL,
);
CREATE TABLE TEMP_LastName
(
    ID int NOT NULL PRIMARY KEY,
    LastName nvarchar(30) Not Null,
);
CREATE TABLE TEMP_PESEL
(
    ID int NOT NULL PRIMARY KEY,
    Pesel varchar(11) Not Null,
);
CREATE TABLE TEMP_NIP
(
    ID int NOT NULL PRIMARY KEY,
    Pesel varchar(10) Not Null,
);
CREATE TABLE TEMP_FathersName
(
    ID int NOT NULL PRIMARY KEY,
    FathersName nvarchar(30) Not Null,
);
CREATE TABLE TEMP_MothersName
(
    ID int NOT NULL PRIMARY KEY,
    MothersName nvarchar(30) Not Null,
);
CREATE TABLE TEMP_BirthDate
(
    ID int NOT NULL PRIMARY KEY,
    BirthDate datetime Not Null,
);
CREATE TABLE TEMP_BirthPlace
(
    ID int NOT NULL PRIMARY KEY,
    BirthPlace nvarchar(30) Not Null,
);
CREATE TABLE TEMP_Email
(
    ID int NOT NULL PRIMARY KEY,
    Email nvarchar(10) Not Null,
);
```

Ad 3) Wypełnianie tymczasowych tabel:

```
Insert Into dbo.TEMP_FirstName values
(1,'Patryk'),
(2,'Karol'),
(3,'Paweł'),
(4,'Walerian'),
(5,'Franciszek'),
(6,'Olgierd'),
(7,'Kamil'),
(8,'Ksawery'),
(9,'Leopold'),
(10,'Leszek'),
(11,'Ferdynand'),
(0,'Władysław')
```

```
Insert Into dbo.TEMP_SecondName values
(1,'Olgierd'),
(2,'Ksawery'),
(3,'Łukasz'),
(4,'Sebastian'),
(5,'Felicjan'),
(6,'Olaf'),
(7,'Klaudiusz'),
(8,'Barnaba'),
(9,'Leopold'),
(10,'Emil'),
(11,'Franciszek'),
(0,'Jan')
```

```
Insert Into TEMP_LastName values
(1,'Achmistrówicz-Wachmistrówicz'),
(2,'Acki'),
(3,'DługieNazwisko'),
(4,'Smergowski'),
(5,'Ulryk von Bater'),
(6,'Fenster'),
(7,'Fernando'),
(8,'Fernelllo'),
(9,'Mięgowa-Warowski'),
(10,'Meerreettiich'),
(11,'Von Gebels'),
(0,'Monstowski-Wycinański')
```

```
Insert Into TEMP_PESEL values
(1,'86032802416'),
(2,'90042606711'),
(3,'38042019458'),
(4,'82013005283'),
(5,'82013005283'),
(6,'18042707003'),
(7,'9007040327'),
(8,'01212418124'),
(9,'88120903887'),
(10,'37082705167'),
(11,'72060918809'),
(0,'39100810743')
```

```
Insert Into TEMP_NIP values
(1,'8603202416'),
(2,'9002606711'),
(3,'3804019458'),
(4,'8201305283'),
(5,'8201305283'),
(6,'1804270003'),
(7,'9007040327'),
(8,'0121248124'),
(9,'8812093887'),
(10,'3708705167'),
(11,'7206918809'),
(0,'3910010743')
```

```
Insert Into dbo.TEMP_FathersName values
(1,'Piotr'),
(2,'Kamil'),
(3,'Paweł'),
(4,'Walerian'),
(5,'Franciszek'),
(6,'Eugeniusz'),
(7,'Krzysztof'),
(8,'Olaf'),
(9,'Fred'),
(10,'Łukasz'),
(11,'Wojciech'),
(0,'Marcin')
```

```
Insert Into dbo.TEMP_MothersName values
(1,'Genowefa'),
```

TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ

```
(2,'Hanna'),  
(3,'Halina'),  
(4,'Kunegunda'),  
(5,'Marianna'),  
(6,'Maria'),  
(7,'Stefania'),  
(8,'Zofia'),  
(9,'Patrycja'),  
(10,'Malwina'),  
(11,'Sabina'),  
(0,'Waleria')  
  
Insert Into TEMP_BirthDate values  
(1,'1990-07-09'),  
(2,'1990-06-01'),  
(3,'1989-07-09'),  
(4,'1989-01-01'),  
(5,'1988-07-09'),  
(6,'1980-12-09'),  
(7,'1989-02-01'),  
(8,'1989-07-04'),  
(9,'1989-07-09'),  
(10,'1987-03-09'),  
(11,'1986-03-29'),  
(0,'1987-12-22')
```

```
Insert Into dbo.TEMP_BirthPlace values  
(1,'Genowefa'),  
(2,'Hanna'),  
(3,'Halina'),  
(4,'Kunegunda'),  
(5,'Marianna'),  
(6,'Maria'),  
(7,'Stefania'),  
(8,'Zofia'),  
(9,'Patrycja'),  
(10,'Malwina'),  
(11,'Sabina'),  
(0,'Waleria')
```

```
Insert Into TEMP_Email values  
(1,'Patryk@o2.pl'),  
(2,'Karol@test.pl'),  
(3,'Pawel@wp.pl'),  
(4,'Walerian@o2.pl'),  
(5,'Franciszek@wp.pl'),  
(6,'Albert@gmail.pl'),  
(7,'Kamil@gmail.pl'),  
(8,'Xawery@yahoo.pl'),  
(9,'Leopold@www.pl'),  
(10,'Leszek@wp.pl'),  
(11,'Ferdynand@o2.pl'),  
(0,'Wladyslaw@gmail.de');
```

Kolumny Createdby oraz CreationDate w tabeli dbo.Person będą wypełniane za pomocą zmiennej @adminId oraz funkcją GETDATE(), dlatego nie ma potrzeby tworzenia tabel tymczasowych dla tych kolumn.

Ad 4) Napisanie procedury, która wypełni tabelę dbo.Person:

```
create procedure Insert_Person  
as  
begin  
declare @i int;  
set @i = 1;  
declare @adminId uniqueidentifier;  
select @adminId = Id from Users.Users where UserName = 'Admin';  
  
while(@i <= 2000)  
begin  
insert into dbo.Person(id, FirstName, Secondname, LastName,  
PESEL, NIP, FatherName, MothersName, BirthDate, BirthPlace,  
Email, CreationDate, CreatedBy) values  
(  
newid(),  
(SELECT [Name] from dbo.TEMP_FirstName where id = cast(rand() *  
(select count(1) from TEMP_FirstName) as int)),  
(SELECT [Name] from dbo.TEMP_SecondName where id = cast(rand() *  
(select count(1) from TEMP_SecondName) as int)),  
(SELECT LastName from TEMP_LastName where id = cast(rand() *  
(select count(1) from TEMP_LastName) as int)),  
(SELECT Pesel from TEMP_PESEL where id = cast(rand() * (select  
count(1) from TEMP_PESEL) as int)),  
(SELECT Pesel from TEMP_NIP where id = cast(rand() * (select  
count(1) from TEMP_NIP) as int)),  
(SELECT FathersName from TEMP_FathersName where id = cast(rand()
```

```
* (select count(1) from TEMP_FathersName) as int)),  
(SELECT MothersName from TEMP_MothersName where id = cast(rand() *  
(select count(1) from TEMP_MothersName) as int)),  
(SELECT BirthDate from TEMP_BirthDate where id = cast(rand() *  
(select count(1) from TEMP_BirthDate) as int)),  
(SELECT BirthPlace from TEMP_BirthPlace where id = cast(rand() *  
(select count(1) from TEMP_BirthPlace) as int)),  
(Select Email From TEMP_Email where id = cast(rand() * (select  
count(1) from TEMP_Email) as int)),  
Getdate(),  
@adminId  
);  
set @i += 1;  
end;  
end;
```

Wyżej zaprezentowana procedura jest w stanie wypełnić w teorii jednoznacznie co najmniej 12^{10} wierszy, nie licząc kolumn z typami UniqueIdentifier oraz Datetime. Procedura Insert_Person pobiera dane z tabel tymczasowych w sposób losowy, a następnie wstawia te dane do tabeli dbo.Person. Wyrażenie cast(rand() * (select count(1) from TEMP_LastName) as int) wybiera tylko jeden rekord z tabeli w sposób całkowicie losowy, co ma na celu zapewnienie jak najmniejszej liczby powtarzających się rekordów.

Ad 5) Kolejny krok to wypełnienie tabel, czyli wywołanie zapytania, które uruchomi procedurę Insert_Person, czyli instrukcję:

```
exec Insert_Person.
```

Ad 6) Po wypełnieniu tabel w bazie TestDataGeneration należy usunąć tymczasowe tabele oraz procedurę, czyli napisać instrukcję:

