

Magazyn programistów i liderów zespołów IT



JAK SCHWYTAĆ ZŁODZIEJA DANYCH

**UNITY – WPROWADZENIE****MATEMATYKA GRAFIKI 2D I 3D****NOWOŚCI W BIBLIOTECE GTK+****VUE.JS – CHWILOWA MODA CZY
DOJRZAŁY FRAMEWORK?**



Join our team of Makers!
career.cybercom.com

Change
tomorrow
with us

www.cybercom.pl

CYBERCOM
GROUP

Cybercom Poland Sp. z o.o.

ul. Hrubieszowska 2, 01-209 Warszawa / ul. Składowa 35, 90-127 Łódź / ul. Unii Lubelskiej 4c, 85-059 Bydgoszcz

Po co mi matematyka?

To pytanie bardzo często zadają sobie osoby odkrywające tajniki branży IT. Wbrew pozorom jest ono zasadne, ale odpowiedź w stylu „bo potrzebna” nie jest satysfakcjonująca w jakimkolwiek stopniu. Możecie zapytać, czemu przytaczam to pytanie w tytule? Ano tak się złożyło, że w aktualnym numerze znajdziecie wyjątkowo dużo wzorów matematycznych.

Nie, nie zamierzamy rozpoczęć kursu „Matematyka w branży IT” prowadzonego, jak to było modne swego czasu, w formie korespondencyjnej :). Po prostu nieprzytoczenie tych wzorów matematycznych bardzo mocno obniżyłoby poziom merytoryczny tekstu, bowiem w tym numerze mamy dla Was „mini kompendium” wiedzy matematycznej dotyczącej programowania grafiki 2D i 3D, którą znajdziecie w artykule Wojtka Sury pt. „Matematyka grafiki 2D i 3D”.

Kolejnym tekstem, który może przy tych temperaturach przerazić co słabsze jednostki wzorami, jest bardzo interesujący, poruszający znany już z naszych łamów temat steganografii od nieco innej strony. Jednak nie samą teorią programista żyje, artykuł zawiera prostą, w pełni działającą implementację omawianych zagadnień, a skrywa się pod tytułem „Jak schwytać złodzieja danych” autorstwa Jarosława Jedyńskiego.

Oczywiście w numerze znajdziecie również inne interesujące teksty, np. omawiający zmiany w jednej z najstarszych bibliotek do tworzenia GUI, będącej głównym składnikiem programu The GIMP, znanej bibliotece GTK+. Marek Sawerwain w artykule „Nowości w bibliotece GTK+” w przystępny i przejrzysty sposób podsumowuje aktualnie wprowadzone w niej zmiany.

Czy pamiętacie, skąd wzięło się u Was zainteresowanie programowaniem? Założę się, że spora część przyszła do tego tematu, bo chciała stworzyć własną grę, jednak złożoność zagadnień pojawiających się przy tym niejedną osobę przypisnęła o ból głowy. Obecnie producenci ułatwiają to zadanie na wiele sposobów. Począwszy od generatorów szkieletów gotowych gier implementujących określone zachowania, skończywszy na kompletnych środowiskach służących do ich tworzenia. Tym drugim rodzajem oprogramowania udostępnionego przez producenta zajmuje się Wojtek Sura w swoim drugim artykule pod tytułem „Unity – wprowadzenie”.

Mamy nadzieję, że upały, jakie ostatnio doświadczają wszyscy, nie wpłyną na Waszą chęć zapoznania się z aktualnym numerem, a czas spędzony nad lekturą tekstu przygotowanych w tym wydaniu pobudzi kreatywność i być może zainspiruje do stworzenia własnego projektu życia.

Mariusz „maryush” Witkowski

BIBLIOTEKI I NARZĘDZIA

Nowości w bibliotece GTK+	4
<i>Marek Sawerwain</i>	

PROGRAMOWANIE APLIKACJI WEBOWYCH

Vue.js – chwilowa moda czy dojrzały framework?	10
<i>Igor Podlaski</i>	

PROGRAMOWANIE GIER

Matematyka grafiki 2D i 3D	20
<i>Wojciech Sura</i>	

Unity – wprowadzenie	36
<i>Wojciech Sura</i>	

BEZPIECZEŃSTWO

Jak schwytać złodzieja danych	50
<i>Jarosław Jedyński</i>	

KLUB LIDERA IT

Współczesne architektury aplikacji biznesowych. Ports and Adapters oraz microservices	58
<i>Mariusz Sierackiewicz</i>	

Wdrożenie Agile z perspektywy managera	64
<i>Tomasz Dutkiewicz</i>	

STREFA CTF

Confidence 2018 – KrkAnalytica	68
<i>Paweł Łukasik</i>	

KLUB DOBREJ KSIĄŻKI

Czysta architektura	70
<i>Rafał Kocisz</i>	

reklama

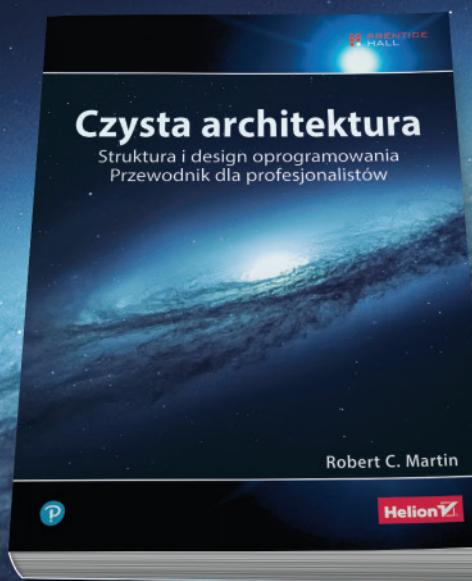


NOWA książka autora „Czystego kodu”!

SKORZYSTAJ z RABATU 25%
na tytuły ROBERTA C. MARTINA!

KOD:
HEL25

Książkę i eBook ZNAJDZIESZ na <http://helion.pl>



Nowości w bibliotece GTK+

Jeśli zapytamy o najbardziej znane projekty ruchu Open Source, to w odpowiedzi usłyszmy o jądrze systemu Linux, programie LibreOffice, kompilatorze GCC czy ostatnio mocno odświeżonym programie GIMP. Ten ostatni jest też interesujący z innego powodu, a mianowicie biblioteki do tworzenia interfejsu użytkownika, o którą oparto cały program GIMP. A jest nią pakiet GTK+, obok QT jedna z podstawowych bibliotek do tworzeniu GUI. W niedalekiej przyszłości możemy się spodziewać wydania wersji o numerze 4.0, toteż warto przedstawić bibliotekę GTK+, aby sprawdzić, jakie zmiany oraz nowości zostały przygotowane w ramach nadchodzącej nowej wersji GTK+.

Pakiet GTK+ to wieloplatformowe (dostępne są wersje dla systemów Windows, macOS, a także Linux) rozwiązanie wspierające tworzenie graficznych interfejsów użytkownika (w skrócie GUI). GTK+ przeszła dłuża drogę, bowiem pierwsza wersja datuje się na kwiecień 1998, czyli minęło już 20 lat w rozwoju tej biblioteki. Warto podkreślić, iż przez te lata nie zmieniło się jedno z pierwotnych założeń, jakie jest charakterystyczne dla pakietu GTK+. Od początku do dzisiaj pakiet jest tworzony z myślą o wykorzystaniu języka C, ale mimo to biblioteka GTK+ posługuje się modelem obiektowym. Istnieją naturalnie dodatkowe rozwiązania dla GTK+ oferujące wsparcie dla C++, Pythona czy Javy (kilka dodatkowych uwag na ten temat znajdziemy w ramce pt. „GTK+ i inne języki programowania”). Jednakże to C jest podstawowym językiem, w jakim jest implementowana całość biblioteki GTK.

GTK+ i inne języki programowania

GTK+ z racji tego, iż zostało zaimplementowane za pomocą języka C, jest nieco łatwiejsze do przeniesienia na inne platformy sprzętowo-programowe, ale też ułatwia to przygotowanie portów API GTK+ dla innych języków programowania. Najważniejsze języki to m.in. C++, gdzie dostępna jest biblioteka GTKMM, oferująca zestaw klas do tworzenia aplikacji. Wspierane są języki D, Java, Python, a także przedstawiciele języków funkcyjnych, np. Haskel. Autorzy GTK+ oraz platformy GNOME pokusili się również o przygotowanie dodatkowego języka programowania o nazwie Vala bezpośrednio opartego o GTK+ i komponenty GNOME, aby ułatwić tworzenie aplikacji w ramach wymienionych rozwiązań.

Warto jeszcze dodać, że mamy obecnie trzy główne gałęzie GTK+, tj. wersję 2.x, 3.x i nadchodzącą 4.x. Nie wszystkie porty języków zostały przeniesione na wydanie 3.x, panuje tu niestety lekki bałagan. Trzeba jednak przyznać, że najważniejsze i najbardziej popularne języki programowania jak np. Python mają dobre i stabilne wsparcie.

PRZYKŁAD ZERO – PROGRAMOWANIE OBIEKTOWE W C

Omówienie GTK+, w tym także nowości i zmian, jakie wprowadzane są w wersji 4.0, rozpoczniemy od podstawowego przykładu, tj. okna głównego aplikacji. Wiele lat rozwoju biblioteki spowodowało, iż tego typu aplikacje możemy napisać na kilka sposobów, w zależności od tego, jak bardzo interesują nas wewnętrzne mechanizmy GTK+.

W Listingu 1 przedstawiono kod źródłowy przykładowu tworzącego okno z tytułem. Został napisany w nowym stylu obecnym w GTK+, ponieważ wykorzystujemy obiekt aplikacji GTK+, a dopiero po sygnale aktywacji tworzymy okno naszego przykładu.

Listing 1. Aplikacja numer zero, puste okno, ale z tytułem

```
#include <gtk/gtk.h>

static void create_main_win(GtkApplication* app, gpointer data) {
    GtkWidget *mwin = NULL;

    mwin = gtk_application_window_new( app );
    gtk_window_set_title( GTK_WINDOW (mwin), "Okno główne
    aplikacji" );
    gtk_window_set_default_size( GTK_WINDOW (mwin), 640, 480 );
    gtk_widget_show_all( mwin );
}

int main (int argc, char **argv) {
    GtkApplication *app;
    int status;

    app = gtk_application_new( "org.gtk.example.zero", G_
    APPLICATION_FLAGS_NONE );
    g_signal_connect ( app, "activate", G_CALLBACK (create_main_
    win), NULL );
    status = g_application_run( G_APPLICATION (app), argc, argv );
    g_object_unref( app );
    return status;
}
```

Ten krótki przykład ujawnia podstawowy styl programowania w GTK+, który nie zmienił się od początku istnienia tej biblioteki. Mamy zestaw funkcji do tworzenia kontrolek ekranowych (nazywanych oryginalnie widgetami – w dalszej części artykułu będziemy stosować to określenie) oraz zmiany ich stanów. Dodatkowo GTK+ ma wbudowany system sygnałów, np. `clicked`, czyli kliknięcie na przycisk. Obsługę sygnałów dołączamy w sposób dynamiczny, również w trakcie działającego programu.

W naszym przykładzie mamy tylko dwie funkcje: główną `main` oraz `create_main_win`. W funkcji głównej tworzymy obiekt aplikacji:

```
GtkApplication *app = gtk_application_new( "org.gtk.example.
zero", G_APPLICATION_FLAGS_NONE );
```

Do utworzonego obiektu należy podłączyć obsługę sygnału `activate`, odpowiedzialnego w naszym przypadku za utworzenie reszty elementów naszej aplikacji, czyli okna głównego:

```
g_signal_connect ( app, "activate", G_CALLBACK (create_main_
win), NULL );
```

W funkcji main pozostało uruchomić aplikację za pomocą funkcji `g_application_run`, powrót z tej funkcji oznacza naturalnie zakończenie działania aplikacji, zatem można nakazać zwolnienie zasobów obiektu app za pomocą funkcji `g_object_unref`. Przy czym trzeba tu dodać, iż wywołanie ostatniej funkcji oznacza obniżenie licznika referencji – dopiero gdy dojdzie on do zera, to można usunąć obiekt i zwolnić pamięć.

Zadanie funkcji `create_main_win` to utworzenie okna głównego aplikacji. Schemat postępowania jest podobny do budowy obiektu aplikacji:

```
GtkWidget *mwin = mwin = gtk_application_window_new( app );
```

Po utworzeniu obiektu możemy zmienić np. tytuł okna: `gtk_window_set_title`. Wolno nam zmienić rozmiary obiektu i na końcu należy zmienić stan widgetów na widoczne. Ponieważ widgety są ustawione zgodnie z kolejnością tworzenia, to funkcja `gtk_widget_show_all` nakaże ich wyświetlenie od okna głównego aplikacji po ostatni przyciski etykietę.

Pozostaje do wyjaśnienia jeszcze, jak skompilować plik źródłowy. Przykład z Listingu 1 nie wymaga stosowania najnowszej wersji GTK+, z repozytorium wystarczy wersja 3.x. Jeśli plik z Listingu 1 zostałby zapisany pod nazwą np. `gtk-example-zero.c`, to jego komplikacja byłaby następującą:

```
gcc `pkg-config --cflags gtk+-3.0` -o gtk-example-zero gtk-
example-zero.c `pkg-config --libs gtk+-3.0`
```

Kompilacja z wersją GTK+4.0 wymaga tylko zmiany numeru wersji. GTK+ oferuje bogaty zestaw kontrolek/widgetów, jednak my ograniczymy się tylko do uzupełnienia aplikacji o przycisk. W Listingu 2 przedstawiono fragmenty kodu tworzącego okno oraz przycisk. Utworzenie przycisku wymaga przygotowania obiektu zarządzającego jego rozmiarem, tj. menadżera układu, ale ponownie, jak przy tworzeniu okna, sprowadza się to do skorzystania z funkcji tworzącej odpowiedni obiekt:

```
button_box = gtk_button_box_new (GTK_ORIENTATION_HORIZONTAL);
```

Utworzony obiekt trzeba dodać do okna:

```
gtk_container_add (GTK_CONTAINER (window), button_box);
```

Następnie możemy utworzyć właściwy obiekt przycisku:

```
button = gtk_button_new_with_label ("Hello World");
```

Kolejne dwie linie z Listingu 2 podłączają obsługę zdarzeń. Pierwsza dotyczy reakcji na kliknięty przycisk, zostanie wywołana funkcja `print_hello`, natomiast druga funkcja `g_signal_connect_swapped` również podłącza obsługę sygnału, ale w nieco bardziej sprytny sposób. Funkcja `g_signal_connect` za argument przyjmuje widget przycisku, natomiast w tym drugim przypadku będzie to widget wskazany w trzecim parametrze funkcji `g_signal_connect_swapped`. Pozwala to na skrócenie kodu, w przypadku gdy chcemy wskazać inny widget niż ten, który emituje dany sygnał. W naszym przypadku nakazujemy usunięcie okna głównego.

Listing 2. Uzupełnienie aplikacji o przycisk

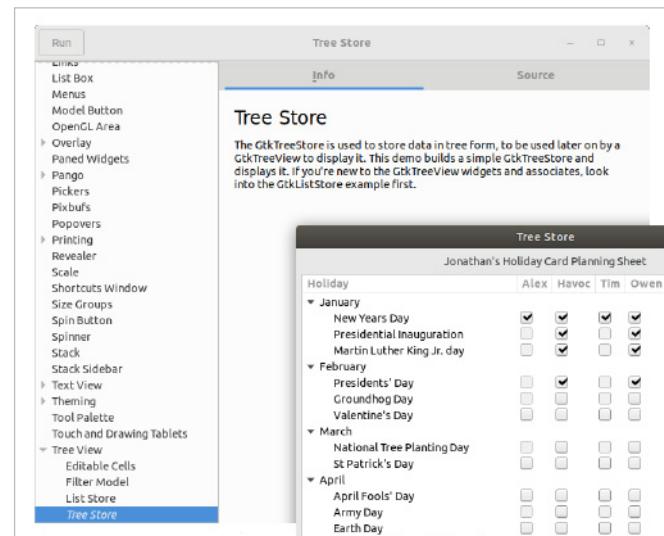
```
static void print_hello (GtkWidget *widget, gpointer data) {
    g_print ("Hello World !!!\n");
}
...
GtkWidget *window;
GtkWidget *button;
GtkWidget *button_box;

window = gtk_application_window_new (app);
gtk_window_set_title (GTK_WINDOW (window), "Window");
gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);

button_box = gtk_button_box_new (GTK_ORIENTATION_HORIZONTAL);
gtk_container_add (GTK_CONTAINER (window), button_box);

button = gtk_button_new_with_label ("Hello World");
g_signal_connect (button, "clicked", G_CALLBACK (print_hello),
NULL);
g_signal_connect_swapped (button, "clicked", G_CALLBACK (gtk_
widget_destroy), window);
gtk_container_add (GTK_CONTAINER (button_box), button);

gtk_widget_show_all (window);
```



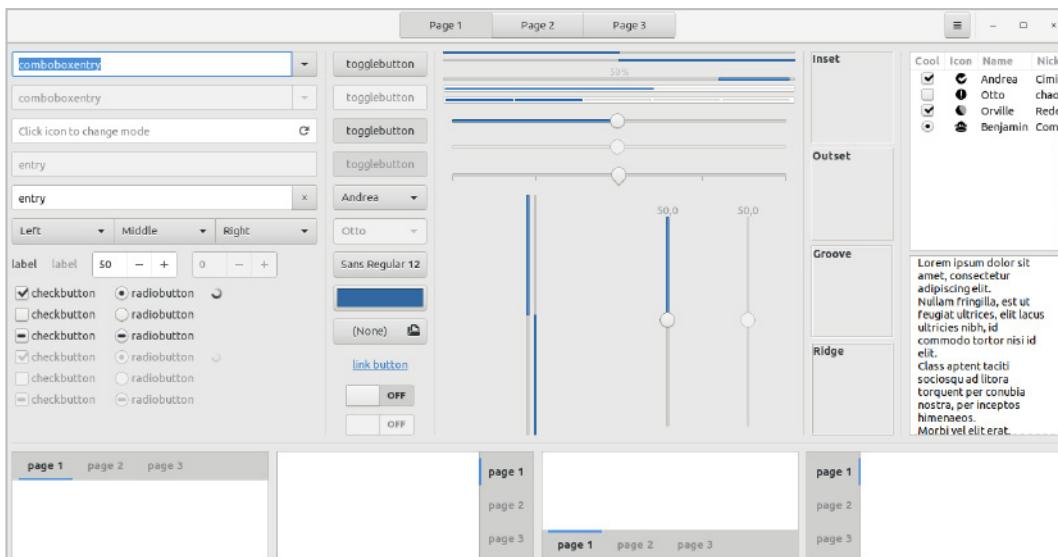
Rysunek 1. GtkDemo – prezentacja możliwości wielu kontrollek GTK+ wraz z kodem źródłowym (na rysunku kontrolka TreeView)

Warto jeszcze wspomnieć o programie GtkDemo. Przedstawia on wiele kontrollek GTK+ wraz z krótkim omówieniem oraz kodem źródłowym danego przykładu. Uruchomić warto także aplikację `GtkWidgetFactory` z Rysunku 2, przedstawia ona bowiem niemal wszystkie ważne kontrolki, jakie oferuje nam GTK+.

GTK+ – NAJWAŻNIEJSZE ZMIANY

Podstawową zmianą jest sposób komplikacji pakietu, który została opisany w ramce pt. „Kompilacja pakietu GTK+”. Choć może się wydawać, że brakuje innych zasadniczych innowacji, np. nowych widgetów, to zmiany, jakie zachodzą w GTK+, są więcej niż gruntowne i dotyczą jej wewnętrznych mechanizmów.

Najbardziej istotną zmianą jest oparcie GTK+ o API przeznaczono dla kart graficznych, tj. Vulkan czy OpenGL. Od wersji 4.0 całość grafiki pakietu GTK+ jest tworzona przez kartę graficzną, co będzie skutkować znaczącym wzrostem płynności, ale przede wszystkim daje możliwość wykorzystania karty graficznej do tworzenia różnego rodzaju efektów specjalnych. Zastosowanie takiego podejścia skutkuje tym, iż wszystkie operacje jak np. przesuwanie zawartości okna, rysowanie ikon itd. są realizowane przez GPU. Związane z tym faktem jest też zastosowanie backendu Wayland, który sys-



Rysunek 2. Przegląd komponentów GTK+ w aplikacji GtkWidgetFactory

Kompilacja pakietu GTK+

Pierwszą zmianą, jaką napotkamy na początku pracy z pakietem GTK+, jest zmieniony sposób komplikacji, a dokładniej zamiast skryptów z rodziny configure mamy system oparty o program meson (wymaga on obecności również pakietu ninja). Mimo to sam proces konfiguracji jest podobny do triumwiratu configure, make oraz make install. Na początek przeprowadzamy konfigurację pakietu np.:

```
meson --prefix=/opt -Dwayland-backend=false _build .
```

co oznacza, iż pakiet GTK+ ma zostać zainstalowany w katalogu /opt, bez wsparcia dla systemu Wayland, a pliki pośrednie mają zostać umieszczone w katalogu _build, natomiast konfigurację przeprowadzamy dla pakietu znajdującego się w katalogu aktualnym, o czym informuje nas kropka na końcu polecenia.

Zaletą mesona jest fakt, iż jeśli brakuje nam określonego pakietu, lub mamy starszą wersję, a jest nam potrzebna nowsza, to meson – o ile autorzy danego pakietu wskazali odpowiednie repozytoria – samodzielnie ściągnie brakujące biblioteki i je skompiluje. Jak bardzo ułatwia to komplikację, nie trzeba wspominać nikomu, kto chciał samodzielnie komplikować tak rozbudowane środowiska jak GNOME czy KDE bezpośrednio ze źródła.

Po przeprowadzeniu konfiguracji należy wejść do katalogu, w którym mają być składowane pliki pośrednie:

```
cd _build
```

Następnie możemy wydać polecenie przeprowadzające komplikację:

```
ninja
```

Po zakończonej komplikacji warto wydać polecenie testowania poszczególnych funkcji biblioteki GTK+:

```
meson test
```

Ponieważ korzystamy z wersji roboczej, to warto spróbować przeprowadzić testy, choć może to zabrać sporo czasu. My jednak od razu przejdziemy do instalacji środowiska:

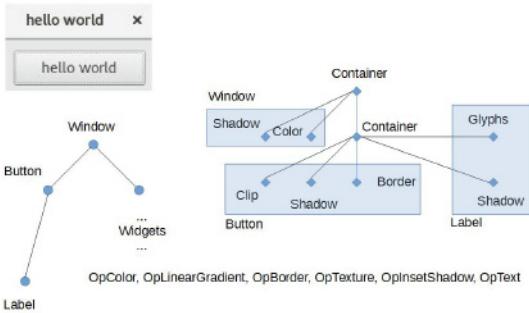
```
sudo ninja install
```

Właściwa komplikacja jest jak widać podobna do poprzedniego rozwiązania, ale główne zalety ze stosowania systemu meson wynikają z możliwości samodzielnego ściągania potrzebnych pakietów. Również plik opisujący konfigurację pakietu `meson.build` posługuje się znacznie nowocześniejszą syntaktyką niż pakiety z rodziny Automake korzystające z preprocesora M4.

tematycznie zyskuje na popularności (choć trzeba wspomnieć, że już wcześniej istniała możliwość stosowania GTK+ bez konieczności korzystania z serwera „X”). Nowa wersja 4.0 została gruntownie przebudowana, aby podstawowe mechanizmy tworzenia grafiki były łatwe w dostosowaniu do nowoczesnych API używanych w kartach graficznych. Dodatkowo modernizacja GTK+ poprawiła łatwość w przenoszeniu GTK+ na nowe platformy sprzętowe.

Wobec tych gruntownych zmian fundamentalnej przebudowie uległ podsystem GDK (tj. The GIMP Drawing Kit, nazwa nie jest błędna, bowiem jak już wspomniano na początku artykułu takie są źródła pakietu GTK+). Wprowadzono nowy sposób rysowania kontrolek GTK+. Schematycznie przedstawiono to na Rysunku 3. Po szczególnie fragmenty kontrolki zostały podzielone na części odpowiedzialne za cień, kolor, obszar wycinania itd. Podział pozwala na budowę grafu opisującego poszczególne elementy okna. Na podstawie utworzonego grafu wyznaczane są operacje (np. typu nałożenie koloru OpColor, wypełnienie gradientem OpLinearGradient itd.), jakie są potrzebne do utworzenia odpowiedniej reprezentacji graficznej danej kontrolki. Na podstawie tych informacji system zarządzania sceną GTK+ (GSK) może dokonać optymalizacji, i nawet wykorzystać wcześniej narysowane elementy np. czcionki. Zadaniem GSK jest przekazanie operacji do systemu rysującego, który może być prezentowany przez standard OpenGL albo Vulkan, i to system graficzny zajmie się procesem rysowania kontrolek. Takie podejście odciąża tradycyjny procesor komputera i pozwala na wykorzystanie potencjału karty graficznej. Można nawet mówić o wykorzystaniu „shaderów”, aby rysować przysłowiowe przyciski i etykiety. Niewątpliwie takie podejście pozwala na lepsze wykorzystanie obecnie dostępnych możliwości kart graficznych.

Kolejną nowością w GTK+ jest pełne wsparcie dla kaskadowych arkuszy stylów (ang. CSS). Były dostępne już w poprzedniej wersji 3.x. Nadchodzące wydanie GTK+ będzie już w pełni wykorzystywać podejście CSS. Wymagało to naturalnie kolejnych zmian w GDK. Podsystem GDK oferuje wiele funkcjonalności obejmujących m.in.: tworzenie grafiki rastrowej, tj. kształtów kontrolek graficznych, rysowanie tekstu oraz przetwarzanie zdarzeń. Obecne zmiany są oczywiście związane z zastosowaniem API kart graficznych, wobec czego możemy korzystać z obiektów `GdkTexture`, które zgodnie ze swoją nazwą reprezentują dane graficzne, a także mamy nowy

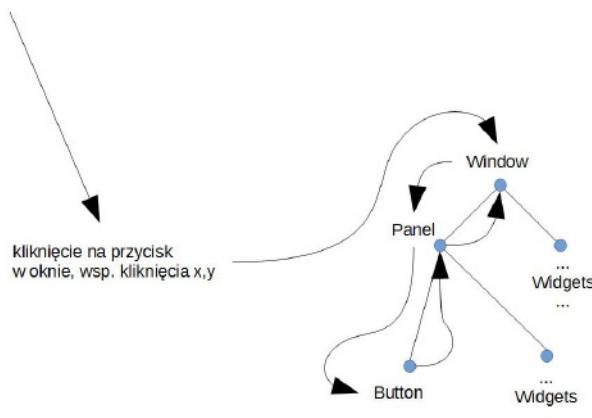


Rysunek 3. Sposób rysowania kontrolek w GTK+4.0 dla okna z jednym przyciskiem, słowo „Widgets” oznacza inne elementy interfejsu, jakie mogłyby wystąpić w schemacie (opracowano na podstawie Matthias Clasen, GTK+ 4 Status Update)

komponent `GdkPaintable`, odpowiedzialny za możliwość tworzenia grafiki.

Zmieniona została także wewnętrzna obsługa odbierania zdarzeń przez kontrolki. W GTK+ 3.x kliknięcie na przycisk istotnie dotyczyło tylko samego przycisku. Sygnał mógł być propagowany do nadrzędnej kontroli.

W najnowszej odsłonie GTK+ sygnał trafia do okna aplikacji, a następnie współrzędne kliknięcia pozwalają na wykrycie, do jakiego komponentu należy skierować obsługę. Schematycznie pokazano to na Rysunku 4. Sygnał z przycisku może naturalnie być propagowany z powrotem do kontrolek nadrzędnych. Mechanizm ten jest jednak zmianą o charakterze wewnętrzny i nadal stosuje się funkcję `g_signal_connect`, aby podłączać obsługę zdarzeń.



Rysunek 4. Sposób przekazywania komunikatów w GTK+4.0 (opracowano na podstawie Matthias Clasen, GTK+ 4 Status Update)

Pośród wielu innych nowości i zmian warto wskazać na istotne poprawki, jakie wniesiono do komponentu `GtkGLArea`. Już wcześniej była możliwość stosowania API OpenGL w GTK+, jednakże najnowsze zmiany pozwalają na jeszcze dalszą integrację. Można stosować nowy obiekt `GdkTexture` nie tylko w kontekście grafiki 3D, ale również w ramach odtwarzanych plików wideo za pomocą systemu GStreamer. Wspomnimy o tym jeszcze w dalszej części artykułu.

GSK – JAK ZARZĄDZAĆ WĘZŁAMI SCENY

Nowo wprowadzony podsystem GSK do pakietu GTK+ ma swoje źródła w starszym rozwiążaniu opartym o widget `GnomeCanvas`. Kontrolka ta pozwala na tworzenie grafiki 2D o bardzo dobrej jakości na poziomie okien aplikacji GTK+/GNOME. Obecnie można w tym celu stosować bibliotekę Cairo, ale będzie to tylko pojedyncza płaszczyzna, na której powstaje rysunek/grafika. Zastosowanie GSK pozwala nam na łączenie węzłów w jedną większą całość.

Ze względu na roboczy charakter GTK+, w aktualnej wersji 3.93/GIT – dostępnej podczas pisania tego artykułu – nie znajdziemy w pakiecie `GtkDemo` żadnego przykładu dotyczącego GSK. Podobnie w katalogu `examples`. Jednak warto sprawdzić katalog `testsuite` i odszukać pliki źródłowe dot. testów poszczególnych podsystemów GTK+. W pliku `test-render-nodes.c` znajdziemy sporo przykładów, jak tworzyć grafikę w GTK+ bezpośrednio za pomocą GSK oraz Cairo. Otwiera to naturalnie drogę do własnych kontrolek graficznych dostępnych w ramach GTK+.

W kodzie źródłowym pliku testowego znajdziemy np. funkcję `opacity`, która sprawdza, jak określona bitmapa jest przedstawiana przy różnych poziomach parametru przezroczystości. Fragmenty kodu źródłowego z tego przykładu zostały zaprezentowane w Listingu 3.

Tworzymy tam zestaw ośmiu rysunków, z inną wartością parametru przezroczystości, określonym już przy tworzeniu węzła: `gsk_opacity_node_new`. Przesunięcie węzła realizujemy za pomocą macierzy utworzonej przez funkcję `graphene_matrix_init_translate`. Funkcja ta pochodzi z dodatkowej biblioteki o nazwie Graphene, odpowiedzialnej za matematyczną stronę przekształceń realizowanych na elementach graficznych.

Po utworzeniu wszystkich ośmiu węzłów tworzony jest kontener `gsk_container_node_new`. Pozostałe węzły nie są nam potrzebne, dlatego za pomocą funkcji z rodziny „`unref`” oznaczamy, iż węzły te będzie można zwolnić w przyszłości. Utworzony kontener można zapisać do pliku `gsk_render_node_write_to_file`, ale także odczytać teksturę reprezentującą utworzoną grafikę: `gsk_texture_node_get_texture`, a teksturę można już dalej przekazywać w ramach GTK+.

Listing 3. Przykład użycia systemu GSK

```

GskRenderNode *child;
GskRenderNode *node;
GskRenderNode *nodes[5];
GskRenderNode *container;
graphene_matrix_t matrix;
int i;

child = create_picture ();

for (i = 0; i < 9; i++) {
    node = gsk_opacity_node_new (child, i / 8.0);
    graphene_matrix_init_translate (&matrix, &(const graphene_point3d_t) {i * 256, 0, 0});
    nodes[i] = gsk_transform_node_new (node, &matrix);
    gsk_render_node_unref (node);
}

gsk_render_node_unref (child);

container = gsk_container_node_new (nodes, 5);

for (i = 0; i < 5; i++)
    gsk_render_node_unref (nodes[i]);

// obiekt container reprezentuje
// utworzoną grafikę

```

PLIKI „AUDIO & VIDEO” W GTK+

W pakiecie GTK+ bardzo brakowało łatwego w użyciu wsparcia dla plików multimedialnych. Jednakże po długim okresie oczekiwania mamy zestaw następujących widgetów o nazwach: `GtkMediaStream`, `GtkMediaFile`, `GtkVideo`, `GtkMediaControls` zapewniających obsługę danych multimedialnych na poziomie podstawowego API biblioteki. Nowe widgety wykorzystują inne znane już pakiety takie jak GStreamer oraz FFmpeg, aby obsługiwać popularne formaty cyfrowego „audio & video”.

Pierwszy z wymienionych widgetów – `GtkMediaStream` – stanowi klasę bazową do utworzenia strumieni dźwiękowych oraz cyfrowe-

go wideo. Stosowany jest widget `GdkPaintable` do rysowania po szczególnych klatek obrazu. Drugi widget – `GtkMediaFile` – jest interfejsem do plików w ramach klasy `GtkMediaStream`, czyli możemy czerpać dane z plików. Zarządzaniem procesem odtwarzania zajmuje się kontrolka `GtkMediaControls`, natomiast `GtkVideo` jest podstawową kontrolką do prezentacji plików wideo.

Obecnie GTK+ z repozytorium korzysta ze wspomnianych już pakietów Ffmpeg oraz GStreamer i to ten drugi pakiet ma być do-myślnym rozwiązaniem. Przykład odtwarzania plików wideo jest równie nieskomplikowany co nasz pierwszy przykład z początku artykułu.

W Listingu 4 przedstawiono najważniejsze fragmenty kodu źródłowego naszego przykładu. W programie `GtkDemo` znajduje się podobny przykład, zatem warto również i tam zatrzymać się, aby poznać kilka dodatkowych detali. W naszym przykładzie pokażemy, jak rozpoczęć odtwarzanie pliku wideo wskazanego za pomocą okna dialogowego. Dodatkowo damy możliwość przełączenia aplikacji w tryb pełnoekranowy.

Pominiemy początkowe etapy tworzenia okna aplikacji, gdyż są takie same jak w naszym pierwszym przykładzie z Listingu 1. Możemy przejść od razu do utworzenia kontrolki odtwarzającej pliki multimedialne i dodania utworzonego obiektu do menadżera układu:

```
video = gtk_video_new ();
gtk_container_add (GTK_CONTAINER (window), video);
```

Następnie wykonujemy dodatkowe czynności, jak ustalenie zawartości paska statusu oraz podłączenie obsługi okna dialogowego, w którym możemy wybrać plik do odtworzenia. Pominiemy opis tego zadania, ponieważ jest to typowy kod dla okien dialogowych GTK+, nas bardziej interesuje wskazanie pliku, a zrobimy to następującą linią kodu:

```
gtk_video_set_file( GTK_VIDEO (video), file );
```

gdzie parametr `file` (typu `Gfile*`) jest rezultatem zwróconym przez okno dialogowe.

Sterowanie procesem odtwarzania jest możliwe za pomocą widgetu `GtkMediaControls`. Sterować można także samym strumieniem, ale wymaga to uzyskania dostępu do strumienia:

```
GtkMediaStream *gmstream = gtk_video_get_media_stream
(GTK_VIDEO(video));
```

Uruchomienie trybu pełnoekranowego także nie jest problematyczne. W aplikacji mamy przycisk odpowiedzialny za włączenie trybu pełnego ekranu, zatem wymagane jest uzyskanie obiektu reprezentującego tzw. widget główny, tj. w naszym przypadku będzie to okno główne:

```
GtkWidget *window = gtk_widget_get_toplevel (button);
```

Ponieważ przycisk znajduje się w oknie, to możemy nakazać przejęcie w tryb pełnoekranowy:

```
gtk_window_fullscreen (GTK_WINDOW (window));
```

Listing 4. Odtwarzanie plików wideo w GTK+

```
GtkWidget *title;
GtkWidget *video;
GtkWidget *open_button;
GtkWidget *fullscreen_button;

window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
// pozostałe operacje
// na oknie aplikacji

video = gtk_video_new ();
gtk_container_add (GTK_CONTAINER (window), video);

title = gtk_header_bar_new ();
gtk_header_bar_set_show_title_buttons (GTK_HEADER_BAR (title), TRUE);
gtk_header_bar_set_title (GTK_HEADER_BAR (title), "Odtwarzacz
plików multimedialnych");
gtk_window_set_titlebar (GTK_WINDOW (window), title);

open_button = gtk_button_new_with_mnemonic ("Open");
g_signal_connect (open_button, "clicked", G_CALLBACK (open_
clicked_cb), video);
gtk_header_bar_pack_start (GTK_HEADER_BAR (title), open_button);

fullscreen_button = gtk_button_new_from_icon_name
("view-fullscreen-symbolic");
g_signal_connect (fullscreen_button, "clicked", G_CALLBACK
(fullscreen_clicked_cb), NULL);
gtk_header_bar_pack_end (GTK_HEADER_BAR (title),
fullscreen_button);
```

PODSUMOWANIE

W artykule zaprezentowaliśmy tylko podstawowe przykłady dotyczące GTK+. Mimo ich skromności pokazują one, jak skonstruować API tego pakietu. Omówiliśmy pokróć najważniejsze nowości, a dotyczą one głównie aspektów graficznych GTK+ i przejście na pełne wykorzystanie API kart graficznych, tj. standardów OpenGL oraz Vulkan. Oczywiście nowe rozwiązania jak GSK i obiekty `GdkTexture` czy też `GdkPaintable` stanowią najważniejsze innowacje, są one jednak przydatne w bardziej zaawansowanych aplikacjach. W „zwykłych” aplikacjach możemy korzystać z bardzo przystępnego API, jak np. do odtwarzania plików wideo, gdzie dosłownie za pomocą kilku linii kodu możemy uzyskać potrzebną nam funkcjonalność. Nowa wersja stabilna ma być dostępna na jesieni 2018 roku, czyli już niebawem. Na razie możemy skorzystać z wersji z repozytorium, aby lepiej poznać nadchodzące nowości w pakiecie GTK+.

W sieci:

- ▶ Matthias Clasen, GTK+ 4 Status Update: <https://mclasen.fedorapeople.org/gtk4-devconf2018.pdf>
- ▶ Oficjalna strona projektu GTK+: <https://www.gtk.org/>
- ▶ Blog dotyczący rozwoju projektu GTK+: <https://blog.gtk.org/>
- ▶ Repozytorium kodu źródłowego GTK+: <https://gitlab.gnome.org/GNOME/gtk>
- ▶ Dodatkowa kopia głównego repozytorium: <https://github.com/GNOME/gtk>
- ▶ Podręcznik dla programisty dla GTK+ wersji 4.x: <https://developer.gnome.org/gtk4/>



MAREK SAWERWAIN

Autor, pracownik naukowy Uniwersytetu Zielonogórskiego, na co dzień zajmuje się teorią kwantowych języków programowania, ale także tworzeniem oprogramowania dla systemów Windows oraz Linux. Zainteresowania: teoria języków programowania oraz dobra literatura.