```
drop table TEMP_FirstName;  
drop table TEMP_SecondName;  
drop table TEMP_LastName;  
drop table TEMP_PESEL;  
drop table TEMP_NIP;  
drop table TEMP_FathersName;  
drop table TEMP_MothersName;  
drop table TEMP_BirthDate;  
drop table TEMP_BirthPlace;  
exec('drop procedure Insert_Person');
```

Napisanie skryptu w takiej kolejności, jak zaprezentowana została w punktach od 1 do 6, pozwoli na wypełnienie tabeli danymi, następnie usunie z bazy zbędne już tabele oraz procedury. W przypadku zaistnienia potrzeby wypełnienia większej ilości tabel należy pamiętać o kolejności uruchamiania procedur, które wstawiają dane, ze względu na klucze obce oraz relacje między tabelami.

PODSUMOWANIE

W artykule przedstawiono podejście uzyskiwania danych, które wypełniają listy na formularzach. Taką strategię można zastosować przed testami manualnymi, testami interfejsu (np. filtrowanie), testami polegającymi na wypełnieniu dużej liczby rekordów. W literaturze istnieje wiele sposobów dostarczania danych dla potrzeb testów. Ale mając procedurę, która to robi, oszczędzamy czas.



MAREK ŻUKOWICZ

bobmarek@o2.pl

Absolwent matematyki na Uniwersytecie Rzeszowskim, pracuje jako tester oprogramowania. Jego zainteresowania skupiają się wokół obszarów testowania oraz jakości oprogramowania. Interesuje się również wybranymi zagadnieniami matematyki oraz zastosowaniami modeli matematycznych w procesie testowania oprogramowania.

Let us bring you home

www.semihalf.com

Semihalf creates software for advanced solutions in the areas of operating systems, virtualization, networking and storage.
We make software which is tightly coupled with the underlying hardware to achieve maximum system capacity for running at scale.
We offer a dynamic and open yet principled work environment with exceptional engineering challenges.
Our partners and customers are semiconductor industry leaders mainly from Silicon Valley.

Join us for a deep dive into the latest microprocessor technologies and high performance software development.

Currently we are looking for

Kernel Hacker

(Krakow)

Your challenges

You will be responsible for bringing up new hardware and creating low-level software running on multicore processors. You will be hacking on operating system kernel (BSD, Linux), writing device drivers and optimizing for the best performance results. Your code will often be submitted to the open source repositories.

Requirements

- ④ Fluency in C code development and debugging
- ④ Low level coding experience (Linux or BSD kernel, bootloaders)
- ④ x86 or ARM assembly experience
- ④ Handling of standard shell utilities and tools like GCC, GDB, GIT, DTrace

Benefits



Flexible working hours aligned to your individual preferences



Small teams (2-6 persons) with real influence on projects



Individual yearly training budget



Unique employee profit sharing programme



Social package, medical health care, multisport card



Chillout room, game console

Dokąd zmierza PHP

Wywiad z Rasmusem Lerdorfem, twórcą języka PHP

Podczas konferencji WeAreDevelopers 2017 w Wiedniu Rasmus Lerdorf wygłosił prelekcję na temat historii języka PHP i planów jego dalszego rozwoju [1]. Na łamach Programisty prezentujemy wywiad dotyczący powiązanych tematów, przeprowadzony podczas tego wydarzenia.

Magazyn Programista: W jaki sposób na chwilę obecną wyznaczany jest kierunek rozwoju języka PHP?

Rasmus Lerdorf: Kierunek rozwoju PHP wyznaczany jest przede wszystkim przez społeczność korzystającą z tego języka. Dużą rolę w tym procesie odgrywają programiści, którzy dołączyli do projektu i poświęcają mu swój czas: dodają oni do języka właściwości, które sami uważają za przydatne. Częścią procesu związanego z rozwojem języka jest oczywiście pisanie dokumentów RFC i głosowanie. W skrócie wygląda to tak, że w sytuacji kiedy programista-ochotnik rozwijający PHP wpadnie na jakiś ciekawy pomysł – opisuje go, po czym podlega on ocenie społeczności w formie głosowania. Jeśli pomysł zdobędzie wymaganą liczbę głosów, to jego implementacja staje się częścią języka. Oczywiście w procesie tym istotną rolę pełnią też wieloletni współpracownicy. Kierunek rozwoju PHP od 23 lat jest raczej stabilny: skupiamy się przede wszystkim na oferowaniu rozwiązań dla sieci Web. W ostatnich latach ta dziedzina zmieniała się bardzo dynamicznie, a my próbując nadążyć za tymi zmianami, zaniedbaliśmy nieco kwestie związane z wydajnością. W związku z tym wydajność stanowi ostatnio, obok obsługi nowości w webowym ekosystemie, główny kierunek rozwoju PHP.

M: Rozumiem jednak, że to Ty nadal masz decydujący głos w sprawach dotyczących obierania kierunków rozwoju PHP, na podobnej zasadzie jak Bjarne Stroustrup steruje rozwojem języka C++?

R: Nie do końca. Społeczność języka PHP od początku swojego istnienia miała bardzo płaską strukturę. W tym układzie postrzegam siebie po prostu jako jednego ze współtwórców tego projektu, w związku z tym mam oczywiście wpływ na jego kształt, jednakże nie w większym zakresie niż jakakolwiek inna osoba poświęcająca większość swojego czasu na rozwój tego języka. Tak jak inni, podczas głosowań nad RFC mam do dyspozycji jeden głos. Jeśli nie wspieram jakiejś konkretnej zmiany, zaś pozostały ją przeglądają, to tak czy inaczej zostaje ona zaimplementowana. W tym ujęciu można powiedzieć, że w kwestiach rozwoju języka nie mam tzw. ostatniego słowa, tak jak Linus czy Stroustrup – jednakże jest to świadoma decyzja: nie chcę tego robić. Chcę, żeby osoby, które dołączają do tego projektu, miały poczucie, że ich decyzje są respektowane, że mają realny wpływ na rozwój tego języka. Myślę, że to daje im dużą motywację do pracy.



M: Dlaczego porzuciłeś pracę nad PHP 6?

R: Taaak... Historia związana z rozwojem PHP 6 jest interesująca. Pojawiły się tam dwa istotne problemy. Po pierwsze, biblioteka ICU od IBM służąca do obsługi Unicode, na której się opieraliśmy, była naprawdę powolna i na dodatek trudna w obsłudze. Pracując z nią, mieliśmy wrażenie, że po ka-walku dodajemy PHP do ICU, a nie odwrotnie.

Złożoność tego zadania stała się przyczyną kolejnego problemu. Programiści pracujący nad tym zagadnieniem zaczęli tracić motywację do pracy i nadzieję, że w ogóle uda się to dokończyć. Obsługa Unicode to specyficzna właściwość języka: z jednej strony jest ona wszystkim potrzebna, a z drugiej strony – nie wzbudza specjalnych emocji, nikt nie czeka na nią z jakimś specjalnym utęsknieniem. Tak więc jeśli da się znaleźć proste rozwiązanie do obsługi tego zagadnienia, nawet jeśli nie jest ono idealne, to i tak wszyscy są zadowoleni. Nikt nie jest specjalnie chętny, aby na rzecz wygodnej obsługi Unicode poświęcać wydajność czy też nadmiernie komplikować implementację języka. Tak mniej więcej wyglądała kwestia wsparcia Unicode w starszych wersjach PHP: dzięki temu, że jego implementacja stanowi rodzaj kanału, przez który przesypane są dane, kilka pomocniczych funkcji do obsługi Unicode przez długi czas było zupełnie wystarczające do obsługi tego standardu. W momencie gdy okazało się, że na rzecz pełnego wsparcia Unicode musimy poświęcić wydajność oraz znaczco podnieść złożoność implementacji języka – kompletnie straciliśmy motywację do pracy nad tym zagadnieniem. Poza trzema czy czterema głównymi programistami wszyscy inni, patrząc na to, mówili: „nie, dziękuję, wystarczy mi to, co jest w PHP 5” – i to był główny problem.

Trudno było kontynuować prace w sytuacji, kiedy zdecydowana większość osób wspierających ten projekt wolała pozostać przy piątej wersji języka, twierdząc, że rzeczy związane z szóstką są zbyt skomplikowane. Doszliśmy wtedy do przekonania, że popełniliśmy błąd, próbując zrobić zbyt duży skok jakościowy pomiędzy wersjami: staraliśmy się wprowadzić zbyt dużo poważnych zmian w jednym czasie, przez co traciliśmy impet. Motywacja i entuzjazm w zespole leciały na łeb na szyję, aż w końcu ktoś powiedział: „ta droga prowadzi w ślepy zaułek, przerwijmy to!”.

Z racji tego, że na tym etapie cały kod znajdował się w gicie, zdecidowaliśmy po kolei „cherry-pickować” kolejne usprawnienia z wersji 6 do gałęzi PHP 5. Można powiedzieć, że PHP 6 umarło śmiercią

naturalną, po prostu przestaliśmy nad tą wersją pracować, a w zamian za to powstaje PHP 7. Wersja ta nie posiada co prawda pełnej obsługi Unicode (takiej, jaką planowaliśmy do wersji 6), ale liczymy, że stosując metodę małych kroków, uda nam się zbudować optymalne rozwiązanie.

M: Podsumowując, główny powód decyzji o porzuceniu prac nad PHP 6 był taki, że osoby wspierające projekt nie zgodziły się na „wielkie porządki”. Czy tak?

R: Cóż, ludzie chcieliby zrobić wielkie porządki, ale nie bardzo uśmiecha im się wizja załamania się ich istniejącego kodu. Nie chcą też integracji z biblioteką ICU, co w praktyce oznaczałoby 30-40 procentowy spadek wydajności i dwa razy większe zapotrzebowanie pamięci w stosunku do PHP 5. Mówią zatem: „wsparcie Unicode to wspaniała rzecz, ale nie chcę, żeby przy okazji moja witryna zwolniła o 30% i zużywała więcej pamięci na serwerze”. Myślę, że to był podstawowy problem i powód, dlaczego straciliśmy poparcie użytkowników.

M: Czy uważasz, że proces rozwoju języka sterowany przez społeczeństwo podąża za potrzebami zwykłych programistów, wykorzystujących język w codziennej pracy? Pytam, bo mam wrażenie, że profile i potrzeby większości członków społeczności rozwijającej PHP w dużej mierze odbiegają od charakterystyk i potrzeb zwykłych, średnio-zaawansowanych użytkowników tego języka.

R: Może po części masz rację, ale na to pytanie trudno dać jednoznaczną odpowiedź. Rzec w tym, że Twoje rozumienie pojęcia „zwykły programista” może różnić się diametralnie od rozumienia innych osób. Użytkownicy PHP to niezwykle szeroka i różnorodna grupa; musimy wspierać zarówno „weekendowych wojowników” z bardzo małym doświadczeniem, jak i profesjonalistów, którzy pracują nad dużymi, złożonymi witrynami. Z tej racji PHP stara się być językiem uniwersalnym, trafiającym w gusta przedstawicieli skrajnych subkultur w masie swoich użytkowników. Oczywiście jest, że osoby, które współtworzą implementację tego języka, reprezentują innych rodzaj programistów: przede wszystkim muszą biegle władać językiem C i dobrze orientować się w niskopoziomowych aspektach programowania. Bez tych umiejętności łatwo zagubić się w zawiłościach implementacji PHP. Z tej racji mamy tutaj do czynienia z pewną auto-selekcją, przez co niektóre właściwości dołączane do języka są bardziej ukлонem w kierunku bardziej doświadczonych programistów. Jakkolwiek uważam, że wykonaliśmy dobrą robotę w zakresie projektowania języka w sposób przyjazny dla grup użytkowników o różnych stopniach zaawansowania. Oczywiście zdaję sobie sprawę, że wiele osób nareka co nieco na kierunek rozwoju języka PHP. Osoby takie muszą zdać sobie sprawę, że język ten rozwijany jest przez ochotników. Nie jesteśmy organizacją komercyjną – nie mogę w tym układzie kazać programistom robić tego, na co nie mają ochoty. Bez etatowych pracowników, choćbym nie wiem jak chciał, nie jestem w stanie – ani groźbą, ani prośbą – zmusić programistę-ochotnika do wykonania pracy, do której nie jest on przekonany.

M: Jak na dzisiaj oceniasz bezpieczeństwo aktualnej implementacji PHP? Jak postrzegasz tę kwestię w dalszej perspektywie, w kolejnych wersjach języka? Jak wiadomo, w przeszłości nie obyło się bez pewnych ekscesów w tym zakresie, co zaważyło mocno na opinii PHP w zakresie bezpieczeństwa... Mam tu na myśli takie elementy języka

jak register globals, magic quotes i te wszystkie wymyślne mechanizmy, które nie zdaly egzaminu w praktyce.

R: Prawdę powiedziawszy, tych problemów z bezpieczeństwem po stronie PHP było bardzo niewiele – większość z nich występowała po stronie aplikacji korzystających z PHP, a to subtelna różnica. Elementy języka pokroju *register globals* czy *magic quotes* nie mają w sobie nic niebezpiecznego, o ile korzysta się z nich we właściwy sposób. Oczywiście, problem nadal pozostaje problemem, ale to jest raczej kwestia braku sposobu na wymuszenie skutecznej edukacji programistów w tej dziedzinie. Patrząc z jeszcze innej perspektywy, bez mechanizmu *register globals* nie byłoby PHP. Rejestrowanie zmiennych globalnych w 1995 roku było nieocenionym narzędziem. Pamiętaj, że wtedy nie mieliśmy ataków XSS, nie było też JavaScriptu. Wszystko to, co dziś można łatwo uzyskać za pomocą wstrzyknięcia kodu, wtedy było nieosiągalne. To, że PHP pozwalał stworzyć formularz i automatycznie przekształcać jego pola w zmienne, było w mojej opinii jedną z istotnych przyczyn sukcesu i gwałtownego wzrostu zainteresowania tym językiem. Łatwo, patrząc z dzisiejszej perspektywy, krytykować rozwiązania stosowane w PHP, zresztą trudno nie zgodzić się, że na dzisiaj jest to beznadziejnie głupie rozwiązanie. Ażkolwiek, oceniając pewne właściwości PHP, trzeba spojrzeć wstecz i postarać się zrozumieć kontekst, w którym zostało ono wprowadzone do języka. Wracając jednak do tematu bezpieczeństwa, uważam, że PHP – na dziś – od strony bezpieczeństwa trzyma się całkiem dobrze. Aktualnie staramy się przekonać ludzi do korzystania z Libsodium na rzecz przestarzałej biblioteki mcrypt. Staramy się również o profesjonalny audit bezpieczeństwa ze strony Libsodium – w tym kontekście uważam, że wyprzedzamy konkurencję w zabiegach o bezpieczeństwo.

M: Kontynuując temat bezpieczeństwa, co sądzisz o Wordpressie?

R: Wordpress... Cóż, jest to kolejny podstarzały projekt. Prawda jest taka, że idealnie bezpieczne oprogramowanie nie istnieje. Im starsze ono jest, tym częściej słyszy się o odnalezionych w nim lukach bezpieczeństwa. O ile aktualizujesz Wordpressa na bieżąco, podobnie jak inne oprogramowanie, to jesteś w miarę bezpieczny. Jeśli nie instalujesz podejrzanych, przestarzałych bądź niebezpiecznych wtyczek pobranych nie wiadomo skąd – możesz spać spokojnie. W tym ujęciu trudno mówić o jakichś wyjątkowych problemach z bezpieczeństwem po stronie Wordpressa. Owszem, jest to stary kod, który ma już swoje lata, ale nie jest bardziej niebezpieczny niż inne rozwiązania stosowane w sieci. Jest on po prostu wykorzystywany na olbrzymią skalę. Z drugiej strony, lepiej chyba stosować rozwiązanie przetestowane przez dużą liczbę użytkowników, ze znanimi problemami, niż to, które pozornie wydaje się bezpieczne ze względu na to, że jeszcze nikt nie znalazł w nim luk. Założę się, że gdyby wybrać dowolną bibliotekę lub framework i nagle zwrócić na niego uwagę takiej rzeszy programistów oraz użytkowników, którzy na tę chwilę pracują z Wordpresssem, to liczba raportów na temat luk bezpieczeństwa byłaby porównywalna. W związku z tym, kiedy ludzie pytają mnie: „czemu Wordpress jest tak mało bezpiecznym rozwiązaniem?”, odpowiadam po prostu: „nie jest!”. Po prostu ciągle słyszy się o problemach z nim związanych ze względu na olbrzymią liczbę jego użytkowników. A czemu Windows ma opinię tak mało bezpiecznego systemu operacyjnego? Zasada jest taka sama. Podsumowując ten temat, pozwolę sobie zacytować Stroustrupa: *są dwa typy języków programowania: takie, na które ludzie ciągle narzekają, i takie, których nikt nie używa.*

M: Zmieniając temat: jakie nowoczesne frameworki oparte na PHP, a wykorzystujące najnowsze właściwości języka (takie jak na przykład generatory lub wskazywanie typów), których zadaniem jest usprawnić pracę programisty, a tworzone oprogramowanie uczynić bardziej bezpiecznym, mógłbyś zarekomendować?

R: Szczerze: nie mam pojęcia. Nie korzystam z frameworków. Pracuję nad rozwojem języka PHP. Najwięcej programuję w C. Zabawne jest, że ludzie zawsze pytają mnie o to, zapominając, że PHP nie jest zaprogramowanie w PHP :). Większość mojego czasu spędżam z narzędziami pokroju GDB czy Valgrind. Znam oczywiście PHP jako język, wiem, jak z niego korzystać w wydajny sposób, ale z pytaniami na temat frameworków lepiej zwrócić się do ich twórców lub do ludzi, którzy z nich na co dzień korzystają. Ja tego nie robię. Mam jako takie pojęcie o Laravelu, o Symphony i o kilku innych rozwiązaniach i w mojej opinii nie wykorzystują one jeszcze w pełni potencjału PHP 7. Myślę, że potrzeba więcej czasu, aby to się stało – przynajmniej rok. Póki co większym problemem jest to, że nadal sporo użytkowników korzysta ze starszych wersji PHP, więc twórcy frameworków muszą dbać o kompatybilność wstępna. Dla przykładu minimalna wersja języka wspierana przez WordPressa to PHP 5.2, mamy więc tu nadal do czynienia z piątą wersją języka. Jak więc myśleć tu o wykorzystywaniu właściwości PHP 7? Liczę jednak, że czas to zmieni.

M: Mówisz, że czujesz się przede wszystkim programistą C i koncentrujesz się na rozwoju implementacji języka. Czy rozważałeś, w odniesieniu do PHP, odejście od modelu interpretera na rzecz innych rozwiązań, np. podobnych do tego, co stosuje Facebook, czyli HHVM?

R: Domyślam się, że masz na myśli technologię JIT? Prawdę mówiąc, PHP 7 od strony wydajności ma podobne osiągi jak HHVM. Ten ostatni również jest interpreterem, tyle że z wbudowanym mechanizmem JIT. PHP 7 póki co jest w podobnym stopniu szybki, czasami nawet szybszy niż HHVM, zaś dodanie obsługi JIT mamy w planach (to, co już jest, można obejrzeć sobie, wpisując w wyszukiwarce GitHub'a frazę „JIT for PHP”). Warto tutaj nadmienić, że dodanie mechanizmu komplikacji Just-In-Time nie podniesie drastycznie wydajności aplikacji opartych na PHP. JIT daje mocnego kopa w przypadku wykonywania obliczeń na procesorze, takich jak generowanie fraktali czy przetwarzanie obrazów, czyli operacji, których nie powinno się raczej obsługiwać za pomocą PHP. W syntetycznych benchmarkach różnice w wydajności są kolosalne. Ale o czym my tu mówimy? O zbiorze zagnieżdżonych pętli? Rzecz w tym, że większość aplikacji opartych na Wordpressie, Laravelu czy Drupalu wykonuje zupełnie inne operacje, tak jak odpytywanie bazy danych czy też obsługa interakcji z użytkownikiem, w przypadku których obecność JIT niewiele zmieni. Komplikacja Just-In-Time pojawi się oczywiście w PHP, licząc, że w przeciągu roku, może dwóch lat. A póki co nie próżnujemy w zakresie dodawania usprawnień związanych z podniesieniem wydajności PHP

– przez ostatnie dwa lata udało się odnieść na tym polu szereg spektakularnych sukcesów: wydajność tego języka wystrzeliła jak rakieta w stosunku do tego, co było wcześniej, i na ten moment jest na naprawdę zadowalającym poziomie. Ponadto cały czas pracujemy nad dodawaniem nowoczesnych usprawnień ramię w ramię z takimi firmami jak Intel czy Microsoft. Szczególnie Intel bardzo mocno wspomogł nas w optymalizacji kodu pod ich procesory. Również zespół GCC wspierał nas w niektórych pracach optymalizacyjnych. Ponadto programiści współtworzący PHP bardzo lubią zadania związane z optymalizacją, więc motywacji do pracy w tym zakresie nam nie brakuje.

M: Czy macie w planach usunięcie jakichś istniejących elementów PHP w przyszłych wersjach języka? Może jakieś przestarzałe elementy składowe lub biblioteki?

R: Tak. Póki co w końcu pozbyliśmy się mcrypta. Usunęliśmy też wszystkie te rozszerzenia MySQL, które uważane były za niebezpieczne. Jednakże tego rodzaju zmiany staramy się wprowadzać bardzo ostrożnie, ze względu na istniejące aplikacje, które mogą z nich nadal korzystać. Każdy element PHP, który usuwamy, jest dla ludzi przeszkodą do wykonania aktualizacji. Jeśli będziemy robić to zbyt szybko, to ludzie w ogóle przestaną instalować nowe wersje PHP i zdecydują się pozostać na zawsze przy wersji 5. Dlatego robimy to wolno i ostrożnie, chcąc uniknąć gwałtownej rewolucji i związanych z tym niedogodności (tak jak stało się w przypadku przejścia z Perla 5 na 6 lub z Pythona 2 na 3).

M: Czy pracujesz nad jakimiś innymi projektami, niezwiązanymi z PHP?

R: To trudne pytanie. Wszystkie moje przedsięwzięcia tak czy inaczej prowadzą do PHP. Jakiś czas temu zacząłem pracę nad narzędziem Fan, służącym do statycznej analizy... dla PHP. Mam też piętnastoletniego syna, więc dużą część mojego wolnego czasu poświęcam jemu. Syn interesuje się modelarstwem, tworzy różne elementy za pomocą drukarki 3D, buduje z nich różne rzeczy, bawi się programowaniem Arduino, a ja często wspieram go w tych przedsięwzięciach. Trudno nazwać to moimi projektami, jest to raczej wspólna zabawa i pomaganie sobie nawzajem. Kiedy masz w domu dziecka, jego hobby stają się po trochu dwiema, wypierając te, które być może są dla ciebie bardziej interesujące, ale nie masz już na nie czasu... Po prostu brakuje dnia na to, aby połączyć wszystko naraz: pracę nad PHP, moją pracę zawodową, zabawę z synem i jeszcze coś innego. Nie mówiąc już o moich wyjazdach na konferencje, dzięki którym mam okazję być tutaj i udzielać tego wywiadu :)

Wywiad przeprowadził Michał Leszczyński
Tłumaczenie na język polski: Rafał Kocisz
Specjalne podziękowania dla Biura Miasta Wiednia, firmy Eurocomm-PR

W sieci

- ▶ [1] <https://www.facebook.com/wearedevelopers.org/videos/1736675446347690/> (moment w filmie: 1:28:27)

P O L E C A M Y :

WARSZAWA / 12-14.07.2017

TECHNICAL LEADERSHIP™
ROLA LIDERA TECHNICZNEGO

1. Rola lidera technicznego
2. Motywacja własna i innych
3. Ludzie
4. Zespół
5. Kompetencje lidera

WARSZAWA / 14-15.12.2017

ZESPOŁY ROZPROSZONE
TECHNIKI SKUTECZNEJ PRACY ZDALNEJ

1. Wprowadzenie
2. Efektywność vs. obecność
3. Strategia wirtualnego biura
4. Strategia zespołu rozprozonego
5. Strategia łączona
6. Metodyki zwinne w zespołach rozproszonych

N A J B L I Ż S Z E S Z K O L E N I A W 2 0 1 7 R O K U :

Wzorce projektowe i refaktoryzacja do wzorców	Warszawa	05-07.07.2017	2100,00 PLN
Technical Leadership™	Warszawa	12-14.07.2017	2100,00 PLN
Nowoczesne architektury aplikacji	Warszawa	19-20.10.2017	1800,00 PLN
Techniki pracy z kodem	Warszawa	29.11-1.12.2017	2100,00 PLN
Kanban w 1 dzień	Warszawa	08.12.2017	1300,00 PLN
Zespoły rozproszone - techniki skutecznej pracy zdalnej	Warszawa	14-15.12.2017	1800,00 PLN
Technical Leadership™	Warszawa	18-20.12.2017	2100,00 PLN

CENY NETTO

Kryteria wartości biznesowej

Jak priorytetyzować backlog, gdy jest wiele sprzecznych interesów

Priorytetyzacja backlogu nie należy do zadań łatwych. A co, gdy pojawia się wiele osób zainteresowanych rozwojem jednego produktu? Opisujemy historię Team Leadera, któremu udało się pogodzić ze sobą priorytety trzech działów biznesowych, nieustannie konkurujących o czas programistów w jego zespole.

ILE TO ZAJMIE?

Typowym pytaniem ze strony biznesu jest: „Ile to zajmie?” W modelu ROI, tj. Return on Investment (z ang. „Zwrot z inwestycji”), jest to pytanie o inwestycję. Ile musimy zainwestować, żeby mieć gotową daną funkcjonalność? Jaki jest koszt naprawienia konkretnego błędu?

Często odpowiedź na pytanie „Ile to zajmie?” jest podstawą priorytetyzacji. Reprezentanci biznesu wybierają zadania mniej kosztowne lub starają się (czego np. w Scrumie nie powinni robić) negocjować oszacowanie czasochłonności z programistami.

Jednak z punktu widzenia korzyści dla firmy odpowiedź na pytanie „Ile to zajmie?” niewiele wnosi. Nie pomaga ona też ostatecznie rozstrzygnąć, które zadanie rzeczywiście warto zaimplementować – szczególnie w przypadku konfliktu interesów.

Dlatego, wracając do modelu ROI, warto przyjrzeć się nie tylko inwestycji, ale też oczekiwanej poziomowi „zwrotu z inwestycji”. W dalszej części artykułu opowiadamy o tym, jak skupienie się na zwrocie z inwestycji dla planowanych zadań programistycznych pomogło w pewnym zespole stworzyć platformę do współpracy z działami biznesowymi. Wypracowane „Kryteria wartości biznesowej” pomogły w codziennej priorytetyzacji.

KILKA SŁÓW NA TEMAT KONTEKSTU

Sytuacja, którą chcemy przedstawić, jest dość często spotykana w dużych organizacjach, gdzie kilka działów biznesowych niezależnie zgłasza swoje zapotrzebowanie na funkcjonalności, spływając one do wspólnego systemu zarządzania zgłoszeniami IT, a dedykowana ku temu osoba z IT musi zdecydować, którymi tematami się zająć, nie mając wystarczającej wiedzy i perspektywy biznesowej. Jednocześnie każdy z działów biznesowych chce, żeby to jego funkcjonalności były realizowane jako pierwsze. Prędzej czy później ma miejsce sytuacja, kiedy zgłoszeń jest zbyt dużo i dochodzi do sporów między biznesem a IT, dlaczego funkcjonalności nie zostały jeszcze zrealizowane. Tak właśnie było w opisywanym przez nas przypadku.

Dodatkowo istniało jeszcze kilka czynników, o których warto wspomnieć:

- » system informatyczny, o którym mowa, używany był najczęściej przez pracowników tej samej firmy co dział IT, zwanych konsultantami;
- » konsultanci używali systemu, aby świadczyć usługi dla klientów końcowych, polegające na rozliczaniu wynagrodzeń;
- » to konsultanci zgłaszały potrzebę zmian w systemie, często na

bazie informacji uzyskanych od klientów (np. klient zmieniał regulamin wynagrodzeń) lub na bazie nadchodzących zmian prawnych dokonywanych przez ustawodawców;

- » osobą, która decydowała o kolejności wykonywanych zadań, był lider po stronie IT;
- » ze względu na brak zaangażowania biznesu w proces decyzyjny główną strategią priorytetyczną była kolejka FIFO – tematy, które się pojawiły wcześniej, najczęściej były realizowane jako pierwsze;
- » po stronie biznesu i konsultantów panowało przekonanie, że trzeba bardzo długo czekać na realizację zgłoszonych tematów – średnio ok. 3 miesiący;
- » nie istniał żaden mechanizm oceny, którymi funkcjonalnościami rzeczywiście warto się zajmować.

Mimo że zaistniała konfiguracja pozwalała realizować kolejne zadania, żadna ze stron nie była z niej zadowolona. Biznes ciągle zadawał pytanie: „Dlaczego tak długo?”, a Team Leader miał poczucie, że jest przeciążony ogromem decyzji, które musiał podejmować w związku z przychodzącymi żądaniami, co do oceny których nie miał wystarczającej wiedzy i kompetencji.

WPROWADZENIE SCRUMA

Jednym z pretekstów, żeby zająć się tematem priorytetyzacji, było wdrożenie Scruma w organizacji. Metodyka ta wymaga zmiany sposobu współpracy między biznesem a IT, więc można było wykorzystać ten fakt, aby wypracować adekwatny nowy model współpracy. Całe przedsięwzięcie, które w efekcie trwało 9 miesięcy i objęło 5 zespołów, zaczęło się od kluczowego zespołu, w którym opisany powyżej problem był najbardziej widoczny. Założenie było takie, aby wypracowany model współpracy wprowadzić do pozostałych zespołów.

Często przy wprowadzaniu zmian oczekiwanie jest takie, że „od teraz wszystko się zmieni”. Management i zespoły widzą to jako „Big Bang” – nagłą zmianę sposobu pracy. W praktyce wypracowanie modelu współpracy zajmuje czas i często odbywa się metodą małych kroków. Tak było i w tym przypadku.

Pracę według Scruma zespół rozpoczął od realizacji zgłoszeń, które miał już wcześniej zaplanowane. Team Leader, teraz w nowej roli Product Ownera, robił w dużej mierze to, co dotychczas – wyznaczał priorytety. Zespół zyskał większą autonomię w kwestii tego, który członek zespołu wykona konkretne zadanie.

PRIORYTETYZACJA BIZNESOWA – PIERWSZE PODĘJCIE

Już w trakcie początkowych Sprintów na pierwszy plan zaczęły się wy- suwać dwa tematy związane ze współpracą z działami biznesowymi:

- » Skąd wiemy, że zajmujemy się najważniejszymi, z punktu wi- dzenia biznesowego, zgłoszeniami? Jak zaangażować przed-stawicieli biznesu w priorytetyzację?
- » Jak zaangażować konsultantów z działów biznesowych w tes-towanie zaimplementowanych rozwiązań? Polityka firmy była taka, że to właśnie ci konsultanci byli odpowiedzialni za prze- prowadzenie testów akceptacyjnych.

W dziale IT zostało nawet ukute hasło:

Scrum = Zaangażowanie + Priorytetyzacja biznesowa

Do tematu zaangażowania konsultantów w testy akceptacyjne wróćmy w dalszej części artykułu. Zaczniemy od kwestii prioryte-tyzacji biznesowej.

Po tym, jak zespół nabrał nieco więcej pewności w przeprowa- dzaniu spotkań Scrumowych, Team Leader zaczął zapraszać na niektóre z nich przedstawicieli działów biznesowych. Na przykład Planowanie Sprintu odbywało się w następujący sposób:

- » Team Leader (Product Owner) przygotowywał *propozycję ko- lejności zgłoszeń z backlogu*;
- » Reprezentanci działów biznesowych mogli ww. kolejność zmienić w zależności od potrzeb biznesowych;
- » Priorytetyzacja była przeprowadzana z użyciem wydrukowa- nych wcześniej kartek z zadaniami z backlogu – dzięki temu dużo łatwiej było prowadzić dyskusję;
- » Po ustaleniu priorytetów zespół dzielił zgłoszenia na mniejsze zadania i ostatecznie szacował, jaki zakres zadań dostarczy. W razie potrzeby Product Owner – wraz z reprezentantami działów biznesowych – decydowali, z których zgłoszeń moż- na zrezygnować, a które muszą zostać zaimplementowane w bieżącym Sprincie.

Spotkania te przebiegały znacznie sprawniej, gdy tydzień przed rozpoczęciem Sprintu przeprowadzano, w podobnym składzie, tzw. Backlog Refinement, którego efektem był m.in. wstępny plan na najbliższe iteracje.

Priorytetyzacja przeprowadzana w ten sposób była krokiem na- przód, miała natomiast jedną wadę – nie była oparta na żadnych danych, a jedynie na wewnętrznym przekonaniu osób z biznesu co do kluczowości zadań. Zaczęły pojawiać się pytania: „Dlaczego moje zgłoszenie nie zostało wybrane do realizacji w tym Sprincie?”. A przede wszystkim: „Skąd wiemy, że wybraliśmy najważniejsze biznesowo zgłoszenia?”. W ten sposób pojawił się pomysł ustalenia wspólnej definicji wartości biznesowej.

PODEJŚCIE ANALITYCZNE

Pierwsze próby mające na celu znalezienie wspólnej i jedno- znaczej definicji wartości biznesowej rozpoczęły się w dziale IT. Przeglądając literaturę i analizując doświadczenia różnych zespołów, natknęliśmy się na koncepcję **Weighted Shortest Job First** (WSJF). Podejście to jest elementem metody Scaled Agile Frame-work (SAFe). Główne założenie WSJF polega na tym, aby podczas priorytetyzacji skupiać się na koszcie opóźnienia (ang. *cost of delay*)

związanym z dostarczeniem danej funkcjonalności w odniesieniu do kosztu wytwarzania funkcjonalności wyrażonej w punktach.

Koszt opóźnienia sam w sobie jest w wielu przypadkach dość rozmytą wartością, toteż SAFe definiuje jej kluczowe składowe:

- » **User-Business Value** – jaką wartość dana funkcjonalność ma dla użytkowników końcowych? Jak jej wprowadzenie może wpływać na wielkość przychodów?
- » **Time Criticality** – jakie mogą być konsekwencje związane ze zmianą czasu wykonania? Czy istnieje termin, którego naru-szenie może mieć istotne konsekwencje finansowe np. wyni- kające z umowy, zmian prawnych wchodzących w życie, akcji marketingowych?
- » **Risk reduction/Sales opportunity** – czy realizacja danego żądania istotnie zmniejsza ryzyko niepowodzenia przedsię- wzięcia? Czy realizacja funkcjonalności daje nowe możliwości sprzedażowe?

Składając powyższe elementy ze sobą, uzyskujemy następujący wzór na wyliczenie wartości biznesowej:

WSJF = (User-Business Value + Time Criticality + Risk reduc-tion/Sales opportunity) / Story Points

W ramach burzy mózgów pojawiły się różne pomysły na to, co można by brać pod uwagę w ramach poszczególnych składników. Ponizej przedstawiamy przykłady:

User-Business Value

- » oszczędność czasu konsultanta w godzinach,
- » poziom przychodu z klienta, którego dotyczy żądanie,
- » poziom zadowolenia klienta ze współpracy (NPS),
- » długość współpracy z danym klientem w latach,
- » zakres zmiany (ilu osób, ilu klientów dotyczy zmiana).

Time criticality

- » kary umowne wynikające z opóźnień,
- » kary wynikające z niewprowadzenia zmian prawnych na czas,
- » błąd znaleziony przez klienta i jego konsekwencje.

Risk reduction/Sales opportunity

- » możliwość wprowadzenia dodatkowej usługi i jej sprzedaży,
- » zwiększenie ilości sprzedaży istniejących usług,
- » uzyskanie nowego kontraktu sprzedawczo-giego dzięki wprowa-dzonemu funkcjonalnościom.

Im dłużej jednak pracowaliśmy nad analitycznym podejściem, tym bardziej wydawało się ono niepraktyczne, gdyż zawierało dużą ilość składowych, które dla wielu funkcjonalności byłyby bardzo trudne do określenia. Drugą wadą tego podejścia było to, że było ono wypracowywane tylko i wyłącznie w ramach IT, co w efekcie niosło bardzo duże ryzyko, że biznes nie będzie chciał go wykorzystywać, traktując jako niepotrzebną fanaberię IT. Potrzebny był inny pomysł.

PODEJŚCIE OPARTE NA WSPÓŁPRACY

W kolejnym podejściu do tematu kryteriów wartości biznesowej przyjęliśmy zupełnie inną strategię. Zamiast wymyślać rozwiązanie dla reprezentantów działów biznesowych, postanowiliśmy zapytać ich o zdanie.

Zorganizowaliśmy warsztaty, w których uczestniczyli liderzy i managerowie z każdego z 4 działów (3 działy biznesowe oraz IT).

- » W pierwszej kolejności poprosiliśmy uczestników, aby wypełnili **wszystkie kryteria**, które mogą powodować, że zgłoszenie jest ważne biznesowo. Powstała lista ok. 20 kryteriów.
- » Następnie każdy z działów otrzymał 3 „głosy” – klasyczną metodą tzw. Dot Voting można było zagłosować na kryteria, które uważa się za **najważniejsze biznesowo**.

Co ciekawe, większość osób uczestniczących w warsztatach zgłosowała podobnie. W efekcie udało się wyodrębnić 3 najważniejsze kryteria wartości biznesowej. Były to:

- » Poprawność naliczeń (ang. *Compliance*),
- » Usprawnienie procesu (mniej manualnej pracy),
- » Ryzyko finansowe.

Ustalone kryteria były później używane podczas dyskusji o priorytetach zgłoszeń w backlogu. Zmienił się też sposób współpracy IT z działami biznesowymi – to przedstawiciele biznesu teraz proponowali zgłoszenia, a nie Team Leader. Dzięki kryteriom łatwiej było np. przekonać pozostałych uczestników o dużej wadze zgłoszenia związanego z poprawnością naliczeń i, w przypadku braku implementacji, niosącego za sobą spore ryzyko finansowe.

MIARY KRYTERIÓW WARTOŚCI BIZNESOWEJ

O ile wprowadzone składowe wartości biznesowej stanowiły bardzo dobrą bazę do dyskusji podczas spotkań planistycznych i znacząco posunęły do przodu myślenie w kategoriach wartości biznesowych, to jednak potrzebny był krok dalej, aby działy biznesowe mogły się lepiej przygotowywać do dyskusji o priorytetach. Nadszedł czas na to, aby skonkretyzować miary dla wypracowanych elementów wartości biznesowej. W efekcie powstały następujące propozycje miar, które miały być zweryfikowane w praktyce:

Poprawność naliczeń

Kryterium: Zmiana wynikająca ze zmian prawnych lub zmian w regulaminie klienta.

Usprawnienie procesu (mniej manualnej pracy)

Kryterium: Zaoszczędzona ilość godzin pracy konsultanta miesięcznie.

Ryzyko finansowe

Kryterium: Jednorazowa kara lub długofalowe konsekwencje finansowe.

Niedługo po stworzeniu kryteriów jeden z działów zaczął przygotowywać własny backlog tematów z uwzględnieniem miar kryteriów wartości biznesowej.

ZAANGAŻOWANIE BIZNESU W TESTOWANIE

Jednocześnie drugim tematem, który cały czas pojawiał się w kontekście współpracy IT z działami biznesowymi, było przeprowadzanie testów akceptacyjnych przez konsultantów. Zdarzało się, że zespół, dużym nakładem sił, dostarczał implementację zgłoszenia zaplanowanego na Sprint, która następnie przez kilka tygodni cekała na przetestowanie.

Działo się to także w przypadku zadań, które spełniały kryteria wartości biznesowej i często były wskazywanie przez reprezentantów biznesu jako priorytetowe. Mimo deklaracji tych zadań jako kluczowych konsultanci nie znajdowali później czasu na ich przetestowanie.

W związku z tym, na wniosek działu IT, została wprowadzona zasada:

*Planujemy zgłoszenie na Sprint tylko, jeśli **konkretna osoba** z biznesu zadeklaruje, że przetestuje implementację tego zgłoszenia **w trakcie Sprintu***

Okazało się, że takie jasne i stanowcze wymaganie ze strony zespołu usprawniło znacząco także proces priorytetyzacji. Działo ono na zasadzie filtra. Część zgłoszeń proponowanych przez biznes do implementacji szybko odpadała z dyskusji w momencie, gdy okazywało się, że nie ma osoby, która mogłaby je przetestować w trakcie Sprintu.

Jednocześnie gdy konkretnemu konsultantowi rzeczywiście bardzo zależało na implementacji danego zgłoszenia (bo np. zmniejszało to ilość manualnej pracy do wykonania lub było kluczowe dla obsługiwanej przez tego konsultanta klienta), przedstawienie dobrze opisanego zgłoszenia, wraz z wyszczególnieniem kryteriów wartości biznesowych wpływających na jego ważność oraz deklaracją przetestowania w trakcie Sprintu, zwykle wystarczało do przekonania pozostałych osób do nadania takiemu zgłoszeniu wysokiego priorytetu.

Id	Nazwa	Poprawność naliczeń	Usprawnienie procesu	Ryzyko finansowe
1234	XYZ1	Zmiana prawna	Można obsłużyć ręcznie niedużym kosztem	brak
1235	XYZ2	Zmiana prawna	Można obsłużyć ręcznie niedużym kosztem	brak
1240	XYZ5	Regulamin klienta	25 godz / miesiąc	brak
1241	XYZ6	Regulamin klienta	brak	Kara w wysokości miesięcznej faktury

Tabela 1. Przykładowa tabela oceny wartości biznesowej wypracowana przez jeden z działów

UZYSKANE EFEKTY

Po kilkunastu Sprintach udało się uzyskać oczekiwany efekt, na który składały się dwa czynniki: zwiększone zaangażowanie biznesu oraz wspólnie wypracowane kryteria wartości biznesowej. Z punktu widzenia lidera zmiana była ogromna – zamiast skupiać się na decyzjach biznesowych, mógł zaangażować się bardziej w tematy związane z pracą zespołu. Po stronie biznesu zmiana również została oceniona bardzo pozytywnie. Poniżej zamieszczamy kilka odpowiedzi osób z biznesu zaangażowanych w procesy decyzyjne:

- » „Nawet jeśli moje zgłoszenie nie zostanie przyjęte na Sprint, wiem chociaż, dlaczego tak się stało”.
- » „Jeśli czegoś nie zrobimy dla klienta, z którym współpracuję, jestem w stanie wyjaśnić mu, dlaczego tak się stało, i że często dostanie coś innego w zamian”.
- » „Współpraca w naszym dziale [biznesowym] znacznie się poprawiła”.

Z punktu widzenia całej organizacji znacząco poprawił się Time to market dla funkcjonalności, które zostały uznane za istotne biznesowo – z trzech miesięcy zmniejszył się on do jednego miesiąca.

PODSUMOWANIE

Wypracowanie kryteriów wartości biznesowej i sposobu współpracy z biznesem w przypadku tego zespołu zajęło kilka miesięcy. Team Leader, który początkowo priorytetyzował zadania bez wiedzy biznesowej, krok po kroku przekazywał odpowiedzialność za ten proces reprezentantom działów biznesowych.

Podejście analityczne okazało się zbyt skomplikowane i zbyt wymagające, aby móc wdrożyć je w codziennej pracy. Dopiero wspólnie ustalone kryteria wartości biznesowej dały podstawę do efektywnej priorytetyzacji. Stanowiły one swego rodzaju „język” do rozmawiania o randze zgłoszeń.

Zdefiniowanie miar do poszczególnych kryteriów pozwoliło na skonkretyzowanie dyskusji o priorytetach. Natomiast zasada wymogu testowania przez biznes w trakcie Sprintu zwiększała zaangażowanie konsultantów w proces wytwarzania oprogramowania.

Mimo że każda firma jest inna i nie ma w takich przypadkach rozwiązań uniwersalnych, ta historia może być dobrą inspiracją do tego, jak w IT skupiać się na maksymalizowaniu dostarczanej wartości biznesowej.



MARIUSZ SIERACZKIEWICZ

m.sieraczkiewicz@bnsit.pl

Od ponad dwunastu lat profesjonalnie zajmuje się tworzeniem oprogramowania. Zdobyte w tym czasie doświadczenia przenosi na pole zawodowe w BNS IT, gdzie jako trener i konsultant współpracuje z czołowymi polskimi zespołami programistycznymi. Jego obszary specjalizacji to: refaktoryzacja, czysty kod oraz technical leadership. Autor metody Naturalnego Porządku Refaktoryzacji®. Autor książki „Technical Leadership. Od eksperta do lidera”.



PAWEŁ WRZESZCZ

p.wrzeszcz@bnsit.pl

Od 11 lat pracuje w branży IT, w tym 7 lat także po stronie biznesowej. Współzałożyciel firmy programistycznej SoftwareMill. Obecnie jako Agile Coach wspiera organizacje we wdrażaniu zwinnych podejść do tworzenia oprogramowania. Posiada bogate doświadczenie w tworzeniu efektywnych zespołów rozproszonych. Doradza także osobom z IT chcącym założyć własny biznes.

reklama



devstyle.pl
ŚWIAT OKIEM PROGRAMISTY

Sztuka udzielania feedbacku

Udzielanie feedbacku tylko z pozoru jest proste. Stosowany właściwie feedback może być świetnym narzędziem wykorzystywanym do poprawy relacji między ludźmi w zespole lub efektywności i sposobu pracy. Niewłaściwie udzielony może być odebrany jak niezasłużona krytyka i zlekceważony lub spowodować reakcję ofensywną. Dobra jest więc znać klika prostych wskazówek, które mogą temu zapobiec.

POWIEDZ, CO MYŚLISZ

Feedback może być pozytywny (ang. *appreciate*) lub negatywny (ang. *learning*). W obu przypadkach celem udzielenia informacji zwrotnej jest pomoc odbiorcy w wypracowaniu lub motywowaniu do konkretnego zachowania. W obu przypadkach feedback, który go udzielasz, powinien być:

- » uzasadniony
- » szczerzy
- » obiektywny i oparty na obserwacjach
- » skierowany bezpośrednio do odbiorcy
- » udzielany na bieżąco, regularnie lub incydentalnie, gdy zajdzie taka potrzeba
- » konkretny, zrozumiały, odwołujący się do faktów i poparte przykładami
- » osobisty

Gdy feedback, który chcesz przekazać, ma charakter negatywny, nie należy go udzielać publicznie. Musisz pamiętać, że feedback służący nauce, pomimo Twoich najlepszych chęci, może zostać odebrany jako niezasadniona krytyka. Dlatego szczególnie ważne jest w tym przypadku odwoływanie się do faktów.