Pierwsza polska edycja Oracle Code już za nami

Warszawa przyciągnęła rekordową liczbę uczestników

W odbywającej się po raz pierwszy w Polsce konferencji Oracle Code wzięło udział ponad 550 uczestników. Organizatorzy przygotowali aż 27 sesji tematycznych, podczas których każdy znalazł coś dla siebie. Blockchain, chatboty, Internet Rzeczy (IoT) – to tylko niektóre z najnowszych technologii, które były tematem referatów, dyskusji i kuluarowych rozmów podczas Oracle Code 2018 w Warszawie. Nie zabrakło nawet prawdziwego robota.

Jako pierwszy głos zabrał gość specjalny Adam Bien, jeden z najbardziej uznanych w Europie programistów Java, konsultant i autor kilku książek. Zgromadzonym gościom i uczestnikom przedstawił najnowsze techniki programowania w języku Java, wygłaszaając referat pt. *Od Java EE 8, przez projekty Jakarta EE i Microprofile, do Javy bezserwerowej*. Na Oracle Code wystąpił także James Allerton-Austin, odpowiedzialny za kreowanie strategii dla Oracle Cloud Platform w regionie EMEA. Podczas spotkania z liczną grupą programistów oraz developerów aplikacji opowiadał o budowaniu interfejsów opartych na technologii chatbotów dla transakcji blockchain.

Już niebawem komunikacja biznesowa może odbywać się przy użyciu chatbotów, a nie stron internetowych czy aplikacji mobilnych. Programiści będą więc w znacznie większym stopniu tworzyć chatboty, które dzięki rozwijającym się procesom samouczenia maszynowego będą w stanie uczyć się zachować użytkownika i w efekcie udzielać znacznie bardziej spersonalizowanych odpowiedzi, m.in. w kontekście transakcji blockchainowych. Pamiętajmy, że to również jest rozwiązanie, które może niebawem na dobre zagościć w branżach takich jak logistyka (zarządzanie łańcuchem dostaw) czy też ubezpieczenia, a nawet branżach do tej pory sceptycznie nastawionych do tej technologii – komentuje wystąpienie Jerzy Suchodolski z Oracle.

W kontekście rozwiązań opartych na sztucznej inteligencji, uczeniu maszynowym szczególnym zainteresowaniem uczestników cieszył się Cloud Chatbot Robot, z wdziękiem towarzyszący wszystkim edycjom Oracle Code. Wspierany Oracle'ową technologią robot chętnie brał udział w rozmowach z licznie zagadującymi go uczestnikami konferencji. Uwagę gości w kuluarach konferencji zwracało także stoisko prezentujące system kontroli procesu wazienia piwa, oparty na technologii cloud computingu i IoT.

W programie konferencji nie zabrakło także licznych warsztatów i sesji tematycznych, pogrupowanych w 6 bloków. Tę część konferencji otworzył Timothy Hall, aktywny bloger i developer z Oracle, który prezentował praktyki związane z projektowaniem aplikacji wykorzystujących funkcje analityczne. W trakcie pozostałych warsztatów na Oracle Code nie zabrakło tematów związanych z kontenerami, mikrousługami, projektowaniem API, DevOps, mikroserwisami, a także technikami programowania w bazach danych. Sporym zainteresowaniem cieszyły się również warsztaty, podczas których uczestnicy, pod czujnym okiem Dorina Paraschiva, architekta aplikacji Java, projektowali aplikację do gry karcianej Blackjack.

Organizowana przez Oracle konferencja była darmowa dla uczestników. Warszawska edycja była jednym z czterech europejskich przystanków Oracle Code w Europie w 2018. Podobne spotkania odbędą się również w Londynie, Berlinie i Paryżu, a także w 10 amerykańskich i azjatyckich miastach. W Polsce konferencja dla developerów odbyła się po raz pierwszy. W zeszłorocznej edycji odwiedzono 21 miast. W konferencjach wzięło udział (także online) ponad 600 tys. developerów i programistów oraz 200 prelegentów z 31 krajów z całego świata.



Vue.js – chwilowa moda czy dojrzały framework?

Od momentu pierwszego publicznego wydania w lutym 2014 roku popularność Vue.js stale rośnie. Do jego najważniejszych zalet można zaliczyć dobrą dokumentację, szybkość działania, wysoką elastyczność i niski próg wejścia. Według raportu „State of Vue.js” 96% użytkowników skorzystałoby z tej technologii w następnych projektach. Antagoniści opisują Vue.js jako jeden z przypadków chwilowej mody, zwolennicy podkreślają jego coraz większą dojrzałość i dostępność narzędzi. Która z grup jest bliżej prawdy?

OD PROSTEJ POTRZEBY DO WIELKIEGO SUKCESU

Koncepcja Vue.js narodziła się w głowie jednego programisty – Evan You podczas pracy w Google Creative Labs (jest to roczny program, podczas którego interdyscyplinarne zespoły, złożone zazwyczaj ze świeżych absolwentów uczelni technicznych, poszukują wspólnie nowych pomysłów, więcej: <https://www.creativelab5.com>). Jego praca wymuszała szybkie prototypowanie front-endu wielu projektów. Okazało się jednak, że do tej pory nie powstało narzędzie, które ułatwiałoby tego typu pracę.

Część zespołów korzystała już z Angulara – również Evan działał w nim wiele zalet – jednocześnie jednak oferował zbyt wiele funkcjonalności jak dla potrzeb wykonywanych przez niego projektów. Pierwotnie Vue powstało jako próba przepisania funkcjonalności Angulara w minimalnej potrzebnej formie. Po pewnym czasie You uznał, że jego projekt ma coraz większy potencjał i warto byłoby podzielić się nim z innymi programistami. Po krótkim uporządkowaniu plików i koncepcji nadał mu nazwę Vue.js. W lutym 2014 roku pierwsza publiczna wersja tego narzędzia pojawiła się w serwisie GitHub.

Kamieniami milowymi w historii Vue było wydanie w połowie i końcowce 2014 roku takich dodatków jak vue-router, vuex czy vue-cli. Choć jak dotąd najważniejszym była wersja 2.0, która była kompletnym przepisaniem framework'a, dzięki czemu zyskał on nowe cechy jak Virtual DOM czy możliwość skorzystania z server side renderingu. Co ważne, nie była to taka rewolucja jak w przypadku Angulara – tym razem obyło się bez większych zmian w samym API, a narzędzie migracyjne dodatkowo ułatwiło proces adaptacji.

W 2018 roku coraz częściej jest zignorować obecność Vue.js. Ze swoimi 93,246 gwiazdkami (stan na moment pisania artykułu) znalazł się on w czołówce najpopularniejszych narzędzi wspierających codzienną pracę front-end developerów. Co więcej, w 2016 roku Vue.js był najpopularniejszym projektem ze wszystkich JavaScriptowych repozytoriów znajdujących się na GitHubie (risingstars2016.js.org), tuż za nim uplasował się React, a na trzecim miejscu znalazł się Yarn. Według statystyk dostarczanych przez npm wzrost pobrań Vue w 2017 roku w porównaniu z 2016 wyniósł aż 446%. Również w raporcie State of JavaScript za rok 2017 możemy znaleźć ciekawe informacje – React utrzymał status dominującego framework'a, jednak Vue.js zaczęło dynamicznie zdobywać swą pozycję kosztem słabnącej popularno-

ści Angulara. Według wspomnianego raportu aż 43% developerów słyszało o Vue i chciałoby go poznać, a 16% programistów z niego skorzystało i zrobiłoby to ponownie w przyszłości. Dla porównania w przypadku Angulara (2 i kolejnych wersji) dane te przedstawiają się następująco: 21% respondentów chce go poznać, a 22% aktualnie z niego korzysta.

Evan You – autor framework'a – utrzymuje się ze wsparcia na portalu Patreon. Co miesiąc 228 prywatnych osób i firm z całego świata wspiera go kwotą w wysokości \$16 000.

Co ciekawe, również Polska ma swój istotny wkład w rozwój Vue.js. Pierwsza oficjalna konferencja (VueConf 2017), skupiona tylko wokół tego framework'a, odbyła się 22 i 23 czerwca 2017 roku we Wrocławiu. Wzięło w niej udział około 300 osób, a wśród prelegentów był między innymi Evan You. Oficjalny newsletter Vue.js również prowadzony jest przez Polaków, a oba te przedsięwzięcia skupiają się wokół firmy Montereal z Wrocławia.

VUE.JS W PRAKTYCE

Vue.js może być wykorzystane zarówno jako prosta biblioteka, jak i w formie pełnoprawnego framework'a. Aby skorzystać z funkcjonalności Vue, wystarczy po prostu umieścić tag `script` ze źródłem do biblioteki w pliku HTML. Aby jeszcze bardziej ułatwić sobie sprawę, skorzystamy z CDN.

Wersja deweloperska vue (zawiera konsolowe ostrzeżenia i tryb debugowania):

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

Wersja produkcyjna vue:

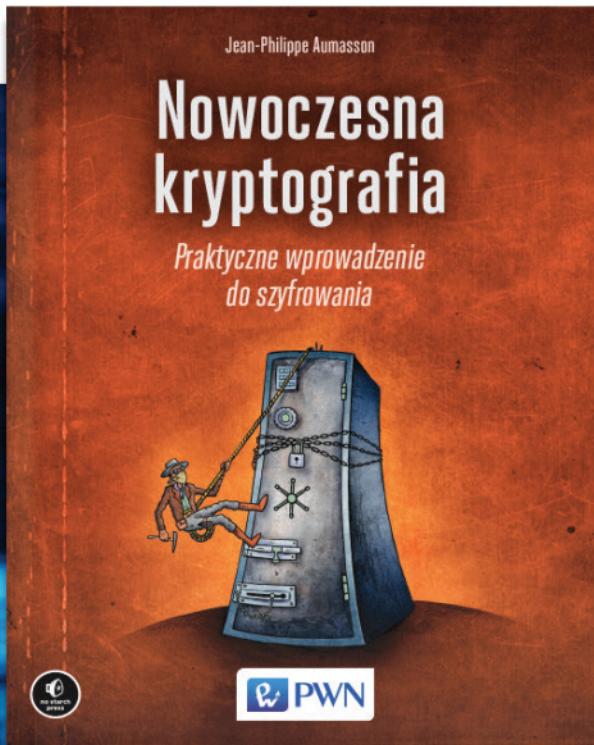
```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.min.js"></script>
```

Wszystko, co teraz musimy zrobić, to utworzyć element HTML, na którym Vue będzie operował, i przypisać go w tagu `script`:

Listing 1. Inicjalizacja Vue

```
<div id="app">
  {{ hello }}
</div>
```

Najlepsze od



Praktyczny przewodnik!



ODWIEDŹ NAS NA



IT.PWN.PL i zapisz się na newsletter!



KSIEGARNIA.PWN.PL

```
<script
src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
<script>
let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World'
  }
})</script>
```

Powyższy kod wyświetli w divie o id app wiadomość „Hello World” – dane ze zmiennej hello zostaną wyświetlone w sposób deklaratywny (nie trzeba określać, w jaki sposób wyświetlić treść zmiennej). Składnia {{ hello }} powinna być już znana użytkownikom Angulara czy Reacta, gdyż jej postać jest analogiczna jak w przypadku tych frameworków.

Warunkowe wyświetlanie

Warunkowe wyświetlanie można uzyskać w bardzo prosty sposób:

Listing 2. Warunkowe wyświetlanie

```
<div id="app">
  <div v-if="helloVisible">
    {{ hello }}
  </div>
</div>

<script>
let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World',
    helloVisible: true
  }
})</script>
```

Dyrektywę v-if można również zastąpić v-show. Efekt dla użytkownika będzie ten sam, jest jednak pewna różnica, którą powiniśmy uwzględnić, wybierając jedną z opcji:

Dyrektyna v-if – w rzeczywistym sposobie usuwa dany element ze struktury szablonu. Wszystkie komponenty i ich event listenery zostaną zniszczone.

Dyrektyna v-show – ukrywa element za pomocą CSS. Pozostań na nadal w DOM, ale otrzyma właściwość display: none.

Dwukierunkowe bindowanie danych i wartości wyliczane

Do powyższego prostego przykładu dodany został input. Chcąc, aby jego wartość zastępowała tekst, który do tej pory był sztywno zapisany w zmiennej hello, należy skorzystać przede wszystkim z dyrektywy v-model, dzięki której można uzyskać w prosty sposób dwukierunkowe bindowanie danych (ang. *two way data binding*) między inputem i stanem aplikacji.

Aby treść wiadomości wyświetlała się tylko wtedy, gdy input ma w sobie więcej niż dwa znaki, należy dodać odpowiedni warunek:

Listing 3. Dwukierunkowe przypisywanie. Powyższy przykład dostępny jest na platformie Codepen: <https://codepen.io/igorpodlaski/pen/GdQXPe>

```
<div id="app">
  zawartość inputa: <input type="text" v-model="hello">
  <div v-show="this.hello.length > 2">
```

```
    treść wiadomości: {{ hello }}
  </div>
</div>

<script>
let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World'
  }
})</script>
```

Vue pozwala na korzystanie z JavaScriptowej logiki w swoich szablonach, nie jest to jednak pożąданie rozwiązanie. Należy zawsze dążyć do jak najszerszej separacji logiki od wyświetlania. Co trzeba zatem zrobić, aby w prosty sposób sprawdzać powyższy warunek i jednocześnie zachować czystość kodu? Z pomocą przychodzą computed properties:

Listing 4. Metody wyliczane (ang. computed methods). Codepen: <https://codepen.io/igorpodlaski/pen/pVaOme>

```
<div id="app">
  zawartość inputa: <input type="text" v-model="hello">
  <div v-show="inputIsBiggerThanTwo">
    treść wiadomości: {{ hello }}
  </div>
</div>

<script>
let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World'
  },
  computed: {
    inputIsBiggerThanTwo: function() {
      return this.hello.length > 2;
    }
  }
})</script>
```

Przypisywanie (bindowanie) klas CSS

Chcąc w inny sposób pokazać użytkownikowi warunki wyświetlnia wiadomości, można skorzystać z dynamicznego przypisywania klas CSS:

Listing 5. Przypisywanie klas CSS

```
<div id="app">
  zawartość inputa: <input type="text" v-model="hello">
  <div v-bind:class="{ default: !inputIsBiggerThanTwo, active: inputIsBiggerThanTwo }">
    treść wiadomości: {{ hello }}
  </div>
</div>

<style>
.default {
  color: FireBrick;
}

.active {
  color: green;
}</style>
```

W powyższym przykładzie treść wiadomości będzie miała kolor żółty, gdy znaków będzie więcej niż 2, i czerwony, gdy będzie ich mniej. Codepen: <https://codepen.io/igorpodlaski/pen/GdQYoV>.

Wyświetlanie elementów listy

W poniższym przykładzie zostało zaimplementowane wyświetlanie elementów z listy w formie HTMLowej listy.

Listing 6. V-for i wyświetlanie elementów listy

```
<div id="app">
  dodaj element: <input type="text" v-model="hello">
<div v-bind:class="{ default: !inputIsBiggerThanTwo, active: inputIsBiggerThanTwo }">
  treść elementu: {{ hello }}
</div>
<ul>
  <li v-for="element in elements">
    {{ element.txt }}
  </li>
</ul>
</div>

<script>
let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World',
    elements: [
      { txt: 'Element 1' },
      { txt: 'Element 2' }
    ]
  },
  computed: {
    inputIsBiggerThanTwo: function() {
      return this.hello.length > 2;
    }
  }
})
</script>
```

W efekcie zostały wyświetlane dwa elementy listy.

Reagowanie na zdarzenia

Do wcześniejszego kodu dodana została metoda `addElement`. Nowym warunkiem jest także atrybut `disabled` pojawiający się, gdy liczba znaków w inputie jest mniejsza niż 2.

Listing 7. Event listener. CodePen: <https://codepen.io/igorpodlaski/pen/zjRmVy>

```
<div id="app">
  dodaj element: <input type="text" v-model="hello">
<div v-bind:class="{ default: !inputIsBiggerThanTwo, active: inputIsBiggerThanTwo }">
  treść elementu: {{ hello }}
</div>
<button :disabled="!inputIsBiggerThanTwo"
v-on:click="addElement">Dodaj element</button>
<ul>
  <li v-for="element in elements">
    {{ element.txt }}
  </li>
</ul>
</div>

<script>
let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World',
    elements: []
  },
  computed: {
    inputIsBiggerThanTwo: function() {
      return this.hello.length > 2;
    }
  },
  methods: {
    addElement: function() {
```

```
      let newElement = {
        txt: this.hello
      }
      this.elements.push(newElement)
      this.hello = ''
    }
  })
</script>
```

W efekcie powstała prosta aplikacja dodająca elementy do listy.

Komponenty

Główną ideą komponentów jest umożliwienie wielokrotnego użycia tego samego kodu oraz poprawienie jego organizacji, dzięki wydzieleniu do osobnych struktur. W najprostszej postaci komponent może przyjąć taką formę:

Listing 8. Komponenty w Vue.js

```
Vue.component("counter", {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++>Kliknij mnie {{ count }} razy.</button>'
```

Dzięki czemu powstanie taki efekt:

Kliknij mnie 8 razy.

Rysunek 1. Przykład najprostszego komponentu

W poniższym przykładzie poprzedni kod został zmodyfikowany tak, aby fragment odpowiedzialny za wyświetlanie treści wiadomości został wyodrębniony do osobnego komponentu. Przydatne okazały się także propsy, służące do przekazywania danych.

Listing 9. Komponent utworzony na bazie wcześniejszego kodu. Codepen: <https://codepen.io/igorpodlaski/pen/wjyRGR>

```
<div id="app">
  dodaj element: <input type="text" v-model="hello">
<div v-bind:class="{ default: !inputIsBiggerThanTwo, active: inputIsBiggerThanTwo }">
  <show-message :message="hello" />
</div>
<button :disabled="!inputIsBiggerThanTwo"
v-on:click="addElement">Dodaj element</button>
<ul>
  <li v-for="element in elements">
    {{ element.txt }}
  </li>
</ul>
</div>
<script>
Vue.component('show-message', {
  props: ['message'],
  template: '<span>treść elementu: {{this.message}}</span>'
});

let app = new Vue({
  el: '#app',
  data: {
    hello: 'Hello World',
    elements: []
  },
  computed: {
    inputIsBiggerThanTwo: function() {
      return this.hello.length > 2;
    }
  },
  methods: {
```

```

addElement: function () {
  let newElement = {
    txt: this.hello
  }
  this.elements.push(newElement)
  this.hello = ''
}
})
</script>

```

W ten sposób powstał nowy „element” <show-message>, który można wykorzystać w szablonie.

Aby nie mieszać kodu szablonu z kodem JavaScript, należy wydzielić zawartość template do osobnego elementu typu <template> i odwołać się do niego w parametrze template, tak jak przy ustawianiu głównego elementu aplikacji:

Listing 10. Wydzielenie kodu komponentu do osobnego elementu.
Codepen: <https://codepen.io/igorpolawski/pen/vjdVQV>

```

<template id="show-message">
  <span>treść elementu: {{ this.message }}</span>
</template>

<div id="app">
  dodaj element: <input type="text" v-model="hello">
  <div v-bind:class="{ default: !inputIsBiggerThanTwo, active: inputIsBiggerThanTwo }">
    <show-message :message="hello" />
  </div>
  <button :disabled="!inputIsBiggerThanTwo"
  v-on:click="addElement">Dodaj element</button>
<ul>
  <li v-for="element in elements">
    {{ element.txt }}
  </li>
</ul>
</div>

<script>
Vue.component('show-message', {
  props: ['message'],
  template: '#show-message'
});

// Dalsza część kodu – wywołanie instancji Vue – analogiczne jak w poprzednim przykładzie.
</script>

```

VUE.JS – ZASADA DZIAŁANIA, WZORCE ARCHITEKTONICZNE I PODSTAWOWE KONCEPCJE

Na stronie głównej projektu określony on jest jako progresywny framework JavaScriptowy. Oznacza to, że można skorzystać z jego możliwości zarówno w prostych projektach, jak i w rozbudowanych aplikacjach SPA (ang. *single page applications* – aplikacje oparte o jedną stronę zmieniającą swoją zawartość). Główne cechy, na których opiera się koncepcja Vue.js, to:

- » Przystępnoś – niski próg wejścia i możliwość szybkiego rozpoczęcia pracy,
- » Wszechstronność – skalowalność ekosystemu od prostej biblioteki do rozbudowanego frameworka front-endowego,
- » Wydajność – zminifikowana i zgzipowana paczka Vue.js mieści się w 20 kB, zaś jego architektura i rozwiązania techniczne – takie jak virtual DOM – wspierają szybkie działanie aplikacji przy minimalnym wysiłku związanym z jej optymalizacją.

MVVM

Najważniejszym wzorcem architektonicznym, na bazie którego powstała Vue, jest MVVM – z ang. *Model-View-ViewModel* (Model-Widok-

-WidokModel). Głównym celem stosowania tego typu architektury jest rozdzielenie logiki biznesowej (model) od graficznego interfejsu (view) przy jednoczesnym uproszczeniu komunikacji między tymi dwoma warstwami w ViewModel. Dzięki temu każda zmiana danych w modelu wywołuje natychmiastową reakcję widoku i na odwrót.

1. **Warstwa modelu** – odpowiada za przetwarzanie logiki biznesowej i dostarczanie danych.
2. **Warstwa widoku** – służy jedynie do wyświetlania wynikowego kodu w postaci interfejsu graficznego końcowego użytkownika. Vue korzysta z szablonów budowanych w oparciu o tradycyjny HTML oraz kilku dyrektywach upraszczających data-binding.
3. **Warstwa ViewModelu** – łączy oba powyższe elementy, tak aby korzystanie z wyniku operacji na modelu było prostsze w warstwie widoku – oraz odwrotnie – aby ułatwić wykorzystanie zmian, które nastąpiły w widoku w części odpowiadającej za logikę biznesową. W tym miejscu Vue wspiera programistę poprzez udostępnienie łatwego w wykorzystaniu dwukierunkowego data-bindingu (przypisywanie danych).

Virtual DOM

Żeby móc lepiej zrozumieć działanie Virtual DOM, warto przypomnieć sobie kilka podstawowych informacji. DOM (Document Object Model) jest reprezentacją struktury dokumentów HTML lub XML w postaci modelu obiektowego, a także zbiorem zasad, wedle których należy odnosić się do tej struktury z poziomu skrypcji. Dzięki DOM możemy odczytać stronę jako strukturalną grupę węzłów. Każda przeglądarka internetowa musi zwrócić wszystkie elementy HTML jako tablicę węzłów.

Bez DOM JavaScript nie mógłby manipulować na elementach umieszczonych na stronach internetowych. Dzięki niemu wszystko, co znajduje się w dokumencie HTML (lub XML), ma swoją reprezentację, na której mogą pracować języki takie jak JavaScript.

Przykładowe metody DOM:

- » document.getElementById(n),
- » document.createElement(n),
- » element.innerHTML.

HTMLowy DOM zawsze oparty jest o strukturę drzewa:

Listing 11. Fragment kodu HTML

```

<div id="kontener" class="container">
  <p><strong>lorem ipsum</strong></p>
</div>

- div
-- p
--- strong
---- lorem ipsum

```

Powyzsza struktura w przypadku JavaScriptu została zapisana w taki sposób:

Listing 12. Tak może wyglądać obiekt Virtual DOM

```

let div = {
  tag: 'div',
  attr: { id: 'kontener', class: 'container' },
  children: [
    // p -> strong -> lorem ipsum
  ]
}

```

Ma to swoje wady i zalety. Dość oczywistą zaletą jest możliwość dotarcia do określonych elementów w prosty sposób. Niestety coraz większa złożoność nowoczesnych SPA wymusza dalszy rozrost tego drzewa oraz konieczność jego szybkich modyfikacji – co z każdą kolejną gałęzią jest coraz bardziej trudne.

Czym jest zatem Virtual DOM? Najprościej można opisać go jako lokalną, uproszczoną kopię HTMLowego DOM przechowywaną w formacie JavaScriptowych obiektów. Reprezentuje to, jak właściwy DOM powinien wyglądać w danym momencie. Virtual DOM jest dodatkową warstwą abstrakcji i zawsze generuje rzeczywisty Document Object Model.

Jakie są zatem główne korzyści wynikające z korzystania z Virtual DOM?

- » Rozdzielenie logiki renderowania od środowiska przeglądarkowego, dzięki czemu możliwe jest skorzystanie z server side rendering w środowisku node.js lub generowanie szablonów w odrębnym od HTML (wykorzystywanych np. w aplikacjach mobilnych).
- » Przyśpieszenie wykonywania wszystkich operacji na drzewie struktury HTML, dzięki operowaniu na obiektach JavaScriptowych zamiast na strukturze drzewa elementów HTML.

Kompilacja kodu szablonu

Wykonywanie kodu Vue.js odbywa się w kilku krokach:

1. Pierwszą fazą jest parsowanie, które generuje kod HTML w postaci drzewa składniowego (AST). Zawiera ono informacje takie jak: atrybuty, rodzice, dzieci, tagi i inne. Ciekawostka: w sieci można znaleźć aplikację pozwalającą podejrzeć wynik renderowania szablonu do drzewa AST (<https://github.com/ktsn/vue-ast-explorer>).
2. Kolejną fazą jest optymalizacja. Podczas niej kompilator przechodzi przez wygenerowane drzewo AST i sprawdza, które z elementów są w pełni statyczne (elementy, które nigdy się nie zmieniają). Zostaną one odpowiednio oznaczone, a następnie będą przechowywane jako stałe, dzięki czemu przy każdym ponownym renderowaniu będzie można je pominąć.
3. Ostatnim etapem jest generowanie kodu (CodeGen). W tym miejscu następuje właściwe wytworzenie funkcji renderujących, które będą wykorzystywane podczas odświeżania DOM.

Reaktywne komponenty – metoda `defineProperty`

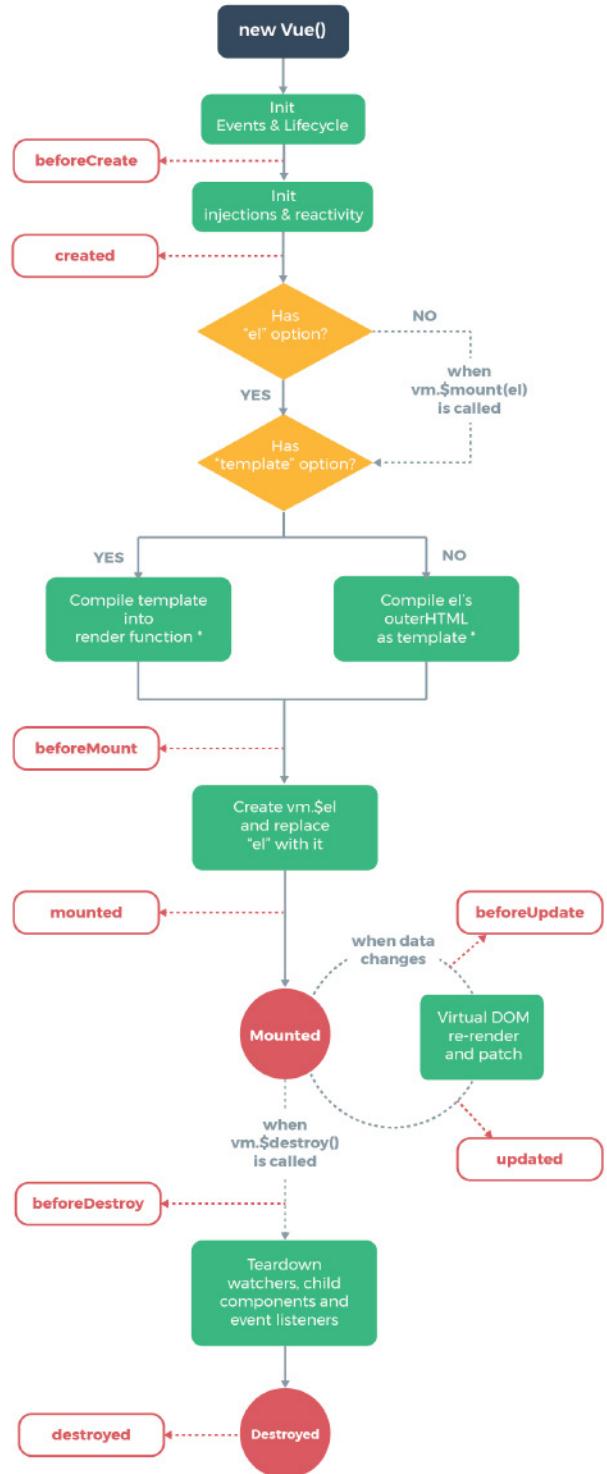
Co dzieje się „pod maską” z danymi przekazanymi przez nas w obiekcie data do instancji Vue? W jaki sposób framework osiąga tak ściśle powiązanie modelu z warstwą widoku?

Vue przechodzi przez wszystkie właściwości w obiekcie data i tworzy dla nich gettery i settery za pomocą `Object.defineProperty`:

Listing 13. Metoda `defineProperty`

```
let ob = Object.create(null);
let text = '';

Object.defineProperty(ob, 'text', {
  get () {
    return text;
  },
  set (nValue) {
    text = nVal;
  }
})
ob.text = 'test'
```



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

Rysunek 2. Cykl życia aplikacji Vue. Źródło: <https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram>

Po każdej zmianie danych funkcja `set` powiadamia dołączony obserwator. Obserwatory powstają dla każdego komponentu podczas inicjalizacji Vue. Po powiadomieniu odpowiednich obserwatorów następuje wywołanie funkcji re-renderowania.

Więcej o `Object.defineProperty` w kontekście Vue: <https://vuejs.org/v2/guide/reactivity.html>.

Cykl życia aplikacji Vue

Aplikacja Vue.js przy każdym wywołaniu przechodzi przez kilka stanów. Do każdego z nich można odwołać się w specjalnie do tego przygotowanych metodach, poniżej znajduje się ich pełny spis wraz z krótkim objaśnieniem. Na Rysunku 2 przedstawiony jest cały cykl życia aplikacji w formie schematu blokowego.

1. `beforeCreate` – występuje natychmiastowo (synchronicznie) po inicjalizacji instancji i przed utworzeniem zależności i obserwatorów.
2. `created` – następuje synchronicznie zaraz po utworzeniu instancji Vue. W tym momencie wszystkie dane w widoku zostały już zsynchronizowane z modelem.
3. `beforeMount` – występuje tuż przed etapem utworzenia realnego DOM na bazie wirtualnego DOM.
4. `mounted` – stan działającej aplikacji tuż po wygenerowaniu właściwego DOM.
5. `beforeUpdate` – wywoływane po zmianie danych, tuż przed nadpisaniem właściwego DOM, przez to, co znajduje się w stanie Virtual DOM.
6. `updated` – po nadpisaniu DOM.
7. `activated` – wywołane po aktywowaniu komponentów `kept-alive`.
8. `deactivated` – po deaktywowaniu komponentów `kept-alive`.
9. `beforeDestroy` – tuż przed usunięciem instancji Vue.
10. `destroyed` – wywołane po usunięciu instancji Vue.
11. `errorCaptured` – wywołane po wystąpieniu błędu w dziedziczącym komponencie.

NARZĘDZIA

Vue.js wciąż jest rozwijającym się frameworkm, jednak już od samego początku duży nacisk został położony na narzędzia i rozwiązania pomocne w codziennej pracy.

Vue-cli – instalacja, możliwości

Vue ma własny interfejs wiersza poleceń (cli – ang. *command line interface*), który ułatwia budowanie projektu.

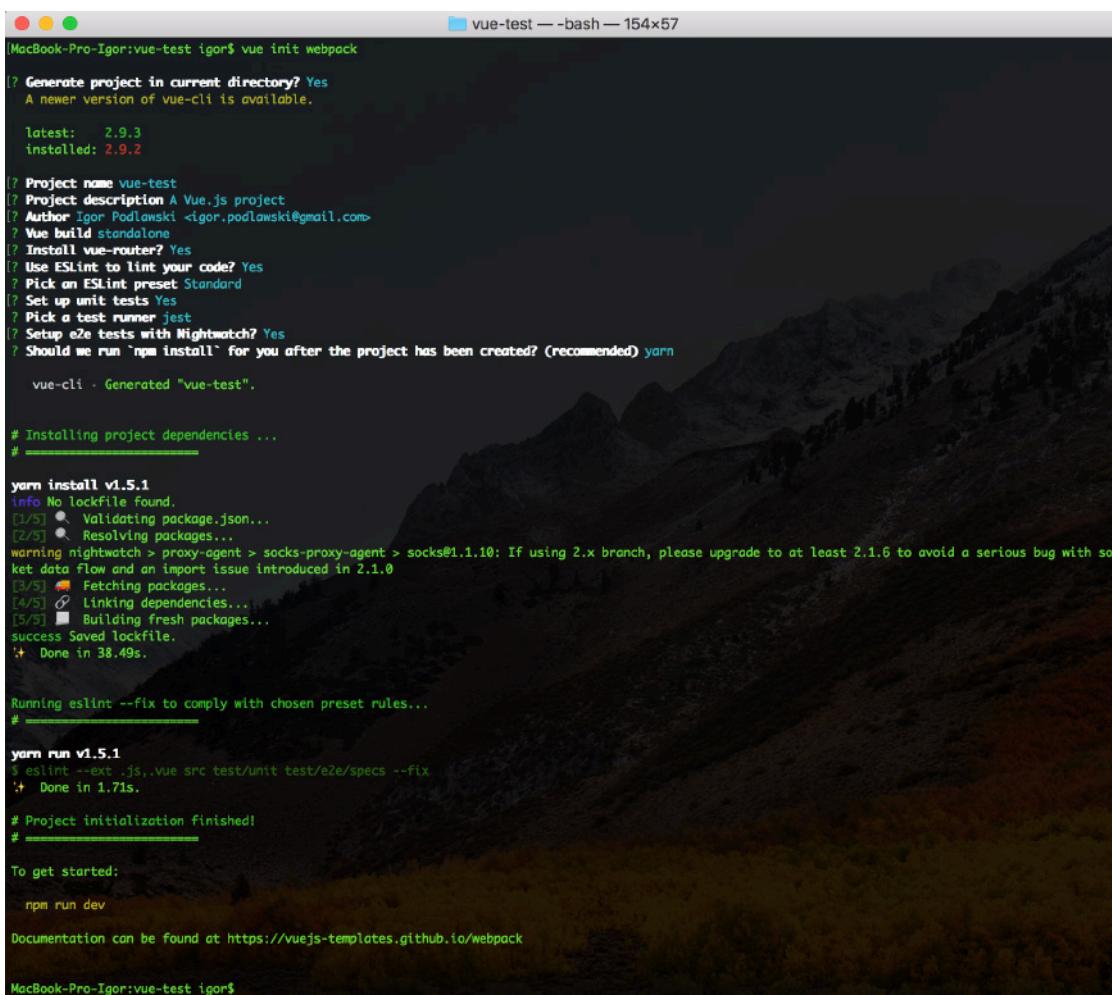
Aby go zainstalować, wystarczy w terminalu (lub innym wierszu poleceń) wpisać polecenie:

```
» npm install -g vue-cli
```

Uwaga: trzeba wcześniej mieć zainstalowane środowisko node.js.

Podczas korzystania z vue-cli możemy utworzyć projekt według następujących szablonów:

1. **browserify** – oferuje szablon oparty na browserify i vueify, dodatkowo: hot-reload, lintowanie i testy jednostkowe,
2. **browserify-simple** – browserify wraz z vueify,
3. **webpack** – szablon oparty na webpacku oferujący wsparcie vue-loadera, hot-reload, lintowanie, wsparcie dla testowania i eksportowania CSS,
4. **webpack-simple** – webpack i vue-loader,
5. **pwa** – szablon PWA (Progressive Web App) bazujący na szablonie webpack,
6. **simple** – szablon, w którym vue jest załadowane w pliku HTML (podobnie jak we wcześniejszych przykładach).



```
MacBook-Pro-Igor:vue-test igor$ vue init webpack
? Generate project in current directory? Yes
A newer version of vue-cli is available.

latest: 2.9.3
installed: 2.9.2

? Project name vue-test
? Project description A Vue.js project
? Author Igor Podławski <igor.podlawski@gmail.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests? Yes
? Pick a test runner jest
? Setup e2e tests with Nightwatch? Yes
? Should we run `npm install` for you after the project has been created? (recommended) yarn
vue-cli - Generated "vue-test".

# Installing project dependencies ...
# -------

yarn install v1.5.1
info No lockfile found.
[1/5] ⚡ Validating package.json...
[2/5] ⚡ Resolving packages...
warning nightwatch > proxy-agent > socks@1.1.10: If using 2.x branch, please upgrade to at least 2.1.6 to avoid a serious bug with socket data flow and an import issue introduced in 2.1.0
[3/5] 🚀 Fetching packages...
[4/5] ⚡ Linking dependencies...
[5/5] 🏺 Building fresh packages...
success Saved lockfile.
� Done in 38.49s.

Running eslint --fix to comply with chosen preset rules...
# -------

yarn run v1.5.1
$ eslint --ext .js,.vue src test/unit test/e2e/specs --fix
� Done in 1.71s.

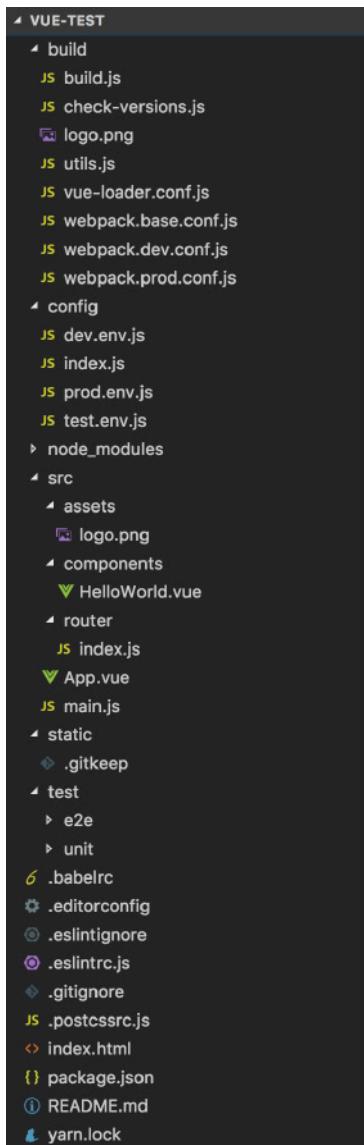
# Project initialization finished!
# -------

To get started:
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack

MacBook-Pro-Igor:vue-test igor$
```

Rysunek 3. Generowanie projektu za pomocą vue-cli



Rysunek 4. Struktura projektu wygenerowanego za pomocą vue-cli

Aby skorzystać z powyższych szablonów, należy wydać komendę:

- » `vue init webpack*`
- » – lub inne w zależności od wyboru

Po wydaniu tego polecenia vue-cli zada nam kilka pytań i wygeneruje dla nas projekt (Rysunek 3).

Natomiast struktura wygenerowanego projektu będzie przedstawiała się w sposób widoczny na Rysunku 4. Znajdziemy w niej konfigurację webpacka podzieloną na wersję produkcyjną i deweloperską, folder z testami jednostkowymi i e2e (jeśli wybierzemy taką opcję), katalog src (główny katalog kodu źródłowego naszej aplikacji) oraz podstawowe, wymagane node_modules (tylko wtedy, gdy zdecydujemy się na opcję zbudowania projektu, możemy też wybrać narzędzie do obsługi paczek – do wyboru yarn lub npm).