W przypadku udzielania negatywnej informacji zwrotnej:

- » Nie używaj ogólników takich jak „zawsze” lub „nigdy”
- » Pamiętaj o emocjach swoich i odbiorcy, szanuj je i rozmawiaj o nich w taki sposób, żeby ich nie podsycać
- » Opisz wydarzenie, opierając się na faktach, nie na emocjach
- » Postaraj się zauważyc i docenić pozytywne aspekty wydarzenia, jeśli takie zaistniały
- » Możesz używać podobnych konstrukcji: „Zauważylem, że świetnie poradziłeś sobie z... i kiedy zdarzyło się... nie pokazałeś tego, co potrafisz. Co o tym myślisz? Jeśli mógłbym coś zaproponować, nastepnym razem warto by było... dlatego, że...”
- » Jeśli najpierw podkreślasz akcent pozytywny, nie używaj spójnika „ale”

Kanapka

Istnieje wiele technik udzielania feedbacku. Najprostszą, ale i najmniej użyteczną wydaje się być „kanapka”, w której negatywną informację zwrotną poprzedza się pochwaleniem odbiorcy, a po zakończeniu przekazywania właściwej treści znów wspomina się o jakimś pozytywnym aspekcie. Zgadzam się, że przed udzieleniem informacji zwrotnej o charakterze negatywnym dobrze jest zmniejszyć dystans pomiędzy Tobą a rozmówcą, a ponadto pokazać mu, że nie zależy Ci na krytykowaniu go. Najłatwiej jest to zrobić, mówiąc mu kilka dobrych słów. Wadą tego modelu jest prostota. Po kilku pierwszych zastosowaniach tej techniki Twój rozmówca

zacznie zauważać wzorzec i straci zaufanie do wszystkiego, co mówisz przed i po krytyce.

Trzeba też pamiętać o odpowiednim dobraniu proporcji składników takiej „kanapki”. Jeśli pochwała będzie zbyt obfita, może sprawić, że słowa dezaprobaty przemkną niezauważone, jeśli zbyt skąpa – przestanie odgrywać jakąkolwiek rolę, gdyż odbiorca skupi się jedynie na krytyce.

Ten model może okazać się pomocny, jeśli będzie używany okazjonalnie z zachowaniem umiaru w doborze parametrów oraz w zastosowaniu do tematów lub zdarzeń drobnych, które nie angażują odbiorcy emocjonalnie. Zbyt często stosowany może dezorientować Twojego rozmówcę. Otrzymuje on bowiem za każdym razem информацию, że radzi sobie dobrze, mimo to po słowach pochwały zawsze następuje krytyka. Twój rozmówca może z czasem zacząć reagować negatywnie na słowa aprobaty, czekając na przysłowie „ale”.

Z-FUKO-PZK

Niektórzy do przekazywania feedbacku wykorzystują model FUKO lub jego rozwinięcie Z-FUKO-PZK. Nazwa tego modelu jest akronimem od słów:

- » Zależy mi
- » Fakty
- » Uczucia
- » Konsekwencje
- » Oczekiwania
- » Propozycja rozwiązań
- » Zgoda
- » Krytykowany

Użycie tego modelu wymaga odrobiny wprawy, ale dobrze sprawdzi się on w przypadku, gdy odbiorca jest silnie emocjonalnie zaangażowany w sytuację. Trudność polega na tym, że musisz nauczyć się wypowiadać w taki sposób, aby w Twoim przesłaniu znalazły się wszystkie elementy modelu, jednocześnie tak, aby nie brzmieć nienaturalnie.

- » Zależy mi – przekazując feedback, najpierw powiedz, na czym Ci zależy
- » Fakty – następnie obiektywnie opisz wydarzenie, którego dotyczy feedback
- » Uczucia – powiedz, jakie emocje wywołało w Tobie to wydarzenie
- » Konsekwencje – następnie opisz, jakie konsekwencje może wywołać zdarzenie, które przebiegło w taki, a nie inny sposób, i co dzieje się z Tobą pod wpływem tych emocji
- » Oczekiwania – wyraź swoją potrzebę i powiedz, jakie masz oczekiwania w stosunku do rozmówcy
- » Propozycja – zaproponuj rozwiązanie, które pozwoli zaspokoić Twoją potrzebę
- » Zgoda – zapytaj rozmówcę, czy zgadza się na twoją propozycję

- » Krytykowany – dowiedz się, co możesz zrobić, aby mu pomóc ją zrealizować

„Zależy mi na naszej współpracy, ponieważ do tej pory przebiegała świetnie i ufałem Ci. Zauważylem jednak, że w tym miesiącu trzykrotnie nie pojawiłeś się w pracy, nie uprzedzając o tym nikogo. Martwi mnie to i powoli tracę zaufanie w Twoją solidność i słowność. W naszej kulturze korporacyjnej możemy sobie czasem pozwolić przyjść lub wyjść wcześniej i odrobić godziny w innym dniu, ale niepojawianie się w pracy bez uprzedniego ustalenia z przełożonym i poinformowania zespołu może być przyczyną zwolnienia. Chciałbym, żebyś przynajmniej dzień wcześniej informował zespół, że nie pojawiłeś się na swojej zmianie, i we własnym zakresie umawiał się z osobą, która Cię w tym dniu zastąpi. Czy to wymaganie jest dla Ciebie akceptowalne? Czy będziesz go przestrzegał i czy widzisz sposób, w jaki mógłbym Ci w tym pomóc?”.

Model Z-FUKO-PZK sprawdzi się równie dobrze w przypadku zastosowania na linii pionowej przełożony-podwładny, jak i poziomej podwładny-podwładny. Wymaga koncentracji, zastanowienia się oraz gruntownego przygotowania tego, co chcesz przekazać odbiorcy. Dzięki odwołaniu się do faktów i oddzieleniu ich od Twoich uczuć oraz oczekiwani masz szansę opisać sytuację w obiektywny, łatwy do zrozumienia sposób. Jest to szczególnie istotne w przypadku udzielania informacji zwrotnej na linii pionowej podwładny-przełożony, gdzie, z oczywistych względów, ważne jest to, aby przekazana informacja nie wywołała u odbiorcy reakcji ofensywnej.

Przekazanie feedbacku w taki sposób wymaga od Ciebie dodatkowej pracy, jednak z pewnością nie zostanie on uznany za nieuzasadniony.

Przestrzegając tych kilku prostych zasad, zwiększasz szansę, że informacja zwrotna od Ciebie naprawdę zostanie wysłuchana przez odbiorcę, a nie tylko usłyszana przez niego. Jednak czy jest sposób, aby zwiększyć prawdopodobieństwo tego, że feedback zostanie przemyślany i wdrożony? Informacja zwrotna przychodząca z zewnątrz, nawet przekazana w mistrzowski sposób, nie będzie miała większej mocy niż to, co odbiorca sam myśli o danej sytuacji. Czy nie byłoby wspaniale pomóc odbiorcy samemu odkryć to, co chciałbyś mu przekazać, udzielając feedbacku? W osiągnięciu tego może Ci pomóc pewien model.

NIE MÓW, SŁUCHAJ

GROW

Model GROW jest używany w coachingu, ma na celu pomoc rozmówcy w określeniu i osiągnięciu ważnego dla niego celu.

GROW, tłumaczony jako ROŚNIJ, jest akronimem:

- » Goal
- » Reality
- » Options
- » Will

Coaching nie polega na bezpośrednim przekazywaniu swoich sugestii czy rad, często jednak umiejętnie korzysta z technik udzielania feedbacku. W coachingu należy skoncentrować się na odbiorcy i za pomocą pytań pomóc mu ustalić, z jaką trudnością się mierzy. Pomóc mu odnaleźć taką drogę do osiągnięcia celu, która pozostała spójna z jego wewnętrznymi zasadami. Model GROW może być w łatwy sposób dostosowany do użycia w sesji feedbackowej. Aby w pełni wykorzystać potencjał modelu, odwołam się do kolejnego narzędzia coachingowego, jakim jest zadawanie pytań.

Aby umożliwić odbiorcy udzielenie sobie feedbacku, najpierw trzeba mu pomóc ustalić jego zakres:

- » Co chciałbyś osiągnąć dzięki naszej sesji?
- » Na czym chcesz się skoncentrować?
- » Który aspekt zdarzenia chcesz zrozumieć?
- » Jakie masz oczekiwania?
- » Czego musisz się dowiedzieć, żeby uznać tę sesję za użyteczną?
- » Czy istnieją pytania, na które chciałbyś teraz znaleźć odpowiedź?

Część *Goal* sesji należy adresować na początku i odwoływać się do niej w trakcie pozostałych części sesji, aby mieć pewność, że cel został osiągnięty.

Część sesji związana z Rzeczywistością (*Reality*) ma na celu zatrzymanie się, skoncentrowanie na bieżącej sytuacji i przeanalizowanie jej. Uświadomienie sobie rzeczywistości przed przejściem do planowania rozwiązań.

W tradycyjnym feedbacku udzielający często przeskakuje bezpośrednio z przeszłości, opisującą to, co się wydarzyło, do przyszłości, w której mówi, jakie są jego oczekiwania w przypadku, gdyby miało się to zdarzyć ponownie. Przy tradycyjnym feedbacku nie jest interesujące dla osoby udzielającej feedbacku to, czy odbiorca dobrze rozumie swoją sytuację. Celem jest przekazanie odbiorcy np. co, dla czego i w jaki sposób ma zrobić inaczej niż poprzednim razem.

W przypadku modelu GROW ważne jest, aby osoba dająca sobie feedback potrafiła jasno i obiektywnie ocenić sytuację, w której się znajduje, i spojrzeć na nią z kilku innych perspektyw. Bez tego nie będzie w stanie odnaleźć innych Opcji (*Options*) ani podjąć żadnych Działań (*Will*).

W trakcie tej części sesji pomocne mogą być pytania:

- » Czy możesz opisać sytuację?
- » Jak wyglądało to z Twojej perspektywy?
- » Jak się poczułeś?
- » Jak czułbyś się, będąc po drugiej stronie konfliktu?
- » Jak ważne było dla Ciebie to zdarzenie?
- » Jak na Ciebie wpłynęło?
- » Co myślisz o tym, co przed chwilą zrobiłeś?
- » Co byś myślał, gdybyś był po drugiej stronie konfliktu?

Pytania osadzające w rzeczywistości mają na celu pomoc w spojreniu na zdarzenie lub problem z szerszej perspektywy. Dzięki koncentracji na rzeczywistości pomagasz rozmówcy uspokoić emocje i skoncentrować się na faktach.

Kiedy uczestnik sesji osiągnie wewnętrzny spokój oraz dostrzega sytuację z innych perspektyw, możesz zacząć zadawać pytania feedbackowe takie jak:

- » Co takiego wiesz teraz, czego nie wiedziałeś przed tym wydarzeniem?
- » Co zauważyleś w swoim zachowaniu?
- » Jak czujesz się z tym, w jaki sposób przed chwilą postąpiłeś?
- » Jeśli mógłbyś przeżyć całą sytuację raz jeszcze, jak byś postąpił?
- » Dlaczego?
- » Co sprawia, że właśnie tak byś postąpił?
- » Co jeszcze?

Podczas tej części sesji możesz odwoływać się do ustalonego na początku sesji celu, upewniając się, czy rozmówca przybliża się do jego osiągnięcia:

- » Czy pamiętasz pytania, na które chciałeś znaleźć odpowiedź?
- » Czy już ją znalazłeś? Jaka to odpowiedź?
- » Czy któreś pytania pozostały jeszcze bez odpowiedzi?

Możesz również płynnie przejść do kolejnej części, którą są Opcje (*Options*):

- » Kto może Ci pomóc w znalezieniu odpowiedzi na Twoje pytania?
- » Gdzie możesz znaleźć potrzebne informacje?
- » Co możesz zrobić, aby w przyszłości zachować się w zgodzie z tym, co przed chwilą powiedziałeś?
- » Co lub kto może Ci w tym pomóc?
- » Czego nie chciałbyś powtórzyć?
- » Jak możesz się przed tym uchronić?

To jest moment sesji, w którym Ty osiągnąłeś swój cel, sprawiłeś, że Twój rozmówca udzielił sobie feedbacku, który Ty zamierzałeś mu przekazać. Być może w szerszym lub węższym zakresie niż zamierzony przez Ciebie, ale ta część, do której doszedł sam, będzie stanowiła dla Twojego rozmówcy najcenniejszą lekcję. Jednak nie poprzestawaj na tym! Pamiętaj, że ta sesja nie jest dla Ciebie.

Ostatnia część Wola (*Will*) jest bardzo ważna dla Twojego rozmówcy. Ma na celu pomoc w podjęciu przez niego decyzji odnośnie tego, które ze znalezionej przez niego opcji chce wcielić w życie. Dzięki niej i dzięki późniejszej wytrwałości w realizacji postanowień, w przyszłości, ma szansę odnieść sukces.

Ważne, żeby osoba udzielająca sobie feedbacku zdecydowała się na realizację tych opcji, na które ma realny wpływ.

W tej części sesji możesz zapytać:

- » Co zrobisz, aby osiągnąć swój cel?
- » Jak chcesz to zrobić – dokładnie w jakich krokach?
- » W jakich ramach czasowych – kiedy?
- » Co zrobisz, aby nie stracić motywacji?
- » Czy możesz zrobić więcej?
- » Jak możesz się bardziej zaangażować?
- » Kto lub co może Ci pomóc w wytrwaniu w postanowieniu?
- » Co musi się stać, żebyś był pewien, że cel został osiągnięty?

Naturalną ścieżką konwersacji w modelu GROW byłoby zapytanie rozmówcy na samym początku o to, co się stało oraz jak się czuje i pozwolenie mu na wyzbycie się wszystkich emocji. Po tym wstępnie obydwoje będącymi mieli lepszy wgląd w to, jak wygląda sytuacja z punktu widzenia Twojego rozmówcy oraz jakie budzi emocje. Kiedy wrażenia ostygnią, łatwiej będzie osobie udzielającej sobie informacji zwrotnej skoncentrować się na sesji, ustalić swój Cel (*Goal*) i spróbować spojrzeć na wydarzenie innymi oczyma.

Czasem również bardziej naturalnym jest połączenie części Rzeczywistość oraz Opcje w jedno i zadawanie pytań w głąb danego zagadnienia

- » Jak wygląda bieżąca sytuacja?
- » Jak chciałbyś żeby wyglądała?
- » Jakie widzisz możliwości doprowadzenia do tego?

Jeśli poruszanych jest wiele kwestii jednocześnie, omawianie ich jako całości jest łatwiejsze niż wypisanie wszystkich aspektów w jednym kroku i analizowanie ich w kolejnym.

Nie poprzestawaj na tej części sesji. Pamiętaj o części związanej z wyrażeniem Woli, dzięki której Twój rozmówca zobowiąże się do podjęcia bardzo konkretnych działań przybliżających go do osiągnięcia sukcesu.

Dzięki zastosowaniu modelu GROW zamiast tradycyjnej sesji feedbackowej, otrzymujesz więcej w zamian za poświęcony czas. Odbiorca feedbacku jest jednocześnie jego właścicielem, więc nie może nie zgadzać się ze swoją opinią. Ma szansę spojrzeć na wydarzenie z kilku perspektyw i obiektywnie ocenić swoje zachowanie oraz zdecydować o podjęciu odpowiednich działań zapobiegawczych.

CO, GDZIE, KIEDY?

Istnieje wiele modeli udzielania feedbacku. Wszystkie jednak opierają się na prostych wskazówkach przedstawionych na początku tego artykułu.

„Kanapkę” należy serwować okazjonalnie. Najlepiej gdy informacja zwrotna dotyczy drobnych kwestii. Należy zachować umiar, aby słowa aprobaty nie przysłoniły feedbacku służącego nauce, który chcesz przekazać. Naturalną okazją do zastosowania tej techniki może być chcąc przekazania odbiorcy pełnej informacji o tym, które aspekty jego zachowania lub pracy wymagają korekty, a które są satysfakcyjne. Można wtedy w „części pozytywnej” odwołać się do tego, co rozmówca już robi wystarczająco dobrze, następnie wymienić, co wymaga poprawy, i wyjaśnić dlaczego, a na zakończenie przypomnieć rozmówcy o tym, co udało mu się osiągnąć do tej pory, i zakończyć słowami otuchy.

W przypadku kwestii „trudniejszych”, gdy w grę zaczynają wchodzić emocje, lepiej jest skorzystać z modelu Z-FUKO-PZK. Ten model sprawdzi się świetnie przy okazji grupowych sesji feedbackowych. Przy okazji takich spotkań obie strony mają zazwyczaj szansę wcześniej przygotować i spisać swoje obserwacje oraz oczekiwania. Dzięki temu model staje się prostszy w użyciu, gdyż na spotkaniu uczestnicy przedstawiają już skonstruowaną wcześniej, dobrze przemyślaną wypowiedź. Dzięki celowej koncentracji na faktach oraz świadomemu odseparowaniu od opisu zdarzenia uczuciu oraz oczekiwaniom znacznie łatwiej jest przekazać informację zwrotną służącą nauce, zachowując przyjemną atmosferę w trakcie rozmowy oraz unikając nieporozumień.

Najtrudniejszym z przedstawionych modeli jest model GROW. Wymaga on już, aby udzielający feedbacku potrafił wejść w relację coachingową z odbiorcą. Może być stosowany przy rozmowach rozwojowo-okresowych lub w przypadku naprawdę drażliwych i istotnych kwestii oraz omawiania zdarzeń, w których rozmówca postąpił w sposób nieetyczny. Szczególnie w takich przypadkach, gdy występuje pewien konflikt wynikający z braku zrozumienia sposobu postrzegania zdarzenia z perspektywy drugiej osoby, warto jest posłużyć się tym modelem i odbyć takie sesje z obydwoma stronami konfliktu. Dzięki swojej konstrukcji model GROW w naturalny sposób pomaga zrozumieć sposób myślenia oraz reakcję drugiej osoby. Umożliwia bardziej krytyczne spojrzenie na własne zachowanie.

W sieci:

- ▶ http://www.coachingcultureatwork.com/wp-content/uploads/2013/05/Facts_about_Feedback.pdf
- ▶ <http://ccnews.pl/2016/05/16/prawidlowo-udzielac-informacji-zwrotnej-model-fuko-pzk/>



KATARZYNA SUSKA

ksuska@e-xu.com.pl | www.e-xu.com.pl

Magister Fizyki. Specjalista ds. Rozwoju Oprogramowania. Zatrudniona w firmie Nokia we Wrocławiu w dziale Control Plane. Obecnie pracuje na stanowisku Scrum Master. W wolnych chwilach programista aplikacji webowych oraz aplikacji na platformę Android. Interesuje się współczesnymi trendami w procesach zarządzania w gospodarce rynkowej. Hobby: ornitologia i historia sztuki.



Największy wybór profesjonalnego oprogramowania w Polsce !

... w ofercie programy ponad 500 producentów ...



Więcej informacji:

📞 (22) 868 40 42



sales@tts.com.pl

Sprzedaż



Dystrybucja



Import na zamówienie

TTS Company Sp. z o.o.

ul. Domaniewska 44A

02-672 Warszawa

www.tts.com.pl

CONFidence 2017 – C64 & Encryption Service

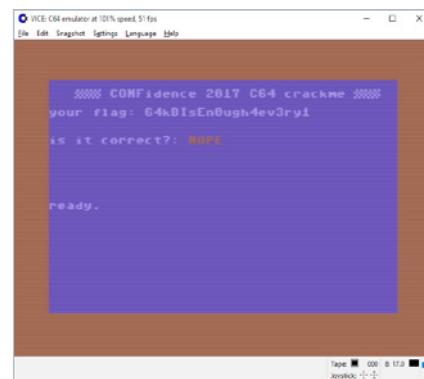
CONFidence to odbywająca się corocznie w Krakowie konferencja poświęcona tematyce bezpieczeństwa – największe tego typu wydarzenie w Polsce. Na towarzyszące jej zawody CTF przybyło w tym roku ponad 20 drużyn z całego świata, w tym z USA, Australii, Tajwanu i Izraela. Zadania były mocno niestandardowe i obejmowały analizę nietypowych architektur, takich jak NES, Xtensa czy Commodore 64.

CONFidence DS CTF		Home	Information	News	Scoreboard	Activity log	Sign in	Sign up
#	Team name	Country	Tasks solved	Points	Last submission			
1	baloom	IL	<div style="width: 75%;"></div>	3220	2017-05-19 14:33:33 CEST Token (Web, 200)			
2	p4	PL	<div style="width: 75%;"></div>	3170	2017-05-19 13:41:25 CEST Token (Web, 200)			
3	Teamless	-	<div style="width: 75%;"></div>	3020	2017-05-19 13:22:41 CEST Shoot the flag (Reverse Engineering, 400)			
4	H4x0rPsch0rr	DE	<div style="width: 75%;"></div>	3020	2017-05-19 14:33:46 CEST Flagcopter Stage 2 (Pwning, 300)			
5	CodiSec	PL	<div style="width: 75%;"></div>	2770	2017-05-19 14:35:00 CEST KeyGenMe (Reverse Engineering, 400)			
6	217	MN	<div style="width: 75%;"></div>	2370	2017-05-19 13:19:06 CEST Token (Web, 200)			
7	ChkSum[0]	IL	<div style="width: 75%;"></div>	2050	2017-05-19 13:37:58 CEST Encryption service (Cryptography, 300)			
8	IspamAndHex	HU	<div style="width: 75%;"></div>	2020	2017-05-19 13:01:05 CEST Token (Web, 200)			
9	Snatch The Root	PL	<div style="width: 75%;"></div>	1820	2017-05-19 14:21:56 CEST Token (Web, 200)			
10	9447	AU	<div style="width: 75%;"></div>	1220	2017-05-19 12:18:45 CEST kore-two (Web, 150)			
11	Just Hit the Core	PL	<div style="width: 75%;"></div>	920	2017-05-19 14:05:08 CEST kore-two (Web, 150)			
12	Shellphish	US	<div style="width: 75%;"></div>	870	2017-05-19 13:24:44 CEST kore-two (Web, 150)			
13	ctcs	PL	<div style="width: 75%;"></div>	550	2017-05-19 14:41:25 CEST Starbase (Reverse Engineering, 250)			

CTF	CONFidence 2017 (Finał) http://2017.confidence.org.pl/
Waga CTFtime.org	44,69 (https://ctftime.org/event/434)
Liczba drużyn (z niezerową liczbą punktów)	22
System punktacji zadania	Od 100 (proste) do 500 (trudne) punktów
Liczba zadań	20
Podium	1. Baloom (Izrael) – 3220 pkt. 2. P4 (Polska) – 3170 pkt. 3. Teamless – 3020 pkt.
Zadania	C64 (RE 100) i Encryption service (Crypto 300)

Narzędzia

Żeby w ogóle zabrać się za analizę, potrzebujemy kilku narzędzi. Warto rozpocząć od uruchomienia programu w emulatorze – wybraliśmy do tego dostępny za darmo VICE¹. Po uruchomieniu naszym oczom ukazuje się flaga: „64kBIsEn0ugh4ev3ry1” i informacja, że jest nieprawidłowa (patrz Rysunek 1).



Rysunek 1. Zadanie C64 widziane w emulatorze VICE

1. <http://vice-emu.sourceforge.net/>

C64

C64 to zadanie typu RE (reverse engineering). Warte było 100 punktów, a rozwiązało je 13 drużyn.

Jak to zwykle bywa w tego typu wyzwaniach, otrzymujemy pojedynczy plik binarny, z którego należy wyciągnąć flagę. W tym przypadku jest to crackme.prg – plik programu dla Commodore 64.

Do dezasembliacji możemy użyć radare2 lub dostępnego online deasemblera WFDIS².

Przyda się też debugger – po kilku minutach poszukiwał najbardziej do gustu przypadł nam darmowy C64 Debugger³. Pomocna może też okazać się mapa pamięci C64⁴. Te narzędzia to wszystko, czego potrzeba, żeby dowiedzieć się, co dzieje się w środku i wydobyć prawidłową flagę.

Analiza

Przed rozpoczęciem właściwej analizy warto wiedzieć coś o samej architekturze. Mamy dostępne trzy 8-bitowe rejesty ogólne – go przeznaczenia: AR, XR i YR.

Przykładowe instrukcje:

- » **lda / ldx / ldy – load;** zapisuje do rejestrów AR / XR / YR wartość spod podanego adresu lub stałą
- » **sta / stx / sty – store;** zapisuje wartość rejestrów AR / XR / YR pod adres podany jako argument
- » **bne – branch if not equal;** wykonuje skok pod adres podany jako argument, jeśli wynik poprzedniej instrukcji był różny od zera (tj. flaga Z nie jest ustawiona)
- » **jmp – jump;** skacze pod adres podany jako argument
- » **jsr – jump to subroutine;** odkłada adres kolejnej instrukcji na stos i skacze pod wskazany w argumencie adres; odpowiednik instrukcji `call` z x86
- » **rts – return from subroutine;** powraca z funkcji pod odłożony na stosie adres
- » **cmp / cpx / cpy – compare;** porównuje wartość podaną jako argument (stałą lub znajdująca się pod podanym adresem) z rejestrów AR / XR / YR i ustawia odpowiednie flagi
- » **inx – increment;** zwiększa wartość rejestrów XR o 1
- » **eor – exclusive or;** wykonuje operację xor na rejestrze AR i wartości podanej jako argument; wynik zapisuje w rejestrze AR

Adresy mogą być podane z offsetem wyznaczonym przez rejestr, np. instrukcja:

```
sta $0123, y
```

załaduje wartość rejestrów AR pod adres 0x0123+YR.

Warto też zaznaczyć, że napisy nie są reprezentowane przez standard ASCII, ale tzw. *screen codes* (patrz Rysunek 2).

Screen code (dec, hex)	Character (up/gfx, lo/up)
0 \$00	█
1 \$01	Ⓐ
2 \$02	Ⓑ
3 \$03	Ⓒ
4 \$04	Ⓓ
5 \$05	Ⓔ
6 \$06	Ⓕ
7 \$07	Ⓖ
8 \$08	Ⓗ
9 \$09	Ⓘ
10 \$0A	Ⓙ
11 \$0B	Ⓙ
12 \$0C	Ⓙ

Rysunek 2. Screen codes⁵

2. <http://www.white-flame.com/wfdis>

3. <https://sourceforge.net/projects/c64-debugger>

4. <http://sta.c64.org/cbm64mem.html>

5. <http://sta.c64.org/cbm64scr.html>

Mając tę wiedzę, możemy przystąpić do właściwej analizy:

```
0x0000080e      lda #$20
0x00000810      ldx #$00
0x00000812      sta $0400,x
0x00000815      sta $0500,x
0x00000818      sta $0600,x
0x0000081b      sta $0700,x
0x0000081e      inx
0x0000081f      bne L0812
```

Na samym początku widzimy pętlę zerującą pamięć znajdującą się pod adresami 0x400-0x420, 0x500-0x520, 0x600-0x620, 0x700-0x720. Jest to pamięć ekranu, czyli ten blok po prostu czyści ekran.