Jednoplikowe komponenty

Jedną z istotnych cech Vue są jednoplikowe komponenty. Można ich użyć dzięki skorzystaniu z narzędzi budujących projekt, takich jak Webpack czy Browserify. Zapisuje się je jako – nomen omen – jeden plik, w którym zawarty jest kod szablonu, CSS i JavaScript.

```

// VUE-TEST
  build
    build.js
    check-versions.js
    logo.png
    utils.js
    vue-loader.conf.js
    webpack.base.conf.js
    webpack.dev.conf.js
    webpack.prod.conf.js
  config
    dev.env.js
    index.js
    prod.env.js
    test.env.js
  node_modules
  src
    assets
      logo.png
    components
      HelloWorld.vue
    router
      index.js
    App.vue
    main.js
    static
      .gitkeep
    test
      e2e
      unit
  .babelrc
  .editorconfig
  .eslintignore
  .eslintrc.js
  .gitignore
  .postcssrc.js
  index.html
  package.json
  README.md
  yarn.lock

```

Rysunek 5. Jednoplikowy komponent z przykładu wygenerowanego przez vue-cli

Jednoplikowe komponenty pozwalają na łatwiejszą pracę przy większych aplikacjach typu SPA. Na Rysunku 5 przykład kodu wygenerowanego przez szablon webpack dostępny w vue-cli.

Routing – vue-router

Vue posiada oficjalną bibliotekę vue-router, dzięki której możliwe jest proste definiowanie ścieżek podstron. Jakie wspiera ona właściwości?

- » Mapowanie zagnieżdzonych ścieżek;
- » Konfiguracja routera w oparciu o komponenty;
- » Parametry routingu;
- » Efekty przejść widoków;
- » Obsługa HTML5 History Api, dzięki czemu można korzystać z domyślnych funkcji przeglądarki (wstecz, do przodu);
- » Automatyczne klasy CSS do linków routera dla aktywnych stron;
- » Modyfikowanie zachowania scrolla po przejściu do podstrony (przesunięcie do góry, do kotwicy itp.).

Zarządzanie stanem aplikacji

Często duże aplikacje wymagają zarządzania ich stanem – jednym z podstawowych przypadków użycia może być obsługa zalogowanych/wylogowanych użytkowników. Aby ułatwić pracę w tego typu sytuacjach, Vue udostępnia oficjalne rozszerzenie o nazwie „vuex”.

Jest ono inspirowane znanym ze świata Reacta reduxem. Dużą zaletą jest integracja z vue devtools.

Vue-devtools

Powstało także oficjalne rozszerzenie do narzędzi developerskich znajdujących się w przeglądarce Chrome (i innych na niej bazujących, np. Opera) oraz Firefox.

Pozwalają one na łatwe przeglądanie zawartości komponentów czy szybką zmianę danych w nich się znajdujących.

Renderowanie po stronie serwera

Często aby móc w pełni wykorzystać możliwości przygotowanej przez nas aplikacji, musi ona wspierać server side rendering. Co prawda boty Google dobrze radzą sobie z renderowaniem JavaScriptu, niestety dotyczy to jednak głównie aplikacji synchronicznych. Najważniejszym problemem, który jest z tym związany, jest kwestia dynamicznego doładowywania treści, które mogą zostać pominięte. Na szczęście społeczność Vue.js przygotowała framework „Nuxt.js”.

PORÓWNANIE Z INNYMI FRAMEWORKAMI

Nie sposób rozmawiać o Vue, nie odnosząc się do jego głównej konkurencji. Za sukcesem dwóch najbardziej znanych rywali stoją wielkie firmy, w przypadku Reacta jest to Facebook, zaś Angular jest dzieckiem Google. Vue.js pierwotnie było dziełem jednej osoby – Evana You. Współcześnie zespół Vue składa się z co najmniej 23 osób, w tym dwóch Polaków – Michała Sajnoga i Damiana Dulisza. Repozytorium Vue wskazuje, że 190 programistów miało swój wkład w rozwój tego frameworka, w przypadku Angulara jest to 625 developerów, zaś współautorami Reacta jest aż 1178 osób.

React

Vue i React są do siebie podobne w kilku istotnych kwestiach:

- » Wykorzystanie Virtual DOM,
- » Reaktywne komponenty,
- » Skupienie się na podstawowej funkcjonalności z możliwością wykorzystywania dodatkowych bibliotek i rozszerzeń.

Natomiast w kwestii różnic:

- » Kod Reacta jest w całości JavaScriptem – jego szablony opierają się o JSX. Również Vue wspiera JSX, jednak domyślnym rozwiązaniem dla warstwy widoku jest system szablonów. Oznacza to, że w Vue można bez przeszkód wykorzystywać naturalną strukturę HTML. React kładzie duży nacisk na znajomość nowoczesnego JavaScriptu.
- » Na ten moment React lepiej wspiera natywne aplikacje mobilne. Również w ekosystemie Vue powstał framework Weex, stworzony przez grupę Alibaba, jednak React Native jest zdecydowanie dojrzałszym i lepiej wypróbowanym rozwiązaniem.
- » Często podkreślana zaletą Vue w stosunku do Reacta jest lepiej prowadzona dokumentacja, co spowodowane jest dynamicznym rozrostem Reacta – przez co nie zawsze nadają ona dostatecznie szybko za zmianami.
- » Vue w przeciwieństwie do Reacta znacznie łatwiej skaluje się do małych aplikacji, w pewnych przypadkach jego implementacja jest tak łatwa jak wdrożenie jQuery.

- » Dostarczane przez Vue narzędzie vue-cli ułatwia tworzenie schematów nowych projektów, create-react-app w tym przypadku jest mniej rozbudowanym narzędziem.

Angular

Główną cechą Angulara jest jego całościowe podejście. W tym aspekcie różni się on od Vue i Reacta jako framework mający dostarczyć wszystkiego, co potrzebne do budowy aplikacji SPA. Duży nacisk w Angularu kładzie się także na wykorzystywanie TypeScriptu. W bardzo wielu przypadkach jest to zaleta, jednak korzystanie z Angulara do prostszych rozwiązań może być jak strzelanie z armaty do wróbla. Zarówno React, jak i Vue są frameworkami ułatwiającymi pracę z widokiem aplikacji, zaś Angular jest całościowym rozwiązaniem MVC.

Warto odnotować, że projekt Vue + vuex + vue router mieści się w 30 kB (min+gzip). Jest to znacząca różnica w stosunku do 130 kB (min+gzip) Angulara. Największym podobieństwem między Angularem a Vue jest system szablonów. W obu przypadkach jest to pewnego rodzaju rozszerzony HTML.

PRZYPADKI UŻYCIA VUE

Coraz więcej znanych przedsiębiorstw decyduje się na wykorzystanie Vue, należą do nich między innymi: Xiaomi, Alibaba, WizzAir, Grammarly, Optimizely, Gitlab, Adobe, Reuters czy Facebook (newsfeed.fb.com). Vue został również głównym front-endowym frameworkiem zalecanym do pracy z Laravel – co więcej, jednym ze sponsorów pracy nad Vue.js jest właśnie Laravel.

Ciekawym projektem, o którym warto wspomnieć, jest w pełni open-source'owy Vue Storefront (www.uestorefront.io) tworzony przez firmę Divante. Jego głównym założeniem jest przygotowanie rozwiązania eCommerce wspierającego dowolne platformy sklepów (typu PrestaShop czy Magento). Do głównych cech projektów opartych o Vue Storefront mają należeć szybkość działania czy wsparcie PWA.

CO DALEJ?

Vue udowodniło, że w dziedzinie front-endu nadal jest miejsce na zmiany. Jego rosnąca popularność tylko to potwierdza. Warto przynajmniej pobić się z tym frameworkiem, bo na pewno coraz częściej będzie wymieniany obok swoich większych braci Reacta czy Angulara. Dzięki Vue wiele rzeczy można zrobić łatwiej – a uproszczenie codziennej pracy jest przecież jednym z głównych zadań frameworków czy bibliotek. Powyższy artykuł jest tylko wstępem, a w sieci jest coraz więcej dobrych źródeł, na bazie których można rozszerzyć swoją wiedzę. Na sam początek polecam w szczególności dokumentację Vue i zachęcam do dalszych eksperymentów i zabawy.



IGOR PODLAWSKI

i.podlaski@kavastudio.pl

Front-end developer i lider zespołu IT w katowickiej agencji Kava Studio. Absolwent Uniwersytetu Ekonomicznego w Katowicach. Fan nowoczesnych technologii, szybkich samochodów i dużych pieniędzy.

SZKOLENIA I DORADZTWO IT
DLA PROFESJONALISTÓW

sages

Vue.js

BUDOWANIE APLIKACJI
PRZEGŁĄDARKOWYCH

[HTTP://WWW.SAGES.COM.PL](http://www.sages.com.pl)



Praktyczne
warsztaty



Nacisk na techniki
i narzędzia



Większy wzrost
wydajności pracy

Matematyka grafiki 2D i 3D

Podczas programistycznej zabawy z grafiką 2D i 3D – szczególnie akcelerowanej przy użyciu karty graficznej – przedżej czy później każdy zderza się z zagadniem, które wielu spędza sen z powiek: z matematyką. Napisanie shadera jest praktycznie niemożliwe bez pewnej podstawowej wiedzy dotyczącej wektorów i macierzy, zimplementowanie prostego algorytmu naprowadzania rakiety w przestrzeni dwuwymiarowej wymaga przynajmniej minimalnej wiedzy z zakresu trygonometrii, a zrealizowanie detekcji kolizji pociągnie za sobą konieczność skorzystania z garści twierdzeń z zakresu geometrii przestrzennej.

Choć już od blisko dekady pracuję zawodowo jako programista, a od przeszło dwóch – programuję w ogóle, tak naprawdę jestem matematykiem – ukończyłem matematykę z informatyką na Uniwersytecie Wrocławskim. I przyznam szczerze, że nie żałuję – wiedza, którą zdobyłem w czasie studiów, już nie raz przydała się podczas pisania programów: od statystyki, przez analizę sygnałów, analizę matematyczną, aż do algebry i pokrewnej jej geometrii. Ukończenie takiego, a nie innego kierunku studiów dało mi też bardzo wyraźny obraz tego, jak bardzo informatyka jest powiązana z matematyką – w końcu przecież wywodzi się z niej w linii prostej.

Gdy ostatnio prowadziłem warsztat z podstaw programowania gier w Unity i wspomniałem, że w tym procesie praktycznie nie da się obejść bez pewnej podstawowej wiedzy z zakresu arytmetyki wektorów, usłyszałem od części uczestników jęk zawodu. Dlatego też zdecydowałem się napisać artykuł poświęcony pewnym aspektom geometrii dwu- i trójwymiarowej, swoiste kompendium, do którego będzie można zatrzymać się w przypadku takich czy innych matematycznych problemów. Postaram się również podać praktyczne przykłady, w jaki sposób można wykorzystać tę wiedzę. Dodam też od razu, że skupię się na samych wzorach i zależnościach, pomijając twierdzenia i dowody, ponieważ chcę, żeby artykuł miał charakter bardziej praktyczny – zainteresowani bez większych problemów odnajdą odpowiednią literaturę w Internecie.

Zakładam, że czytelnik ma przynajmniej podstawowe pojęcie na temat takich zagadnień, jak układ współrzędnych, funkcje, argumenty i wartości itd.

FUNKCJA KWADRATOWA

Naszą matematyczną podróż rozpoczęliśmy jeszcze w szkole podstawowej. Są to podstawy, można by powiedzieć, antyczne, więc poniżej zawrę tylko krótkie przypomnienie – poza tym funkcja kwadratowa przyda się też do wyjaśnienia zagadnienia modelowania funkcji.

Zatem do rzeczy. Funkcją kwadratową nazywamy taką funkcję, która ma postać:

$$ax^2 + bx + c = 0$$

Wzór 1. Funkcja kwadratowa

Gdzie x jest parametrem funkcji, zaś a , b oraz c to pewne wartości stałe. Aby rozwiązać to równanie, liczymy pomocniczo wartość zwaną deltą, od której zależy liczba wyników. Pół życia zachodziłem w głowę, skąd taka dziwna nazwa i symbol, bo nie skojarzyłem, że przecież jest to po prostu zwykła, wielka grecka litera.

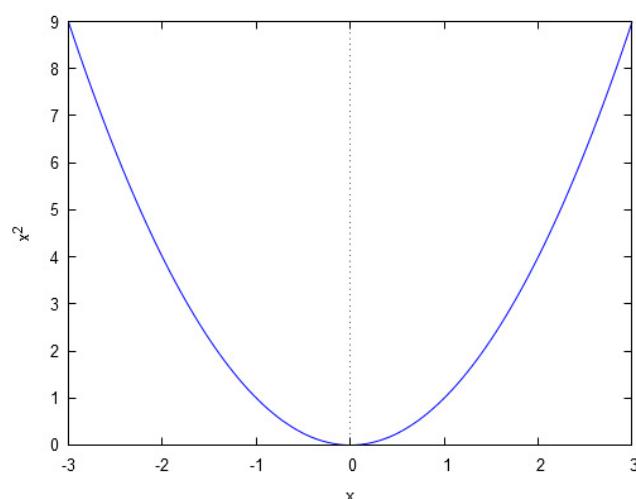
$$\Delta = b^2 - 4ac$$

$$\begin{cases} \text{Jeżeli } \Delta > 0, \text{ wtedy } x_1 = \frac{-b - \sqrt{\Delta}}{2a} \text{ oraz } x_2 = \frac{-b + \sqrt{\Delta}}{2a} \\ \text{Jeżeli } \Delta = 0, \text{ wtedy } x = \frac{-b}{2a} \\ \text{Jeżeli } \Delta < 0, \text{ wtedy nie ma rozwiązań} \end{cases}$$

Wzór 2. Rozwiązywanie równania kwadratowego

Jeżeli podczas prowadzenia obliczeń natknemy się na równanie kwadratowe, jest to ważna wskazówka, że mogą istnieć dwa rozwiązania problemu. Stanie się tak na przykład wówczas, gdy będziemy poszukiwać punktu oddalonego równo od dwóch innych punktów – w rzeczywistości istnieją zawsze dwa rozwiązania tego problemu i od kontekstu zależy, które z nich nas interesuje. Zdarza się również, że choć istnieją dwa rozwiązania, to tylko jedno z nich ma sens – na przykład wówczas, gdy niewiadoma oznacza czas, który musi upływać do jakiegoś wydarzenia, a jeden z wyników jest ujemny.

Wykresem funkcji kwadratowej jest parabola, którą możemy zobaczyć na Rysunku 1.



Rysunek 1. Parabola – wykres funkcji $y=x^2$

Modelowanie funkcji

Od czasu do czasu podczas pisania aplikacji zachodzi potrzeba wymodelowania funkcji matematycznej. Założmy na przykład, że

SZKOLENIA Z JAVA

KATOWICE 02.07.2018

POZNAŃ 11.07.2018

WARSZAWA 25.07.2018

KRAKÓW 06.08.2018

- TESTY AUTOMATYCZNE
- WZORCE PROJEKTOWE
- PROGRAMOWANIE REAKTYWNE

SZKOLENIA Z PYTHON

WROCŁAW 09.07.2018

ŁÓDŹ 16.07.2018

POZNAŃ 23.07.2018

WARSZAWA 13.08.2018

- PROGRAMOWANIE I TWORZENIE APLIKACJI
- PODSTAWY PROGRAMOWANIA



KOMPLEKSOWO SZKOLIMY NOWOCZESNY BIZNES



SPRAWDŹ SZKOLENIA NA WWW.HELIONSZKOLENIA.PL

Akcja	Przekształcenie	Przykład dla funkcji $\sin(x) + x^2$ oraz $c = 5$
Przesunięcie wykresu w lewo o c jednostek	$f(x + c)$	$\sin(x) + x^2 \rightarrow \sin(x + 5) + (x + 5)^2$
Przesunięcie wykresu w prawo o c jednostek	$f(x - c)$	$\sin(x) + x^2 \rightarrow \sin(x - 5) + (x - 5)^2$
Przesunięcie wykresu w górę o c jednostek	$f(x) + c$	$\sin(x) + x^2 \rightarrow \sin(x) + x^2 + 5$
Przesunięcie wykresu w dół o c jednostek	$f(x) - c$	$\sin(x) + x^2 \rightarrow \sin(x) + x^2 - 5$
Odbicie lustrzane według osi OY (poziome)	$f(-x)$	$\sin(x) + x^2 \rightarrow \sin(-x) + (-x)^2$
Odbicie lustrzane według osi OX (pionowe)	$-f(x)$	$\sin(x) + x^2 \rightarrow -(sin(x) + x^2)$
Skalowanie wykresu wzduż osi OX (poziome) c razy	$f(\frac{x}{c})$	$\sin(x) + x^2 \rightarrow \sin(\frac{x}{5}) + (\frac{x}{5})^2$ (wykres będzie pięciokrotnie szerszy, licząc od osi OY)
Skalowanie wykresu wzduż osi OY (pionowe) c razy	$f(x) \cdot c$	$\sin(x) + x^2 \rightarrow (\sin(x) + x^2) \cdot 5$ (wykres będzie pięciokrotnie wyższy, licząc od osi OX)

Tabela 1. Modelowanie funkcji matematycznych

chcemy, aby panel – element interfejsu – chował się i pokazywał poprzez animację. Najprostszym sposobem byłoby oczywiście zrealizowane animacji liniowej, to znaczy zmiany szerokości lub przesuwania panelu proporcjonalnie do upływającego czasu. Rozwiążanie to, choć najprostsze, nie jest jednak zbyt estetyczne, ponieważ element porusza się w sposób nienaturalny – na początku gwałtownie przyspiesza, potem porusza się cały czas z tą samą prędkością, a następnie gwałtownie zatrzymuje.

Znacznie bardziej naturalnym (dla oka) zachowaniem byłoby, gdyby element zaczynał chować się powoli, a potem coraz bardziej przyspieszał. Jeżeli przyjrzymy się funkcji kwadratowej, możemy zwrócić uwagę na to, że w pobliżu wierzchołka funkcja ta zachowuje się w pożądany przez nas sposób: najpierw zmienia się bardzo powoli, a potem coraz szybciej (wprawdzie trochę nie w tym kierunku, co trzeba, ale zaraz temu zaradzimy).

Załóżmy, że chcemy, by element o szerokości 240 pikseli schował się w czasie 1 sekundy. Potrzebujemy więc funkcji, która z jednej strony ma kształt paraboli – żeby zachowywała się w oceniany przez nas sposób, ale z drugiej dla argumentu 0 (czyli w momencie rozpoczęcia animacji) osiągała wartość 240, a dla argumentu 1 (czyli w momencie zakończenia animacji) – wartość 0.

Modelowanie funkcji odbywa się na jeden z dwóch sposobów: modyfikujemy przekazywane jej argumenty bądź jej wartości. W Tabeli 1 pokazano, w jaki sposób możemy przekształcać funkcję, by osiągnąć odpowiednie efekty, a za chwilę spróbujemy zastosować wymienione sposoby do rozwiązania naszego problemu.

Bardzo ważna uwaga jest taka, że jeżeli chcemy wykonać kilka przekształceń, musimy każde następne przekształcenie nakładać na wynik poprzedniego – może to brzmieć jak coś oczywistego, ale naprawdę łatwo się tu pomylić. Trochę zaskakujące może również wydać się to, że przeskalowanie wykresu poziomo wymaga podzielenia argumentu przez zadaną stałą, ale tak właśnie jest – proponuję sprawdzić w dowolnym programie matematycznym albo choćby na Wolfram Alpha: [http://www.wolframalpha.com/input/?i=sin\(x\)%2Bx%5E2;+sin\(x%2F2\)%2B\(x%2F2\)%5E2](http://www.wolframalpha.com/input/?i=sin(x)%2Bx%5E2;+sin(x%2F2)%2B(x%2F2)%5E2).

Przećwiczmy teraz modelowanie funkcji na poprzednim przykładzie. Na wejściu mamy funkcję $y = x^2$, którą chcemy dopasować tak, by na przedziale $[0, 1]$ malała od 240 do 0. Pierwszym krokiem

będzie odbicie jej względem osi OX, ponieważ $= x^2$ jest funkcją rosnącą, a my potrzebujemy malejącej:

$$y = x^2 \rightarrow y = -x^2$$

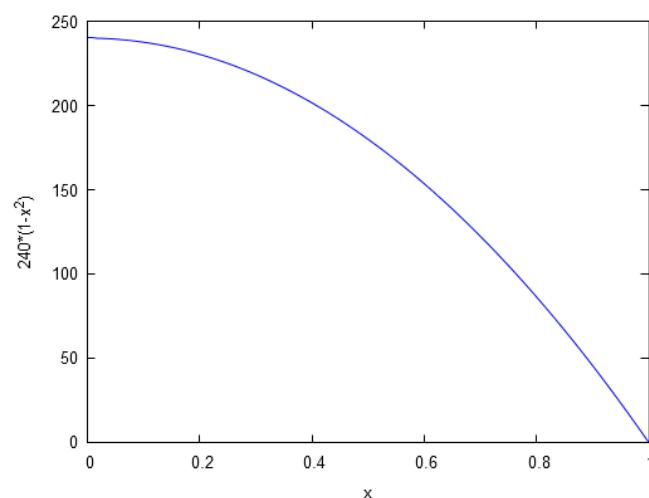
Odbita funkcja schowała się całkiem pod osią OX, więc potrzebujemy przesunąć ją do góry tak, aby na odcinku $(0, 1)$ osiągała wartości $(1, 0)$. W punkcie 1 osiąga ona wartość -1, więc musimy podnieść ją o jednostkę do góry. Pamiętajmy o tym, by przekształcać wynik poprzedniego przekształcenia:

$$y = -x^2 \rightarrow y = -x^2 + 1$$

Na koniec potrzebujemy rozciągnąć funkcję pionowo tak, by osiągała wartości w zakresie $(240, 0)$ zamiast $(1, 0)$. Korzystamy więc z odpowiedniego wzoru z tabelki i otrzymujemy następujący wynik:

$$y = -x^2 + 1 \rightarrow y = (-x^2 + 1) \cdot 240$$

W ten sposób otrzymujemy ostateczną funkcję, która wygląda następująco:



Rysunek 2. Wymodelowana funkcja

Teraz możemy napisać funkcję, która obliczy szerokość elementu uzależnioną od czasu – dla wygody początkową szerokość możemy przekazać jako parametr:

Listing 1. Funkcja obliczająca położenie w czasie

```
public float EvalWidth(float time, float initialWidth)
{
    if (time < 0.0f || time > 1.0f)
        throw new ArgumentOutOfRangeException(nameof(time));

    return -(time * time) + 1) * initialWidth;
}
```

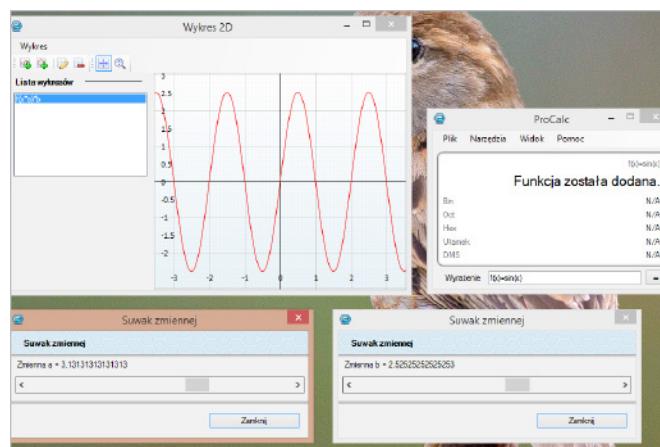
A jeżeli taka animacja nam nie odpowiada? Element wprawdzie zaczyna animować się powoli, ale potem z impetem zatrzymuje. W takiej sytuacji musimy odnaleźć taką funkcję, która bardziej nam odpowiada, zachowując się na pewnym odcinku w pożądany sposób. Funkcją taką może być na przykład wykres funkcji $\cos(x)$, która w przedziale $(0, \pi)$ najpierw powoli się rozpędza, a potem łagodnie wyhamowuje. Proponuję spróbować swoich sił w modelowaniu funkcji tak, aby spełniała nasze pierwotne wymagania, a poniżej – dla sprawdzenia – wynik:

$$\frac{\cos(x \cdot \pi) + 1}{2} \cdot 240$$

Wzór 3. Wymodelowana funkcja łagodnej animacji

Zawsze warto sprawdzić, czy wynik jest dopasowany do naszych potrzeb, rysując wykres w dowolnym programie matematycznym takim jak na przykład Maxima czy Octave lub w jednym z internetowych serwisów takich jak na przykład Wolfram Alpha.

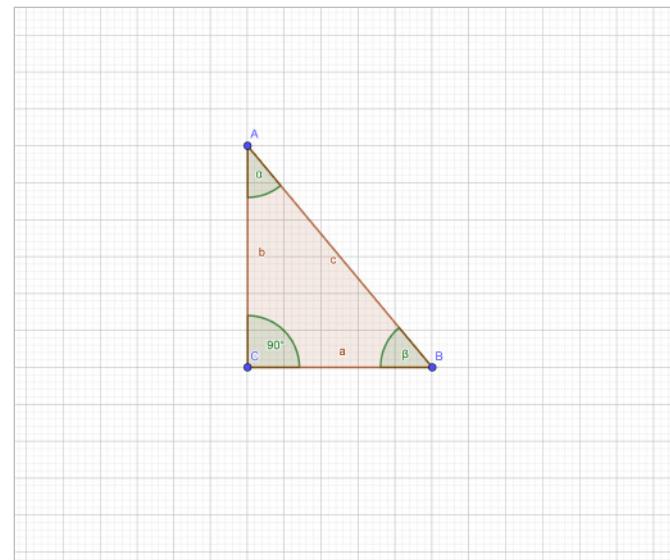
Miedzy innymi w celu eksperymentowania z modelowaniem funkcji napisałem też aplikację narzędziową – kalkulator programistyczny, który oprócz obliczania wyrażeń matematycznych pozwala na obserwowanie zachowania wykresu funkcji podczas manipulacji jej parametrami (Rysunek 3). ProCalca można ściągnąć za darmo z mojej strony domowej (link na końcu artykułu).



Rysunek 3. ProCalc

TRYGONOMETRIA

Przechodzimy teraz powoli do geometrii, a na wstępnie przypomnijmy sobie definicję funkcji trygonometrycznych. Dla dowolnego trójkąta prostokątnego definiujemy:



Rysunek 4. Trójkąt prostokątny

$$\sin(\alpha) = \frac{a}{c}$$

$$\cos(\alpha) = \frac{b}{c}$$

$$\sin(\beta) = \frac{b}{c}$$

$$\cos(\beta) = \frac{a}{c}$$

Wzór 4. Funkcje trygonometryczne

Oczywiście każda szanująca się biblioteka matematyczna (ba, standardowa) pozwala obliczyć sinus czy cosinus zadanego kąta bez budowania odpowiedniego trójkąta prostokątnego. Musimy jednak mieć na uwadze, że przeważająca większość z nich jako argument przyjmuje kąt podany w radianach, a nie w stopniach.

Radiany są sposobem wyrażenia kąta w postaci wycinka obwodu jednostkowego okręgu. Jak pamiętamy, obwód okręgu jest równy $O = 2\pi r$, co w przypadku okręgu jednostkowego (czyli o promieniu równym 1) sprowadza się do wartości 2π . Wartość ta, oznaczająca cały obwód, jest równoważna wartości 360° . Idąc dalej, połowa obwodu, równa π , odpowiada 180° itd. Aby więc przeliczyć stopnie na radiany, musimy wykonać następujące obliczenia:

$$rad = deg \cdot \frac{\pi}{180} \approx deg \cdot 0,01745329$$

$$deg = rad \cdot \frac{180}{\pi} \approx rad \cdot 57,2957795$$

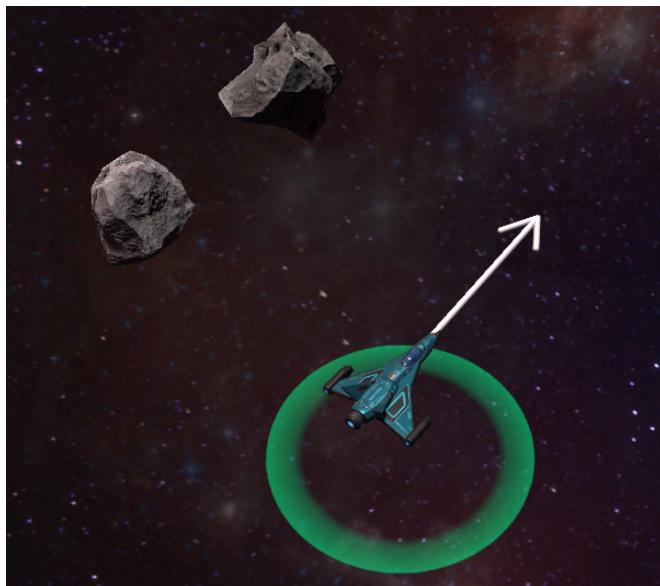
Wzór 5. Przeliczanie radianów i stopni

Jeszcze słowo komentarza – wśród podstawowych funkcji trygonometrycznych jest też oczywiście tangens i cotangens. Nie wspominam o nich jednak dlatego, że – przynajmniej z mojego doświadczenia – podczas programowania korzystałem z nich naprawdę sporadycznie.

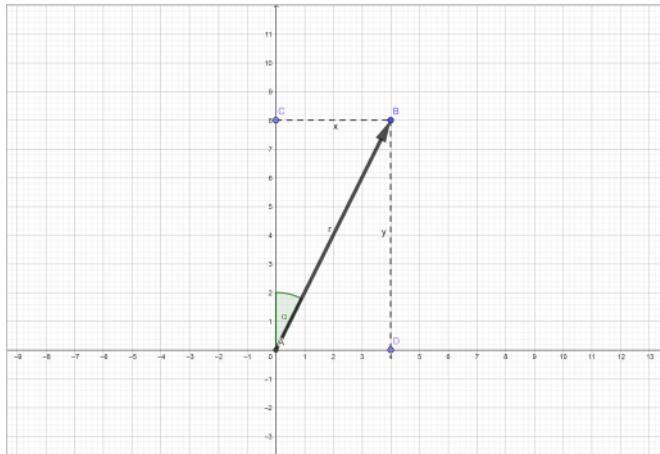
Budowanie odcinków

Funkcje trygonometryczne mają mnóstwo zastosowań, ale bezpośrednio przydają się najbardziej wówczas, gdy potrzebujemy zbu-

dować odcinek nachylony o określony kąt bądź na odwrót: mając dany odcinek, dowiedzieć się, pod jakim kątem jest on nachylony. Przykład z życia – piszę teraz grę 2,5D w Unity, w której samodzielnie modeluję fizykę (2,5D oznacza grę wyświetlaną w trzech wymiarach, ale rozgrywającą się na płaszczyźnie). W każdej klatce muszę obliczyć, o ile przesunie się statek kosmiczny (Rysunek 5). W zakresie danych mam jednak dostępną tylko jego orientację α (odchylenie od osi OY) oraz prędkość r (czyli długość przeciwprostokątnej). Matematyczny model problemu możemy znaleźć na Rysunku 6.



Rysunek 5. Jak obliczyć współrzędne docelowe?



Rysunek 6. Matematyczny model problemu

Jeżeli dobrze przyjrzymy się zastanowej sytuacji, odkryjemy trójkąt prostokątny, którego przyprostokątnymi (czyli prostopadłymi ramionami) są szukane przez nas współrzędne, zaś odcinek, o który chcę przesunąć statek nachylony pod zadanym kątem, tworzy jego przeciwprostokątną. Możemy teraz przekształcić wzory na funkcje trygonometryczne, by obliczyć szukane wartości:

$$\sin(\alpha) = \frac{x}{r} \rightarrow x = r \sin(\alpha)$$

$$\cos(\alpha) = \frac{y}{r} \rightarrow y = r \cos(\alpha)$$

Wzór 6. Obliczanie współrzędnych budowanego odcinka

Co ważne, powyższe wzory zadziałyają nawet wówczas, gdy technicznie nie będzie możliwe zbudowanie trójkąta prostokątnego – to znaczy gdy kąt α będzie większy lub równy niż 90° .

Obliczanie kąta

Funkcje trygonometryczne mają również funkcje odwrotne – i tak dla sinusa mamy arcus sinus, zaś dla cosinusa – arcus cosinus, działające według wzoru:

$$\sin(\alpha) = r \rightarrow \arcsin(r) = \alpha$$

$$\cos(\alpha) = r \rightarrow \arccos(r) = \alpha$$

Wzór 7. Funkcje odwrotne do funkcji trygonometrycznych

Tu oczywiście natychmiast istotna uwaga. Pamiętajmy, że funkcje trygonometryczne nie są różnowartościowe, to znaczy zwracają tę samą wartość dla wielu różnych argumentów. Na przykład:

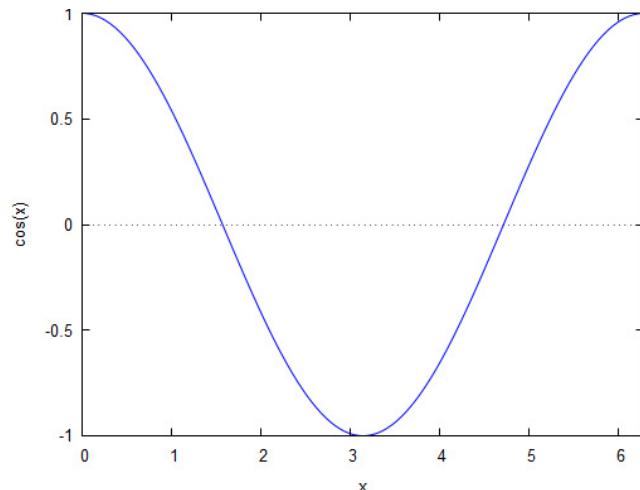
$$\sin(0^\circ) = 0$$

$$\sin(180^\circ) = 0$$

$$\sin(360^\circ) = 0$$

...i tak dalej

Wzór 8. Funkcje trygonometryczne są nieróżnowartościowe!



Rysunek 7. Wykres funkcji $\cos(x)$

W konsekwencji funkcja odwrotna do sinusa zwraca tylko wąski zakres wyników o szerokości 180° (lub π w przypadku radianów), np. $[0, 180]$ lub $[-90, 90]$. Dlatego też musimy wykonać dodatkowe sprawdzenie, czy zwracanego przez funkcję kąta nie musimy poprawić. Przyjmijmy oznaczenia z Rysunku 6.

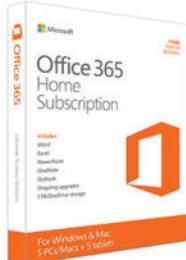
Listing 2. Liczymy kąt na bazie zadanych współrzędnych

```
public float GetAngle(float x, float y)
{
    // Jeżeli współrzędne są zerowe, kąt nie istnieje
    if (Math.Abs(x) < 0.000001 && Math.Abs(y) < 0.000001)
        throw new ArgumentException("Coordinates are too close to zero!");

    // Liczymy przeciwprostokątną
    float r = (float)Math.Sqrt(x * x + y * y);
    // Cosinus kąta to y / r, stosujemy funkcję odwrotną
    float angle = (float)Math.Acos(y / r);

    // Obliczony kąt należy od 0 do pi - sprawdzamy, czy
    // nie musimy go poprawić
```

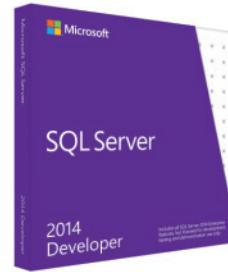
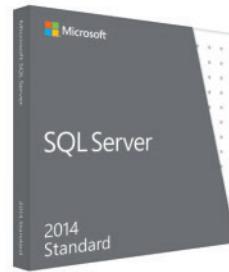
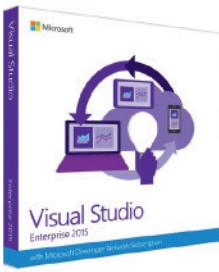
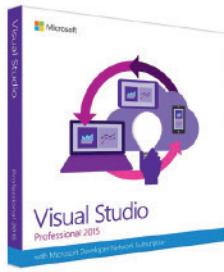
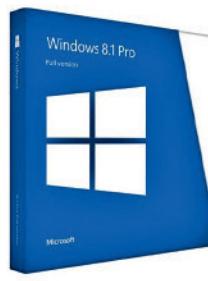
TTS Company rekomenduje oprogramowanie Microsoft ®



Microsoft Partner



Gold Datacenter
Gold Small and Midmarket Cloud Solutions
Silver Data Platform
Silver Data Analytics
Silver Cloud Productivity



www.OprogramowanieKomputerowe.pl

Microsoft Azure

Office 365

OneDrive

Więcej informacji: ☎ (22) 868 40 42 ✉ sales@tts.com.pl

Sprzedaż



Dystrybucja



Import na zamówienie

```

if (x < 0)
{
    angle = (float)(2 * Math.PI - angle);
}

// Tu możemy potencjalnie przeliczyć kąt na stopnie
return angle;
}

```

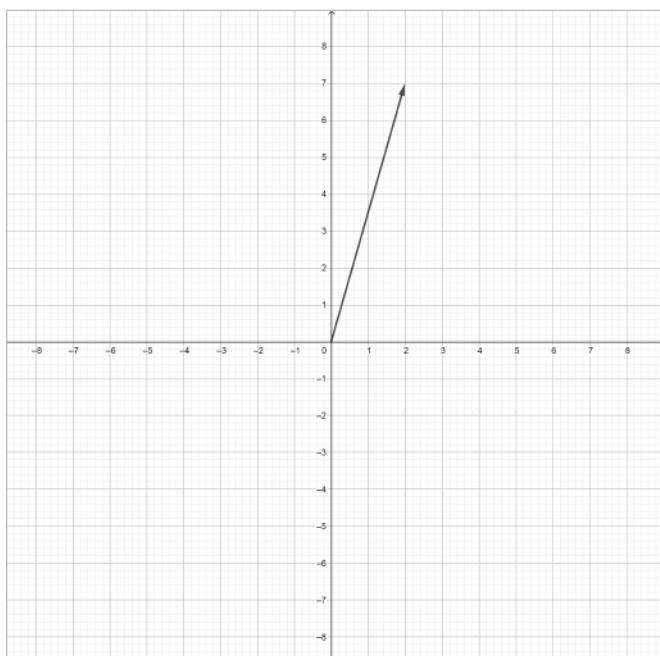
WEKTORY I MACIERZE

Wektorem nazywamy ciąg liczb, zapisywanych zwykle w następujący sposób:

$$\begin{bmatrix} 2 \\ 7 \end{bmatrix}$$

Wzór 9. Wektor

Każdy tak zapisany wektor ma swoje odzwierciedlenie w układzie współrzędnych, na przykład powyższy wektor zwizualizujemy następująco:



Rysunek 8. Wektor

Wektory w kontekście geometrycznym mają kilka ważnych cech. Po pierwsze, są generalnie utożsamiane z punktami, na które wskazują, dlatego często pojęć „wektor” i „punkt” używa się zamiennie, w zależności od kontekstu. Wektory zawsze zaczynają się w środku układu współrzędnych. Jeżeli chcemy opisać wektor, który zaczyna się gdzieś indziej (tak zwany wektor zaczepiony), będziemy musieli zrobić to za pomocą pary: punktu rozpoczęcia oraz samego wektora. W praktyce z takiej postaci wektorów korzysta się rzadziej, ale na przykład silnik Unity ma dla takiej pary własną nazwę: Ray (promień).