```
0x00000821      jsr S0863
0x00000824      jsr S0869
0x00000827      jsr S0879
0x0000082a      jsr S0899
0x0000082d      jsr S0889
0x00000830      jsr S083a
0x00000833      jsr S084b
0x00000836      jmp L0839
0x00000839      rts
```

Dalej następuje wywołanie siedmiu funkcji. Przeanalizujmy niektóre z nich:

```
0x00000869      ldx #$00
0x0000086b      lda $0900,x
0x0000086e      bne L0871
0x00000870      rts
0x00000871      sta $042b,x
0x00000874      inx
0x00000875      jmp L086b
```

Mamy tu pętlę kopującą ciąg bajtów z adresu 0x900 do 0x42b aż do napotkania bajtu zerowego. 0x42b to pamięć ekranu, czyli funkcja ta wypisuje coś na ekran – rzeczywiście, zaglądając debugerem pod adres 0x900, widzimy nagłówek „CONFidence 2017 C64 crackme” w postaci *screen code* (patrz Rysunek 3).



Rysunek 3. Zrzut pamięci od adresu 0x900

Następne funkcje działają analogicznie, wypisując kolejne napisy:

```
0x0879 - "your flag:"
0x0899 - fałszywa flaga "64kBIsEn0ugh4ev3ry1"
0x0889 - "is it correct?:"
```

Warto przy tym zapamiętać adres, spod którego funkcja 0x0899 kopiuje flagę – jest to 0x95a (patrz Rysunek 4).



Rysunek 4. Zrzut pamięci od adresu 0x940

Wreszcie dochodzimy do naprawdę interesujących funkcji:

```
0x0000083a      ldx #$00
0x0000083c      ldy $09b4,x
0x0000083f      lda $0aca,y
0x00000842      sta $0bcb,x
0x00000845      inx
0x00000846      cpx #$13
0x00000848      bne L083c
0x0000084a      rts
```

Funkcję tę moglibyśmy zapisać w następujący sposób:

```
for (x = 0; x < 19; x++)
{
    y = indexes[x];
    a = values[y];
    output[x] = a;
}
```

W powyższym przykładzie przyjęliśmy następujące odwzorowanie nazw na adresy:

```
indexes - 0x09b4
values - 0x0aca
output - 0x0bcb
```

Możemy wobec tego stwierdzić, że funkcja kopiuje 19 bajtów z dużej (największy indeks to aż 216) tablicy z indeksów wskazanych przez tablicę znajdująca się pod adresem 0x09b4 i ładuje je pod adres 0x0bcb.

Żeby zobaczyć wynik funkcji, możemy użyć debuggera. Ustawimy breakpoint na wyjściu z funkcji (adres 0x084a) i przeczytajmy pamięć z adresu 0x0bcb. Wynikiem jest tablica:

```
[0xed, 0x09, 0x90, 0xf8, 0x1b, 0x77, 0x62, 0x28, 0x88,
 0x4f, 0x20, 0xb2, 0x98, 0x69, 0x7e, 0x24, 0x81, 0x2a,
 0x91]
```

Co ważne uwagi, tablica ma 19 bajtów – dokładnie tyle, ile zapisana w programie nieprawidłowa flaga.

Ostatnią funkcją do przeanalizowania jest ta znajdująca się pod adresem 0x084b:

```
0x0000084b      ldx #$00
0x0000084d      inc $095a,x
0x00000850      lda $095a,x
0x00000853      eor $0bcb,x
0x00000856      cmp $09a0,x
0x00000859      bne L08c9
0x0000085b      inx
0x0000085c      cpx #$13
0x0000085e      bne L084d
0x00000860      jsr S08a9
0x00000863      lda #$17
0x00000865      sta vMemControl
0x00000868      rts
```

Mamy tu wykonującą się 19 razy pętlę. W każdym kroku brany jest kolejny bajt z tablicy pod adresem 0x095a (jak sprawdziliśmy wcześniej, znajduje się tam fałszywa flaga). Bajt ten zwiększany jest o 1, wykonywana jest alternatywna wykluczająca na odpowiadającym bajcie tablicy z adresu 0x0bcb (stworzonej jako wynik poprzedniej funkcji), a potem następuje porównanie z odpowiadającym bajtem z tablicy pod adresem 0x09a0. Jeśli którykolwiek bajt się nie zgadza, następuje skok pod adres 0x08c9, gdzie – jak łatwo sprawdzić za pomocą debuggera – następuje wypisanie informacji o tym, że flaga jest fałszywa.

Potrzebujemy jeszcze tylko zawartości tablicy z 0x09a0 – na szczęście jest ona stała i jej zawartość możemy sprawdzić w dowolnym momencie w widoku pamięci w debuggerze (patrz Rysunek 5).

Rysunek 5. Zrzut pamięci od adresu 0x9a0

```
[0xef, 0x1c, 0xd2, 0xeb, 0x2a, 0x6f, 0x66, 0x1d, 0xd0,
 0x4d, 0x2d, 0xa9, 0xd0, 0x7f, 0x70, 0x74, 0x99, 0x2e,
 0xa4]
```

Powyższą funkcję możemy więc zapisać w języku wysokiego poziomu w następujący sposób:

```
tab1 = [0xed, 0x09, 0x90, 0xf8, 0x1b, 0x77, 0x62, 0x28, 0x88,
        0x4f, 0x20, 0xb2, 0x98, 0x69, 0x7e, 0x24, 0x81, 0x2a,
        0x91]
tab2 = [0xef, 0x1c, 0xd2, 0xeb, 0x2a, 0x6f, 0x66, 0x1d, 0xd0,
        0x4d, 0x2d, 0xa9, 0xd0, 0x7f, 0x70, 0x74, 0x99, 0x2e,
        0xa4]

for (x = 0; x < 19; x++)
{
    flag[x]++;
    flag[x] ^= tab1[x];
    if (flag[x] != tab2[x])
    {
        wrong_flag();
    }
}
```

Rozwiążanie

Postarajmy się znaleźć taki 19-bajtowy ciąg, który spełni warunki powyższej funkcji.

Funkcję tę możemy bardzo łatwo odwrócić:

```
tab1 = [0xed, 0x09, 0x90, 0xf8, 0x1b, 0x77, 0x62, 0x28, 0x88,
        0x4f, 0x20, 0xb2, 0x98, 0x69, 0x7e, 0x24, 0x81, 0x2a,
        0x91]
tab2 = [0xef, 0x1c, 0xd2, 0xeb, 0x2a, 0x6f, 0x66, 0x1d, 0xd0,
        0x4d, 0x2d, 0xa9, 0xd0, 0x7f, 0x70, 0x74, 0x99, 0x2e,
        0xa4]
flag = [0]*19

for x in range(19):
    flag[x] = tab1[x] ^ tab2[x]
    flag[x] -= 1

print ['%02x' % i for i in flag]
```

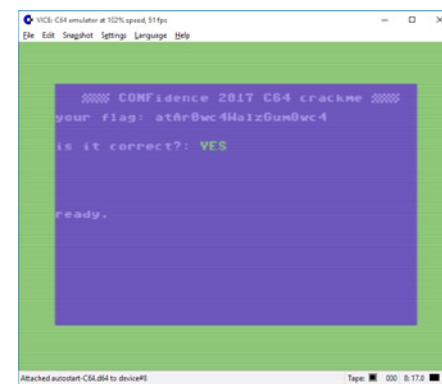
Wynik:

```
['01', '14', '41', '12', '30', '17', '03', '34', '57', '01',
 '0c', '1a', '47', '15', '0d', '4f', '17', '03', '34']
```

Otrzymamy wynik to flaga w postaci *screen code*. Spróbujmy ją podać programowi i sprawdźmy, czy zadziała. Ponieważ flaga jest zapisana na stałe i nie mamy możliwości podania jej z klawiatury, jedyna możliwość to edycja pliku binarnego. Uruchommy więc nasz ulubiony hexedytor i odnajdżmy fałszywą flagę – jak sprawdziliśmy wcześniej, jej reprezentacja to:

```
[0x36, 0x34, 0x0b, 0x42, 0x49, 0x13, 0x45, 0x0e, 0x30, 0x15,
 0x07, 0x08, 0x34, 0x05, 0x16, 0x33, 0x12, 0x19, 0x31]
```

Ciąg ten znajduje się pod offsetem 0x15b względem początku pliku. Zamieńmy 19 bajtów zaczynających się pod tym adresem na wynik naszych obliczeń i... voilà!



Rysunek 6. Rozwiążanie

ENCRYPTION SERVICE

Kolejne zadanie, które chcemy omówić, to warte 300 punktów Crypto. Poradziło sobie z nim 5 zespołów. Głównym problemem tego zadania był czas, jaki mieliśmy na jego rozwiązanie – zostało ono opublikowane drugiego dnia, na kilka godzin przed końcem konkursu.

Jako treść zadania dostaliśmy adres i kod źródłowy programu serwera, który zamieszczały poniżej. Dla poprawy czytelności usunęliśmy z niego obsługi błędów i nagłówki. #define MAX_KEY_