Wektor charakteryzuje trzy cechy: kierunek, długość i zwrot. Przez kierunek rozumiemy prostą rozpinaną przez wektor; długość jest raczej oczywista, natomiast zwrot decyduje o tym, w którym kierunku pokazuje strzałka wektora; na przykład wektor przeciwny (czyli o przeciwnym zwrocie) do podanego wcześniej będzie miał współrzędne:

$$\begin{bmatrix} -2 \\ -7 \end{bmatrix}$$

Liczba współrzędnych wektora definiuje jego wymiar. Powyższe przykładowe wektory mają wymiar 2 i można przedstawić je na

płaszczyźnie; w grafice 3D stosuje się oczywiście wektory trójwymiarowe, natomiast większe liczby wymiarów, choć teoretycznie możliwe, pozostawmy już do rozważenia matematykom.

Długość wektora możemy obliczyć za pomocą następujących wzorów:

$$\left\| \begin{bmatrix} a \\ b \end{bmatrix} \right\| = \sqrt{a^2 + b^2}, \text{ oraz}$$

$$\left\| \begin{bmatrix} a \\ b \\ c \end{bmatrix} \right\| = \sqrt{a^2 + b^2 + c^2}$$

Wzór 10. Długość wektora

Operacje na wektorach

Istnieje kilka kluczowych operacji na wektorach. Najbardziej podstawowe z nich to dodawanie i odejmowanie.

$$\begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a + c \\ b + d \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} - \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a - c \\ b - d \end{bmatrix}$$

Wzór 11. Dodawanie i odejmowanie wektorów

Wektor możemy też przemnożyć przez skalar (wartość skalarną, czyli liczbę).

$$\begin{bmatrix} a \\ b \end{bmatrix} \cdot c = \begin{bmatrix} a \cdot c \\ b \cdot c \end{bmatrix}$$

Wzór 12. Mnożenie wektora przez skalar

Mnożenie przez skalar ma pewną ciekawą właściwość – długość wynikowego wektora jest równa długości wektora źródłowego, przemnożonej przez ten sam skalar. Istnieje również operacja zwana normalizacją, polegająca na podzieleniu wektora przez jego długość – otrzymamy wówczas wektor o tym samym kierunku i zwrocie, ale o długości równej 1. Jest to jedna z najczęściej wykonywanych operacji – Unity pozwala na znormalizowanie wektora przy użyciu metody `normalize()` lub pobranie jego znormalizowanej wersji przy użyciu własności `normalized`.

Wektor prostopadły

Okazuje się, że mając dany niezerowy, dwuwymiarowy wektor, bardzo łatwo jest zbudować wektor do niego prostopadły. W tym celu zamieniamy współrzędne miejscami i zmieniamy znak jednej z nich. Wektorem prostopadłym do:

$$\begin{bmatrix} 2 \\ 7 \end{bmatrix}$$

będzie więc na przykład wektor:

$$\begin{bmatrix} 7 \\ -2 \end{bmatrix}$$

Iloczyny wektorów

Nie istnieje bezpośrednio coś takiego jak mnożenie wektorów, ale mamy dla nich zdefiniowane dwa iloczyny – skalarny i wektorowy.

Iloczyn skalarny polega na obliczeniu sumy iloczynów odpowiadających sobie współrzędnych:

$$\begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} c \\ d \end{bmatrix} = a \cdot c + b \cdot d$$

Wzór 13. Iloczyn skalarny

Nazwa iloczynu skalarnego wzięła się z tego, że jego wynikiem jest skalar, czyli liczba.

Iloczyn skalarny jest dosyć ciekawy, ponieważ ma swoją interpretację geometryczną. Jeżeli pierwszy z wektorów nazwiemy \vec{A} , zaś drugi \vec{B} , to prawdziwa jest następująca równość ($|\vec{A}|$ oznacza długość wektora):

$$\vec{A} \cdot \vec{B} = |\vec{A}| \cdot |\vec{B}| \cdot \cos(\varphi(\vec{A}, \vec{B}))$$

Wzór 14. Geometryczna interpretacja iloczynu skalarnego

Jeżeli oba wektory są znormalizowane, to możemy w stosunkowo łatwy sposób obliczyć kąt pomiędzy nimi, traktując iloczyn skalarny arcus cosinusem – co więcej, kąt pomiędzy dwoma wektorami nigdy nie przekroczy 180° , więc nie trzeba również robić żadnego dodatkowego testowania widocznego w Listingu 2. Idąc dalej, łatwo jest też sprawdzić, czy dwa wektory są prostopadłe, bo w takim przypadku ich iloczyn skalarny będzie równy zero.

Iloczyn wektorowy jest nieco bardziej skomplikowany, a jego definicja wygląda następująco:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2 \cdot b_3 - a_3 \cdot b_2 \\ a_1 \cdot b_3 - a_3 \cdot b_1 \\ a_1 \cdot b_2 - a_2 \cdot b_1 \end{bmatrix}$$

Wzór 15. Iloczyn wektorowy

Ten skomplikowany wzorek również jest dosyć przydatny. Jeżeli bowiem mnożone wektory nie leżą na jednej linii, otrzymany wektor będzie prostopadły do nich obu (a ściślej, do powierzchni, jaką wektory te wyznaczają). W ten sposób możemy bardzo łatwo obliczyć tzw. normalną do powierzchni, która przydatna jest przy pisaniu shaderów.

Macierz

Macierzą nazywamy tablicę liczb, na której zdefiniowane są odpowiednie działania.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Wzór 16. Przykładowa macierz o wymiarach 3×3

Dodawanie i odejmowanie macierzy jest stosunkowo łatwe i sprawdza się generalnie do tej samej metody, co w przypadku wektorów: dodajemy lub odejmujemy odpowiadające sobie współczynniki.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} + \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} a+j & b+k & c+l \\ d+m & e+n & f+o \\ g+p & h+q & i+r \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} - \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} a-j & b-k & c-l \\ d-m & e-n & f-o \\ g-p & h-q & i-r \end{bmatrix}$$

Wzór 17. Dodawanie i odejmowanie macierzy

Równie pożytecznym co skomplikowanym działaniem jest mnożenie macierzy, które realizowane jest według reguły: wynikowy element w i -tym wierszu i j -tej kolumnie powstaje z iloczynu skalarnego i -tego wiersza pierwszej macierzy i j -tej kolumny drugiej macierzy. Na szczęście operacji tej nie trzeba zwykle implementować samodzielnie, każda szanująca się biblioteka matematyczna potrafi na pewno to zrobić – podobnie jak karta graficzna. Poniżej przedstawiam wzór na przemnożenie dwóch macierzy trójwymiarowych – kolorami i pogrubieniem pokazuję, skąd bierze się wartość jednego z elementów macierzy wynikowej.

$$\begin{bmatrix} a & b & c \\ \textcolor{blue}{d} & \textcolor{blue}{e} & \textcolor{blue}{f} \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} j & k & \textcolor{blue}{l} \\ m & n & o \\ p & q & \textcolor{blue}{r} \end{bmatrix} = \begin{bmatrix} a \cdot j + b \cdot m + c \cdot p & a \cdot k + b \cdot n + c \cdot q & a \cdot l + b \cdot o + c \cdot r \\ d \cdot j + e \cdot m + f \cdot p & d \cdot k + e \cdot n + f \cdot q & \textcolor{blue}{d} \cdot \textcolor{blue}{l} + \textcolor{blue}{e} \cdot o + \textcolor{blue}{f} \cdot r \\ g \cdot j + h \cdot m + i \cdot p & g \cdot k + h \cdot n + i \cdot q & g \cdot l + h \cdot o + i \cdot r \end{bmatrix}$$

Wzór 18. Mnożenie macierzy

Pewną szczególną cechą mnożenia macierzy jest fakt, że można mnożyć macierze o różnych wymiarach, o ile tylko szerokość pierwszej jest równa wysokości drugiej. W wyniku otrzymamy macierz, której szerokość jest równa szerokości drugiej macierzy, a wysokość – wysokości pierwszej.

Wyznacznik macierzy

Najbardziej interesują nas jednak macierze kwadratowe – za chwilę dowiemy się dlaczego. Szczególnym działaniem na kwadratowej macierzy jest operacja obliczenia jej wyznacznika. Można ją zdefiniować rekurencyjnie:

- » Wyznacznik macierzy o wymiarze 1 jest równy jej jedynemu elementowi;
- » Wyznacznik macierzy o wymiarze n jest równy następującej sumie:

$$\det(M) = \sum_{i=1}^n (-1)^{i+1} \cdot m_{i,1} \cdot \det(M'_{i,1})$$

Wzór 19. Wyznacznik macierzy

gdzie

- » $m_{i,1}$ oznacza element znajdujący się w pierwszej kolumnie i i -tym wierszu;
- » $M'_{i,1}$ oznacza macierz M , z której usunięta została pierwsza kolumna i i -ty wiersz.

Powyższy zapis oznacza mniej więcej: wyznacznik macierzy M to naprzemienna suma (ze znakami zmieniającymi się co składnik) iloczynów elementów z pierwszej kolumny i i -tego wiersza oraz wy-

znaczników macierzy powstałych z usunięcia z macierzy M pierwszej kolumny i i-tego wiersza.

Ponieważ zarówno powyższy zapis, jak i moja próba wytłumaczenia go dla niektórych może wydawać się niejasna, prześledźmy, jak obliczyć wyznaczniki macierzy o wymiarach 2 i 3. W nawiasach klamrowych zarysowałem, skąd biorą się konkretne wartości. Zwróćmy szczególną uwagę na naprzemienność znaków poszczególnych składników sumy – łatwo o tym zapomnieć, co owocuje trudnymi do wyśledzenia błędami.

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \left\{ \begin{pmatrix} a & b \\ c & \textcolor{red}{d} \end{pmatrix} - \begin{pmatrix} a & \textcolor{red}{b} \\ \textcolor{blue}{c} & d \end{pmatrix} \right\} = a \cdot d - b \cdot c$$

$$\det \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \left\{ \begin{pmatrix} a & b & c \\ d & \textcolor{red}{e} & \textcolor{red}{f} \\ g & \textcolor{blue}{h} & \textcolor{blue}{i} \end{pmatrix} - \begin{pmatrix} a & \textcolor{red}{b} & c \\ \textcolor{blue}{d} & e & f \\ g & \textcolor{blue}{h} & \textcolor{blue}{i} \end{pmatrix} + \begin{pmatrix} a & \textcolor{red}{b} & \textcolor{red}{c} \\ \textcolor{blue}{g} & h & i \end{pmatrix} \right\} = a(e \cdot i - h \cdot f) - d(b \cdot i - h \cdot c) + g(b \cdot f - e \cdot c)$$

Wzór 20. Wyznaczniki macierzy

Układy równań liniowych

Wyznacznik sam w sobie nie niesie zbyt dużo pożytecznej informacji, ale można go wykorzystać do uproszczenia i zautomatyzowania niektórych obliczeń. Jednym z najskuteczniejszych jego zastosowań jest metoda obliczenia układu n równań liniowych z n niewiadomymi.

Przyjrzymy się temu rozwiązaniu dla układu stopnia trzeciego. Jego uogólniona postać wygląda następująco:

$$\begin{cases} a_1 \cdot x + b_1 \cdot y + c_1 \cdot z = d_1 \\ a_2 \cdot x + b_2 \cdot y + c_2 \cdot z = d_2 \\ a_3 \cdot x + b_3 \cdot y + c_3 \cdot z = d_3 \end{cases}$$

Wzór 21. Układ trzech równań z trzema niewiadomymi

Układ taki możemy zapisać w postaci następującej macierzy prostokątnej:

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{bmatrix}$$

Wzór 22. Współczynniki równań liniowych ułożone w macierzy

Aby szybko i automatycznie obliczyć taki układ równań, na początku musimy policzyć tak zwany wyznacznik główny – W , a jest on wyznacznikiem macierzy powstały z usunięcia z macierzy równań kolumny wyrazów wolnych, czyli tych, przy których nie ma żadnej zmiennej (w tym przypadku kolumny d) :

$$W = \det \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}$$

Wzór 23. Wyznacznik główny

Jeżeli wyznacznik główny jest niezerowy, istnieje dokładnie jedno rozwiązanie równania. Liczymy wówczas wyznaczniki poszczególnych niewiadomych, budując macierze w podobny sposób, co wyznacznik główny, to jest na przykład macierz wyznacznika W_x utworzymy poprzez usunięcie z macierzy równania wszystkich

współczynników przy niewiadomej x (czyli w naszym przypadku kolumny a):

$$W_x = \det \begin{pmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \end{pmatrix}$$

$$W_y = \det \begin{pmatrix} a_1 & c_1 & d_1 \\ a_2 & c_2 & d_2 \\ a_3 & c_3 & d_3 \end{pmatrix}$$

$$W_z = \det \begin{pmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{pmatrix}$$

Wzór 24. Wyznaczniki niewiadomych

Na koniec pozostało nam tylko obliczenie wartości niewiadomych:

$$\begin{cases} x = \frac{W_x}{W} \\ y = \frac{W_y}{W} \\ z = \frac{W_z}{W} \end{cases}$$

Wzór 25. Obliczamy wartości niewiadomych

Choć jest to rzadko przydatne, w przypadku gdy wyznacznik główny jest zerowy i jednocześnie wszystkie wyznaczniki niewiadomych są zerowe, możemy wywnioskować, że rozwiązań jest nieskończenie wiele, natomiast gdy przy zerowym wyznaczniku głównym choć część wyznaczników niewiadomych jest niezerowa, a część zerowa, rozwiązanie nie istnieje w ogóle.

Miejsce trafienia promienia w płaszczyźnie

Zobaczmy, jak można zastosować metodę wyznaczników w przypadku konkretnego problemu. Gdy w mojej grze użytkownik kliknie w dowolnym miejscu, muszę obliczyć współrzędne miejsca, w które kliknął. Ponieważ jednak gra jest trójwymiarowa, kliknięcie więc jest nie tyle punktem, co bardziej promieniem – prostą przechodzącą przez pozycję kamery oraz przez miejsce kliknięcia na wirtualnym ekranie (Rysunek 9).



Rysunek 9. Kliknięcie w grze 3D

Płaszczyznę gry (OXZ) rozpinają dwa wektory:

$$\vec{X} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

oraz

$$\vec{Z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Aby odnaleźć przecięcie promienia myszy (opisanego przez jego początek – M oraz wektor \vec{D}) z tą płaszczyzną, musimy rozwiązać równanie:

$$t \cdot \vec{X} + s \cdot \vec{Z} = M + u \cdot \vec{D}$$

Wzór 26. Poszukujemy punktu przecięcia

Powyzszy wzór oznacza poszukiwanie punktu, który z jednej strony jest równy sumie (pewnych) przedłużen wektorów X i Z , a z drugiej – (pewnego) przedłużeniu wektora D zaczepionego w punkcie M . Jeżeli rozbijemy teraz powyższe równanie na poszczególne współrzędne, otrzymamy trzy równania z trzema niewiadomymi (t, s, u):

$$\begin{cases} t \cdot Xx + s \cdot Zx = Mx + u \cdot Dx \\ t \cdot Xy + s \cdot Zy = My + u \cdot Dy \\ t \cdot Xz + s \cdot Zz = Mz + u \cdot Dz \end{cases}$$

Wzór 27. Układ równań liniowych

Wystarczy teraz przenieść na drugą stronę równań niewiadomą u , by móc zapisać wszystkie współczynniki w prostokątnej macierzy:

$$\begin{bmatrix} Xx & Zx & -Dx & Mx \\ Xy & Zy & -Dy & My \\ Xz & Zz & -Dz & Mz \end{bmatrix}$$

Wzór 28. Macierz zbudowana z równań

Teraz możemy już zastosować metodę wyznaczników. Zauważmy też, że efektywnie interesuje nas tylko punkt przecięcia, więc wystarczy obliczyć samą niewiadomą u . Gdy obliczoną wartość podstawimy do równania osi myszy, możemy ów punkt łatwo wyznaczyć, oszczędzając sobie (i procesorowi) niepotrzebnych obliczeń.

Przekształcenia przestrzeni dwuwymiarowej

Macierze kwadratowe są ciekawe również dlatego, że mają one swoją interpretację geometryczną: opisują przekształcenia wektorów.

Zacznijmy od przestrzeni dwuwymiarowej.

Identyczność

Podstawową macierzą, o której warto wiedzieć, jest macierz identycznościowa, która ma następującą postać:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Wzór 29. Macierz identyczności

Macierz identyczności ma to do siebie, że jeżeli przemnożymy przez nią wektor, otrzymamy ten sam wektor, bez żadnych zmian:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Wzór 30. Działanie macierzy identyczności

Skala

Kolejną macierzą jest macierz skali. Definiujemy ją przez określenie współczynnika skali poziomej S_x oraz pionowej – S_y i budujemy następująco:

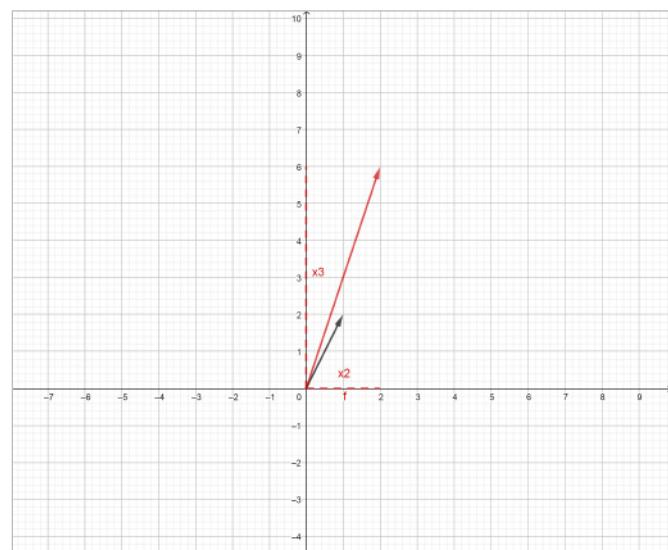
$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

Wzór 31. Macierz skali

Spróbujmy zobaczyć, jak macierz skali działa w praktyce.

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 \\ 3 \cdot 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$$

Wzór 32. Działanie macierzy skali



Rysunek 10. Skalowanie wektora

Szczególnym przypadkiem macierzy skali są również macierze odbicia lustrzanego względem osi OX lub OY:

$$M_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Wzór 33. Odbicie lustrzane

Obrót

Macierz obrotu wokół środka układu współrzędnych wygląda następująco:

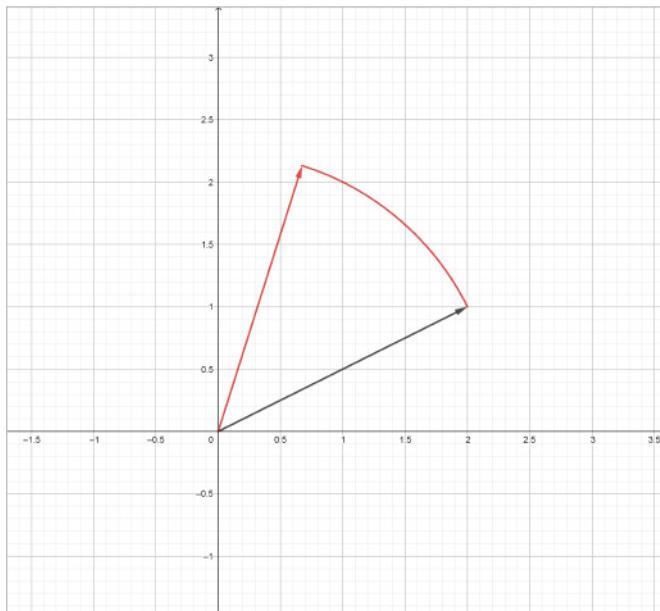
$$R_\alpha = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Wzór 34. Macierz obrotu

Sprawdźmy, w jaki sposób działa ta macierz:

$$\begin{bmatrix} \cos(45^\circ) & -\sin(45^\circ) \\ \sin(45^\circ) & \cos(45^\circ) \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2\sqrt{2}}{2} - \frac{\sqrt{2}}{2} \\ \frac{2\sqrt{2}}{2} + \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{3}{\sqrt{2}} \end{bmatrix} \approx \begin{bmatrix} 0.70 \\ 2.12 \end{bmatrix}$$

Wzór 35. Obracamy wektor wokół środka układu współrzędnych



Rysunek 11. Obrót wokół środka układu współrzędnych

Przekształcenia przestrzeni trójwymiarowej

Ponieważ wiemy już, w jaki sposób macierze przekształcają wektory (w przestrzeni trójwymiarowej dzieje się to w analogiczny sposób, jak w przypadku dwuwymiarowej), ograniczę się tylko do podania odpowiednich macierzy.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Wzór 36. Macierz identyczności

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix}$$

Wzór 37. Macierz skali

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Wzór 38. Macierze obrotów wokół odpowiednich osi

Zauważycie, że brakuje tutaj macierzy translacji? Jest tak dla tego, że translacji (czyli przesunięcia o wektor) nie da się zapisać w postaci macierzy o tym samym wymiarze, co wektor. Dlatego często stosuje się sztuczkę i sztucznie zwiększa wymiar wektora po to, by można było zapisać również macierz translacji. Wektor:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

staje się wówczas wektorem:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

a dla takiego wektora możemy już skonstruować macierz translacji:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

dla wektorów dwuwymiarowych, oraz:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dla wektorów trójwymiarowych.

Wzór 39. Macierz translacji

Jako ciekawostkę mogę przytoczyć fakt, że karty graficzne wewnętrznie operują na wektorech i macierzach czterowymiarowych – właśnie po to, by móc również wykonywać translację w przestrzeni trójwymiarowej.

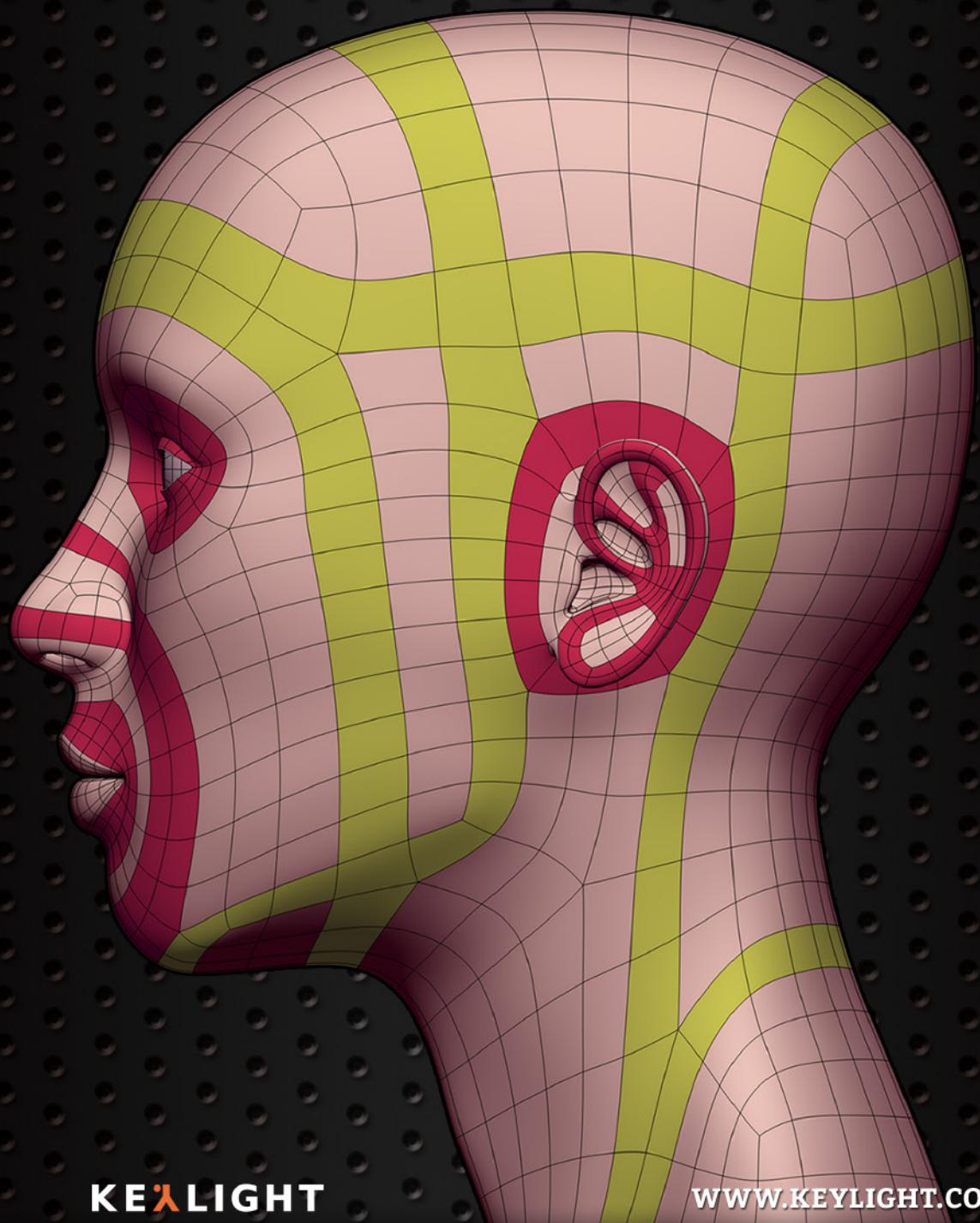
Składanie przekształceń

Teraz mogę już napisać, dlaczego mnożenie macierzy jest tak ważne: otóż mnożenie macierzy przekształceń jest równoważne złożeniu tych przekształceń. Jeżeli na przykład przemnożymy macierz skali S przez macierz obrotu O , a następnie otrzymaną macierz przemnożymy przez jakiś wektor, otrzymamy taki sam wynik, jak gdybyśmy najpierw przemnożyli macierz O przez ten wektor, a potem wynik tego działania – przez macierz S (czyli najpierw obrócili wektor, a następnie go przeskaliowali).

$$S \cdot O \cdot \vec{V} = S \cdot (O \cdot \vec{V})$$

Wzór 40. Składanie przekształceń

Tu musimy utrważyć bardzo ważną zasadę: pamiętajmy, że przekształcenia składają się zawsze **od końca**, to znaczy ostatnie przekształcenie zostanie nałożone jako pierwsze. Niektóre interfejsy API pozwalają więc zdefiniować podczas konstruowania końcowej macierzy przekształcenia, czy chcemy dane przekształcenie domnożyć z lewej, czy z prawej strony, by móc poukładać je we właściwej kolejności. I jeszcze jeden ważny fakt: mnożenie macierzy nie jest przemienne.



KEYLIGHT

WWW.KEYLIGHT.COM.PL

A po co mi to?

Wbrew pozorom zapis przekształceń przestrzeni w postaci macierzy można znaleźć w jawniej lub zakamuflowanej postaci w wiele miejscach. Najbardziej oczywistym przypadkiem są shadery, w których operacje na wektorach i macierzach są na porządku dziennym. Przyjrzyjmy się poniższemu kodowi prostego shadera:

Listing 3. Kod przykładowego shadera

```

cbuffer SceneConstantBuffer : register(b0)
{
    matrix Object;
    matrix Camera;
    matrix Projection;
    float4 LightPos;
}

cbuffer ObjectConstantBuffer : register(b1)
{
    float4 DiffuseColor;
}

struct VS_MESH_INPUT
{
    float4 Pos : POSITION;
    float4 Normal : NORMAL;
};

struct VS_MESH_OUTPUT
{
    float4 Pos : SV_POSITION;
    float4 LightPos : LIGHT0;
    float4 VertexPos : VERTEX0;
    float4 Normal : NORMAL0;
    float4 Color : COLOR0;
};

VS_MESH_OUTPUT VsMesh(VS_MESH_INPUT input)
{
    VS_MESH_OUTPUT output = (VS_MESH_OUTPUT)0;

    // Docelowa pozycja wierzchołka
    output.Pos = mul(input.Pos, Object);
    output.Pos = mul(output.Pos, Camera);
    output.Pos = mul(output.Pos, Projection);

    // Pozycja światła
    output.LightPos = LightPos;

    // Pozycja wierzchołka (w świecie)
    output.VertexPos = mul(input.Pos, Object);

    // Normalna do wierzchołka
    output.Normal = normalize(mul(input.Normal, Object));

    // Kolor wierzchołka
    output.Color = DiffuseColor;

    return output;
}

float4 PsMesh(VS_MESH_OUTPUT input) : SV_Target
{
    // Wektor wskazujący światło
    float4 LightDir = normalize(input.LightPos - input.VertexPos);

    // Cieniowanie Phonga
    float dotValue = dot(LightDir, input.Normal);
    if (dotValue < 0)
        dotValue = dot(LightDir, -input.Normal);

    return input.Color * dotValue;
}

```

Powyższy listing zawiera kod najprostszych vertex- i pixel-shaderów, które realizują projekcję perspektywiczną i dwustronne cieniowanie Phonga. Nie będę tu tłumaczył, w jaki sposób działają – zainteresowanych odsyłam do mojego artykułu o shaderach

w Direct3D (Programista 3/2012). Jednak już w powyższym kodzie możemy znaleźć dużo z opisanych wcześniej wzorów: mnożenie wektora przez macierz, iloczyn skalarny oraz normalizacja wektora.

Idąc dalej: podczas rysowania za pomocą obiektu Graphics dostarczanego przez Windows Forms można również zdefiniować przekształcenie, któremu będzie poddawany obszar wydruku, a przekształcenie to jest typu – niespodzianka – Matrix, czyli macierz. Podobnie jest z Windows Presentation Foundation, gdzie również przekształcenie jest macierzą, tylko uwaga: WPF trzyma wektory w postaci wierszy, a nie – jak wcześniej pokazałem – kolumn. Wpływają to na wszystkie wykonywane obliczenia, między innymi wektor jest mnożony przez macierz, a nie na odwrót, co odwraca też kolejność składania przekształceń – wektor najpierw jest przekształcany przez pierwsze z serii.

SKRZYNKA Z NARZĘDZIAMI

Na tym etapie wyposażymy się w całą garść narzędzi, za pomocą których możemy rozwiązywać różne matematyczne zagadnienia podczas pisania aplikacji graficznych lub gier. Zobaczmy na koniec, z jakiego rodzaju wyzwaniami można próbować mierzyć się przy użyciu zaprezentowanego przeze mnie aparatu matematycznego.

Naprowadzanie rakiet

Zabrałem się ostatnio za pisanie gry będącej wariacją na temat kosmicznej strzelaniny i w ramach implementowania różnych rodzajów uzbrojenia stanąłem przed problemem zaimplementowania mechanizmu naprowadzania rakiet. Gra jest trójwymiarowa, ale akcja toczy się na płaszczyźnie – tylko w dwóch wymiarach. Po wystrzeleniu rakiety zaczyna działać algorytm poszukujący celów, a po jego znalezieniu powinien uruchomić się kolejny, którego zadaniem jest pokierowanie rakietą w taki sposób, by w niego trafila.

W wielu grach stosowane jest – czasem bardzo wyraźnie widoczne – oszustwo polegające na tym, że rakiety wyjęte są spod ograniczeń mobilności, czyli maksymalnej prędkości, zwrotności lub przyspieszenia. Dysponując prawie nieograniczonymi możliwościami przemieszczania, nawet najbardziej prymitywny algorytm spokojnie poradzi sobie z trafieniem w cel. Ja nie chciałem postępować w ten sposób i rakiety w mojej grze muszą respektować swoje ograniczenia na równi ze wszystkimi innymi mobilnymi obiektami.

Najprostszym do zaimplementowania algorymem naprowadzania jest algorytm pościgu (ang. *simple chase*). Działa on w bardzo prosty sposób – rakieta zawsze próbuje lecieć w stronę swojego celu. Spróbujmy teraz opisać ten algorytm nieco bardziej matematycznie.

Niech T będzie bieżącym położeniem celu, R – bieżącym położeniem rakiety, zaś \vec{R} – wektorem jej prędkości (kierunek i zwrot takiego wektora określają orientację rakiety, zaś długość – jej prędkość). Przed algorytmem stoi zadanie odnalezienia docelowego wektora prędkości rakiety, który powinien doprowadzić ją do trafienia w cel. W przypadku algorytmu pościgu obliczenia są bardzo proste:

$$\vec{V} = \vec{T} - \vec{R}$$

Wzór 41. Obliczenie docelowego wektora dla algorytmu pościgu

Po obliczeniu tego wektora działanie podejmuje algorytm nawigujący, który poprzez obrót i przyspieszenie rakiety – uwzględniając jej możliwości – stara się sprawić, by wektor prędkości pokrył się z wektorem docelowym, ale o nim innym razem.

Algorytm prostego pościgu jest banalny w implementacji, ale daje bardzo marne efekty i nadaje się tak naprawdę głównie do strzelania do celów nieruchomych. Wynika to przede wszystkim z faktu, iż nie uwzględnia on przemieszczania się celu i stosunkowo łatwo można go oszukać poprzez zmuszenie rakiety do wykonywania gwałtownych manewrów. Mało tego, rakieta często nie jest w stanie trafić nawet w taki cel, który po prostu leci do przodu (Rysunek 12).



Rysunek 12. Rakietka ominęła cel

Zaimplementowany algorytm pozostawiłem dla, powiedzmy, tątszych rakiet, po czym zacząłem pracować nad nieco bardziej zaawansowanym. Zrobiłem małe rozpoznanie i zauważylem, że większość prawdziwych rakiet nie leci wcale od razu w kierunku celu, tylko stara się raczej skonstruować kurs kolizyjny, uwzględniając w ten sposób fakt, że cel się przemieszcza.

Spróbujmy więc wykonać odpowiednie obliczenia – tak aby rakieta próbowała obrać kurs kolizyjny z celem zamiast lecieć bezpośrednio w jego kierunku.

Podstawowymi danymi, którymi dysponujemy, są oczywiście położenie rakiety R oraz jej bieżący wektor prędkości, \vec{V} . Bezpośrednie skorzystanie z informacji o położeniu i prędkości celu mogłoby wydawać się oszustwem, ale okazuje się, że rzeczywiste rakiety potrafią w pewnym stopniu informacje te oszacować, więc możemy założyć, że również je znamy – nazwijmy więc położenie celu T oraz jego obecny wektor prędkości \vec{T}_s .

Wektor prędkości skonstruowany jest zawsze w taki sposób, by można było w łatwy sposób obliczyć przesunięcie obiektu w zadanym czasie. Jeżeli na przykład przemnożymy go przez 1, otrzymamy przesunięcie obiektu w czasie 1 sekundy. Możemy więc ogólnie przyjąć, że w czasie t rakieta przemieści się o wektor $t \cdot \vec{V}$, czyli jej docelowym położeniem będzie $R + t \cdot \vec{V}$. Podobnie jest oczywiście z jej celem.

Mamy do rozwiązania następujący problem: w jakim kierunku, z bieżącą prędkością, powinna lecieć rakieta, by trafić w cel? Zauważmy, że choć nie znamy współrzędnych docelowego wektora rakiety, to jednak znamy jego długość (która musi być równa właśnie bieżącej prędkości), a to jest informacja, która bardzo się przyda. Drugą, nieco mniej oczywistą niewiadomą jest czas, po którym cel zostanie trafiony – bo skoro zostanie trafiony, to do miejsca spotkania przyleci w takim samym czasie co rakieta.

Możemy teraz spróbować pozbierać wszystkie fakty i ułożyć z nich równania:

$$\begin{cases} R + t \cdot \vec{V} = T + t \cdot \vec{T}_s \\ |\vec{V}| = |\vec{T}_s| \end{cases}$$

Wzór 42. Równanie trajektorii kolizyjnej

Pierwsze z równań mówi tyle, że po upłynięciu czasu t rakieta oraz cel znajdą się w tym samym miejscu. Drugie równanie nakłada zaś ograniczenie długości na docelowy wektor.

Choć równanie jest technicznie prawdziwe, dosyć trudno będzie nad nim pracować, więc przekształćmy je najpierw na równania operujące na skalarach, czyli liczbach. W tym celu rozbijemy każdy wektor na jego składowe, na przykład V na Vx i Vy , \vec{T}_s na Tx i Ty itd. Możemy też przyjąć, że bieżąca prędkość rakiety to *speed* – usuniemy z równania niewygodny wektor, zastępując go wartością stałą. Nasze równania będą teraz wyglądały następująco:

$$\begin{cases} Rx + t \cdot Vx = Tx + t \cdot Tsx \\ Ry + t \cdot Vy = Ty + t \cdot Tsy \\ \sqrt{Vx^2 + Vy^2} = speed \end{cases}$$

Wzór 43. Równania do rozwiązywania

Aż kocri, by skorzystać tu z wyznacznikowej metody rozwiązywania równań, ale niestety trzecie równanie nie jest liniowe – niewiadome są podniesione do kwadratu – więc nie możemy z niej skorzystać. Dlatego musimy zrobić to po staremu, uzależniając Vx i Vy od t w dwóch pierwszych równaniach i podstawiając je do trzeciego.

Oczywiście da się to wszystko zrobić na kartce, ale od czego mamy komputer? Skorzystałem z fenomenalnego (i darmowego) programu Maxima, który potrafi wykonywać przekształcenia równań matematycznych (a nawet w wielu przypadkach automatycznie je rozwiązać).

Listing 4. Wprowadzamy równania do Maxima

```
(%i1) Rx+t*Vx=Tx+t*Tsx;
(%o1) Vxt + Rx = Tsxt + Tx
(%i2) Ry+t*Vy=Ty+t*Tsysty;
(%o2) Vyt + Ry = Tsysty + Ty
(%i3) sqrt(Vx^2+Vy^2)=speed;
(%o3) sqrt(Vx^2+Vy^2) = speed
```

Teraz możemy poprosić Maximę, żeby z dwóch pierwszych równań wyznaczyła odpowiednio Vx i Vy :

Listing 5. Wyznaczamy niewiadome z dwóch pierwszych równań

```
(%i4) solve(%o1, Vx)
(%o4) [Vx = \frac{Tsxt+Tx-Rx}{t}]
(%i5) solve(%o2, Vy)
(%o5) [Vy = \frac{Tsysty+Ty-Ry}{t}]
```

Ponieważ wiemy, że prędkość jest dodatnia, możemy podnieść trzecie równanie do kwadratu, pozbywając się niewygodnego pierwiastka:

Listing 6. Upraszczamy trzecie równanie

```
(%i6) %o3^2
(%o6) Vy^2 + Vx^2 = speed^2
```

Teraz możemy zamienić Vx i Vy w wyrażeniu 6 na, odpowiednio, wyrażenia 4 i 5:

Listing 7. Konstruujemy równanie z jedną niewiadomą

```
(%17) append(%o4, %o5)
(%o7) [Vx =  $\frac{Tsxt+Tx-Rx}{t}$ , Vy =  $\frac{Tsyt+Ty-Ry}{t}$ ]
(%18) subst(%o7, %o6)
(%o8)  $\frac{(Tsyt+Ty-Ry)^2}{t^2} + \frac{(Tsxt+Tx-Rx)^2}{t^2} = speed^2$ 
```

Czas jest dodatni i zakładamy, że niezerowy, więc możemy obie strony przemnożyć przez t^2 , a następnie rozwinąć nawiasy i przemieścić wszystkie składniki na lewą stronę równania.

Listing 8. Przekształcamy równanie, by obliczyć t

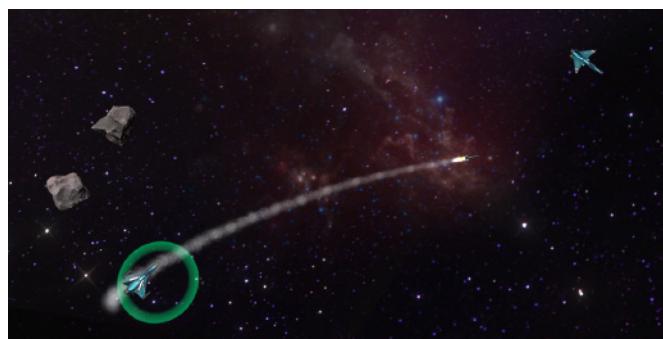
```
(%i9) %o8*t^2
(%o9) (Tsyt + Ty - Ry)^2 + (Tsxt + Tx - Rx)^2 = speed^2*t^2
(%i10) expand(%o9)
(%o10) Tsy^2*t^2 + Tsx^2*t^2 + 2TsyTyt + 2TsxtTxt - 2RyTsyt -
2RxTsxt + Ty^2 - 2RyTy + Tx^2 - 2RxTx + Ry^2 + Rx^2 = speed^2*t^2
(%i11) (%o10) - speed^2 * t^2
(%o11) Tsy^2*t^2 + Tsx^2*t^2 + 2TsyTyt + 2TsxtTxt - 2RyTsyt -
2RxTsxt + Ty^2 - 2RyTy + Tx^2 - 2RxTx + Ry^2 + Rx^2 - speed^2*t^2 = 0
```

Spokojnie, nawet mnie – technicznie matematyka – łatwo przerzą powyższe równanie. Zwróćmy jednak uwagę na dwa kluczowe fakty: wiemy, że tylko t jest w nim niewiadomą i że występuje ono tylko w dwóch postaciach: t^2 lub t . Choć zatem równanie wygląda potwornie, to mamy przed sobą tak naprawdę zwykłe równanie kwadratowe. Możemy nawet poprosić Maximę, żeby wyznaczyła z niego współczynniki a , b , i c , aby potem normalnie policzyć deltę, a potem – w konsekwencji – również t .

Listing 9. Wyznaczamy współczynniki równania kwadratowego

```
(%i12) coeff(%o11, t, 2)
(%o12) -speed^2 + Tsy^2 + Tsx^2 = 0
(%i13) coeff(%o11, t, 1)
(%o13) 2TsyTy + 2TsxtTx - 2RyTsyt - 2RxTsxt = 0
(%i14) coeff(%o11, t, 0)
(%o14) Ty^2 - 2RyTy + Tx^2 - 2RxTx + Ry^2 + Rx^2 = 0
```

Mając współczynniki równania kwadratowego praktycznie podane na tacy (bo wszystkie wartości w nich występujące są nam znane), obliczamy deltę i sprawdzamy, czy istnieją rozwiązania (bo może się zdarzyć, że ich nie będzie). Jeśli są dwa, to bierzemy to, które jest mniejsze, ale jednocześnie większe od zera (w końcu obliczamy t , czyli czas – musi być dodatni). Na koniec korzystamy z równań 4 i 5 w Listingu 5, by obliczyć docelowy wektor – i oglądamy wyniki (Rysunek 13). Prawdę mówiąc, algorytm jest tak skuteczny, że przed rakietą mającą maksymalną prędkość większą od celu praktycznie nie da się uciec.



Rysunek 13. Rakietka na kursie kolizyjnym

PODSUMOWANIE

Wszystkie wzory oraz matematyczne konstrukcje zaprezentowane w tym artykule wybrałem nieprzypadkowo – należą one do przeważającej większości matematycznych narzędzi, z których korzystam podczas pisania aplikacji oraz – ostatnio – gier. W przypadku tych ostatnich bez wahania powtórzę to, co powiedziałem na moim warsztacie: przynajmniej elementarna wiedza na temat arytmetyki wektorów i macierzy jest tam po prostu konieczna. I choć – oczywiście – nie wyczerpałem całkowicie wszystkich zagadnień matematycznych pojawiających się w aplikacjach i grach, to niniejszy artykuł z powodzeniem może stać się wstępem do zgłębiania bardziej zaawansowanych zagadnień, z których jedno – bardzo ciekawe – możecie odnaleźć w ramce „W Sieci”.

W sieci

- ▶ ProCalc, mój kalkulator dla programisty: <http://spooksoft.pl/procalc-features/>
- ▶ Tak naprowadzane są prawdziwe rakiety: https://en.wikipedia.org/wiki/Proportional_navigation
- ▶ Implementacja powyższego algorytmu w modzie do gry wraz z filmami: <http://www.moddb.com/mods/wicmw/features/flint-lead-pursuit-guidance-principles>



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.

Lepsza jakość oprogramowania



Dołącz do naszego zespołu – testerzy.pl/kariera

Unity – wprowadzenie

Myślę, że przeważająca większość programistów – włączając w to mnie – przeszła w swojej karierze okres, w czasie którego chcieli napisać grę. I przeważająca większość tej przeważającej większości stosunkowo szybko rezygnowała, a działało się to zwykle w momencie nauki renderowania grafiki w OpenGLu albo DirectX11, gdy okazywało się, że wyświetlenie na ekranie prostego sześcianu zajmuje dobre kilkadziesiąt linijek kodu. Gdy w tym momencie programista uświadadniał sobie, że po oprogramowaniu silnika grafiki stoi przed nim jeszcze zaimplementowanie animacji i efektów specjalnych, fizyki z detekcją kolizji, oprawy dźwiękowej, ubranie tego w spójną całość i dopiero wtedy może nieśmiało myśleć o jakiejkolwiek logice gry, projekt okazywał się grubo ponad siły jednej osoby. Sytuacja zmienia się jednak diametralnie, gdy skorzystamy z jednego z dostępnych na rynku silników gier. Dziś opowiem o jednym z dwóch bodaj najpopularniejszych, czyli o Unity.

PREHISTORIA, CZYLI 1994

Czy mieliście może do czynienia z takimi aplikacjami, jak Klik&Play albo The Games Factory? Pozwalały one na tworzenie gier poprzez układanie obiektów na polu gry oraz definiowaniu ich logiki poprzez proste serie regułek budowanych na zasadzie par: warunek i akcja (lub akcje). Grę można było również szybko testować w czasie jej rozwoju – po wcisnięciu „Play” aplikacja przechodziła bowiem do trybu symulacji, gdy wszystkie reguły zaczynały działać. Zakończone dzieło można było wyeksportować jako plik wykonywalny, który trzeba było rozpowszechniać z odpowiednimi bibliotekami DLL dostarczonymi lub wygenerowanymi przez aplikację.

Myślę, że wspomniany Klik&Play możemy swobodnie nazwać prekurem współczesnych silników gier takich jak Unity czy Unreal Engine. Oczywiście poziom zaawansowania tych ostatnich jest radykalnie wyższy, ale ogólna idea jest wciąż taka sama – zdjąć z projektanta ciężar implementowania tych elementów gry, które w większości z nich i tak się powtarzają, ale same w sobie wymagają dużo pracy. Tym sposobem może on skupić się na logice gry, jej wyglądzie, oprawie dźwiękowej itd., oszczędzając bardzo wymierne ilości czasu, a oszczędność ta może sięgnąć poziomu, w którym napisanie gry przez pojedynczego programistę staje się całkiem realne.

Jakiego rodzaju funkcjonalności dostarczają nam współczesne silniki gry? Przed wszystkim bardzo uproszczonego sposobu wyświetlania grafiki 3D. Ci, którzy czytali moje wprowadzenie do Direct3D (Programista 1/2012), wiedzą, że takie pojęcia jak kamera czy źródło światła na niskim poziomie w ogóle nie istnieją; wprowadzenie tych abstrakcji naprawdę bardzo ułatwia życie. Dodatkową zaletą jest też fakt, że silnik korzysta z tego interfejsu, który na danej platformie jest dostępny, na przykład z DirectX pod Windowsem i z OpenGLa pod Linuksem. Kolejnym elementem jest obsługa fizyki – ciał sztywnych (ang. *rigid body*), grawitacji i sił oraz – często kłopotliwej obliczeniowo – detekcji kolizji. Obsługa wejścia również jest bardzo uproszczona – do tego stopnia, że twórca nie musi implementować osobnej logiki dla klawiatury i joysticka lub ga-

mepada – silnik ujednolica wejście z różnych urządzeń i dostarcza konkretnych wartości liczbowych, które można wykorzystać w obliczeniach. Dostajemy również obsługę przestrzennego dźwięku: możemy zdefiniować jego źródła, ich zasięg i głośność oraz położenie „słuchacza”, by silnik sam zmięksował wszystkie próbki, biorąc pod uwagę ich kierunek i odległość. Mało? W standardzie mamy też niesamowicie konfigurowalne systemy cząstek, które pozwalają w stosunku łatwy sposób wygenerować takie efekty specjalne, jak fajerwerki, różne rozbłyski, dym, eksplozję czy ogień.

Oprócz uproszczenia w zakresie kwestii audiowizualnych dostajemy również gotową abstrakcję dla mechanizmów bardziej niskopoziomowych, jak obsługa gier sieciowych, ujednolicony sposób oprogramowywania logiki gry czy uproszczona obsługa wielowątkowości.

ALE PEWNI NIE ZA DARMO?

Ano właśnie za darmo, a przynajmniej prawie za darmo. Zarówno Unity, jak Unreal Engine mają bardzo liberalne cenniki, które pozwalają każdemu stosunkowo łatwo wejść w świat programowania gier. Unity oferuje trzy modele opłat:

- » **Personal** – darmowy, zawierający pełną, kompletną, nieograniczoną wersję edytora, dostęp do sklepu (Asset Store), możliwość budowania na wszystkie platformy, implementowania reklam, zakupów wewnętrz aplikacji oraz grę 20 graczy naraz w grach wieloosobowych. Jedynym większym ograniczeniem jest jedynie niekonfigurowalny splash-screen z logo Unity, który pokazuje się zawsze po uruchomieniu gry lub wizualizacji. Warunkiem skorzystania z tego modelu jest osiągnięcie przez osobę lub organizację przychodów nie większych niż 100 000 dolarów rocznie (włączając w to np. kampanie kickstarterowe i tym podobne). Dla studenta, pojedynczego programisty lub zespołu programującego gry typu indie – rewelacja.
- » **Plus** – kosztujący 35 dolarów miesięcznie, zawierający wszystko, co jest w wersji Personal, ale dodatkowo rozszerzone mechanizmy analizy wydajności, stałe promocje na zakup



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

wielu składników (assets) ze sklepu oraz czasowy dostęp do zaawansowanych narzędzi (na przykład generatora terenu i drzew o nazwie Gaia). Oprócz tego ekran startowy jest konfigurowalny. Warunkiem skorzystania z tego modelu jest osiąganie przez osobę lub organizację przychodów nie większych niż 200 000 dolarów rocznie.

- » **Pro** – kosztujący 125 dolarów miesięcznie, zawierający wszystko, co jest w wersji Plus, z dodatkową możliwością zmiany lub usunięcia ekranu startowego, dostępu do częstych aktualizacji (miesięczne i kwartalne) i możliwości hostowania gier sieciowych dla większej liczby graczy. Model Pro nie ma żadnych warunków dotyczących rocznych przychodów.

Choć będę opowiadał o Unity, wspomnę jeszcze, że Unreal Engine ma również atrakcyjny model, choć działający w nieco inny sposób – twórcy tego silnika życzą sobie 5% od każdej sprzedanej sztuki gry.

INSTALACJA

Dwa słowa na temat instalacji Unity. Edytor Unity można obecnie zainstalować w trakcie instalacji Visual Studio 2017 po wybraniu stosownej opcji w instalatorze; odradzam jednak, ponieważ jest tam dostępna stara wersja Unity – 2017.2, a dosłownie na dniach pojawiła się już wersja 2018.1. Jeżeli planujemy korzystać z Visual Studio jako edytora skryptów i debugera (a jest to obecnie rozwiązanie zalecane przez zespół Unity – porzucają bowiem MonoDevelop jako wspierane środowisko programistyczne), to zdecydowanie warto zainstalować Visual Studio Tools for Unity. Dzięki temu rozszerzeniu Visual Studio będzie odpowiednio kolorować składnię skryptów, zyska nowe opcje podpowiadania wraz z pomocnymi komentarzami oraz – co chyba najważniejsze – ułatwi debugowanie projektowanych gier bezpośrednio w środowisku programistycznym (które daje – nie ukrywajmy – potężne wsparcie w tym zakresie).

Jako alternatywę dla Visual Studio można również podać Visual Studio Code, które po zainstalowaniu rozszerzenia C# oraz Debugger for Unity bardzo dobrze sprawdzi się w charakterze edytora skryptów.

INTERFEJS UŻYTKOWNIKA

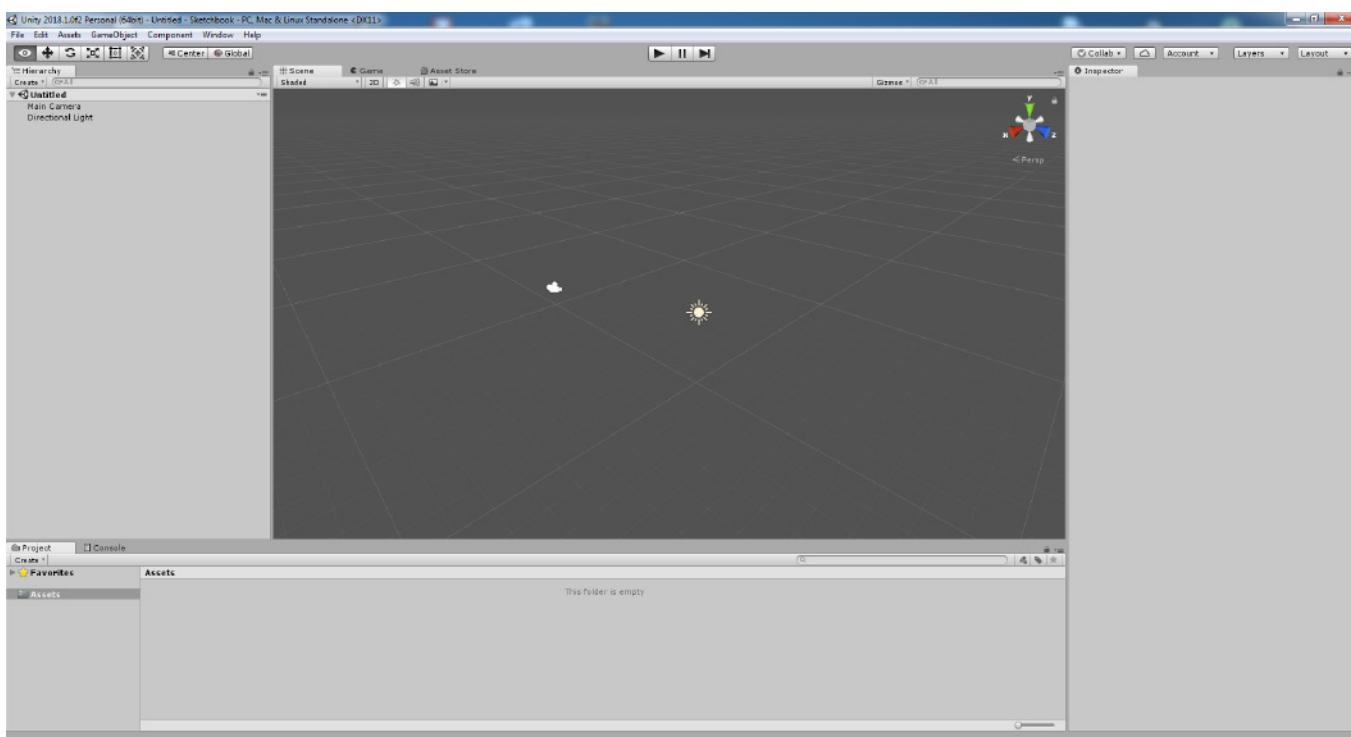
Naszą przygodę z Unity zacznijemy od poznania interfejsu użytkownika. Możemy go zobaczyć w całości na Rysunku 1; edytor składa się z kilku obszarów, którym przyjrzymy się nieco dokładniej.

Pierwszym obszarem, widocznym na Rysunku 2, a położonym zaraz pod głównym menu aplikacji jest obszar wyboru narzędzia do manipulacji obiektem na scenie.



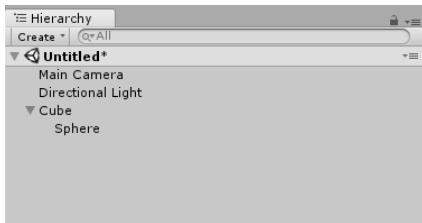
Rysunek 2. Manipulacja sceną

Możemy tu wybrać jedno z podstawowych narzędzi, od lewej: przesuwanie podglądu (panning), przenoszenie obiektów, ich obracanie oraz skalowanie, zaznaczenie prostokątne oraz narzędzie uniwersalne (przenoszenie+obracanie+skalowanie). Kolejny przycisk pozwala na zdecydowanie, czy punktem kluczowym dla przekształceń ma być geometryczny środek zaznaczenia, czy tak zwane punkty zaczepienia poszczególnych obiektów. Ma to znaczenie głównie wówczas, gdy zaznaczymy wiele obiektów; w pierwszym przypadku będą one wszystkie obracać się wokół tego samego punktu, w drugim zaś każdy z nich będzie obracał się wokół osi przechodzącej przez jego punkt zaczepienia. Wreszcie ostatni przycisk określa, czy podczas przekształcania obiektu uchwyty mają być wyrównane do jego lokalnego układu odniesienia, czy globalnego (światowego). Jeżeli na przykład obrócimy obiekt, jego lokalne osie obrócą się razem z nim – po ustaleniu lokalnego



Rysunek 1. Interfejs edytora Unity

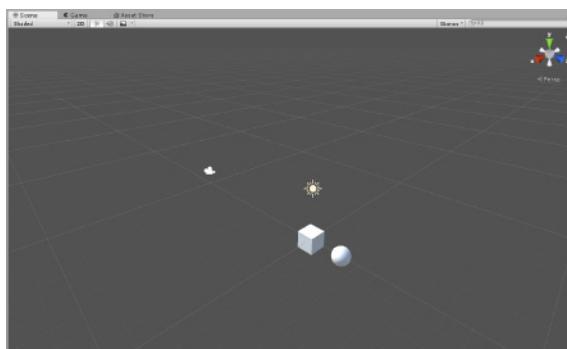
układu odniesienia możemy go wtedy przesunąć zgodnie z jego bieżącą orientacją.



Rysunek 3. Okno hierarchii

Drugim obszarem jest widok hierarchii sceny (Rysunek 3). Zawiera on drzewo wszystkich obiektów znajdujących się w bieżącej scenie, reprezentujące ich wzajemne zależności. Podstawową konsekwencją tego, że jeden obiekt jest zależny od drugiego, jest dziedziczenie podstawowych przekształceń. Jeżeli więc przesuniemy obiekt-rodzica, wraz z nim zostaną przesunięte obiekty-dzieci. Dotyczy to również pozostałych przekształceń: obrotu oraz skali. Przekształcenia zdefiniowane dla obiektów-dzieci są nakładane w kolejności. Jeżeli na przykład obiekt-rodzic ma współrzędne (0, 1, 0), a obiekt-dziecko – (1, 0, 0), efektywnie położenie obiektu-dziecka we współrzędnych globalnych będzie wynosiło (1, 1, 0).

Mechanizm ten wbrew pozorom jest niesamowicie wygodny i pozwala osiągnąć bardzo dużo efektów niewielkim kosztem. Jeżeli na przykład wstawiamy na scenę rakietę i chcemy, by ciągnęła ona za sobą smugę dymu, wystarczy dodać odpowiednio skonfigurowany emiter cząstek jako dziecko bryły reprezentującej rakietę – gdy będzie się ona poruszać, wraz z nią będzie poruszał się emiter.

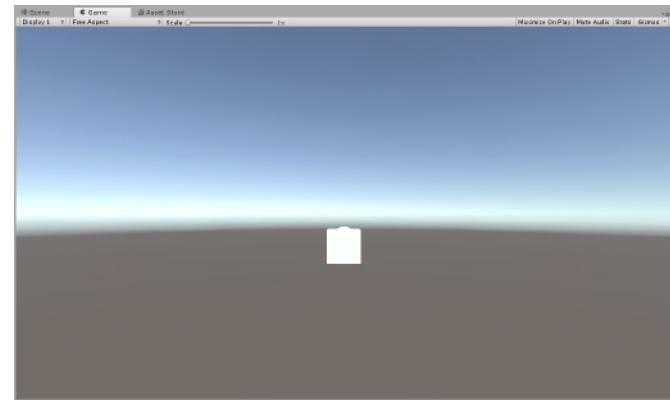


Rysunek 4. Podgląd sceny

Na Rysunku 4 przedstawiono jeden z ważniejszych obszarów – podgląd sceny. Jest to widok projektanta – patrzymy na scenę niezależnie od tego, jak będzie patrzył na nią gracz. Kontrolki w lewym, górnym rogu okna pozwalają na wybór sposobu renderowania sceny (cieniowane obiekty, ich siatki, kanał alfa, etapy renderowania itd.). Oprócz tego lista rozwijana w prawym, górnym rogu umożliwia regulowanie wyświetlania tzw. gizmów, czyli ikonek reprezentujących obiekty nie bezpośrednio widoczne (kamery, źródła światła, dźwięku itp.), a znajdujące się obok pole tekstowe – wyszukać obiekt na scenie.

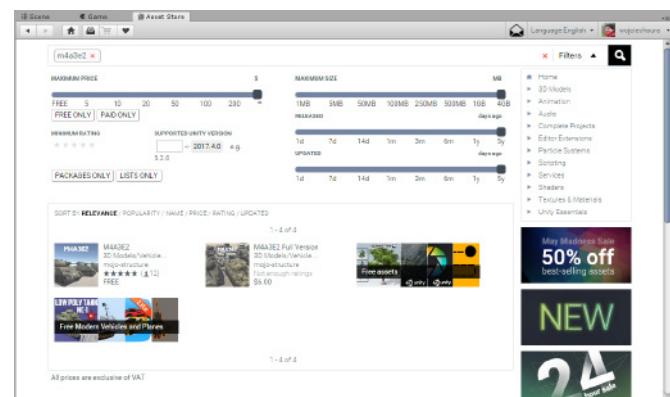
Przemieszczanie się po widoku sceny jest zrealizowane w dosyć ciekawy sposób – możemy zrobić to klasycznie przy użyciu narzędzia przesuwania (Rysunek 2) lub wcisnąć i przytrzymać prawy przycisk myszy – umożliwi to przesuwanie widoku w stylu gry FPS, czyli za pomocą klawiszy WSAD oraz dodatkowo Q i E, odpowiednio do opuszczania się w dół i wznoszenia w górę. Choć na początku nie mogłem się do tego przyzwyczaić, okazało się to bar-

dzo wygodnym rozwiązaniem. Wciśnięcie przycisku Shift podczas przemieszczania zwiększa jego prędkość.



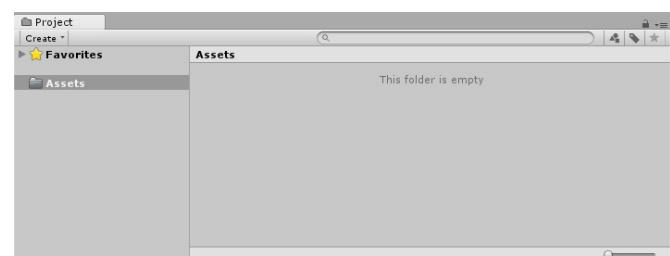
Rysunek 5. Podgląd gry

Zaraz obok widoku sceny znajdziemy również podgląd gry – można dostać się do niego po przełączeniu na drugą zakładkę. Tutaj nie możemy już poruszać się swobodnie – widok ten przedstawia, jak będzie widział edytowaną scenę gracz. Kiedy uruchomimy grę, to właśnie w tym miejscu będziemy mogli testować wprowadzane do gry zmiany.



Rysunek 6. Sklep

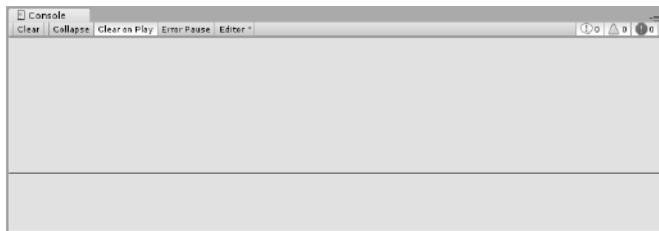
Kolejna zakładka przenosi nas do sklepu Unity o nazwie Asset Store. Jest to miejsce, gdzie możemy kupić i ściągnąć różne zasoby przydatne podczas pisania gry – od obiektów 3D przez efekty (np. eksplozje, ogień itp.), gotowe skrypty i inne. Warto pamiętać, że skorzystanie ze sklepu wymaga wcześniejszego założenia konta.



Rysunek 7. Widok projektu

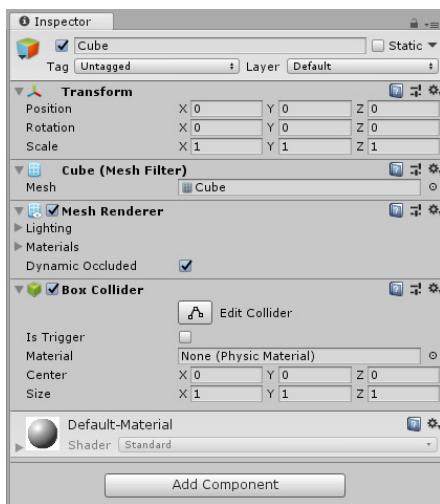
Na Rysunku 7 znajdziemy podgląd projektu – jest to po prostu hierarchia katalogów w projekcie, w których możemy umieszczać różnego zasoby (modele, tekstury, materiały, dźwięki, skrypty itd.). Unity

aktywnie interesuje się zawartością tego katalogu – jeżeli wrzucimy tam jakiś plik, po chwili powinien on pojawić się w edytorze – i od razu możemy go użyć. Warto jednak wspomnieć, że Unity dla każdego zasobu generuje dodatkowy plik z metadanymi i najbezpieczniej jest przenosić zasoby za pomocą edytora, bo wtedy zadba on o przeniesienie plików metadanych wraz z zasobami. Co ważne – poza kilkoma szczególnymi katalogami (jak Assets czy Resources) struktura jest zupełnie dowolna i możemy przechowywać zasoby w taki sposób, w jaki jest nam wygodnie.



Rysunek 8. Konsola

Przedostatnim obszarem jest konsola, w której będziemy mogli znaleźć wszystkie informacje na temat tego, co dzieje się w projekcie – od błędów komplikacji, poprzez ostrzeżenia związane z silnikiem gry, aż do komunikatów, które sami możemy tam wrzucać przy użyciu komendy Debug.Log.



Rysunek 9. Okno inspektora

Na koniec zostaje inspektor – okno wyświetlające wszystkie parametry zazначенego obiektu. Zestaw modyfikowanych parametrów zależy oczywiście od zazначенego obiektu, a o szczegółach opowiem za chwilę. Warto jednak pamiętać, że w całym edytorze bardzo sprawnie działa mechanizm drag&drop, jeśli więc jeden z parametrów obiektu wymaga na przykład wybrania tekstuury, możemy ją po prostu odnaleźć w oknie projektu, a potem przeciągnąć do odpowiedniego pola w inspektorze. Bardzo ułatwia to konfigurowanie obiektów na scenie.

Oprócz opisanych obszarów edytora bardzo istotne są oczywiście przyciski uruchamiania gry, które znajdziemy u góry okna. Wciśnięcie „Play” komplikuje projekt, uruchamia grę i przełącza automatycznie widok na zakładkę podglądu gry (potencjalnie maksymalizując ją, jeżeli sobie tego życzymy). Pozwala to na bardzo szybkie i dynamiczne przetestowanie projektowanej gry w dowolnym momencie jej rozwoju.

ARCHITEKTURY EC I ECS

Pierwszym moim wielkim zaskoczeniem była specyficzna architektura niejako wymuszona przez silnik Unity. Nie ma co ukrywać: większość programistycznego świata MVVMem stoi. Webowy front-end i backend, aplikacje desktopowe i mobilne – wszędzie znajdziemy taką czy inną postać popularnej architektury Model-View-ViewModel. Niejako więc z automatu zacząłem zastanawiać się, w jaki sposób wdrożyć go w projektowanej przez mnie grze, gdy tymczasem okazało się, że w tym świecie znacznie popularniejszą i wygodniejszą architekturą okazuje się być ECS (lub jej uproszczenie na wersja EC).

ECS to skrót od Entity-Component-System. Każdy obiekt na scenie jest obiektem, encją (ang. *entity*), która może zawierać jeden lub więcej komponentów (ang. *component*). Każdy komponent decyduje o pewnym pojedynczym aspekcie encji; powinien on opisywać możliwie najmniejszą zamkniętą jej cechę – przykładami mogą być: posiadanie punktów życia, zadawanie obrażeń podczas kolizji, możliwość zaznaczenia przez gracza, parametry ruchu, przynależność do jednej z frakcji (na przykład gracz/wrogowie) itd. System jest zaś pojedynczym fragmentem logiki gry realizującym akcje w oparciu o encje i ich konkretne komponenty.

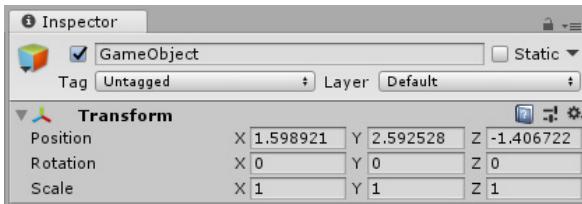
Powiedzmy na przykład, że gra toczy się w kosmosie i mamy na scenie wieżyczkę, która atakuje rakietami nadlatujące statki. Encjami w takim przypadku będą: wieżyczka, statki oraz lecące rakiety. Wieżyczka może mieć komponent, który reguluje, jak często rakiety mogą być odpalone, inny komponent przechowujący bieżące zapasy amunicji, kolejny informujący o punktach życia wieżyczki, następny definiujący parametry poszukiwania celu (na przykład kąty pola widzenia oraz maksymalny dystans do potencjalnego celu). System może natomiast zawierać logikę, która wyszukuje wszystkie encje, które zawierają ten zestaw komponentów, a następnie na bazie zawartych w nich parametrów wyszukuje potencjalne cele, instancjonuje rakiety i odpowiednio zmniejsza zapasy amunicji.

Architektura ta okazuje się być niesamowicie elastyczna. Jeżeli na przykład potrzebujemy, żeby jeden z wrogich statków kosmicznych zachowywał się podobnie jak wieżyczka – możemy go po prostu wyposażyć w taki sam zestaw komponentów i od tego momentu będzie on zachowywał się dokładnie tak samo. Dzięki temu możemy każdy oprogramowany mechanizm użyć ponownie dla dowolnej liczby obiektów na scenie. Im zaś bardziej będziemy trzymać się zasad SRR (Single Responsibility Rule, zasada pojedynczej odpowiedzialności), tym łatwiej będzie dopasowywać zachowanie konkretnych obiektów – na przykład strzelać pociskami zamiast rakiet lub zastosować inny mechanizm namierzania celów.

Unity w wersjach do 2017.4.2 obsługiwało nieco uproszoną wersję architektury ECS, którą możemy chyba nazwać EC, czyli Entity-Component. Różnica polega na tym, że implementacja logiki znajdowała się wewnątrz komponentów – upraszcza to nieco proces jej oprogramowania, ale wymaga dosyć dużej dyscypliny, by nie wprowadzić do projektu zbyt dużego chaosu. Sam złapałem się na tym, że w trakcie refaktoryzowania gry rozdzielałem takie komponenty-kombo na dwa mniejsze: jeden zawierający tylko parametry związane z jakimś zachowaniem oraz drugi, który implementował samą logikę. Unity 2018.1 wprowadza już kompletny mechanizm ECS; w tym artykule skupię się na razie na poprzedniej wersji architektury (która wciąż jest obecna w Unity i możemy z niej korzystać), a w późniejszych częściach postaram się opowiedzieć, jak programować już w czystym ECS.

GAMEOBJECT I KOMPONENTY

W Unity encją (tą z architektury ECS czy EC) jest klasa o nazwie `GameObject`. Stanowi ona swoisty kontener na komponenty i dopóki jakichś komponentów do niej nie dodamy, jest „niczym” – obiektem pustym. Jednak jest kilka cech `GameObject`, na które warto zwrócić uwagę.



Rysunek 10. Właściwości obiektu pustego

Po pierwsze, obiektovi możemy nadać tak zwany tag. Jest on po prostu ciągiem znaków, który może pomóc zidentyfikować dany obiekt; tagi są nadawane przez programistę i głównie przez niego używane – silnik dostarcza zbiór metod pomagających na przykład odnaleźć obiekty po tagu, ale sam z tych danych nie korzysta. Unity jest jednak skonstruowane w ten sposób, że można stosunkowo szybko i wydajnie wyszukać wszystkie obiekty, które mają konkretny tag. Pewnym ograniczeniem jest jedynie fakt, że dany obiekt może mieć tylko jeden tag.

`GameObject` można również przyporządkować do konkretnej warstwy. Jest ich 30, z czego pierwszych kilka jest zarezerwowanych przez silnik Unity, natomiast pozostałe można wykorzystywać według własnych potrzeb. Warstwy używane są przeważnie do optymalizacji obliczeń; na przykład można ustawić kamerę w taki sposób, by renderowała tylko konkretne warstwy, wykonać tzw. raycasting (czyli poszukiwanie obiektów znajdujących się na zadanej prostej) tylko na jednej z nich lub wreszcie wyłączyć testowanie kolizji dla obiektów znajdujących się na zadanych warstwach. Jest to w miarę prosty i skuteczny sposób na to, by poprawić wydajność gry stosunkowo niewielkim kosztem.

Trzecią cechą `GameObject` jest to, że możemy oznaczyć go jako obiekt statyczny. Uwaga na nomenklaturę – nie ma to nic wspólnego z klasami czy metodami statycznymi; termin ten w przypadku `GameObject` dotyczy bowiem tych obiektów, co do których mamy pewność, że nie będą się w czasie gry poruszać (na przykład teren, platformy, elementy dekoracyjne itp.). Pozwala to Unity wykonać część obliczeń jeszcze w edytorze, przyspieszając w ten sposób grę. Unity może na przykład scałić kilka sąsiednich obiektów statycznych, by wyrenderować je hurtem w tak zwanym pakiecie (ang. *batch*).

Wśród optymalizacji, które może wykonać Unity dla obiektów statycznych, są: możliwość obliczenia zaawansowanego oświetlenia dla sceny, przyspieszenie obliczania okluzji (czyli pominięcia renderowania obiektów całkowicie lub częściowo zasłoniętych), renderowanie grup obiektów (batching) polegające na scaleniu kilku obiektów w jeden większy, przyspieszenie obliczania nawigacji pomiędzy dwoma punktami (silnik wykorzystuje informację o przeszkodach, a także tzw. off-mesh-links, które definiują nawigację pomiędzy dwoma obszarami, pomiędzy którymi nie ma bezpośredniego połączenia) czy tzw. reflection probe – służącego do renderowania odbić lustrzanych.

Wreszcie `GameObject` może być aktywny lub nie. Zdeaktywowanie obiektu powoduje tymczasowe wyłączenie go ze sceny i wyłącza działanie podpiętych do niego komponentów i skryptów. Warto pamiętać o tym, że parametr `isActive` rodzica przewyszczza parametry `isActive` dzieci. Innymi słowy obiekt jest aktywny, gdy sam ma zapaloną flagę `isActive` oraz wszyscy jego przodkowie w hierarchii mają ją również zapaloną.

KOMPONENTY

Unity jest bardzo jednolity, jeśli chodzi o encje i komponenty. Na przykład – technicznie rzecz biorąc – nie istnieje bezpośrednio coś takiego jak kamera czy światło. Ich rolę pełnią takie obiekty `GameObject`, do których podwieszone są komponenty `Camera`, `Flare` layer oraz `Audio Listener` w przypadku kamery oraz `Light` w przypadku światła. Jeżeli dodamy odpowiednie komponenty do innego, dowolnego `GameObject`, uczynimy z niego w prosty sposób kamerę lub światło. Takie ujednolicenie bardzo upraszcza programowanie, bo zmniejsza liczbę typów obiektów na scenie. Jest to również bardzo wygodne, bo – na przykład – nie trzeba tworzyć dodatkowego obiektu, by sprawić, że eksplozja oświetli okoliczne obiekty – wystarczy tylko dodać symulującemu eksplozję emiterowi częstek komponent `Light`, a zacznie on również pełnić rolę światła.

Spójrzmy na kilka podstawowych komponentów dostarczanych przez Unity.

Transform

Podstawowym komponentem, który mają (i obowiązkowo muszą mieć) prawie wszystkie obiekty na scenie, jest `Transform`, który zawiera parametry trzech podstawowych transformacji – położenia, obrotu i skali. Widać to na Rysunku 10 – nawet obiekt pusty posiada również komponent `Transform`. Cechą ta odzwierciedlona jest również w bibliotece standardowej – o ile dostanie się do konkretnego komponentu wymaga wykonania odpowiedniego zapytania, to `Transform` jest dostępny bezpośrednio, w postaci właściwości klasy `GameObject`.