```
LEN 16
#define KEY_SIZE_LEN 1
#define ENTRY_SIZE (MAX_KEY_LEN + KEY_SIZE_LEN)
#define NUM_KEYS 16
#define STORAGE_SIZE (ENTRY_SIZE * NUM_KEYS)
#define MASTER_KEY_INDEX 0
#define DATA_SIZE 16

uint8_t keys[STORAGE_SIZE];
uint8_t data[DATA_SIZE];
uint8_t current_key[MAX_KEY_LEN];

void regenerate_key(unsigned int index, unsigned int len) {
    unsigned int offset = index * ENTRY_SIZE;
    if (offset > STORAGE_SIZE - ENTRY_SIZE || len > MAX_KEY_LEN)
        return;
    int fd = open("/dev/urandom", O_RDONLY);
    read(fd, &keys[offset], len);
    close(fd);
    keys[offset + MAX_KEY_LEN] = len;
}

void load_key(unsigned int index) {
    unsigned int offset = index * ENTRY_SIZE;
    unsigned int key_len = keys[offset + MAX_KEY_LEN];
    if (offset > STORAGE_SIZE - ENTRY_SIZE || key_len > MAX_KEY_LEN)
        return;
    memcpy(current_key, &keys[offset], key_len);
}

void encrypt(void) {
    uint8_t data_out[DATA_SIZE];
    AES128_ECB_encrypt(data, current_key, data_out);
    write(1, data_out, 16);
}

void load_data(void) {
    read(0, data, DATA_SIZE);
}

int main(void) {
    for (int i = 0; i < NUM_KEYS; i++)
        regenerate_key(i, MAX_KEY_LEN);

    int fd = open("flag.txt", O_RDONLY);
    read(fd, data, DATA_SIZE);
    close(fd);

    load_key(MASTER_KEY_INDEX);
    encrypt();
    memset(data, 0, DATA_SIZE);
    regenerate_key(MASTER_KEY_INDEX, MAX_KEY_LEN);

    char cmd;
    unsigned int p1, p2;
    while(1) {
        read(0, &cmd, 1);
        if (cmd == 'l') {
            load_data();
        } else if (cmd == 'e') {
            encrypt();
        } else if (cmd == 'r') {
            read(0, &p1, sizeof(p1));
            read(0, &p2, sizeof(p2));
            regenerate_key(p1, p2);
        } else if (cmd == 'k') {
            read(0, &p1, sizeof(p1));
            load_key(p1);
        }
    }
}
```

Zgodnie z tytułem zadania mamy do czynienia z serwisem pozwalającym zdalnie szyfrować wiadomości szyfrem symetrycznym. W swojej pamięci trzyma on 16 losowych, nieznanych nam kluczy, spośród których możemy wybierać w procesie szyfrowania.

Sposób działania programu:

1. wygeneruj 16 różnych kluczy,
2. wczytaj flagę z pliku do zmiennej data,
3. załaduj pierwszy klucz do zmiennej current_key,
4. zaszyfruj data kluczem z current_key i zwróć szyfrogram użytkownikowi,
5. wyzeruj data,
6. zresetuj pierwszy klucz.

Następnie serwis pozwala nam:

- » ustawić dowolny klucz na losowy o zadanej długości (regenerate_key),
- » wczytać do data nasz dowolny 16 znakowy tekst (load_data),
- » załadować dowolny klucz do current_key (load_key),
- » zaszyfrować data kluczem z current_key (encrypt).

Aby łatwiej było nam później napisać eksplota, stworzymy na początku program implementujący całą komunikację z serwerem:

```
# -*- coding: utf-8 -*-
# pwntools - najlepsza biblioteka do CTF-ów
from pwn import remote, p32

NUM_KEYS = 16
MAX_KEY_LEN = DATA_SIZE = 16
KEY_SIZE_LEN = 1
ENTRY_SIZE = NUM_KEYS + KEY_SIZE_LEN
STORAGE_SIZE = ENTRY_SIZE * NUM_KEYS

server = remote('adres_serwera', 1337)

def regenerate_key(index, length):
    server.send('r')
    server.send(p32(index))
    server.send(p32(length))

def load_key(index):
    server.send('k')
    server.send(p32(index))

def encrypt():
    server.send('e')
    return server.read(DATA_SIZE)

def load_data(data):
    assert len(data) == DATA_SIZE
    server.send('l')
    server.send(data)

if __name__ == '__main__':
    ciphertext = server.read(DATA_SIZE)
    print ciphertext
```

Flaga jest szyfrowana algorytmem AES-128-ECB. Szyfrogram i tekst jawnny mają dokładnie 16 znaków, czyli jeden blok. W dodatku flaga została usunięta z pamięci serwisa. Takie założenia nie dają nam żadnego wektora ataku na sam sposób szyfrowania. Aby odszyfrować flagę, musimy w jakiś sposób odzyskać klucz.

Nasuważą się podstawowe pytania, które zawsze musimy sobie zadać: czy źródło losowości jest przewidywalne? Czy możemy coś zakładać o przyszłych kluczach? Odpowiedzi niestety są przeczące – /dev/urandom z naszego punktu widzenia daje prawdziwie losowe wyniki.

Zastanówmy się, co w tej chwili wiemy o kluczu, którym została zaszyfrowana flaga, i co możemy potencjalnie z nim zrobić. Na pierwszy rzut oka wydawałoby się, że został on utracony, ale na szczęście tak nie jest – został jeszcze w zmiennej current_key.

Co możemy zrobić z głównym kluczem? W tej chwili nic sensownego, tylko użyć go do szyfrowania naszego tekstu. Co możemy zrobić z resztą kluczy? Możemy je zresetować albo załadować. Warto zwrócić uwagę na to, jak są one przechowywane – trzymane

są w tablicy keys, będącej ciągłym obszarem pamięci.

Po dokładnej analizie funkcji regenerate_key zauważymy dwie bardzo ciekawe linie kodu:

```
unsigned int offset = index * ENTRY_SIZE;
keys[offset + MAX_KEY_LEN] = len;
```

Ponieważ offset i index są zmiennymi typu unsigned int – ten typ jest 32-bitowy – są one ograniczone: mogą przyjmować jedynie wartości z przedziału $[0; 2^{32}-1]$, co powoduje, że wynik mnożenia zmiennej index i ENTRY_SIZE może wyjść poza zakres (gdy index będzie większe od $2^{32}/\text{ENTRY_SIZE}$).

Tutaj potrzebujemy pewnego matematycznego twierdzenia.

Tożsamość Bézouta⁶ mówi nam, że dla niezerowych liczb całkowitych a oraz b o największym wspólnym dzielniku d istnieją liczby całkowite x oraz y spełniające równanie $a*x + b*y = d$, a powne x i y możemy otrzymać za pomocą rozszerzonego algorytmu Euklidesa⁷.

ENTRY_SIZE == 17 i 2^{32} są względnie pierwsze (ich największy wspólny dzielnik jest równy 1), co po podstawieniu do wzoru oznacza istnienie takich x i y, że $\text{ENTRY_SIZE}*x + 2^{32}*y == 1$. Wykonując modulo 2^{32} stronami, możemy przekształcić tę postać na: $\text{ENTRY_SIZE}*x \bmod 2^{32} == 1$. Takie x nazywamy odwrotnością modularną i do jej wyznaczania użyjemy już gotowego algorytmu: funkcji invert z biblioteki gmpy2.

Skoro umiemy znaleźć nasz x, to bardzo łatwo możemy nadpisać prawie dowolne komórki tablicy keys wartością parametru len.

Niestety linijka:

```
read(fd, &keys[offset], len);
```

nadpisuje nam dokładnie len komórek, ale nic nie stoi na przeszkodzie, abyśmy nadpisywali klucze zerami (gdy len == 0).

```
from gmpy2 import invert
def set_null(key, index):
    # Procedura zeruje index-bajt w kluczu o numerze key.
    UINT_MAX = 2**32 # De facto max+1
    offset = (key * ENTRY_SIZE + index)
    new_offset = (invert(ENTRY_SIZE, UINT_MAX) * (offset - MAX_KEY_LEN)) % UINT_MAX
    # Sanity check
    assert (new_offset * ENTRY_SIZE + MAX_KEY_LEN) % UINT_MAX == offset
    assert (new_offset * ENTRY_SIZE) % UINT_MAX <= STORAGE_SIZE - ENTRY_SIZE
    regenerate_key(new_offset, 0)
```

Przyjrzyjmy się funkcji load_key, a konkretnie poniższej linijce:

```
memcpy(current_key, &keys[offset], key_len);
```

Gdy ładowany klucz jest krótszy niż MAX_KEY_LEN, tylko część current_key zostanie nadpisana, co w połączeniu z naszą świeżo nabityą zdolnością zerowania każdego bajtu klucza pozwala nam nadpisać dowolny prefix naszej zmiennej current_key zerami.

```
def nullify_current_key_prefix(prefix_len, tmp_key=1):
    regenerate_key(tmp_key, prefix_len)
    for i in xrange(prefix_len):
        set_null(tmp_key, i)
    load_key(tmp_key)
```

Cała ta wiedza jest już praktycznie wystarczająca, aby odzyskać

⁶ https://pl.wikipedia.org/wiki/To%C5%BCCsamo%C5%9B%C4%87_B%C3%A9zouta

⁷ https://pl.wikipedia.org/wiki/Algorytm_Euklidesa#Rozszerzony_algorytm_Euklidesa

klucz.

Za pomocą funkcji load_data i encrypt możemy szyfrować dowolne teksty o długości 16 znaków. Oczywiście do tej operacji zostanie użyty current_key, nad którym właśnie zyskaliśmy częściową kontrolę.

Wyzerujmy pierwsze n-1 bajtów klucza i zaszyfrujmy nim wybrany tekst. Teraz możemy lokalnie znaleźć ostatni bajt przez sprawdzenie wszystkich 256 możliwości i porównanie ich wyników z wcześniej uzyskanym szyfrogramem (patrz Rysunek 7).

Klucz	Szyfrogram
00 00 00 00 00 00 00 ?? 6F 0F 86 0B B1 60 75	551a3f70dbb286e0c70e3740a18b8293
00 00 00 00 00 00 00 00 6F 0F 86 0B B1 60 75	a0ecc9bfb336cb0cec14d5cf59dc54
00 00 00 00 00 00 00 01 6F 0F 86 0B B1 60 75	b7aa92f6970e924de739698d1bf7dc0b
⋮	⋮
00 00 00 00 00 00 00 EB 6F 0F 86 0B B1 60 75	551a3f70dbb286e0c70e3740a18b8293

Rysunek 7. Wybór pojedynczego bajtu klucza

Po odgadnięciu jednego znaku przesuwamy się rekurencyjnie w stronę początku klucza i w analogiczny sposób zgadujemy pozostałe bajty (patrz Rysunek 8).

↑	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??
rekurencja	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ?? ?? 75
⋮	⋮
	00 ?? 3D EF EF 96 C1 14 EB 6F 0F 86 0B B1 60 75
	?? ?? 3D EF EF 96 C1 14 EB 6F 0F 86 0B B1 60 75

Rysunek 8. Kierunek łamania klucza

```
def calculate_key(plaintext, n=0):
    nullify_current_key_prefix(n)
    if n == MAX_KEY_LEN:
        assert encrypt() == AES.new('\0'*n).encrypt(plaintext)
        return ''
    encrypted = encrypt()
    recovered_key = calculate_key(plaintext, n+1)
    for k in xrange(256):
        key_candidate = '{}{}'.format(chr(k), recovered_key).
        rjust(MAX_KEY_LEN, '\0')
        if AES.new(key_candidate).encrypt(plaintext) == encrypted:
            return str(chr(k)) + recovered_key
```

Sam main ostatecznie wygląda następująco:

```
if __name__ == '__main__':
    ciphertext = server.readn(DATA_SIZE)
    plaintext = 'A' * DATA_SIZE
    load_data(plaintext)
    key = calculate_key(plaintext)
    print AES.new(key).decrypt(ciphertext)
```

A nasza flaga to: Drgns{f2Br#@!#d}

Krzysztof "sasza" Stopczarski, Robert "rodbert" Tomkowski



Rozwiązania zadań C64 i Encryption service zostały nadesłane przez Codilime – polską drużynę CTF-ową złożoną z pracowników firmy Codilime, obecnie na 12. miejscu rankingu CTFTIME. <https://codisec.com/>.

*Włamiemy się legalnie do Twojego systemu,
zanim nielegalnie zrobią to inni*

BEZPIECZEŃSTWO SYSTEMÓW IT
SECURITUM



BEZPIECZEŃSTWA

aplikacje webowe | certyfikowani

aplikacje mobilne | pentesterzy

sieci | OSCE | OSCE

socjotechnika | CISSP | CEH

PRZESZŁO 150 TESTÓW BEZPIECZEŃSTWA
REALIZOWANYCH ROCZNIE

SECURITUM.PL/OFERTA/

Przygoda z testami UX nie tylko dla stron WWW i aplikacji

Ostatnio kolega, który prowadzi firmę zajmującą się UX w Krakowie, zapytał mnie, czy mógłbym mu pomóc, biorąc udział w testowaniu prototypu gadżetu dla kuchni. W związku z tym, że pracuję w agencji interaktywnej, jestem zaznajomiony z UX. Na co dzień obserwuję, jak się tworzy user stories, przygotowuje wireframe, projektuje odpowiednie IA i tworzy user journey na podstawie zdefiniowanych potrzeb użytkownika. Tym razem miałem okazję do doświadczenia UX testów w innym kontekście – hardware.

Jako wielbiciel UX, gadżetów i gotowania byłem pewny, że przypadnie mi do gustu testowanie nowego urządzenia. Oto, czego się nauczyłem:

1. Upewnij się, że użytkownik czuje się komfortowo

Zakomunikuj testerowi miejsce i czas testów. W moim przypadku testy odbywały się na obrzeżach miasta, gdzie ciężko było się dostać. Jeśli jest to możliwe, uzgodnij z testerem, kto pokryje koszty dojazdu.

Kiedy dojechałem, poczułem się komfortowo i swobodnie, gdy pokazano mi urządzenie i zaproszono do zbadania go i poznania jego funkcji. Poproszono mnie o głośne komentowanie tego, co widzę i robię. Zadanie wydało się być proste, ponieważ z natury głośno myśle!

2. Nie można zastąpić testowania przez prawdziwego użytkownika czymkolwiek innym

Mówiąc „testowanie przez prawdziwego użytkownika”, chciałbym podkreślić, że użytkownik biorący udział w teście powinien być rekrutowany na podstawie podobieństwa profilu rzeczywistego użytkownika.

Zatem jeśli będziemy testować np. stronę dla szkoły, to powinny zostać przetestowane osoby z dziećmi w wieku szkolnym. Jeśli strona została zaprojektowana tak, aby przyciągnąć uwagę rodziców, którzy rozważają wybór tej szkoły, to kolejny test powinien zostać przeprowadzony na rodzicach, których dzieci nie są jeszcze w wieku szkolnym. Jeśli testy wykazują, że obie grupy użytkowników mogą korzystać z serwisu i wykonać to, czego potrzebują, to znaczy, że test użyteczności sprawdził założenia i rozwiązania przeprowadzone podczas projektowania i budowania strony.

Jeśli natomiast istnieją problemy, zakończone testy powinny dostarczyć nam odpowiednich wskazówek do projektowania i budowania portalu oraz wyjaśnić, które problemy wymagają zmian i jak ich dokonać.

Budżety i naciski czasowe mogą prowadzić do pomijania lub ignorowania testów użyteczności. Niektórzy klienci myślą, że projektant „jest ekspertem i z góry powinien wiedzieć, jak tworzyć użyteczne produkty”. W pewnym stopniu jest to słusne. Do użyskania właściwego UX wykorzystywane są wszystkie umiejętności projektanta, ale w wielu sytuacjach nie ma idealnej odpowiedzi

i to, co może działać dobrze dla jednej publiczności w jednym projekcie, może sprawiać kłopoty innej grupie odbiorców.

Przeprowadzenie udanego UX testu na prawdziwych użytkownikach może być naprawdę cennym sposobem na usunięcie osobistej opinii z decyzji projektowych.

Zadanie, które zostało mi powierzone, było bardzo proste i możliwe, że nie oddawało rzeczywistych możliwości urządzenia. Jednak przeprowadzenie testu w prawdziwym mieszkaniu, prawdziwej kuchni sprawiło, że poczułem się, jakbym używał tego urządzenia w rzeczywistym scenariuszu.

Jedną z pierwszych rzeczy, jaką mi powiedziano, było „nie testujemy twoich możliwości jako kucharza, jesteśmy zainteresowani twoją szczerą opinią na temat produktu, którego będziesz używać”. Przypominam sobie podobne powiedzenie w testach użyteczności witryny, które przeprowadziłem kilka lat wcześniej. Trudno mi było nie pokazać moich umiejętności kuchennych i czułem się zakłopotany, gdy nie zauważałem ikony, która przeniosła mnie do następnego etapu testu i skończyło się tak, że musiano mi o tym kilka razy przypomnieć „To nie jest intuicyjne” było zdaniem, które kilkakrotnie powtarzałem.

3. Upewnij się, że co najmniej dwie osoby są zaangażowane w przeprowadzenie testu

We wczesnych dniach mojego nieformalnego testowania użyteczności pamiętam, jaki przeżywałem koszmar, gdy musiałem zanotować to, co użytkownik mówił, jednocześnie zachęcając go do podjęcia innego scenariusza. Posiadanie kolegi, który zajmowałby się wszystkim innym, niż mówienie do testera, sprawiłoby realną różnicę i ułatwiłoby znacznie pracę pomiędzy testerem a asystentem.

4. Zanim rozpoczęcie się test, upewnij się, że to, co ma być testowane, rzeczywiście dobrze działa

Kilka razy musielismy przerwać sesję, ponieważ urządzenie przestało działać. Rozwiązywanie – technik z sąsiedniego pokoju podszedł do urządzenia, wyłączył zasilanie i włączył je ponownie, a potem popatrzył i powiedział „znowu działa!”. Próbowałem sam tej sztuczki przez godzinę, ale straciłem przez to czas, żeby wrócić do odpowiedniej pozycji w testach – był to potencjalnie kolejny użyteczny feedback dla zespołu, który zakończył test!

Nie jestem pewien, czemu urządzenie przestało działać, ale z całą pewnością przeszkodziło to w płynności przeprowadzania

testu i jako nowemu użytkownikowi dało mi dużo mniej zaufania dla urządzenia. Moje późniejsze komentarze były skażone „czasem to tylko nie działa...”. Przypuszczam, że miałem pecha, że urządzenie podczas testu się psuło, ale nie spytałem, czy miało to miejsce w przypadku innych testerów.

Kiedy zajmowałem się przygotowaniem testów użyteczności, prosiłem współpracowników o ich wykonanie i ukończenie, abym miał pewność, że scenariusze, które stworzyłem, są realistyczne i możliwe. Dlatego każdemu radzę: przetestuj swoje testy przed testem!

5. Rozpoznanie swoich błędów poznawczych (ang. cognitive bias) jest pierwszym krokiem do rozwiązywania ich ewentualnych negatywnych skutków

W teście gadżetu kuchennego agencja mojego przyjaciela testująca urządzenie była całkowicie niezależna od firmy/projektantów urządzenia. Nie mieli żadnego interesu w tym, czy urządzenie działa, czy nie. Jeśli testujemy własne projekty i rozwiązania, istnieje zagrożenie, że potencjalne nieplanowane zmiany produktu mogą pociągnąć za sobą koszty. W efekcie możemy błędnie wybrać ignorowanie pewnych zaobserwowanych wad podczas testów. Co gorsza, nawet nie zauważymy wady lub spróbujemy ją ukryć!



WILL PARKER

Head of Project Management w The Cogworks. Doświadczony Digital Project Manager, prowadzący projekty w sektorze komercyjnym, rządowym i trzecim (organizacji pozarządowych). Specjalizujący się w zwinnym zarządzaniu (Agile), zarządzaniu treścią (CMS) oraz UX. Do jego codziennych obowiązków należy także account management. Jest pasjonatem technologii w spoteczeństwie, stara się zrozumieć więcej na temat unikalnej przestrzeni, jaka powstaje w interakcji człowiek-komputer.

reklama

Oczywiście wszystko zależy od tego, jaki mamy rodzaj kontraktu z klientem... i jeśli wpadniemy w opisaną powyżej pułapkę i zaczniemy ignorować wady, to istnieje duże prawdopodobieństwo, że klient już do nas nie wróci!

Drogą wyjścia jest zaakceptowanie swoich błędów poznawczych, a następnie zminimalizowanie ich skutków. Jako projektanci aplikacji jesteśmy otwarci na pokazanie projektów prawdziwym użytkownikom – i oczywiście na otrzymanie feedbacku, który może wpływać na uzyskanie lepszego efektu końcowego.

ZaUXujmy razem!

W swojej pracy szukam ciekawych projektów i klientów, którzy wykorzystaliby umiejętności i doświadczenie UX, jakie mój zespół posiada. Z punktu widzenia uczestnika testowania UX jest zabawa, której można być częścią – zwłaszcza jeśli jesteś podobny do mnie i dość stanowczy, jeśli chodzi o nowe technologie. Z punktu widzenia klienta lub projektanta testy UX mogą być istotnym punktem w projekcie, w którym podejmowane są decyzje ostatecznie wpływające na końcowy wynik.

A teraz zadaj sobie pytania – kiedy ostatnio testowałeś swój nowy projekt na kilku prawdziwych użytkownikach? Czy nauczyłeś się czegoś cennego? A może dzięki testom znalazłeś lepsze rozwiązanie?

Obserwuj nas na Twitterze



@PROGRAMISTAMAG



Kurs angielskiego dla programistów.

Lekcja 5

Przedstawiam piątą lekcję minikursu angielskiego dla programistów. Tym razem tematem przewodnim jest język PHP. Zachęcam do wielokrotnego wykonywania ćwiczeń, aby dobrze utrwalić sobie przyswojony materiał. Rozwiązania do ćwiczeń zamieszczono na stronie internetowej, której adres podano na dole artykułu.

INTRODUCTION TO PHP

PHP is a server-side scripting language **designed by** Rasmus Lerdorf in 1994 primarily for web development but also used as a **general-purpose programming language**.

PHP code may be embedded into HTML markup, or it can be **used in combination with** various web **template systems**, **web content management systems** and web frameworks. PHP code is usually **processed** by a PHP **interpreter implemented** as a module in the web server or as a Common Gateway Interface (CGI) **executable**. The web server software combines the results of the interpreted and **executed** PHP code.

The PHP interpreter only executes PHP code **within its delimiters**. Anything **outside** its delimiters is not processed by PHP. The most common delimiters are <?php to open and ?> to close PHP sections. This shortened delimiter **makes script files less portable**, since support for them can be **disabled** in the local PHP **configuration**.

STORING DATA AND DATA TYPES

Variables are **prefixed** with a **dollar symbol**, and a **type** does not **need to be** specified **in advance**. Unlike **function** and **class** names, variable names are **case sensitive**. Both **double-quoted** ("") and heredoc **strings provide the ability to interpolate** a variable's **value** into the string. PHP treats **newlines** as **whitespace**, and **statements are terminated by a semicolon**. PHP has three types of **comment** syntax: /* */ marks **block and inline comments**; // as well as # are used for **one-line comments**. The echo statement is one of several **facilities** PHP provides to **output** text, e.g., to a web browser.

PHP **stores integers** in a **platform-dependent** range, either a **64-bit or 32-bit signed integer** equivalent to the C-language **long type**. Integer variables can be assigned using **decimal** (positive and negative), **octal**, **hexadecimal**, and **binary** notations.

Floating-point numbers can be specified using floating point notation, or two forms of **scientific notation**. PHP has a native **Boolean type**. Using the Boolean type conversion rules, **non-zero values are interpreted as true** and zero as false.

The null data type **represents** a variable that has no value; NULL is the only allowed value for this data type.

Tekst lekcji oparty na stronie (CC BY-SA 3.0 Unported License)
<https://en.wikipedia.org/wiki/PHP>

Ćwiczenia

1. Dopasuj słowa lub wyrażenia z lewej kolumny do słów lub wyrażeń z prawej. Potrafisz je wszystkie przetłumaczyć na język polski?

signed	management system
block/inline	dependent
Boolean	integer
double-	comment
scientific	quoted
general-purpose	type
to provide	notation
non-zero	programming language
platform-	the ability to
web content	value

2. Odpowiedz na poniższe pytania. Możesz cytować tekst. Powtarzaj, aż będziesz w stanie udzielić odpowiedzi bez patrzenia na tekst.

1. What is PHP?
2. Who designed PHP?
3. How can you use PHP?
4. What does the PHP interpreter execute?
5. How do you declare variables in PHP?
6. In what way do function and class names differ from variable names?
7. What ability do double-quoted and heredoc strings provide?
8. What types of comments does PHP have?
9. How can you specify floating-point numbers?
10. What does the null data type represent?

3. Utwórz zdania wg wzoru „Double-quoted strings provide the ability to interpolate a variable's value into the string”.

Przykład: heredoc strings + interpolate a variable's value into the string -> Heredoc strings provide the ability to interpolate a variable's value into the string.

1. variables + store values in the memory.
2. dollar symbol + declare variables
3. double-quotes + define strings
4. semicolon + terminate statements
5. the syntax /* */ + insert comments into the code
6. PHP + use floating-point values

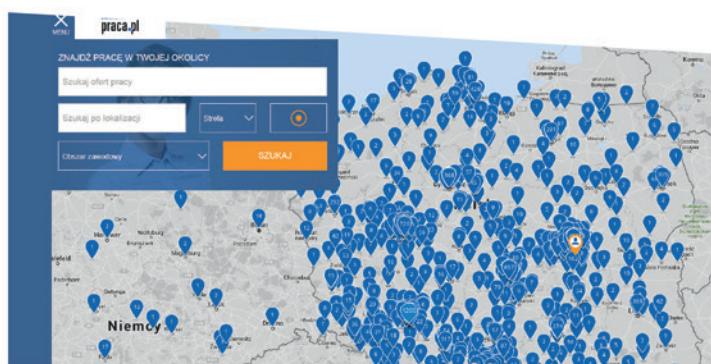
Odpowiedzi

Odpowiedzi do ćwiczeń znajdują się na stronie <http://shebang.pl/programista-minikurs>.



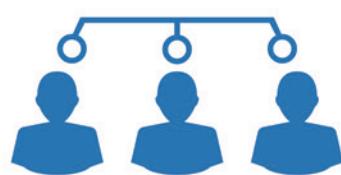
Chcesz dobrze zarobić?

Na Praca.pl codziennie znajdziesz ponad 3 000 ofert pracy z obszaru IT i nowe technologie



Znajdź pracę w Twojej okolicy

Lokalna.praca.pl



Poleć znajomego do pracy
i zgarnij 1 000 zł

[Praca.pl/rekomendacje.html](#)

SŁOWNIK

64-bit/32-bit signed integer – całkowita 64-/32-bitowa liczba ze znakiem	output – wyjście, zwracać
binary – binarny, dwójkowy	outside – na zewnątrz, poza
block/inline comment – komentarz blokowy/liniowy	platform-dependent – zależny od platformy
Boolean type – typ logiczny, typ boole'owski	portable – przenośny
case sensitive – rozróżniający wielkość liter	positive – dodatni
class – klasa	prefix – przedrostek
comment – komentarz	scientific notation – notacja naukowa
configuration – konfiguracja	statement – instrukcja
decimal – dziesiętny, decymalny	string – łańcuch
delimiter – ogranicznik	template system – system szablonów
designed by – zaprojektowany przez	terminated by a semicolon – zakończony średnikiem
dollar symbol – symbol dolara	to disable – wyłączyć
double-quoted – w podwójnym cudzysłowie	to execute – wykonywać
executable – plik wykonywalny	to implement – zaimplementować
facility – narzędzie	to interpolate – interpolować
floating-point number – liczba zmiennoprzecinkowa	to interpret sth as – interpretować coś jako
function – funkcja	to need to – musieć
general-purpose programming language – ogólny język programowania	to process – przetwarzanie
hexadecimal – szesnastkowy, heksadecymalny	to provide the ability to – umożliwiać
in advance – z góry	to represent – reprezentować
integer – liczba całkowita	to store – przechowywać
interpreter – interpreter	type – typ
long type – typ długi	unlike – w odróżnieniu
negative – ujemny	use in combination with – używać w połączeniu z
newline – nowy wiersz	value – wartość
non-zero value – wartość różna od zera	variable – zmienna
octal – ósemkowy, oktalny	web content management system – internetowy system zarządzania treścią
one-line comment – komentarz liniowy	whitespace – białe znaki
	within – wewnętrz, w obrębie

**ŁUKASZ PIWKO**

piwko.lukas@gmail.com

Tłumacz angielskiej i francuskiej literatury programistycznej z około 70 książkami na koncie, nauczyciel, wykładowca i maniak technologii programistycznych.

PRENUMERATA

Zamów prenumeratę magazynu Programista
przez formularz na stronie:

<http://programistamag.pl/typy-prenumeraty/>

lub zrealizuj ją na podstawie faktury Pro-forma. W spawie faktur Pro-forma prosimy kontaktować się z nami drogą mailową:
redakcja@programistamag.pl.

Prenumerata realizowana jest także przez RUCH S.A.

Zamówienia można składać bezpośrednio na stronie: www.prenumerata.ruch.com.pl

Pytania prosimy kierować na adres e-mail: prenumerata@ruch.com.pl

lub kontaktując się telefonicznie z numerem:

801 800 803 lub 22 717 59 59, godz. 7:00 – 18:00 (koszt połączenia wg taryfy operatora).

Magazyn Programista wydawany jest przez Dom Wydawniczy Anna Adamczyk

Wydawca/Redaktor naczelny: Anna Adamczyk (annaadamczyk@programistamag.pl).

Redaktor prowadzący: Michał Leszczyński (mlesczynski@programistamag.pl).

Korekta: Tomasz Łopuszański. **Kierownik produkcji:** Havok. **DTP:** Havok.

Dział reklamy: reklama@programistamag.pl, tel. +48 663 220 102, tel. +48 604 312 716.

Prenumerata: prenumerata@programistamag.pl.

Współpraca: Michał Bartylewski, Mariusz Sieraczkiewicz, Dawid Kaliszewski, Marek Sawerwain, Łukasz Mazur, Łukasz Łopuszański, Jacek Matulewski, Sławomir Sobótka, Dawid Borycki, Gynael Coldwind, Bartosz Chrabski, Rafał Kocisz, Michał Sajdak, Michał Bentkowski, Mariusz „maryush” Witkowski, Paweł „KrazaQ” Zakrzewski.

Adres wydawcy: Dereniowa 4/47, 02-776 Warszawa.

Druk: Drukarnia Edit ul. Dworcowka 2, 05-462 Wiązowna, Nakład: 4500 egz.

Nota prawna

Redakcja zastrzega sobie prawo do skrótów i opracowań tekstów oraz do zmiany planów wydawniczych, tj. zmian w zapowiadanych tematach artykułów i terminach publikacji, a także nakładzie i objętości czasopisma.

O ile nie zaznaczono inaczej, wszelkie prawa do materiałów i znaków towarowych/firmowych zamieszczanych na łamach magazynu Programista są zastrzeżone. Kopiowanie i rozpowszechnianie ich bez zezwolenia jest zabronione.

Redakcja magazynu Programista nie ponosi odpowiedzialności za szkody bezpośredni i pośredni, jak również za inne straty i wydatki poniesione w związku z wykorzystaniem informacji prezentowanych na łamach magazynu Programista.



Szkolenia i warsztaty eksperckie - **bo umysł to Twoje najważniejsze narzędzie**



DDD



ARCH



TEST&CRAFT



AGILE&SOFT



JAVA



.NET



C&CPP



WEB



BAZY



MOBILNE



EIP

SPRAWDŹ **200**
AUTORSKICH
PROGRAMÓW
SZKOLEŃ



CONTMAN

www.contman.pl

Praca z możliwościami

SPRAWDŹ NASZE OFERTY PRACY



NORMAL

Developer

7 000 - 9 000 PLN + VAT

SENIOR

Developer

9 000 - 13 000 PLN + VAT

CO OFERUJEMY



atrakcyjne
wynagrodzenie



kursy
i szkolenia



biuro w centrum
Poznania



karnet
OK System



kawę i owoce
na co dzień



imprezy
integracyjne