`Transform` pełni jeszcze jedną ważną rolę – to właśnie on jest odpowiedzialny za przechowywanie hierarchii obiektów na scenie. Może się to wydawać nieco dziwne, ale tak właśnie jest: jeżeli chcemy dostać się do obiektu `GameObject`, który jest rodzicem innego obiektu, musimy dostać się najpierw do komponentu `Transform` tego ostatniego, z niego przejść do komponentu `Transform` jego rodzica, skąd – również właściwością – możemy dostać się już do `GameObject`:

Listing 1. Wyszukujemy rodzica w hierarchii sceny

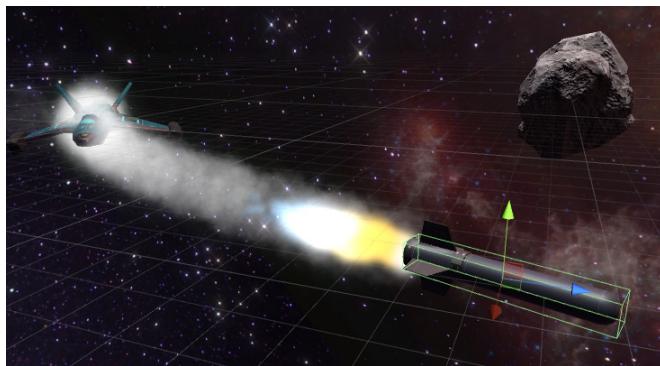
```
var myParent = this.transform.parent.gameObject;
```

MeshFilter oraz MeshRenderer

Oba komponenty: `MeshFilter` oraz `MeshRenderer` są potrzebne, jeżeli dany `GameObject` ma być widoczny na ekranie w postaci bryły. `MeshFilter` jest tylko kontenerem zawierającym informację o tym, która bryła ma zostać użyta do wyrenderowania obiektu, a samym renderowaniem zajmuje się już `MeshRenderer`. Ten ostatni pozwala również określić parametry renderowania. Możemy na przykład zdecydować, czy na renderowany obiekt mogą być rzucane cienie oraz czy sam obiekt rzuca cień.

Collidery

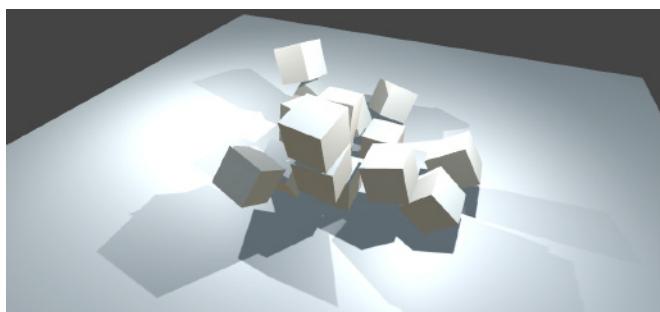
Komponenty z serii Collider, czyli BoxCollider, SphereCollider, MeshCollider i jeszcze kilka innych, określają kształt bryły, która będzie używana do detekcji kolizji. Wbrew pozorom przeważnie bryła obiektu i jej obiekt kolizji (ang. *collider*) różnią się od siebie, a wynika to z faktu, że detekcja kolizji jest procesem bardzo złożonym obliczeniowo. Dlatego też często zamiast konkretnej bryły lepiej użyć na przykład dobrze dopasowanego prostopadłościanu czy kuli i o ile z perspektywy gry nie będzie widać większej różnicy, to znacznie obniżymy koszty obliczeniowe konieczne do wykonania detekcji kolizji. MeshCollider jest natomiast szczególnym obiektem kolizji, który umożliwia wskazanie konkretnej bryły jako obszaru kolizji; trzeba się jednak liczyć z tym, że testowany będzie każdy jej trójkąt, co w przypadku złożonych brył może spowodować znaczny spadek wydajności.



Rysunek 11. Rakietę i jej collider

Rigidbody

Rigidbody nadaje obiektowi charakter ciała sztywnego – zaczyna na niego wpływać grawitacja, będzie on również reagował zderzeniami z innymi obiektami, o ile tylko każdy z nich będzie miał również zdefiniowany collider (tym razem niekoniecznie zawierający również Rigidbody). Dla obiektu będącego ciałem sztywnym możemy zdefiniować między innymi masę, opór powietrza podczas przesuwania i obracania, a także wprowadzić dodatkowe interpolacje, jeżeli obiekt nie zachowuje się zgodnie z naszymi oczekiwaniemi. Oprócz tego możemy na przykład zablokować ruch lub rotację względem jednej lub więcej osi.



Rysunek 12. Symulacja fizyki w akcji

Komponentów jest o wiele więcej, ale mam nadzieję, że tych kilka, które przedstawiłem, da jakieś wyobrażenie na temat sposobu, w jaki wpływają one na obiekt, do którego zostały podwieszone.

SKRYPTY

Siłą silnika gier jest nie tylko liczba gotowych rozwiązań, których dostarcza, ale chyba przede wszystkim elastyczność w zakresie definiowania logiki gry. Unity korzysta z jednego z dwóch języków jako języków „skryptów” – C# oraz UnityScript, będącego zmodyfikowaną wersją JavaScriptu (choć chodzą słuchi, że wsparcie dla UnityScript zostanie wycofane i jedynym oficjalnym językiem pozostanie C#). Tak naprawdę nie są one jednak językami skryptowymi, bo napisane skrypty są potem normalnie komplilowane do assembly .NET. Dobra wiadomość jest taka, że możemy dzięki temu korzystać z całego dobrodziejstwa inwentarza dostarczanego wraz z bibliotekami standardowymi .NET, ale zła jest taka, że obecną oficjalnie wspieraną wersję języka C# jest wersja 4. Zespół Unity zapewnia jednak, że trwają intensywne prace nad tym, żeby wprowadzić kompatybilność z .NET 4.6, co między innymi da dostęp do wszystkich nowości obecnych w C# 6.0.

Skrypty integrują się z mechanizmami obecnymi w Unity w bardzo prosty sposób – po prostu każdy z nich jest komponentem, który możemy podwiesić do dowolnego obiektu. Gdy dodamy do projektu nowy skrypt i otworzymy go, ujrzymy następujący szkielet:

Listing 2. Pusty skrypt

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SlowRotation : MonoBehaviour {

    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
    }
}
```

Na początku zwrócić uwagę na brak namespace – najwyraźniej zabrakło go w szablonie pustego skryptu. Warto wyrobić sobie nawyk i od razu go dodać, żeby nie zaśmiecać globalnej przestrzeni nazw; silnikowi i edytoriowi Unity nie robi większej różnicy, czy dany skrypt jest osadzony w jakiejś przestrzeni nazw, czy nie.

Każdy skrypt musi dziedziczyć po klasie MonoBehaviour, która dostarcza pewnego zestawu podstawowych funkcjonalności, z których możemy korzystać. Co więcej, ważnym wymaganiem jest również dopasowanie nazwy klasy oraz pliku – w przeciwnym wypadku Unity nie pozwoli dołączyć go do obiektu.

Szkielet skryptu zawiera domyślnie dwie metody: Start oraz Update. Są one dodane dla wygody programisty – jeżeli nie ma potrzeby implementowania ich, można je swobodnie usunąć. Co ciekawe, metody te mają widoczność internal i co jeszcze ciekawego – nie są prześlonione (ang. override). Unity stosuje bowiem mechanizm „metod-komunikatorów” oparty o wywoływanie w skryptach metod o określonych nazwach – jest to najprawdopodobniej sposób na omijanie dziedziczenia i metod wirtualnych, których wywoływanie wprowadza niewielki narzuł czasowy. W aplikacjach użytkowych czas ten jest zwykle pomijalny, ale w grze, w której niektóre metody są wywoływane dziesiątki lub setki razy na sekundę, może on urosnąć do zauważalnych wartości. Dodam jeszcze jako ciekawostkę, że programista może sam skorzystać z mechanizmu komunikatorów i zlecić wywołanie we wszystkich skryptach metody o określonej nazwie.

W Unity pierwotnie funkcjonował mechanizm komunikatów oparty o metodę `SendMessage`. Działał on w ten sposób, że w danym `GameObject` wyszukiwał wszystkie komponenty, które zawierały metodę o danej nazwie, a następnie wywoływał na nich te metody. Z nowymi wersjami Unity wprowadzony jednak został nowy system oparty o interfejsy. Aby z niego skorzystać, należy przygotować interfejs dziedziczący po `IEventSystemHandler`, zawierający określone metody, które chcemy wywoływać. Każdy komponent, który chce odbierać komunikaty, musi zaimplementować ten interfejs, zaś wysyłanie komunikatów odbywa się poprzez statyczną metodę `ExecuteEvents.Execute`. Więcej informacji na stronie podręcznika Unity: <https://docs.unity3d.com/Manual/MessagingSystem.html>.

Podobnie jak obiekty, skrypty mogą być w czasie gry aktywowane i dezaktywowane. Nieaktywny skrypt nie będzie otrzymywać komunikatów, to jest nie będą wywoływanego jego metody-komunikaty, ale wciąż można się do niego dostać i wywoływać jego metody, operować na jego właściwościach itd.

Zobaczmy teraz, jakie podstawowe komunikaty (czyli metody o określonych nazwach) obsługuje `MonoBehaviour`.

Start

Metoda-komunikat `Start` jest wywoływana w pierwszej klatce, w której skrypt staje się aktywny. Metoda ta przeważnie jest wykorzystywana do zainicjowania obiektu, ponieważ programista nie ma bezpośredniego dostępu do konstruktora klasy. Oczywiście nic nie stoi na przeszkodzie, by go samodzielnie dopisać, jednak twórcy Unity zalecają, by tego nie robić. Konstruktor jest bowiem wywoływany w wielu nieoczekiwanych momentach (na przykład zaraz po skompilowaniu skryptu), co w skrajnym przypadku może doprowadzić do błędu i zatrzymania edytora Unity. Ograniczenie to dotyczy jednak tylko skryptów (czyli klas dziedziczących po `MonoBehaviour`), w przypadku wszystkich pozostałych klas można korzystać z nich bez ograniczeń.

Update

Teraz zaczyna się zabawa. Metoda `Update` jest wywoływana dokładnie raz na klatkę i tak naprawdę jest jednym z dwóch podstawowych „koni roboczych” skryptu – zwykle to w nich implementujemy logikę gry.

Jest jednak pewien haczyk. Unity w żaden sposób nie gwarantuje, że `Update` będzie wywoływane w stałych interwałach. Ogólnie silnik stara się wywoływać tę metodę tak często, jak jest to możliwe, ale może zdarzyć się sytuacja, że poziom skomplikowania sceny (liczba obiektów, ich logika, oświetlenie itd.) jest na tyle duży, że liczba klatek na sekundę wyraźnie spadnie. W takiej sytuacji możemy oczywiście spodziewać się również spadku częstotliwości wywoływania metody `Update`.

Metoda ta jest wykorzystywana często do przekształcania obiektów – przesuwania, obracania i skalowania. Łatwo jednak po pełnić w takim przypadku bardzo podstawowy błąd polegający na użyciu stałych wartości, jak w Listingu 3.

Listing 3. Przesuwanie obiektu co klatkę – błędne!

```
void Update () {
    this.transform.position += new Vector3(1.0f, 0.0f, 0.0f);
}
```

Dlaczego takie rozwiązanie jest nieprawidłowe? To proste – uzależnia prędkość przesuwania się obiektu od liczby klatek na sekundę. Jeżeli scena będzie nieskomplikowana i w czasie sekundy uda się wyrenderować 60 klatek, to obiekt przesunie się o 60 jednostek. Gdy zaś scena będzie bardziej skomplikowana i liczba klatek na sekundę spadnie do 30, obiekt zauważalnie zwolni – będzie przesuwał się 30 jednostek na sekundę. Innymi słowy prędkość przesuwania się obiektu zapisana w postaci stałej (`new Vector3(1.0f, 0.0f, 0.0f)`) nie jest wyrażona w jednostkach na sekundę, tylko jednostkach na klatkę.

Konieczne jest więc wprowadzenie korekty, która uwzględnia częstotliwość renderowania klatek. W tym celu możemy skorzystać ze statycznej klasy `Time`, która dostarcza bardzo pożyteczną własność o nazwie `deltaTime`. Jej wartość wyraża czas (w sekundach), który upłynął od wyrenderowania poprzedniej klatki. Możemy więc poprawić kod z Listingu 3:

Listing 4. Poprawiony kod przesuwania obiektu

```
void Update () {
    this.transform.position += new Vector3(1.0f * Time.deltaTime,
    0.0f, 0.0f);
}
```

Obiekt będzie się teraz przesuwał proporcjonalnie do częstotliwości renderowania klatek: im częściej będą renderowane, tym wolniej; im rzadziej będą renderowane, tym szybciej. W ten sposób efektywna prędkość obiektu będzie stała i wyrażona w jednostkach na sekundę.

FixedUpdate

`FixedUpdate` jest drugim ze wspomnianych „koni roboczych” skryptu. W odróżnieniu od `Update` jest on wywoływany w (możliwie) stałych interwałach czasu. Oczywiście również i tu nie ma gwarancji, że uda się to zrobić, jednak tym razem silnik dokłada starań, by tak właśnie było – nawet jeśli stanie się to kosztem opóźnienia wywołania `Update`. Przeznaczeniem `FixedUpdate` jest głównie realizowanie logiki operującej na fizyce, ponieważ wszystkie wywołania tej metody mają miejsce bezpośrednio przed obliczeniami fizycznymi. Przykładem może być przyłożenie przez określony czas siły do obiektu, aby zaczął się poruszać.

Jest jeszcze jedna ciekawostka, o której warto wspomnieć – otóż w Unity stosunkowo łatwo można manipulować czasem. Klasa `Time` ma statyczną własność `timeScale`, która określa, w jakim tempie ma upływać czas w grze w stosunku do normalnego czasu. Wartość 0.5 spowoduje, że czas będzie płynął o połowę wolniej niż w rzeczywistości, zaś wartość 2.0 sprawi, że czas będzie płynął dwukrotnie szybciej. Należy mieć na uwadze, że efektywnie wpływa to tylko na wartości zwracane przez `Time.deltaTime` oraz `Time.fixedDeltaTime`; ponieważ jednak wszystkie mechanizmy symulujące fizykę wbudowane w Unity korzystają z tych wartości, można w ten sposób łatwo opóźnić lub zatrzymać czas (na przykład w celu spauzowania gry).

LateUpdate

Metoda ta jest wywoływana podobnie jak `Update`, czyli co klatkę. Różnica polega na tym, że `LateUpdate` wywoływane jest dopiero wówczas, gdy wszystkie `Update` wszystkich komponentów zostały już wywołane. Jest to wygodne rozwiązanie, za pomocą którego

możemy wymusić konkretną kolejność wywoływania skryptów; na przykład jeżeli kamera śledzi jakiś obiekt, samo śledzenie może zostać zrealizowane właśnie w LateUpdate, ponieważ podczas Update obiekt mógł zostać już przesunięty. W obsłudze tej metody należy korzystać z tej samej wartości czasu co w Update, czyli Time.deltaTime.

Awake

Wywołanie Awake następuje w momencie, gdy obiekt został zainicjowany, a dzieje się to tylko raz w czasie życia danego obiektu (w przeciwieństwie do Start, które wywoływane jest zawsze w momencie, gdy staje się on aktywny). Co więcej, programista ma gwarancję, że pierwsze Awake zostanie wywoływane w momencie, gdy *wszystkie* obiekty zostały już zainicjowane, co oznacza, że można je już wtedy wyszukiwać i wchodzić z nimi w interakcję. Jest to również dobre miejsce na zainicjowanie skryptu, ponieważ – z uwagi na tylko jednokrotne wywołanie – Awake zachowuje się trochę jak konstruktor.

Opisane metody należą do zestawu podstawowych komunikatów, które może odebrać skrypt; pełną listę można znaleźć w (składnią świątynnym) manualu Unity pod adresem: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>.

JAKIŚ PRZYKŁAD?

Możemy zacząć od prostego skryptu, który spowoduje, że obiekt, do którego zostanie on dołączony, będzie obracał się w losowym kierunku.

Zanim zaczniemy cokolwiek pisać, kilka słów wprowadzenia w matematykę rządzącą obiektami w Unity. Wszystkie obliczenia dotyczące geometrii realizowane są na liczbach zmiennoprzecinkowych pojedynczej precyzji, czyli float. Ponieważ większość metod .NETowej klasy Math przyjmuje parametry i zwraca wartości typu double, Unity zawiera swoją własną jej implementację o prostej nazwie Mathf. Można tam znaleźć większość – jeżeli nie wszystkie – metody z klasy Math, ale i sporo innych – upraszczających obliczenia, które często występują w grach. Przykładem takich metod mogą być: Clamp (przycięcie liczby do zadanego zakresu), Lerp (liniowa interpolacja pomiędzy dwiema wartościami), SmoothDamp (dążenie z wartością a do b z zadaną maksymalną prędkością i łagodnie zwalniające) itd. Co ciekawe, choć funkcje trygonometryczne operują na radianach, to sporo metod Mathf operuje na stopniach, na przykład LerpAngle, DeltaAngle, MoveTowardsAngle itd. Mamy też dwie stałe: Deg2Rad i Rad2Deg, ułatwiające szybkie przeliczanie stopni pomiędzy różnymi jednostkami.

Operacje przestrzenne realizowane są zwykle na strukturze Vector3, która – jak można się łatwo domyśleć – przechowuje współrzędne (x, y, z). Również i w tym przypadku dostajemy cały szereg metod, które ułatwiają obliczenia; instancję Vector3 możemy odpytać o długość (magnitude), kwadrat długości (sqrMagnitude), wektor znaleziony – czyli w tym samym kierunku, ale o długości 1 (normalized), możemy go też znaleziony (metoda Normalize) lub przekształcić o zadaną wartość (metoda Scale). Klasa Vector3 dostarcza z kolei kolejnego zbioru pomocniczych metod statycznych, takich jak Angle (kąt pomiędzy dwoma wektorami), Cross (iloczyn wektorowy), Dot (iloczyn skalarny), Project (rzutowanie wektora na inny wektor), ProjectOnPlane (rzutowanie wektora na płaszczyznę) itd.

Trochę trudniej jest z reprezentacją orientacji. Większość programistów przyzwyczajona jest do tego, że orientacja przedstawiana jest w postaci trzech wartości obrotów – wokół osi X, Y i Z. Twórcy Unity zdecydowali jednak o reprezentowaniu jej nieco inaczej, za pomocą tak zwanego kwaternionu.

Główną przyczyną takiego stanu rzeczy jest fakt, że zapis euklowski (czyli X, Y, Z) jest podatny na efekt o nazwie „gimbal lock”. W skrócie: wykonanie obrotu o konkretne kąty wokół dwóch osi może spowodować, że trzecia z nich pokryje się z jedną z dwóch pierwszych – zmniejszając w ten sposób stopień swobody obrotu. Bardzo dobrze opisuje ten efekt artykuł w angielskiej Wikipedii pod adresem https://en.wikipedia.org/wiki/Gimbal_lock. Innym argumentem jest też to, że zastosowanie kwaternionów pozwala obniżyć koszt pamięciowy i obliczeniowy podczas wykonywania koniecznych matematycznych przekształceń.

Kwaternion jest konstrukcją matematyczną, która nie jest podatna na ten efekt, ale za cenę tego, że wartości w nim przechowywane nie są wprost czytelne dla człowieka. Odpowiednie przekształcenia pomiędzy kątami euklowskimi i kwaternionami możemy znaleźć na Wikipedii – https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles – i szerzej mówiąc, niespecjalnie zacząć, żeby z nich korzystać.

Trudno byłoby wymagać, by programiści tak drastycznie przestawili sposób myślenia dotyczący obrotów w przestrzeni, więc biblioteka Unity dostarcza zestaw metod, które znacznie ten proces upraszczają. Mamy więc metodę statyczną Quaternion.Euler, która przekształca kąty euklowskie do kwaternionu, czy Quaternion.LookRotation – obliczającą kwaternion dla obrotu we wskazanym kierunku (i opcjonalnie z podanym kątem definiującym „góra”).

Istnieje również metoda przeliczająca kwaternion na kąty euklowskie, ale w większości przypadków nie wróci ona wyników, których się spodziewamy. Wynika to z faktu, że obroty nie są jednoznaczne, czyli wiele różnych zestawów kątów euklowskich może dać w wyniku ten sam obrót. Oznacza to, że po przeliczeniu jednego zestawu kątów na kwaternion, a potem po przeliczeniu go z powrotem na kąty euklowskie możemy dostać zupełnie inne wartości, które choć technicznie prawidłowe, nie będą przedstawiały dla nas żadnej wartości.

Dlatego też prawidłową metodą pracy z kwaternionami jest przechowywanie obrotów w kątach euklowskich i wykonywanie konwersji tylko w jedną stronę. Teraz możemy napisać już pierwszy skrypt.

Listing 5. Losowy obrót obiektu

```
public class SlowRotation : MonoBehaviour
{
    private Vector3 rotationSpeeds;
    private Vector3 currentRotations;

    void Start ()
    {
        currentRotations = Vector3.zero;
        rotationSpeeds = new Vector3(
            Random.Range(0, 30),
            Random.Range(0, 30),
            Random.Range(0, 30));
    }

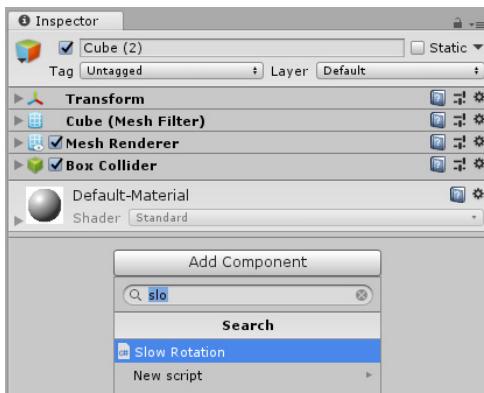
    void Update ()
    {
        currentRotations += rotationSpeeds * Time.deltaTime;
        this.transform.rotation = Quaternion.Euler(currentRotations);
    }
}
```

Na początku zauważmy, że będziemy potrzebować trzech zmiennych (po jednej dla każdej osi), w których znajdą się prędkości obrotów (w stopniach na sekundę), oraz kolejnych trzech zmiennych, w których będziemy przechowywać bieżące kąty obrotów. Dla uproszczenia zamiast trzech osobnych pól możemy przechować je w jednej strukturze Vector3, upraszczając w ten sposób zapis. Jest to dosyć powszechnie rozwiązanie, a świadczy o tym choćby fakt, że metoda Quaternion.Euler w jednej ze swoich wersji przyjmuje właśnie pojedynczy Vector3.

Na początku zerujemy bieżące kąty obrotów oraz losujemy prędkości obrotów; klasy Random nie dostarcza w tym przypadku .NET, lecz biblioteka Unity; jest ona znacznie wygodniejsza w użyciu niż ta pierwsza, ponieważ udostępnia dużo dodatkowych, pomocniczych metod, jak choćby Range, która zwraca liczbę losową z zadanego przedziału.

Operacje realizowane co klatkę są stosunkowo proste – zwiększamy bieżące kąty o prędkość obrotu (oczywiście przeskalowaną przez wartość Time.deltaTime), a następnie ustawiamy bieżącą rotację obiektu na obliczone nowe bieżące obroty.

Teraz możemy wstawić na scenę kilka brył, na przykład sześciów, a następnie dodajemy do nich nasz skrypt jako komponent (Rysunek 13).



Rysunek 13. Dodawanie skryptu

Teraz uruchamiamy grę, by otrzymać spodziewany efekt – widoczny na Rysunku 14.



Rysunek 14. Efekt działania skryptu

PARAMTRY SKRYPTÓW

Skrypt działa bez zarzutu, ale nie mamy żadnej kontroli nad jego parametrami wejściowymi. Przypuśćmy, że chcielibyśmy móc regulować zakres prędkości obrotu, obecnie zapisany na sztywno w kodzie. Komponenty wbudowane w Unity dają możliwość zmiany parametrów w oknie inspektora – czy da się tak zrobić również ze skryptami?

Na szczęście twórcy Unity przewidzieli taką możliwość i można wprowadzić własne parametry stosunkowo łatwo – poprzez wprowadzenie globalnych pól. Nasz skrypt będzie wymagał niewielkiej modyfikacji.

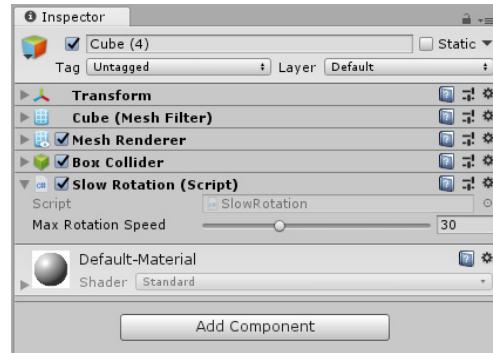
Listing 6. Wprowadzamy parametr do skryptu

```
public class SlowRotation : MonoBehaviour
{
    private Vector3 rotationSpeeds;
    private Vector3 currentRotations;

    void Start ()
    {
        currentRotations = Vector3.zero;
        rotationSpeeds = new Vector3(
            Random.Range(0, maxRotationSpeed),
            Random.Range(0, maxRotationSpeed),
            Random.Range(0, maxRotationSpeed));
    }

    void Update ()
    {
        currentRotations += rotationSpeeds * Time.deltaTime;
        this.transform.rotation = Quaternion.Euler(currentRotations);
    }

    [Range(0, 90)]
    public int maxRotationSpeed = 30;
}
```



Rysunek 15. Parametr skryptu

Wystarczy, że zapiszemy skrypt i przełączymy się na edytor Unity – w okienku inspektora przy dodanym skrypcie pojawi się suwak umożliwiający zmianę parametru (Rysunek 15). Edytor Unity jest na tyle sprytny, że identyfikator zapisany w postaci camel-case (`maxRotationSpeed`) przerabia na bardziej czytelną nazwę: „`Max Rotation Speed`”.

Zastosowane rozwiązanie ma niestety sporą wadę – wprowadziliśmy do klasy publiczne pole, co jest bodaj pierwszym grzechem głównym przeciw enkapsulacji. Szczęśliwie jednak również prywatne pola mogą być widoczne w inspektorze – musimy tylko oznać je odpowiednim atrybutem. Jeżeli chcemy, by do zapisanej wartości miały dostęp inne komponenty, możemy opublikować wartość pola poprzez zwykłą właściwość (Listing 7)

Listing 7. Naprawiamy enkapsulację w skrypcie

```
public class SlowRotation : MonoBehaviour
{
    [SerializeField, Range(0, 90)]
    private int maxRotationSpeed = 30;

    private Vector3 rotationSpeeds;
    private Vector3 currentRotations;

    // Use this for initialization
    void Start ()
    {
        currentRotations = Vector3.zero;
        rotationSpeeds = new Vector3(
            Random.Range(0, maxRotationSpeed),
            Random.Range(0, maxRotationSpeed),
            Random.Range(0, maxRotationSpeed));
    }
}
```

```
// Update is called once per frame
void Update ()
{
    currentRotations += rotationSpeeds * Time.deltaTime;
    this.transform.rotation = Quaternion.
        Euler(currentRotations);
}

public int MaxRotationSpeed
{
    get
    {
        return maxRotationSpeed;
    }
    set
    {
        maxRotationSpeed = value;
    }
}
```

Idźmy dalej – a jeżeli parametr skryptu jest typem złożonym? Powiedzmy na przykład, że chcielibyśmy móc określić maksymalną prędkość obrotu w każdej z trzech osi, a wartości te przechować w pojedynczej klasie.

Jest to również możliwe, jeżeli klasę, która będzie przechowywać dane wartości, oznaczymy atrybutem [Serializable]. Oprócz tego musimy pamiętać, by jej składowe również spełniały warunki publikacji w inspektorze (publiczne pola lub prywatne oznaczone atrybutem [SerializeField])

Listing 8. Typ opisujący zakresy prędkości obrotów

```
namespace Assets
{
    [Serializable]
    public class RotationSpeedCap
    {
        [SerializeField, Range(0, 90)]
        private int x;
        [SerializeField, Range(0, 90)]
        private int y;
        [SerializeField, Range(0, 90)]
        private int z;

        public RotationSpeedCap()
        {
            X = 30;
            Y = 30;
            Z = 30;
        }

        public int X
        {
            get { return x; }
            set { x = value; }
        }

        public int Y
        {
            get { return y; }
            set { y = value; }
        }

        public int Z
        {
            get { return z; }
            set { z = value; }
        }
    }
}
```

Teraz możemy użyć jej w skrypcie.

Listing 9. Używamy złożonego typu w skrypcie

```
public class SlowRotation : MonoBehaviour
{
    [SerializeField]
    private RotationSpeedCap speedCap = new RotationSpeedCap();

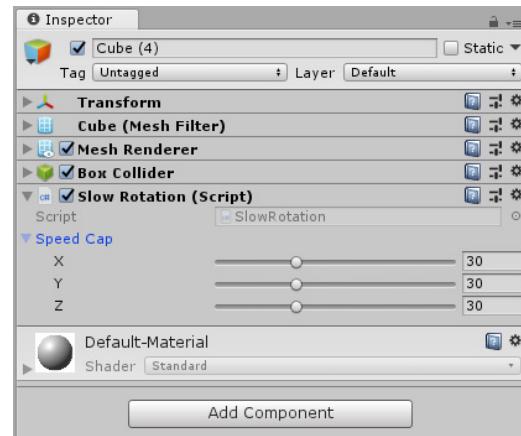
    private Vector3 rotationSpeeds;
```

```
private Vector3 currentRotations;

// Use this for initialization
void Start ()
{
    currentRotations = Vector3.zero;
    rotationSpeeds = new Vector3(
        Random.Range(0, speedCap.X),
        Random.Range(0, speedCap.Y),
        Random.Range(0, speedCap.Z));
}

// Update is called once per frame
void Update ()
{
    currentRotations += rotationSpeeds * Time.deltaTime;
    this.transform.rotation = Quaternion.
        Euler(currentRotations);
}

public RotationSpeedCap SpeedCap
{
    get
    {
        return speedCap;
    }
    set
    {
        speedCap = value;
    }
}
```



Rysunek 16. Parametr jako typ złożony

Od tej pory okienko naszego skryptu będzie wyglądało nieco inaczej (Rysunek 16).

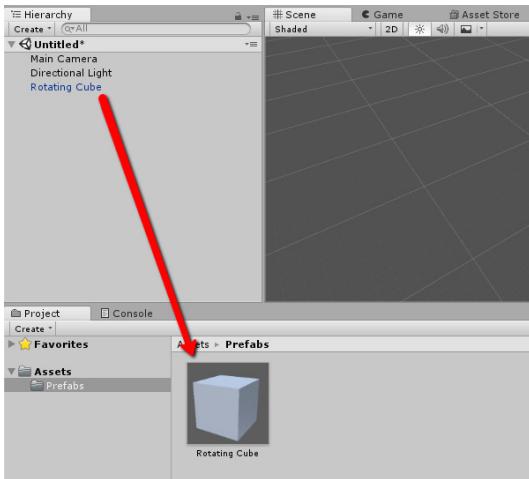
PREFABY

W grach bardzo często zdarza się, że korzystamy z obiektów, które są powtarzalne. Monety i skarby do zdobycia, rakiety i pociski, wrogie statki, ba, nawet drzewa i skały – pojawiają się one w wielu miejscach, ale za każdym razem mają dokładnie takie same parametry i zachowanie.

W mojej demonstracyjnej scenie mam pięć sześcianów, które są praktycznie takie same – z dokładnością do ich położenia. Jednak za każdym razem, gdy chciałem przetestować inną wartość maksymalnej prędkości obrotu, musiałem to robić dla każdego z nich z osobna. Na szczęście Unity i w tym przypadku ma bardzo prosty, a jednocześnie potężny mechanizm, który pomaga zaradzić takim sytujom, a mowa o tak zwanych prefabach (lub bardziej po polsku – prefabrykach).

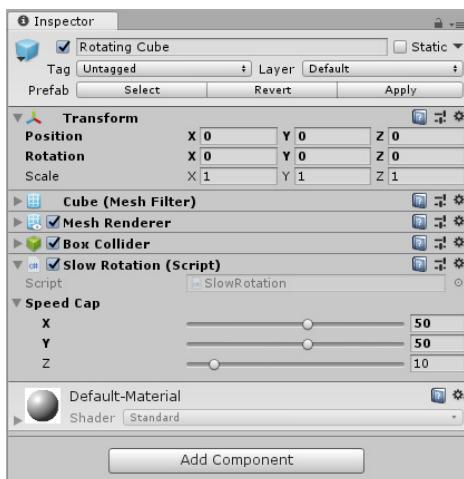
W momencie, w którym uznam, że obiekt jest na tyle dopracowany, bym mógł go zwielokrotnić, mogę przeciągnąć go z hierarchii do okna projektu, by uczynić z niego tak zwany prefab (Rysunek 17). Zwrócić uwagę, że w okienku hierarchii obiekt zmienił

swój kolor na granatowy – jest to sygnał, że jest to tak zwana instancja prefaba – jego klon umieszczony na scenie.



Rysunek 17. Tworzymy prefab

Unity w bardzo przewidywalny sposób zachowuje się wobec parametrówprefaba i jego instancji. Jeżeli zmienimy jakiś parametr w komponentachprefaba, automatycznie we wszystkich jego instancjach wartość ta zostanie również zmieniona. Jeśli jednak zmienimy ją w którejśinstancji, Unity zapamięta, że została ona zmieniona, i nie będzie przenosić już zmian oryginału. Co ważne, możemy jasno zobaczyć, że jeden z parametrów obiektu został zmieniony w stosunku do oryginału – Unity oznacza takie parametry poprzez pogrubienie jego nazwy i wartości (Rysunek 18).



Rysunek 18. Zmodyfikowane parametry instancji prefaba

Bardzo łatwo można również przywrócić wartości do zapisanych w oryginalu – wystarczy kliknąć daną wartość prawym przyciskiem myszy i wybrać „Revert Value to Prefab”. Menu GameObject skrywa jeszcze dwie dodatkowe funkcje: „Apply Changes To Prefab”, która aplikuje zmiany zrobione w instancji prefaba do oryginału (co oczywiście przeniesie się na wszystkie inne jego instancje), oraz „Break Prefab Instance”, która spowoduje zerwanie powiązania z prefabem – instancja stanie się wówczas samodzielnym obiektem.

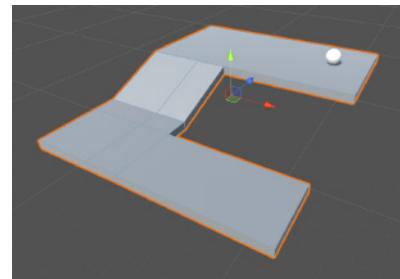
INTERAKCJA Z INNYMI OBIEKTAMI

Modyfikowanie transformacji obiektu jest jednym z najprostszych działań, które możemy podjąć w skrypcie – prawdziwą potęgą

architektury ECS jest możliwość interakcji komponentów ze sobą oraz encji z innymi encjami.

Unity daje praktycznie nieograniczony dostęp do komponentów w skryptach. Możemy w nich robić w zasadzie wszystko to, na co pozwala edytor: dodawać, usuwać, pobierać oraz zmieniać wartości ich parametrów. Ponieważ najłatwiej będzie pokazać to na przykładzie, spróbujmy na koniec napisać prostą grę polegającą na sterowaniu przez gracza kulki po planszy zawieszonej w powietrzu – ot, taki klon gry „Balance”.

Na początku przygotowujemy scenę – budujemy na niej jakiś prosty poziom i umieszczamy na nim kulkę; może to wyglądać na przykład tak jak na Rysunku 19. Po ustawieniu flagi static dla podłogi i dodaniu komponentu Rigidbody do kulki powinna ona spaść na podłogę i zatrzymać się na niej.



Rysunek 19. Zaprojektowana scena

Podstawowym narzędziem, z którego możemy zawsze skorzystać, zarówno wewnętrz skryptu, jak i w kontekście konkretnego obiektu GameObject, jest generyczna metoda GetComponent. Za jej pomocą możemy pobrać komponent i nawiązać z nim interakcję. Pobranie komponentu zazwyczaj realizowane jest w momencie inicjalizacji skryptu; naturalnie może zdarzyć się sytuacja, w której komponentu nie uda się pobrać, bo po prostu nie będzie go posiadał – dostaniemy wówczas wartość null. Częściej jednak zdarza się, że jeden komponent po prostu wymaga drugiego dla prawidłowej pracy i nie możemy sobie pozwolić na jego brak. W takiej sytuacji Unity pozwala zdefiniować zależności komponentu przy użyciu atrybutu [RequireComponent]. Gdy wówczas dodamy do obiektu komponent, Unity automatycznie doda też wszystkie jego brakujące zależności i nie pozwoli ich usunąć, dopóki nie usuniemy wcześniej samego komponentu.

Na początku napiszmy kontroler kulki – umożliwimy w ten sposób graczowi sterowanie nią.

Listing 10. Sterowanie kulką

```
[RequireComponent(typeof(Rigidbody))]
public class BallController : MonoBehaviour {
    private Rigidbody ballRigidbody;
    [SerializeField]
    private float acceleration = 5.0f;
    [SerializeField]
    private float maxVelocity = 20.0f;
    void Awake()
    {
        ballRigidbody = GetComponent<Rigidbody>();
    }
    void FixedUpdate()
    {
        float horizontal = Input.GetAxis("Horizontal");
        float vertical = Input.GetAxis("Vertical");
        var force = new Vector3(horizontal * acceleration,
```

```

        0,
        vertical * acceleration);

ballRigidbody.AddForce(force);
ballRigidbody.velocity = Vector3.
    ClampMagnitude(ballRigidbody.velocity, maxVelocity);
}
}

```

Ponieważ oczekujemy, że nasza kulka będzie zachowywać się jak ciało sztywne, wymagamy komponentu Rigidbody, który pobieramy zaraz w momencie, gdy nasz skrypt zostanie zainicjowany (w metodzie Awake). Metoda FixedUpdate wprawia kulkę w ruch w czterech krokach:

- » Pierwszym krokiem jest pobranie bieżącego wychylenia wirtualnych osi stanowiących abstrakcję dla wejścia (ang. *input*). Nazwa „os” nawiązuje do joysticka, w którym możemy mieć dwie lub więcej osi, w zależności od tego, ile potencjometrów mamy do dyspozycji. W przypadku klawiatury Unity traktuje wciśnięcia klawiszy jako wychylenie wirtualnych osi i w taki sposób możemy je odczytać – dzięki temu nie musimy zastanawiać się, z jakiego kontrolera korzysta gracz. W pustym projekcie zdefiniowane jest domyślnie 18 osi, z których dwie pierwsze – nazywające się „Horizontal” i „Vertical” – sterowane są klawiszami strzałek, a skonfigurować je możemy po wybraniu z menu Edit | Project Settings | Input. Wartość wychylenia osi otrzymamy jako liczbę w zakresie od -1 do 1.
- » Drugim krokiem jest obliczenie siły, która zostanie przyłożona do kulki, wprawiając ją w ten sposób w ruch. Wartości wychylenia mnożymy tu przez pewną stałą, która jest parametrem – jej wartość możemy ustalić w edytorze.
- » Trzecim krokiem jest zaaplikowanie siły do kulki za pomocą jej komponentu Rigidbody.
- » Wreszcie metoda upewnia się, że kulka nie porusza się szybciej niż na to pozwolimy, przycinając długość wektora prędkości do określonej wartości (która również mamy sparametryzowaną).

Po dołączeniu skryptu na obiekcie kulki powinniśmy już móc nią poruszać przy użyciu klawiszy strzałek – konieczne może być co najwyżej dopasowanie wartości parametrów.

Aby gra była ciekawsza, możemy wprowadzić przeszkodę – drzwi, które trzeba otworzyć za pomocą przycisku. Zmodyfikujmy więc naszą scenę, wprowadzając dodatkowe obiekty: przycisk oraz proste drzwi. Pamiętajmy o tym, aby w komponencie Rigidbody drzwi zablokować pozycję i obrót obiektu, w przeciwnym razie będziemy mogli kulką przepchnąć lub przewrócić drzwi (Rysunek 21).

Spróbujmy tu zastosować dobre praktyki programowania obiektowego – to ułatwi nam potem ponowne użycie komponentów w innych obiektach bez niepotrzebnych zmian w kodzie. Przycisk z założenia powinien coś uruchomić, odetnijmy go jednak od konkretnej implementacji akcji za pomocą interfejsu.

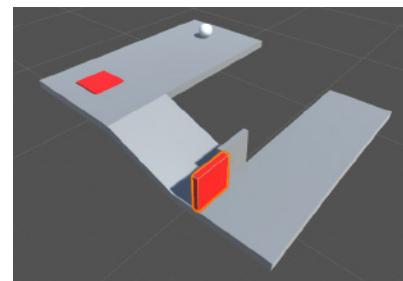
Listing 11. Interfejs obiektu „uruchamialnego”

```

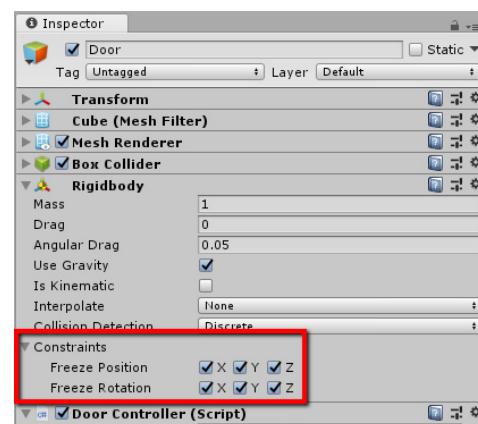
namespace Assets.Scripts
{
    public interface IExecutable
    {
        void Execute();
    }
}

```

W ten sposób skryptowi obsługującemu przycisk możemy przekazać obiekt, którego jeden z komponentów implementuje nasz interfejs. Po wykryciu kolizji przycisku z kulą spróbujmy pobrać ten



Rysunek 20. Przycisk i przeszkoda



Rysunek 21. Konfiguracja Rigidbody dla drzwi

komponent i wywołać Execute, pozostawiając tej metodzie zrealizowanie odpowiedniej akcji. Aby komponent ten działał prawidłowo, pamiętajmy o nadaniu kulce wbudowanego taga „Player”.

Listing 12. Kontroler przycisku

```

public class ButtonController : MonoBehaviour
{
    [SerializeField]
    private GameObject target;

    private bool buttonEnabled = true;

    private void OnCollisionEnter(Collision collision)
    {
        if (buttonEnabled && collision.gameObject.tag == "Player")
        {
            buttonEnabled = false;

            IExecutable executable = (IExecutable)target.GetComponent(
                typeof(IExecutable));
            if (executable != null)
                executable.Execute();
        }
    }
}

```

Na wstępie zdefiniowaliśmy „cel” przycisku – pole target. Oczekujemy, że jeden z jego komponentów implementuje interfejs IExecutable, co umożliwi nam przekazanie mu sterowania w momencie, gdy przycisk zostanie uruchomiony. Pole buttonEnabled zostało wprowadzone, by przycisku można było użyć tylko raz.

Metoda OnCollisionEnter zostaje wywołana wówczas, gdy kolidują ze sobą collidery dwóch obiektów, z których przynajmniej jeden ma komponent Rigidbody. W takiej sytuacji sprawdzamy, czy kolidującym obiektem jest gracz (poprzez sprawdzenie jego taga), a jeżeli tak, to z docelowego obiektu pobieramy komponent implementujący IExecutable i o ile taki znajdziemy, wywołujemy na nim metodę Execute. Myślę, że kod tego kontrolera jest stosunkowo czytelny i nie wymaga bardziej szczegółowego komentarza.

Na koniec pozostała nam jeszcze kontroler drzwi.

Listing 13. Kontroler drzwi

```
namespace Assets.Scripts
{
    public class DoorController : MonoBehaviour, IExecutable
    {
        [SerializeField]
        private Vector3 openVector = new Vector3(1.0f, 0.0f, 0.0f);
        [SerializeField]
        private float openTime = 1.0f; // 1 sekunda

        private bool isOpening = false;
        private bool opened = false;
        private Vector3 initialPosition;
        private Vector3 targetPosition;
        private float currentTime;

        void Start()
        {
            initialPosition = transform.position;
            targetPosition = transform.position + openVector;
            currentTime = 0.0f;
        }

        public void Execute()
        {
            if (!opened)
            {
                isOpening = true;
            }
        }

        void FixedUpdate()
        {
            if (isOpening)
            {
                currentTime += Time.fixedDeltaTime;
                if (currentTime >= openTime)
                {
                    transform.position = targetPosition;
                    opened = true;
                    isOpening = false;
                }
                else
                {
                    transform.position = Vector3.Lerp(initialPosition,
                        targetPosition, (currentTime / openTime));
                }
            }
        }
    }
}
```

Dla uproszczenia skorzystamy z drzwi przesuwanych. Na wstępie deklarujemy dwa parametry: `openVector`, który wskazuje kierunek i odległość, na jaką powinny przesunąć się drzwi po ich otwarciu, oraz `openTime`, który pozwala określić czas otwierania. Pozostałe pola definiują, czy drzwi są w trakcie otwierania (`isOpening`), czy zostały już otwarte, aby zapobiec ponownemu otwarciu (`opened`), początkową oraz końcową pozycję (`initialPosition` oraz `targetPosition`), a także czas, który upłynął od rozpoczęcia otwierania (`currentTime`).

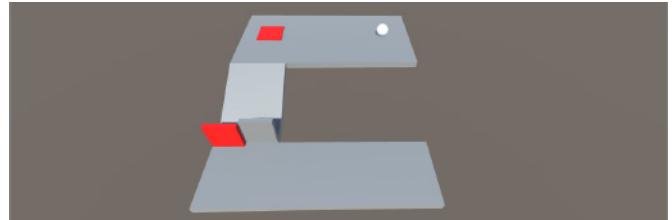
W momencie aktywowania komponentu obliczamy pozycję początkową i końcową (po otwarciu drzwi) – będą one potrzebne do wykonania animacji. Zerujemy również czas otwierania – będzie on zwiększany w momencie, gdy nastąpi otwarcie drzwi.

Ponieważ chcemy, by skrypt został aktywowany przyciskiem, implementujemy interfejs `IExecutable`, zaś w metodzie `Execute` zapalamy flagę `isOpening` (o ile drzwi nie zostały już otwarte).

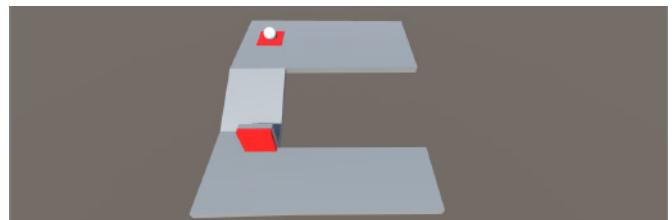
Zadaniem metody `FixedUpdate` jest w zasadzie tylko zrealizowanie animacji otwierania drzwi. Podczas każdego jej uruchomienia zwiększana jest wartość `currentTime` – dzięki temu wiemy, ile czasu upłynęło od momentu rozpoczęcia otwierania drzwi. Jeśli nie przekroczyliśmy jeszcze czasu otwarcia, obliczamy pośrednią pozycję za pomocą metody `Vector3.Lerp` (dla wartości 0 zwróci

ona `initialPosition`, dla 1 – `targetPosition`, zaś dla wartości pomiędzy 0 i 1 – pozycję pośrednią). Po osiągnięciu czasu otwierania blokujemy drzwi w pozycji otwartej i zapalamy flagę `opened` – dzięki temu drzwi nie uda się otworzyć ponownie.

Klikamy przycisk „Play” i sprawdzamy, czy wszystko działa (Rysunki 22 i 23)



Rysunek 22. Przed otwarciem drzwi



Rysunek 23. Drzwi otwarte

Gotowy projekt możnaściągnąć ze strony <https://programistamag.pl/download/> (pamiętajmy o tym, że projekty nie są kompatybilne wstecz, więc do jego otwarcia potrzebna będzie obecnie najnowsza wersja Unity).

NA ZAKOŃCZENIE

Na pomysł napisania artykułu wpadłem dlatego, że w Internecie bardzo brakuje dobrego wprowadzenia do Unity w postaci tekstu. Choć można znaleźć tam setki filmów – skądinąd naprawdę świetnych – to brakowało mi artykułu, który mógłbym przeczytać i mieć jakieś pojęcie na temat tego, jak w ogóle zacząć pisanie własnej gry.

Z braku lepszego rozwiązania zacząłem po prostu czytać manual do Unity, dostępny na stronie <https://docs.unity3d.com/Manual> – w zasadzie od początku, strona za stroną – do momentu gdy zacząłem mniej więcej rozumieć, jakie są zależności pomiędzy różnymi elementami silnika i jak je oprogramowywać. Na marginesie: to wcale nie był zły pomysł, manual jest napisany bardzo dobrym językiem i naprawdę przystępnie – jeżeli po przeczytaniu tego artykułu czujecie niedosyt, możecie po prostu zrobić to samo, co ja.

W nadchodzących artykułach postaram się napisać więcej o różnych elementach Unity, a także o innych tematach związanych z grafiką 3D. A w międzyczasie zapraszam do zabawy w programowanie gier – daje to naprawdę mnóstwo frajdy, zwłaszcza że dużo można osiągnąć naprawdę małym kosztem: szybki efekt jest zawsze bardzo silnym motywatorem, a Unity definitelynie jest środowiskiem, które pomoże go osiągnąć.



WOJCIECH SURA

wojciechsura@gmail.com

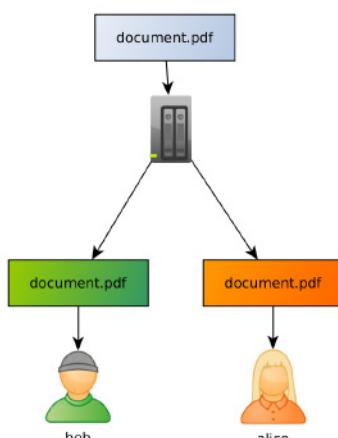
Programuje od przeszło dziesięciu lat w Delphi, C++ i C#, prowadząc również prywatne projekty. Obecnie pracuje w polskiej firmie PGS Software S.A., zajmującej się tworzeniem oprogramowania i aplikacji mobilnych dla klientów z całego świata.

Jak schwytać złodzieja danych

Co łączy wielkie korporacje, służby specjalne oraz gildie w grach komputerowych? Na pierwszy rzut oka niewiele, ale jest co najmniej jedna wspólna cecha: wszystkie mają swoje sekrety oraz chcą je ukryć przed resztą świata. Nieważne, czy mówimy o projekcie nowego produktu, szczegółach operacji antyterrorystycznej albo planach ataku – niektóre informacje powinny zostać w zaufanej grupie.

Niestety, wycieki danych się zdarzają. Nawet NSA¹ czy CIA² nie są od nich wolne. Bo jak zabezpieczyć dane przed wyciekiem w cyfrowym świecie? W przypadku ekstremalnie wrażliwych danych możemy trzymać je na komputerach odizolowanych od sieci, uniemożliwić podłączanie do komputera dysków USB oraz nagrywarek CD, a nawet korzystania z telefonów w pobliżu. Ciężko jednak wyobrazić sobie narzucanie takich ograniczeń ludziom w biurze, albo w przypadku organizacji działających głównie w Internecie.

Jest jednak metoda, która może znaczco utrudnić upublicznienie danych. Wystarczy upewnić się, że osoba wynosząca informacje nie pozostanie anonimowa. W końcu mało kto zaryzykowałby pracę i odpowiedzialnością karną, nawet w słusznej sprawie. Na przykład, jeśli kontrolujemy serwer z dokumentami, możemy napisać małe przezroczyste proxy, które automatycznie dokleja do każdego dokumentu informację, który użytkownik go pobrał (Rysunek 1).



Rysunek 1. Interakcja z serwerem

Bob może o tym wiedzieć albo nie, ale w tym momencie, kiedy document.pdf znajdzie się w Internecie, będzie łatwo wskazać winnego.

Cały ten artykuł jest napisany z punktu widzenia osoby chcączej zaimplementować takie zabezpieczenie – wydaje się to naturalne, skoro czasopismo jest skierowane dla programistów. Jednak te same informacje są przydatne także dla ludzi, którzy chcą ominąć takie systemy. W szczególności dla (różnie ocenianych) demaskatorów (ang. *whistleblowers*), czyli ludzi, którzy upubliczniają nielegalnie zdobyte informacje, żeby zwiększyć świadomość publiczną na temat nieuczciwych lub niemoralnych działań wielkich organizacji.

1. UKRYWANIE INFORMACJI W PLIKACH

Mniej poetycko ujmując, ten artykuł będzie o tym, jak można sprytnie ukryć pewne dane (na przykład nazwę użytkownika) w plikach. Nie chodzi jednak o zwykłe doklejenie informacji. Zależy nam na tym, by dodane dane nie rzucały się w oczy oraz aby było je jak najtrudniej usunąć. Inaczej mówiąc, chcemy stworzyć cyfrowy **znak wodny**. Temat jest bardzo szeroki, więc w tym artykule skoncentrujemy się tylko na dwóch przykładach – plikach graficznych oraz tekście.

Ale nie będzie to zwykły artykuł na temat steganografii. Wiele cyfrowych metod steganograficznych koncentruje się na detalach formatu pliku – czyli np. wykorzystania detali tego, jak działa wewnętrznie format .png. Jest tutaj jednak jeden problem – z definicji bardzo łatwo usunąć takie informacje, nawet przypadkowo (np. konwertując obraz do formatu .jpg). Tymczasem my chcemy zapisać dane w maksymalnie trwały sposób. Z tego powodu zignorujemy zupełnie te metody (zostało na ich temat napisane wystarczająco dużo) i zastanowimy się, jak schować dane „głębiej”.

2. UKRYWANIE INFORMACJI W OBRAZACH

Zacznijmy od plików graficznych, ponieważ są chyba najbardziej popularnym przykładem.

Klasycznym sposobem na ukrycie informacji w obrazku jest zapisanie informacji na najniższych bitach każdego piksela. W celu zakodowania wiadomości zmieniamy ją na bity i zapisujemy subtelną wartość każdego piksela. W najprostszej wersji w celu zakodowania „0” zmieniamy piksel na parzysty, a w celu zakodowania „1” zmieniamy piksel na nieparzysty.

Ta metoda jest tak popularna, że (jako jedyna) została nawet wspomniana na polskiej Wikipedii³.

Weźmy pożyczony bezpośrednio z Wikipedii przykład: niepozorne drzewo (Rysunek 2). Okazuje się jednak, że jeśli przeczytamy najniższe dwa bity każdego piksela, otrzymamy zdjęcie kota (Rysunek 3).

Na pierwszy rzut oka ta metoda spełnia wszystkie nasze oczekiwania. Patrząc na Rysunek 2, trudno się domyślić, że ukryte w nim są jakieś dane. Dodatkowo gęstość zakodowanych informacji jest bardzo wysoka – udało się zakodować aż 6 bitów na każdym pikselu. Co więcej, zapisane dane są w stanie przetrwać pewne drobne modyfikacje – na przykład skonwertowanie obrazu do innego, bezstratnego formatu albo screen-shot ekranu. Niestety, trwałość zakodowanych danych nie jest dla nas satysfakcjonująca. Żeby je

1. https://pl.wikipedia.org/wiki/Edward_Snowden

2. <https://www.spidersweb.pl/2017/03/wikileaks-cia.html>

3. <https://pl.wikipedia.org/wiki/Steganografia>



Rysunek 2. Niepozorny obrazek (źródło: https://pl.wikipedia.org/wiki/Plik:Steganography_original.png, użytkownik Cyp, licencja CC-BY 2.0)



Rysunek 3. Ukryta grafika (źródło: https://pl.wikipedia.org/wiki/Plik:Steganography_recovered.png, użytkownik Cyp, licencja CC-BY 2.0)

wymazać, wystarczy nawet nieświadomie zapisać obraz w strategijnym formacie (tak jak np. jpg) albo lekko przeskalać. Tymczasem my jako cel postawiliśmy sobie zakodowanie informacji tak, żeby usunięcie ich było maksymalnie trudne.

Czy jednak możemy wymyślić coś znacznie lepszego? Odpowiedź brzmi: oczywiście tak, inaczej spora część tego artykułu nie miałaby sensu. Wcześniej jednak musimy cofnąć się o kilka kroków i przerobić trochę teorii.

2.1 TRANSFORMACJA FOURIERA

Konkretnie, w dalszej części artykułu wykorzystamy Dyskretną Transformację Fouriera⁴. Większość czytelników prawdopodobnie miała (nie)przyjemność zapoznać się z nią już na wczesnych latach studiów. W takim przypadku zazwyczaj prezentowana jest przyjazna definicja w rodzaju:

$$A_n = \sum_{n=0}^{N-1} a_n e^{\frac{2\pi i}{N} kn}$$

Wzór 1. Klasyczna postać Dyskretnej Transformaty Fouriera

Albo równoważnie:

$$A_n = \sum_{n=0}^{N-1} a_n \left[\cos\left(\frac{2\pi kn}{N}\right) - i \sin\left(\frac{2\pi kn}{N}\right) \right]$$

Wzór 2. Rozwiniecie wzoru 1

4. https://en.wikipedia.org/wiki/Discrete_Fourier_transform

Wspomina się też wtedy, że reprezentuje ona ciąg próbek wejściowych w przestrzeni częstotliwości (ang. *frequency domain*). Ale co to naprawdę znaczy? I do czego można to wykorzystać? Odpowiedź na to pytanie nie zawsze jest podawana bezpośrednio.

Tymczasem transformacja Fouriera jest niezwykle przydatna przy dowolnym przetwarzaniu sygnałów (np. obrazów, dźwięku, wibracji lub sygnałów elektrycznych), więc warto rozumieć ją bardziej intuicyjnie. O co tu właściwie chodzi? W dużym uproszczeniu – Fourier pozwala nam zapisać dowolną funkcję jako sumę cosinusów (i sinusów, jeśli przejmujemy się wartościami złożonymi). Ponieważ artykuł nie traktuje o matematyce, jak również bardziej chodzi mi o pokazanie intuicji niż dokładnej definicji, popatrzmy na przykłady działania algorytmu w praktyce:

Listing 1. Prosty Przykład transformacji oraz jej odwrotności

```
> import scipy.fftpack as fp
> print fp.rfft([1, 1, 1, 1]) # wykonaj FFT na liście [1, 1, 1, 1]
[ 4.  0.  0.  0.] # wynik: lista w przestrzeni częstotliwości
> print fp.irfft([4, 0, 0, 0]) # operacja odwrotna
[ 1.  1.  1.  1.] # wynik: oryginalna lista
```

Ten wynik oznacza, że sygnał [1, 1, 1] można zapisać jako $4 \cdot \cos(2\pi \cdot 0)/4$ (patrz: wzór 2), czyli po prostu $1 \cdot \cos(0) = 1$. Spróbujmy pobawić się trochę z odwrotną transformacją:

Listing 2.Więcej przykładów dla pierwszego elementu

```
> print fp.irfft([4, 0, 0, 0])
[ 1.  1.  1.  1.]
> print fp.irfft([5, 0, 0, 0])
[ 1.25 1.25 1.25 1.25]
> print fp.irfft([8, 0, 0, 0])
[ 2.  2.  2.  2.]
> print fp.irfft([8, 0, 0, 0, 0, 0, 0, 0])
[ 1.  1.  1.  1.  1.  1.  1.  1.]
```

Jak widać, pierwszy element odpowiada za stały czynnik dodawany do każdego elementu. Rodzi się oczywiście pytanie, za co odpowiada drugi element? Można łatwo wydedukować to ze wzoru, ale jeszcze prościej sprawdzić interaktywnie:

Listing 3. Działanie drugiego elementu

```
> print fp.irfft([0, 4, 0, 0])
[ 2.  0. -2.  0.]
```

Ciekawe – w końcu widać, że to faktycznie cosinus. Jeszcze łatwiej to zauważyc kiedy zwiększymy ilość elementów:

Listing 4. Cosinus na całym zakresie danych

```
print fp.irfft([0, 8, 0, 0, 0, 0, 0, 0, 0])
[ 2, 1.4142, 0, -1.4142, -2, -1.4142, 0, 1.4142]
```

Im dalszy element, tym mniejszy cykl funkcji cosinus, aż przy ostatnim elemencie:

Listing 5. Cosinus o dużej częstotliwości

```
> print fp.irfft([0, 0, 0, 4])
[ 1. -1.  1. -1.]
> print fp.irfft([0, 0, 0, 0, 0, 0, 0, 8])
[ 1. -1.  1. -1.  1. -1.  1. -1.]
```

Mówimy cały czas o odwrotnej transformacji. Prawdziwa magia polega na transformacji we właściwą stronę, czyli szybkiej zmianie dowolnego sygnału na sumę cosinusów:

Listing 6. Transformata dla dowolnych danych

```
> print fp.rfft([1337, 1234, 13, 42])
[ 2626.  1324. -1192. 74.]
```

No dobrze, ale jaki to ma związek z właściwym tematem tego artykułu? Okazuje się, że można tę operację zastosować też w przypadku większej ilości wymiarów⁵. W szczególności w przypadku dwóch wymiarów oznacza to, że możemy konwertować obrazy do przestrzeni częstotliwości i z powrotem. Zobaczmy, jak wygląda to w praktyce.

Temat samodzielnej implementacji DFT wykracza poza ramy tego artykułu. O ile relatywnie prosto jest napisać kod prosto z definicji, to zaimplementowanie szybkiego i pewnego algorytmu jest trudne i czasochłonne. Zamiast tego skorzystamy z gotowych bibliotek naukowych numpy oraz scipy:

Listing 7. Szablon kodu do modyfikacji obrazów

```
import numpy as np
import scipy.fftpack as fp
import sys
from PIL import Image

def im2freq(data):
    """ Skonwertuj obraz do przestrzeni częstotliwości """
    return fp.rfft(fp.rfft(data, axis=0), axis=1)

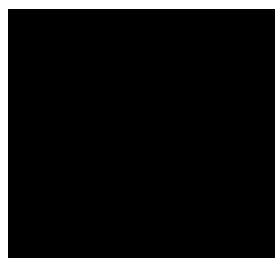
def freq2im(data):
    """ Konwersja "z powrotem", do pikseli """
    return fp.irfft(fp.irfft(data, axis=1), axis=0)

def touint8(data):
    """ Skalowanie wartości tablicy z powrotem do (0, 255) """
    data = data - np.amin(data, axis=(0,1), keepdims=True)
    data = data / max(data.max(), 1)
    return (data * (256-1e-4)).astype(int)

def save(data, fname):
    """ Zapisanie tablicy jako obrazu na dysk """
    out = Image.new('RGB', data.shape[1:-1])
    out.putdata(map(tuple, data.reshape(-1, 3)))
    out.save(fname)

def main():
    freq = im2freq(np.array(Image.open(sys.argv[1])))
    # freq to tablica reprezentująca obraz w przestrzeni
    # częstotliwości. W tym miejscu wykonamy na niej
    # jakieś operacje, a następnie skonwertujemy z powrotem
    save(touint8(freq2im(freq)), sys.argv[2])
```

Weźmy na przykład zupełnie czarny obraz – jak na Rysunku 4. Wykonajmy na nim DFT. Co będzie wynikiem? Oczywiście – dalej ten sam zupełnie czarny obraz. Dane wejściowe składają się z sa- mych zer, więc wynik również będzie zerami (reprezentowanymi przez czarne piksele).



Rysunek 4. Obrazek z czarnym tłem

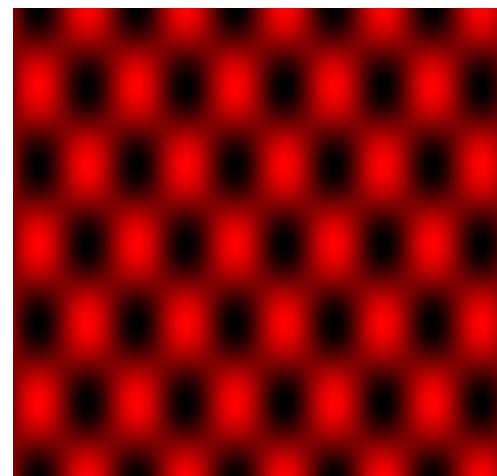
5. Samo FFT generalizuje się do wielu wymiarów. Mimo tego w tym artykule używamy prostszego do implementacji i zrozumienia jednowymiarowego FFT, za to wykonujemy go dwa razy.

Wykonajmy teraz prostą operację w przestrzeni częstotliwości – zmieńmy jakąś wartość na znacznie wyższą. Graficznie mówiąc, rozjaśniemy znacznie jeden piksel:

Listing 8. Pierwsza próba – zmiana jednego piksela

```
freq[5][10][0] = 50000
# ustawienie czerwonego kanału „piksela” (5, 10)
# na wysoką wartość
```

Po wykonaniu transformacji odwrotnej pojawią się odpowiednie „cosinusy”. Wynikiem będzie „szachownica”, zademonstrowana na Rysunku 5.



Rysunek 5. Szachownica cosinusów

Pokazuje to, jak faktycznie wygląda pojedyncza „składowa” w przestrzeni częstotliwości. Nic nie stoi na przeszkodzie, żebyśmy ustawiли więcej składowych jednocześnie, tworząc psychodeliczną tęczę – Rysunek 6.

Listing 9. Trzy piksele jednocześnie.

```
freq[5][10][0] = 50000 # pozycja (5, 10), czerwony kanał
freq[7][8][1] = 40000 # pozycja (7, 8), zielony kanał
freq[9][4][2] = 30000 # pozycja (9, 4), niebieski kanał
```



Rysunek 6. Cosinusowa tęcza

Jak widać, wyprodukowane w ten sposób obrazy są bardzo abstrakcyjne. Jednakże teoria Fouriera mówi, że ustawiając odpowiednio dużo punktów, otrzymamy dowolnie dokładne przybliżenie



Zapraszamy na autorskie szkolenia
z zakresu **bezpieczeństwa IT**

- { Bezpieczeństwo aplikacji WWW }
- { Offensive HTML, SVG, CSS and other Browser-Evil }
- { Wprowadzenie do bezpieczeństwa IT }
- { Szkolenie przygotowujące do egzaminu CEH
(Certified Ethical Hacker) }

www.securitum.pl/oferta/szkolenia

Patroni medialni: sekurak.pl



rozwal.to



dla każdego obrazu! Na przykład dla Rysunku 7 możemy za pomocą pokazanego kodu wykonać transformację w obie strony i wrócić do stanu nieroźnialnego od oryginału (dla ludzkiego oka).



Rysunek 7. Zdjęcie kaczki, na „której” testowaliśmy prezentowane algorytmy

Jak wygląda graficzna reprezentacja tego obrazu w przestrzeni częstotliwości? Mało ciekawie. Nawet po sztucznym zwiększeniu kontrastu dwudziestokrotnie cała składa się z jednolitego koloru. Jedynie po przybliżeniu obrazu, w lewym górnym rogu widać jakąś entropię: Rysunek 8. Mimo wszystko da się z takiej reprezentacji otrzymać z powrotem oryginalną kaczkę, wykonując odwrotną transformację.



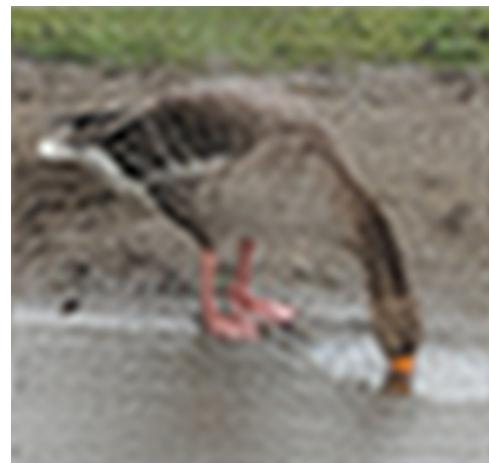
Rysunek 8. Dyskretna Transformata Fouriera z Rysunku 7

Wychodzi na to, że po DFT większość informacji znajduje się przy początku układu współrzędnych (tzn. w lewym górnym rogu obrazka), a reszta nie „wnosi” wiele. Okazuje się, że to całkiem praktyczne spostrzeżenie. Sprytny sposób na kompresję obrazów to właśnie transformacja obrazów do przestrzeni częstotliwości, a następnie odrzucenie wszystkiego poza tym interesującym kawałkiem. Możemy łatwo zaimplementować to w Pythonie:

Listing 10. Wyzerowanie prawie całego obrazka

```
width, height = freq.shape[0:2]
for x in range(width):
    for y in range(height):
        if x > width/10 or y > height/10:
            freq[x][y] = (0, 0, 0)
```

Jak widać, zerujemy wszystko, poza małym prostokątem szerokim i wysokim na 10% oryginalnego rozmiaru (czyli zostaje ledwo 1% danych). Po uruchomieniu go na naszej kaczce dostaniemy bardzo rozmarzaną, ale wciąż rozpoznawalną grafikę (Rysunek 9).



Rysunek 9. Mało wyraźna kaczka z Rysunku 7. Zakodowana za pomocą tylko 1% informacji z oryginału!

Przy zachowaniu 20% obrazu wynik byłby już praktycznie nieroźnialny dla ludzkiego oka – a o ile mniej przestrzeni dyskowej by wymagał! Oczywiście, ktoś już na to wpadł – tak właśnie działa (w wielkim uproszczeniu) format jpg.

2.2 PRAKTYCZNE UKRYWANIE DANYCH

Ale nie jest to artykuł o przetwarzaniu obrazów (było już ich parę na łamach „Programisty”). Do czego zmierzam? Skoro duża część danych w obrazie jest prawie niewidoczna dla ludzkiego oka, to po co się ma marnować? Możemy ukryć tam nasze dane!

Inaczej mówiąc, zamiast ukrywać dane bezpośrednio w obrazie, możemy ukryć je w niewidzialnej dla człowieka przestrzeni częstotliwości. Spróbujmy zaimplementować to w języku Python:

Listing 11. Enkodowanie i dekodowanie

```
def main():
    op = sys.argv[1]
    input = sys.argv[2]
    if op == 'decode':
        freq = im2freq(np.array(Image.open(input)))
        for x, y, z in select_pixels(int(sys.argv[3])):
            value = freq[x][y][z]
            print '{:10.0f} {}'.format(value, int(value > 50000))
            # wypisz surowe wartości zakodowanej wiadomości
    elif op == 'encode':
        freq = im2freq(np.array(Image.open(input)))
        output = sys.argv[3]
        bits = bin2bits(sys.argv[4]) # zmień wiadomość na bity
        for bit, (x, y, z) in zip(bits, select_pixels(len(bits))):
            print bit
            freq[x][y][z] = int(bit) * 100000
            # zmień pixel na 0 dla bitów 0, i 100000 dla bitów 1
        save(touint8(freq2im(freq)), output)
    else:
        print "Unknown operation"
```

I to w zasadzie wszystko, poza dwoma funkcjami pomocniczymi: `select_pixels`, która wybiera, na jakich współrzędnych zapisać nasze n bitów danych:

Listing 12. Funkcja pomocnicza wybierająca n mało znaczących pikseli

```
def select_pixels(n):
    # distance - jak „daleko” ukryć dane.
    # im wyższe, tym więcej danych można ukryć
    # bez widocznej zmiany obrazka, ale też
    # powoduje, że łatwiej je uszkodzić.
    distance = 10
    while True:
        for y in range(distance + 1):
            for channel in range(3):
                yield (distance-y, y, channel)
        n -= 1
        if n <= 0:
            return
    distance += 1
```

A także bin2bits, która zamienia bajty na bity (konkretnie napis składający się z zer i jedynek):

Listing 13. Zmiana bajtów na ciąg bitów

```
def bin2bits(data):
    out = ''
    for byte in data:
        out += '{:08b}'.format(ord(byte))
    return out
```

Oczywiście jest to bardzo prymitywny kod, tak zwany „proof of concept”. Można go użyć do zakodowania wiadomości, a później przeczytania wyników:

Listing 14. Prosta interakcja z programem

```
$ python fft.py encode dziob.png hidden.png xyz
# (program wypisuje bity wiadomości)
$ python fft.py decode hidden.png 24
# (program wypisuje kolejne zdekodowane wartości)
```

Przykład wykonania powyższych komend można zaobserwować na Rysunku 10. Po lewej stronie wynik dekodowania: pierwsza liczba to surowa wartość w przestrzeni częstotliwości, a druga to zdekodowany bit. Po prawej stronie dla porównania poprawne wartości wyliczone podczas kodowania. Jak widać, metoda działa – oryginalna wiadomość została spokojnie i bezbłędnie odtworzona.

Left Column (Output of ./fft.py decode)	Right Column (Output of ./fft.py encode)
30 (0)	0
95007 (1)	1
94953 (1)	1
94905 (1)	1
94897 (1)	1
41 (0)	0
-16 (0)	0
17 (0)	0
-122 (0)	0
94837 (1)	1
94993 (1)	1
94997 (1)	1
94813 (1)	1
-51 (0)	0
-51 (0)	0
94922 (1)	1

Rysunek 10. Interakcja z programem kodującym

Czas porównać ją z przedstawionym na początku ukrywaniem informacji w najniższych bitach obrazka. Czy faktycznie coś zyskujemy?

Okazuje się, że dane ukryte w ten sposób są **znacznie** bardziej odporne na manipulacje. W testach laboratoryjnych, które przeprowadziłem, zakodowane dane przetrwały bez większych problemów:

- » Zapisanie obrazka w stronnym formacie .jpg,
- » Rozjaśnienie, przyciemnienie i zmianę kontrastu obrazka,
- » Przeskalowanie obrazka do 60% rozmiaru i z powrotem,

- » Zakrycie niewielkiej części obrazka (np. narysowanie kilku kresek programem graficznym),
- » A nawet wyświetlenie zdjęcia na monitorze komputera, zrobienie zdjęcia komórką, poprawienie perspektywy zdjęcia w programie graficznym i dopiero zdekodowanie!

W momencie kiedy nasz znak wodny da się, przy pewnej staranności i odrobinie szczęścia, odtworzyć nawet z samego zdjęcia kiepskiej jakości, możemy uznać, że odnieśliśmy sukces⁶. Do osiągnięcia tak ekstremalnego efektu trzeba niestety zapisywać dane blisko początku układu współrzędnych (niski parametr *distance*), co z kolei powoduje, że można zakodować jedynie kilkanaście-kilkadziesiąt bitów bez zmian widocznych dla ludzkiego oka – ale tyle spokojnie wystarczy do naszych zastosowań (na przykład unikalny identyfikator użytkownika powinien zajmować najwyższej 32 bitów).

Trocę historii

Pierwotnie w artykule miały znaleźć się również informacje o ukrywaniu danych w formacie .pdf, ale ostatecznie nie stało się ani miejsca, ani czasu. W ramach ciekawostki można tu nawiązać do historii „Programisty”. Jako że wersja elektroniczna miała tendencję do wyciekania tuż po premierze, autor niniejszego artykułu uczestniczył swego czasu w nieformalnej rozmowie na temat możliwości fingerprintowania elektronicznych wydań magazynu. Powstał nawet prototyp! Niestety (albo na szczęście) system nigdy nie został wykorzystany w praktyce, ale można się zastanawiać, czy istnieją czasopisma, które praktykują takie działania.

3. UKRYWANIE INFORMACJI W CZYSTYM TEKŚCIE

Drugim ciekawym przypadkiem, który chciałbym omówić, jest ukrywanie danych w tekście. Będzie tu znacznie mniej matematyki niż w przypadku grafiki, ale zaprezentowane podejścia są również sprytne.

Ukrywanie danych w tekście niepokoi mnie szczególnie dla tego, że jako programista spędżam większość swojego czasu, pracując właśnie z tekstem. Wydaje się, że nie da się nic ukryć bez zwracania uwagi – w końcu łatwo zauważać dodatkowe znaki, słowa lub zdania. No i faktycznie – w większości byłaby to prawda... gdyby nie standard zwany Unicode.

Unicode to standard kodowania oraz zestaw znaków, z którego korzystają wszystkie współczesne programy przetwarzające tekst. Okazuje się też, że jest fantastycznie skomplikowany i bardzo trudno jest napisać program, który działa z nim dobrze w 100% przypadków. Ale nie będziemy tu narekać na komplikacje kodowania znaków (konsorcjum Unicode i tak świetnie sobie poradziło z tematem, biorąc pod uwagę, jak chaotyczne są naturalne języki). Przeciwne – wykorzystamy je do własnych celów.

3.1. Niewidzialne znaki

Jedna z prostszych metod ukrycia danych w tekście to użycie niewidzialnych znaków. W Unicode jest ich dostatek – na przykład znaki „Zero-width non-joiner” oraz „Zero-width space”. Oba znaki mają zerową szerokość (jak sama nazwa wskazuje), przez co są niewidoczne w większości sytuacji. Pokazują się one jedynie w niektórych specjalizowanych edytorach dla programistów, oraz oczywiście widać je przy przeglądaniu bajtów w hexedytorze.

6. W zasadzie tak zakodowane dane przetrwały też wydrukowanie obrazka w gazecie i zeskanowanie go...

Jak można ich użyć? Wystarczy skonwertować wiadomość, którą chcemy zakodować, na bity (używając np. poprzednio przedstawionej metody `bin2bits`), zamienić każde zero na „Zero-width non-joiner”, a każdą jedynkę na „Zero-width space” i dokleić taki znacznik gdzieś do tekstu (może nawet kilka razy). Tak przedstawiony algorytm jest możliwy do zaimplementowania nawet w kilku linijkach Pythona:

Listing 15. Ukrywanie danych za pomocą niewidocznych znaków

```
def main():
    ZWSP = u'\u200B'
    ZWNJ = u'\u200C'

    input = sys.stdin.read()
    secret = sys.argv[1]
    encoded_secret = ''.join(
        ZWSP if bit == '0' else ZWNJ
        for bit in bin2bits(secret)
    )
    output = input[0] + encoded_secret + input[1:]
    sys.stdout.write(output.encode('utf-8'))
```

Czy to faktycznie działa? Łatwo sprawdzić w konsoli (Rysunek 11):

```
msm@europa /home/msm/demo$ echo "plain text"
plain text
msm@europa /home/msm/demo$ echo "plain text" | xxd
00000000: 706c 6169 6e20 7465 7874 0a
msm@europa /home/msm/demo$ echo "plain text" | python hide.py magic
plain text
msm@europa /home/msm/demo$ echo "plain text" | python hide.py magic | xxd
00000000: 70e2 808b e280 8ce2 808c e280 8be2 808c
00000010: e280 8ce2 808b e280 8ce2 808b e280 8ce2
00000020: 808c e280 8be2 808b e280 8be2 808b e280
00000030: 8ce2 808b e280 8ce2 808c e280 8be2 808b
00000040: e280 8ce2 808c e280 8ce2 808b e280 8ce2
00000050: 808c e280 8be2 808c e280 8be2 808b e280
00000060: 8ce2 808b e280 8ce2 808c e280 8be2 808b
00000070: e280 8be2 808c e280 8c6c 6169 6e20 7465
00000080: 7874 0a
```

Rysunek 11. Niewidzialne dane ukryte w tekście

Mamy tu po kolej cztery komendy. Najpierw wypisywany jest tekst „plain text”. Wynik jest zgodny z oczekiwaniami. Następnie konwertujemy poprzedni tekst na bajty – ponownie bez zaskoczeń: każdy znak napisu odpowiada jednemu bajtowi. Dopiero w trzecim poleceniu uruchamiamy nasz skrypt. Nie ma on jednak żadnych widocznych efektów – tekst wygląda dokładnie tak samo jak na początku. Ukryte dane pojawiają się dopiero wtedy, kiedy przedstawimy wynik jako bajty – od razu widać, że danych jest więcej niż powinno być dla tak krótkiej wiadomości.

Tego typu metody są o tyle sprytne, że mało kto spodziewa się ukrytych danych w czymś tak prostym jak tekst. Nie jest to też czysta teoria – istnieją źródła, które mówią, że dokładnie ta metoda była wykorzystywana w praktyce do namierzania „źródeł” dziennikarzy. Są też udokumentowane historie wykorzystywania jej w równie poważnej sprawie, czyli... prywatnych forach gildii w grze EVE Online. Jak widać, niektórzy gracze podchodzą do grania bardzo poważnie.

Główną wadą jest to, że stosunkowo łatwo pozbyć się takiego znaku wodnego, jeśli ktoś wie, czego się spodziewać. Jest skończona liczba niewidzialnych znaków w Unicode i żaden z nich nie powinien się znajdować w zwykłym tekście. Wystarczy więc usunąć wszystkie podejrzane znaki i problem zostanie rozwiązany.

Homografy: a czy a?

Pewną wariacją tej metody jest używanie tak zwanych homografów. Są to pary znaków, które wyglądają tak samo w większości czcionek, ale z punktu widzenia komputera są innymi znakami. Przykładem jest rosyjska litera „a” (bajty: 0xd0, 0xb0), która wygląda często dokładnie tak samo jak łacińska litera „a” (bajty: 0x61). Czyli zamiast dorzucać sztuczny znak wodny, możemy podmieniać niektóre znaki w tekście na inne. Ta metoda ma dużą zaletę – ciężko się jej pozbyć w sposób automatyczny (nie wiedząc dokładnie, jakie homografy są używane). Jedyny pewny sposób to „opcja nuklearna”, czyli usunięcie wszystkich znaków spoza kresu ASCII – ale w ten sposób usunięte zostaną też na przykład polskie znaki, co przysporzy sporu dodatkowej pracy.

W praktyce ten sposób jest wykorzystywany znacznie częściej do ataków na niewinnych klientów banków i znany pod nazwą IDN homograph attack¹. Sztuczka polega na tym, że użytkownik znajduje w złośliwym e-mailu link do www.dobrybank.pl i nie spodziewa się, że jedna literka w adresie nie jest znakiem, na jaki wygląda... Dobre, że przeglądarki stają się (z sukcesami) wykrywać takie ataki i bronić nas przed nimi, bo inaczej nie można by było już ufać nawet paskowi adresu.

1. https://en.wikipedia.org/wiki/IDN_homograph_attack

3.2. Prawdziwe i fałszywe ogonki

Jak wspominałem, Unicode bywa dziwne. Jedną z ciekawostek są tak zwane *combining characters*. To grupa specjalnych kodów, które nie reprezentują konkretnych liter alfabetu, a modyfikują litery przed nimi. Jednym z nich jest na przykład swojsko brzmący U+0328 COMBINING OGONEK⁷. Jak nazwa wskazuje, służy on do dodawania ogonków do znaków – dzięki temu technicznie nie stoi na przeszkode, żeby poza ą i ē używać w tekście litery „d z ogonkiem” – d. Możemy w ten sposób dowolnie „wzbogacać” tekst – trzeba tylko pamiętać, żeby



Jednocześnie jednak większość typowych kombinacji ma swoje dedykowane znaki. Co oczywiście oznacza, że „ą” można zapisać na dwa sposoby – ze znakiem łączącym oraz bezpośrednio. Jak zwykle – takie niejednoznaczności możemy wykorzystać do naszych celów (patrz też: ramka o homografach). Wystarczy np. znaleźć wszystkie znaki „ę” w wiadomości, zmienić kodowaną wiadomość na bity, i jeśli kolejny bit to „0”, zostawić oryginalne „ę”, a w przeciwnym wypadku zamienić na „ę” z doklejonym ogonkiem.

Jaka jest zaleta tej metody? Głównie taka, że wszystkie szanującej się edytory muszą obsługiwac poprawnie *combining characters*. Jeszcze ciekawsze jest to, że technicznie te dwa znaki są sobie równoważne. To znaczy, że poprawnym zachowaniem dla języka programowania byłoby zwrócenie true dla porównania „ę” z „e[COMBINING OGONEK]”.

Wadą jest to, że relatywnie łatwo pozbyć się takich artefaktów, używając normalizacji Unicode – ale trzeba wcześniej wiedzieć o tym, że istnieje.

3.3. Zastosowanie dla synonimów

Na koniec chyba najbardziej trwała metoda (ale jednocześnie najtrudniejsza w implementacji). Co jeśli zamiast koncentrować się na poszczególnych słowach, będziemy podmieniać całe słowa? Na przykład słowo „szybko” można w większości przypadków zamie-

7. <https://codepoints.net/U+0328>

nić na „błyskawicznie”. Można napisać prosty program wyświetlający wszystkie możliwe podmiany:

Listing 16. Wyszukiwarka synonimów

```
import sys, re
ansi = {
    "GREEN": "",
    "RESET": "",
}

def replace(test, word, part):
    repl = '{' + ','.join(
        "\x1b[32m{}\x1b[39m".format(p, **ansi) for p in part
    ) + '}' # kolorowanie na zielono
    return re.sub(
        "\\\b{}\\\b".format(word), repl, test)

def replace_all(test, word, part):
    test = replace(test, word, part)
    test = replace(test, word + 's', part)
    if word.endswith('e'):
        test = replace(test, word + 'e', part)
    else:
        test = replace(test, word + 'ed', part)
    return test

def main():
    synonyms = eval(open('synonyms.txt').read())
    test = sys.stdin.read().lower()
    for part in synonyms:
        for synonym in part:
            test = replace_all(test, synonym, part)
    print test

main()
```

Przy odpowiednim słowniku synonimów⁸ można osiągnąć nawet dobre rezultaty (Rysunek 12). Na trzecim przykładzie widać też jednak, że naiwne podmienianie słów potrafi dać bezsensowne rezultaty, więc trzeba z nim uważać.

8. Ja do testów skorzystałem z <https://www.englisch-hilfen.de/en/words/synonyms.htm>, ale w Internecie znajduje się wiele bogatszych źródeł.

```
xmsm@xmsm1 /tmp/test
$ cat test.txt
you did a great job on your report.
when did you start?
you won't fool me.
xmsm@xmsm1 /tmp/test
$ cat test.txt | python syn.py
you did a {fantastic,great} job on your report.
when did you {begin,start}?
you won't {fool,idiot} me.
```

Rysunek 12. Trzy miejsca na ukrycie informacji

W tym przypadku można zapisać ledwo trzy bity, ale w przypadku dłuższej wiadomości i lepszego słownika (np. mając po 4-5 synonimów do jednego słowa) spokojnie doszłibyśmy do potrzebnych nam około 30 bitów.

4. PODSUMOWANE

Powyższym artykułem chciałem zwrócić uwagę na to, jak łatwo ukryć dane w trudny do wykrycia i usunięcia sposób. Myślę, że może się to przydać zarówno osobom, które próbują zabezpieczyć swoje informacje, jak i tym, którzy mają dobry powód, żeby je naglaśniać. Prywatnie nie stoję po żadnej stronie, za to jestem pod wrażeniem złożoności nawet najprostszych otaczających nas cyfrowych elementów, jak obrazki i tekst.

JAROSŁAW JEDYNAK

msm@tailcall.net

Rozpoczął karierę jako programista, wyspecjalizował się w security. Do niedawna jako analityk malware w CERT Polska zajmował się między innymi inżynierią wstępnej złośliwego oprogramowania i automatyzacją jego analizy. Obecnie pracuje jako security engineer w Google, spędza czas na reverse-engineeringu oraz prowadzi badania nad nowymi sposobami wykrywania złośliwego oprogramowania. W wolnym czasie programuje do szuflady, gra w konkursy CTF i ogląda koty w Internecie.

reklama



devstyle.pl

ŚWIAT OKIEM PROGRAMISTY

Współczesne architektury aplikacji biznesowych. Ports and Adapters oraz microservices

Ten cykl artykułów ma na celu dokonać przeglądu różnych trendów architektonicznych, które pojawiły się w ciągu ostatnich kilku lat, po to aby je uporządkować, zestawić ze sobą, wskazać główne powody zastosowania, jednocześnie układaając je w ewolucyjną ścieżkę, którą może podążać system na tle zmian architektonicznych. Poprzednio analizowaliśmy klasyczną architekturę trójwarstwową oraz Domain-Driven Design. W niniejszej części przyjrzymy się bliżej podejściom Ports and Adapters oraz architekturze microservices.

PORTS AND ADAPTERS

Szkielety aplikacji (ang. *frameworks*) są wszędzie, już po względnie niedługim czasie większa część aplikacji jest przesiąknięta zależnościami w kodzie od obcych bibliotek, których niełatwo się pozbyć. Trudno przetestować aplikacje w oderwaniu od technologii np. od serwera aplikacji, a przecież ten nie zawsze jest potrzebny, aby zweryfikować, jak system podejmuje decyzje podczas realizacji scenariuszy przypadków użycia. Poza tym brak bezpośredniej zależności od szkieletu aplikacji to czytelniejszy kod, łatwiejsze wnioskowanie i mniej magii.

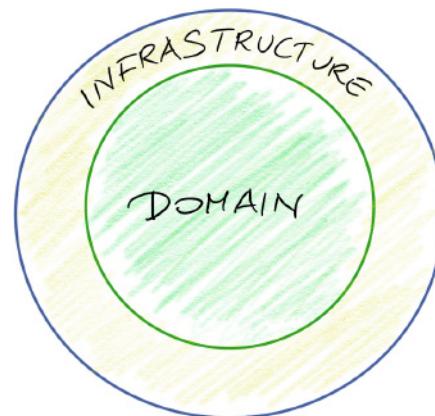
Zależności architektoniczne, o których powyżej mowa, to między innymi:

- » komunikacja z użyciem konkretnego protokołu (np. HTTP/REST),
- » technologie widoku,
- » korzystanie z baz danych,
- » silniki reguł i mechanizmy workflow,
- » komunikacja z użyciem kolejek komunikatów.

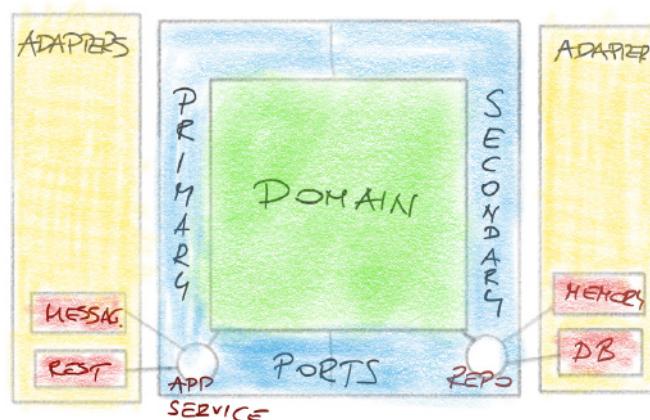
Najczęściej są one przemieszane z logiką tworzonej aplikacji. Z jednej strony elementy infrastruktury mogą wołać metody kodu biznesowego (np. kontroler REST wykonuje operację biznesową) albo kod biznesowy wywołuje operacje związane z infrastrukturą (np. zapisanie wyników do bazy danych).

Pomysł na Ports and Adapters wynikł z przewrotnego pytania: a gdyby tak logicznie zorganizować system, żeby jego esencjonalna, domenowa, core'owa część była całkowicie pozbawiona zależności od technologii? W ten sposób stanowiłaby niejako jądro systemu, a elementy infrastruktury byłyby na zewnątrz, co zostało zobrazowane na Rysunku 1.

Taki koncept jako pierwszy przedstawił Alistair Cockburn w 2005 roku¹. Pozostało tylko pytanie, jak to technicznie rozwiązać? Tutaj z pomocą przychodzi zasada *Dependency inversion*, jedna ze składowych zasad SOLID (patrz ramka), którą w tym przypadku moglibyśmy sprowadzić do stwierdzenia: wszędzie tam, gdzie masz do czynienia z obcymi elementami (czytaj: infrastrukturą, technologią, frameworkami), nie twórz bezpośredniej zależności, a wprowadź



Rysunek 1. Ogólna koncepcja Ports and Adapters



Rysunek 2. Architektura Ports and Adapters z uwzględnieniem interfejsów i ich adapterów

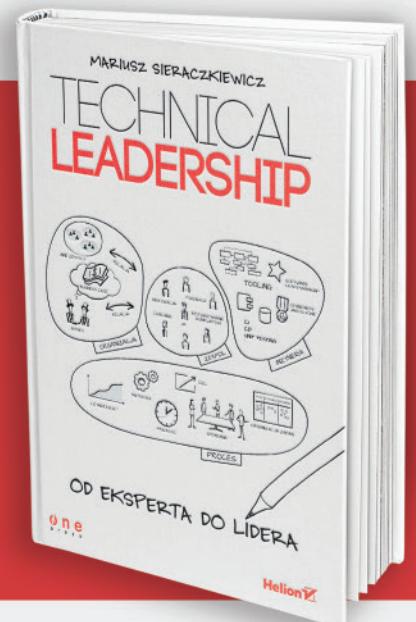
interfejs reprezentujący w sposób abstrakcyjny operacje wejściowe (ang. *primary ports*) i operacje wyjściowe (ang. *secondary ports*). Jeśli logika domenowa potrzebuje zapisać lub pobrać dane np. z bazy danych, korzysta z interfejsu, który jest zaimplementowany z użyciem konkretnej technologii (tzw. adapter). Jeśli aplikacja webowa lub inny klient zewnętrzny potrzebuje wywołać jakiś przypadek użycia, robi to poprzez interfejs wystawiony przez logikę domenową. Bardziej szczegółowy koncept znajduje się na Rysunku 2.

¹ <http://alistair.cockburn.us/Hexagonal+architecture>

BNS IT - SZKOLENIA OTWARTE

WARSZAWA / 04-06.07.2018
TECHNICAL LEADERSHIP™
ROLA LIDERA TECHNICZNEGO

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



ŁÓDŹ / 11-13.06.2018
WZORCE PROJEKTOWE
I REFAKTORYZACJA DO WZORCÓW

1. Wprowadzenie do wzorców projektowych
2. Jakość kodu źródłowego
3. Refaktoryzacja
4. Wzorce GoF

WARSZAWA / 11-13.07.2018
NOWOCZESNE
ARCHITEKTURY APLIKACJI

1. Wprowadzenie do pojęcia architektura oprogramowania
2. Domain-Driven Design
3. Micro Services
4. Ports & Adapters
5. Clean and Onion Architecture
6. Reactive Architecture
7. Serverless Architecture

P O Z O S T A Ł E T E R M I N Y :

Zbieranie wymagań i współpraca z klientem

Łódź 14-16.11.2018 2100,00 PLN

Wzorce projektowe i refaktoryzacja do wzorców

Łódź 17-19.12.2018 2100,00 PLN

CENY NETTO



Rysunek 3. Diagram fragmentu systemu z uwzględnieniem podejścia Ports and Adapters

SOLID – zbiór reguł tworzenia kodu w paradygmacie obiektowym, nazwa jest akronimem od pierwszych liter składowych zasad:

- ▶ Single responsibility principle – klasa powinna mieć tylko jedną odpowiedzialność.
- ▶ Open/closed principle – klasy powinny być otwarte na rozszerzanie, a zamknięte na modyfikację.
- ▶ Liskov substitution principle – obiekty powinny być podmienialne przez podtypy bez wpływu na poprawność działania kodu.
- ▶ Interface segregation principle – różni klienci powinni mieć dedykowane interfejsy zamiast jednego generycznego.
- ▶ Dependency inversion principle – kod powinien zależeć od abstakcji, a nie konkretnych bytów.

Architektura Ports and Adapters może być wykorzystana w dowolnym kontekście. Jeśli zastosujemy ją w połączeniu z Domain-Driven Design, przykładowym portem wejściowym może być interfejs Application Service, zaś portem wyjściowym obiekt typu Repository.

Na powyższym diagramie (Rysunek 3) znajduje się przykład fragmentu aplikacji do wypożyczania książek, co do którego zastosowano podejście DDD przeksztalcone do architektury Ports and Adapters. Główna różnica techniczna polega na tym, że pojawiły się dodatkowe interfejsy odgradzające domenę od reszty świata. Ta niepozorna zmiana niesie jednak ogromne konsekwencje z punktu widzenia osłabienia zależności od elementów infrastruktury i znacząco ułatwia testowanie kodu na różnych poziomach.

Clean Architecture

Samo podejście doczekało się różnych wariacji, z czego dwie najbardziej znane to Onion Architecture, próbujący osadzić Port and Adapters w specyfice Domain-Driven Design, oraz Clean Architecture autorstwa Uncle Boba², który w sposób bezpośredni czerpie z głównej idei stojącej za Ports and Adapters, dodając pewne szczegóły realizacyjne. Pierwszym z nich jest to, iż każdy przypadek użycia (zwany tutaj Interactorem) jest reprezentowany przez osobną klasę (wzorzec Command), który przyjmuje na wejściu obiekt żądania (Request) oraz generuje odpowiedź w postaci obiektu odpowiedzi (Response). Interfejsy reprezentujące porty nazywane są Gateway, zaś obiekty dziedzinowe to Entity. Ideę rozwiązania można zobaczyć na Rysunku 4.

O ile w oryginalnej wersji Ports and Adapters połączonej z Domain-Driven Design przypadek użycia to po prostu metoda w ApplicationService z parametrami oraz zwracanym wynikiem, o tyle tutaj jest to osobna klasa. Model zaproponowany w Clean Architecture wprowadza więcej abstrakcyjności, a co za tym idzie – jeszcze więcej elastyczności z punktu widzenia rozwoju aplikacji, jej testowania oraz dalszego rozpraszania. W Clean Architecture również jest zaproponowane pewne rozwiązanie po stronie widoku: Presenter,

ViewModel oraz View. Zainteresowanych zachęcam do zatrudnienia do literatury.

Zalety i wady

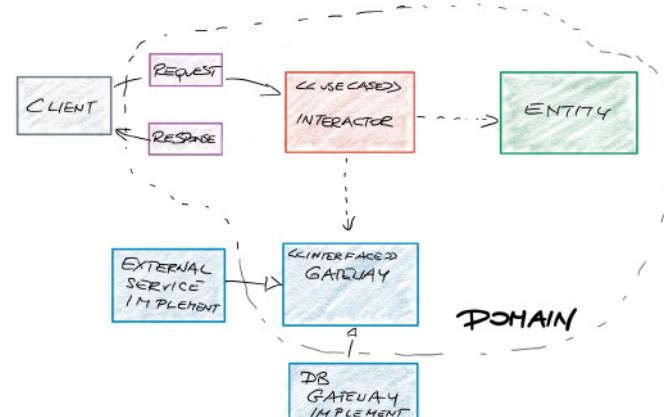
Wracając do bazowej koncepcji Ports and Adapters, to, co otrzymujemy dzięki temu podejściu, to czysta implementacja części domenowej. Implementacje konkretnych portów możemy dowolnie podmieniać, w tym również stosować mocki i fake'i, do wcześniejszej weryfikacji sposobu działania systemu. Co za tym idzie, jest to architektura, która daje ogromne możliwości, jeśli chodzi o testowalność. Możemy na przykład testować widok, a założyć resztę aplikacji. Możemy testować część domenową (przypadki użycia) w oderwaniu od technologii albo z uwzględnieniem tylko wybranej części infrastruktury. Możemy testować implementacje portów drugorzędnych (secondary) w oderwaniu od reszty. Możliwości pod tym względem są ogromne.

Główna wada tego podejścia jest jego duża abstrakcyjność, a co za tym idzie – złożoność, praktycznie na każdym styku mamy interfejsy, co powoduje duży narzut przy jakichkolwiek zmianach w komunikacji (będzie wymagana zmiana wielu klas). Jest to architektura zdycydowana dla sprawnych i kompetentnych zespołów oraz użyteczna w systemach, które będą rozwijane wiele lat, gdzie jest duży nacisk na testowanie, a narzut na zbudowanie warstw abstrakcji z pewnością się zwróci.

Wprowadzenie tego typu architektury również może się odbywać ewolucyjnie jako kolejny krok, po dobrze wydzielonych Bounded Contextach i warstwach. Na przykład można zastosować to podejście tylko w wybranych Bounded Contextach, gdzie wymagana jest duża testowalność.

MICROSERVICES

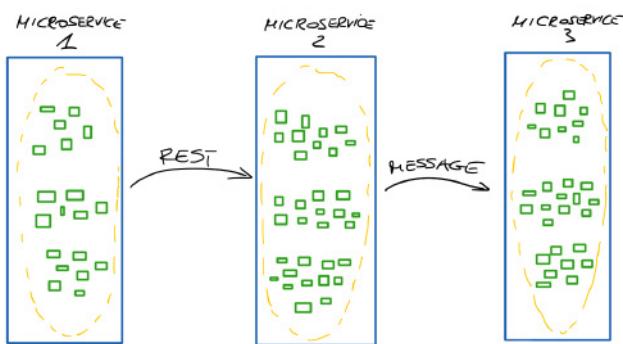
Nazwa „microservices” zaczęła pojawiać się wraz z intensywnym rozwojem systemów działających na dużą skalę: Twitter, Facebook,



Rysunek 4. Architektura Clean Architecture

2. https://en.wikipedia.org/wiki/Robert_C._Martin

Netflix, Uber. W sytuacji kiedy skala zaczyna odgrywać kluczowe znaczenie, dużo trudniejszymi problemami do rozwiązań są kwestie związane z obsługą ogromnej ilości żądań czy efektywnym pobieraniem danych, niż implementacja poszczególnych funkcjonalności. Klasyczne sposoby skalowania serwerów czy baz danych są w tym przypadku bardzo ograniczone i trzeba w pewnym momencie sięgnąć po metody związane z tworzeniem systemów rozproszonych. Bo system oparty o architekturę microservices to nic innego jak system rozproszony, w którym (posługując się nazewnictwem zaczerpniętym z DDD) Bounded Contexty są fizycznie niezależnymi aplikacjami, często wdrażanymi na osobnych serwerach, a komunikacja między nimi odbywa się z użyciem mechanizmów sieciowych. Jeszcze innymi słowy (nie odnosząc się do DDD) jest to system złożony z niezależnych aplikacji, które grupują zbliżone biznesowo funkcjonalności (np. koszyk, obsługa zamówień, obsługa użytkowników). Ideowo można by to przedstawić jak na Rysunku 5.



Rysunek 5. Idea podziału systemu na microservices

System w pierwszej kolejności jest podzielony funkcjonalnie, a nie technicznie (czyli podobnie jak w DDD i w przeciwieństwie do klasycznej architektury warstwowej) na części zwane usługami. Każda usługa w środku może być zrealizowana z użyciem dowolnej architektury i technologii.

Główne wyróżniki tego typu usługi są następujące:

- » pojedyncza odpowiedzialność oparta o funkcje biznesowe,
- » możliwość autonomicznego wdrażania niezależnie od pozostałych usług,
- » artefakt wdrożeniowy zawiera wszystkie zależności,
- » oparta o lekkie technologie (np. Spring Boot, ASP.NET Core Web API).

Duże kontrowersje budzi kwestia rozmiaru. Istnieją różne szkoły i różne preferencje. Nie zawsze microservice musi być „mikro” i zawierać kilkaset wierszy (co jest propagowane przez niektórych guru tego tematu). Złożona biznesowa usługa może mieć rozmiar kilkunastu lub kilkudziesięciu tysięcy wierszy i jeśli dobrze skonstruowana, szczególnie w myśl zasad zaczerpniętych z DDD, też będzie mikrosługą. Dużo istotniejsza jest myśl tworzenia większego systemu jako zbioru mniejszych, autonomicznych aplikacji.

Najczęściej pojedynczy zespół zajmuje się jedną lub kilkoma usługami (ale jedna usługa jest tylko i wyłącznie pod władcą jednego zespołu). Istnieje duża swoboda co do wyboru architektury i technologii, w której będzie zaimplementowana.

Należy jednak pamiętać, że nie ma nic za darmo. Podejście oparte o microservices powoduje, że poszczególne usługi są prostsze. Jednak złożoność systemu nie znika, a przesuwa się w inne miejsce, powodując:

- » większą złożoność komunikacji pomiędzy usługami,
- » bardziej złożoną infrastrukturę.

Komunikacja między usługami

Komunikacja pomiędzy usługami nie odbywa się w tym przypadku poprzez wywołanie funkcji w innym module, a przez wywołanie zdalne. Obecnie najczęściej ta komunikacja jest realizowana poprzez komunikację typu REST z użyciem protokołu HTTP oraz poprzez systemy kolejkowe (messaging) i wymianę komunikatów. W przypadku tego podejścia musimy radzić sobie z sytuacjami, kiedy usługa może być niedostępna, zablokowana, obciążona lub sieć może zawieść. Wtedy pojawia się potrzeba użycia różnego rodzaju technik, aby radzić sobie z tego typu problemami. Stosuje się wiele wzorców wspierających tworzenie aplikacji rozproszonych, w tym m.in.:

- » Retry – mechanizm umożliwiający ponownie wywołanie operacji, jeśli z jakiegoś powodu nie było to możliwe,
- » Circuit Breaker – mechanizm, który jeśli wykryje powtarzający się problem z wywołaniem usługi, powoduje odcięcie wywołań aż do czasu prawidłowego działania usługi; może też zwracać jakiś uproszczony lub zaciagnięty z cache wynik,
- » Gateway API – brama do systemu, balansująca obciążenia, pewnego rodzaju proxy dla klienta, który chce wywoływać konkretne usługi.

Infrastruktura

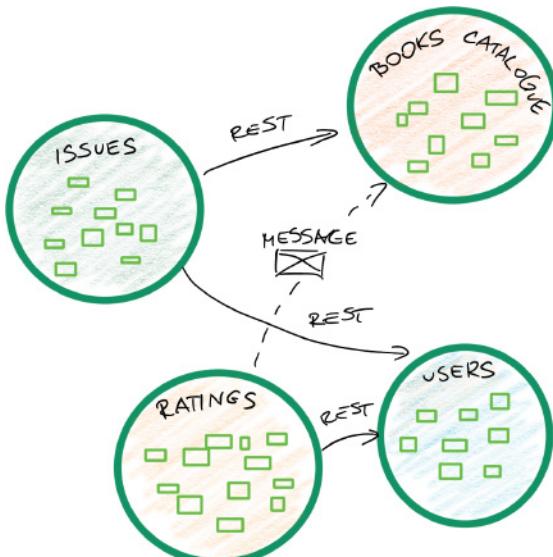
Infrastruktura w podejściu microservices znacząco się komplikuje:

- » należy zaimplementować mechanizm integrujący logi z różnych aplikacji, aby móc wnioskować o tym, co się dzieje w aplikacji,
- » potrzebny jest rejestr usług, aby móc się do nich odwoływać, używając logicznych nazw, a nie konkretnych adresów sieciowych,
- » niezbędne będą mechanizmy pozwalające w czasie rzeczywistym monitorować stan działających aplikacji, tak aby można było szybko reagować w przypadku awarii jednej z nich,
- » w systemach o dużym i zmiennym obciążeniu kluczowe będą mechanizmy autoskalowania, które dadzą możliwość modyfikacji działających instancji danej usługi w zależności od bieżącego obciążenia.

Przykład

W omawianym przez nas przykładzie systemu do wypożyczania książek poszczególne Bounded Contexty stają się osobnymi aplikacjami, które udostępniają API w stylu REST. Ponieważ wcześniej stosowaliśmy zasady DDD podczas tworzenia aplikacji, wyodrębnienie niezależnych usług jest naturalnym krokiem. Metody udostępniane przez ApplicationService zostaną udostępnione na zewnątrz jako API (w przypadku technologii opartych o Javę może to być aplikacja Spring Boot, w przypadku technologii .NET może to być aplikacja ASP.NET Core). Jeśli chcemy skorzystać z innej usługi, będziemy potrzebować klienta, który wywoła owo API. Ponieważ zastosowaliśmy architekturę Ports and Adapters, z punktu widzenia aplikacji wykorzystanie zewnętrznej usługi jest przezroczyste – to tylko wywołanie metody z interfejsu, którego implementacją jest wspomniany wcześniej klient.

Ogólny schemat rozwiązania znajduje się na Rysunku 6, a przykładowa struktura jednego z Bounded Contextów może wyglądać jak na Rysunku 7. Jest ona bardzo podobna do tej z Rysunku 3,



Rysunek 6. Przykład komunikacji między usługami w architekturze microservices

gdyż wewnętrzna konstrukcja jest bardzo zbliżona. Różnica polega na tym, że ponieważ w podejściu microservices każda usługa to niezależny byt, toteż klient (którym może być na przykład interfejs użytkownika albo inna usługa) nie wywołuje operacji w formie lokalnego wywołania metody, tylko w formie wywołania zdalnego. Dlatego na Rysunku 7 adapterem jest API usługi w formacie REST.

Główna różnica w odniesieniu do diagramu z Rysunku 3 polega na tym, że tutaj adapterem dla operacji wejściowych (primary port) jest usługa REST API. Jak się dowiemy bardziej szczegółowo w kolejnej części artykułu, w przypadku komunikacji z użyciem komunikatów alternatywnie adapterem dla operacji wejściowych może być obiekt obsługujący komunikaty.

Zalety i wady

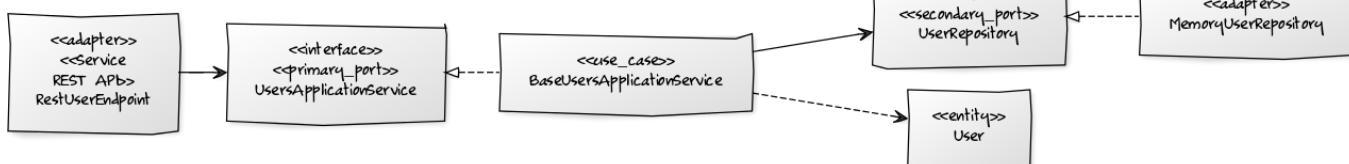
Główną zaletą tego podejścia jest możliwość niemal nieograniczonego skalowania aplikacji zarówno pod kątem ilości obsługiwanych żądań, jak i danych. Ponadto same usługi są prostsze w porównaniu do konstrukcji monolitycznych. Zespoły, które mają pod opieką całą usługę, mogą działać bardziej autonomicznie i podejmować decyzje techniczne w dużym stopniu niezależnie od reszty organizacji.

Wadą jest natomiast dużo większa złożoność całego systemu, która wymaga dużego nakładu sił na zaprojektowanie i zimplementowanie komunikacji między usługami oraz rozwój i utrzymanie dość złożonej infrastruktury (rejestry usług, logowanie, monitoring, autoskalowanie, zarządzanie całością).

Dlatego architektura tego typu powinna być wprowadzana w toku ewolucji systemu, na początku bardziej jako wizja systemu. Po pierwsze, trudno jest dobrze zaprojektować taki system, gdy nie wiemy jeszcze dobrze, w jakim kierunku usługi będą się rozwijać. Po drugie, poszczególne usługi oddzielają granice fizyczne (np. osobne serwery, sieć), więc późniejsze zmiany w designie tego typu systemu będą bardzo kosztowne. Dlatego dojście do zbalansowanej architektury opartej o microservices powinno odbywać się krokowo. Na przykład możemy zacząć od wydzielania Bounded Contexts na poziomie logicznym, a z czasem, kiedy będziemy mieć wyraźne przesłanki wynikające z potrzeb wydajnościowych lub związane z niezależnością wdrażania, wyodrębnić je jako autonomiczną usługę.

PODSUMOWANIE

W tym artykule zajęliśmy się kolejnymi dwoma koncepcjami architektonicznymi: Ports and Adapters oraz microservices. Pierwsza z nich pomaga oddzielić wyraźnie część domenową aplikacji od infrastruktury, upraszczając konstrukcję domeny oraz znacząco ułatwiając testowanie. Druga natomiast to popularny w ostatnich latach model architektury rozproszonej, który pozwala na łatwe skalowanie części systemów oraz ich niezależne wdrażanie. Obydwa podejścia należy wprowadzić w życie w sposób ewolucyjny jako kolejne etapy rozwoju systemu, co w szczególności nie powinno być problemem, kiedy wcześniej wprowadziliśmy bloki budujące i Bounded Contexts rodem z DDD.



Rysunek 7. Fragment diagramu klas dla usługi Users



MARIUSZ SIERACZKIEWICZ

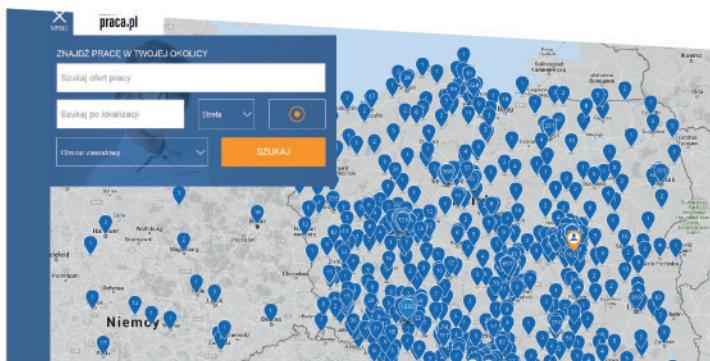
m.sieraczkiewicz@bnsit.pl

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”.



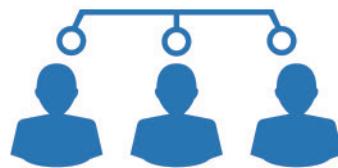
Chcesz dobrze zarobić?

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

Wdrożenie Agile z perspektywy managera

Od najmłodszych lat byłem uczyony, że aby osiągać sukcesy, muszę być indywidualnie najlepszy. Dotyczyło to przede wszystkim szkoły, gry na instrumencie muzycznym czy czegokolwiek, gdzie pojawiał się pierwiastek oceny. Mało kogo interesowało to, czy potrafię współpracować z innymi, czy umiem szukać kompromisów, czy potrafię inspirować albo motywować innych do działania. Nikt specjalnie nie zwracał na to uwagi. Nie jesteśmy w szczególny sposób uczeni, jak współpracować, jak sobie pomagać. Nie jesteśmy z tego tytułu oceniani. To raczej wychodzi samo, jako wypadkowa naszych miękkich cech, wychowania w domu i okoliczności, w jakich akurat się znajdujemy.

ZA SUKCESAMI CZEŚCIEJ STOJĄ JEDNAK ZESPOŁY

Jednak jak pokazuje rzeczywistość, za sukcesami częściej stoją zespoły, niż indywidualności. Lubię sport, więc może te przykłady do mnie najlepiej przemawiają.

Ostatnimi czasami tylko dwaj piłkarze zdobywali tytuł Najlepszego Piłkarza Globu – Ronaldo i Messi. Indywidualnie na przemian osiągający liczne sukcesy w różnego rodzaju konkursach, plebiscytach, tytuły najlepszych strzelców w rozgrywkach ligowych, międzynarodowych. Nie osiągnęliby ich jednak, gdyby nie ich koledzy z Realu Madryt, Barcelony, reprezentacji Portugalii lub Argentyny. Gdyby nie mieli odpowiedniego wsparcia w sztabie treneriskim, medycznym, PRowym czy też marketingowym, nie mieliby tylu trofeów w swoich gablotach.

Indywidualnie wybitni, ale mogący pokazać pełnię swoich możliwości jedynie w wybitnym otoczeniu. To recepta na sukces dzisiaj.

Zanim Agile zawitał do mBanku...

Sport sportem, a ja przecież nie o nim chcę pisać. To, o czym chcę Wam opowiedzieć, to o moich doświadczeniach jako managera, który wdrożył Agile w hurtowni danych w mBanku.

Jak to wyglądało kiedyś, zanim zaczęliśmy podchodzić do rozwoju oprogramowania w zwinny sposób?

Nie będzie odkryciem, jeśli napiszę, że po prostu radziliśmy sobie, stosując głównie Waterfall lub inne mniej uświadomione agilowe hybrydy waterfallowo-kanbanowe.

JAK TO WYGLĄDAŁO KIEDYSZ PERSPEKTYWY PRACOWNIKA...

Z perspektywy developera proces wyglądał często tak, że praca trafiała do pracownika przez koordynatora biznesowego lub managera. Była to praca głównie samodzielna, gdzie występowało mało interakcji z kolegami developerami lub z tzw. biznesem. Pracownicy nie byli specjalnie niepokojeni spotkaniami, chyba że z inicjatywy managerów. Oczekiwane było dowiezienie tematu w czasie, w budżecie i w uzgodnionym zakresie. Pojawiała się specjalizacja, na przykład jeśli temat dotyczy „zwrotu opłat z kart”, to wiadomo, że trzeba z tym iść do Krzyśka. Jeśli Krzyśka nie ma

to musimy poczekać, aż wróci z urlopu. Awaria? Czy Krzysiek wziął laptopa na urlop?

JAK TO WYGLĄDAŁO KIEDYSZ PERSPEKTYWY MANAGERA...

Z perspektywy managera ta praca głównie skupiała się na rozdzielaniu zadań, odbiorze zadań, monitorowaniu pracy, ustalaniu z Biznesem zadań do realizacji. Manager miał zamiast zespołu grupę indywidualistów, łączoną od czasu do czasu w zespoły na potrzeby projektów, które były rozwijane zaraz po zakończeniu projektu.

Z uwagi na tzw. bieżączkę występował permanentny brak czasu na wizję i strategię zespołu, cykliczne rozmowy 1 na 1, systematyczne spotkania z całym zespołem, stymulowanie rozwoju pracowników. Ale sobie radziliśmy. Produkowaliśmy soft, biznes się kręcił.

A JAK AGILE POJAWIŁ SIĘ W ORGANIZACJI?

Pewnego dnia do mBanku zawitał nowy szef informatyki, który z sukcesem wdrażał zwinne podejście do tworzenia oprogramowania w kilku instytucjach w przeszłości. Do tej pory nikt go nie przekonał, że Agile „nie działa”.

CO JA WIEDZIAŁEM O AGILE

Co ja wiedziałem o edżaju? Tyle co przeczytałem na Wikipedii, w książce wydawnictwa Helion – „Zarządzanie projektami IT. Przewodnik po metodach”, z opowieści znajomych pracujących poza mBankiem. Zrobiłem również kurs Agile Project Management, zakończony uzyskaniem certyfikatu na poziomie Foundation. Czyli miałem sporo informacji, ale mało praktycznych doświadczeń.

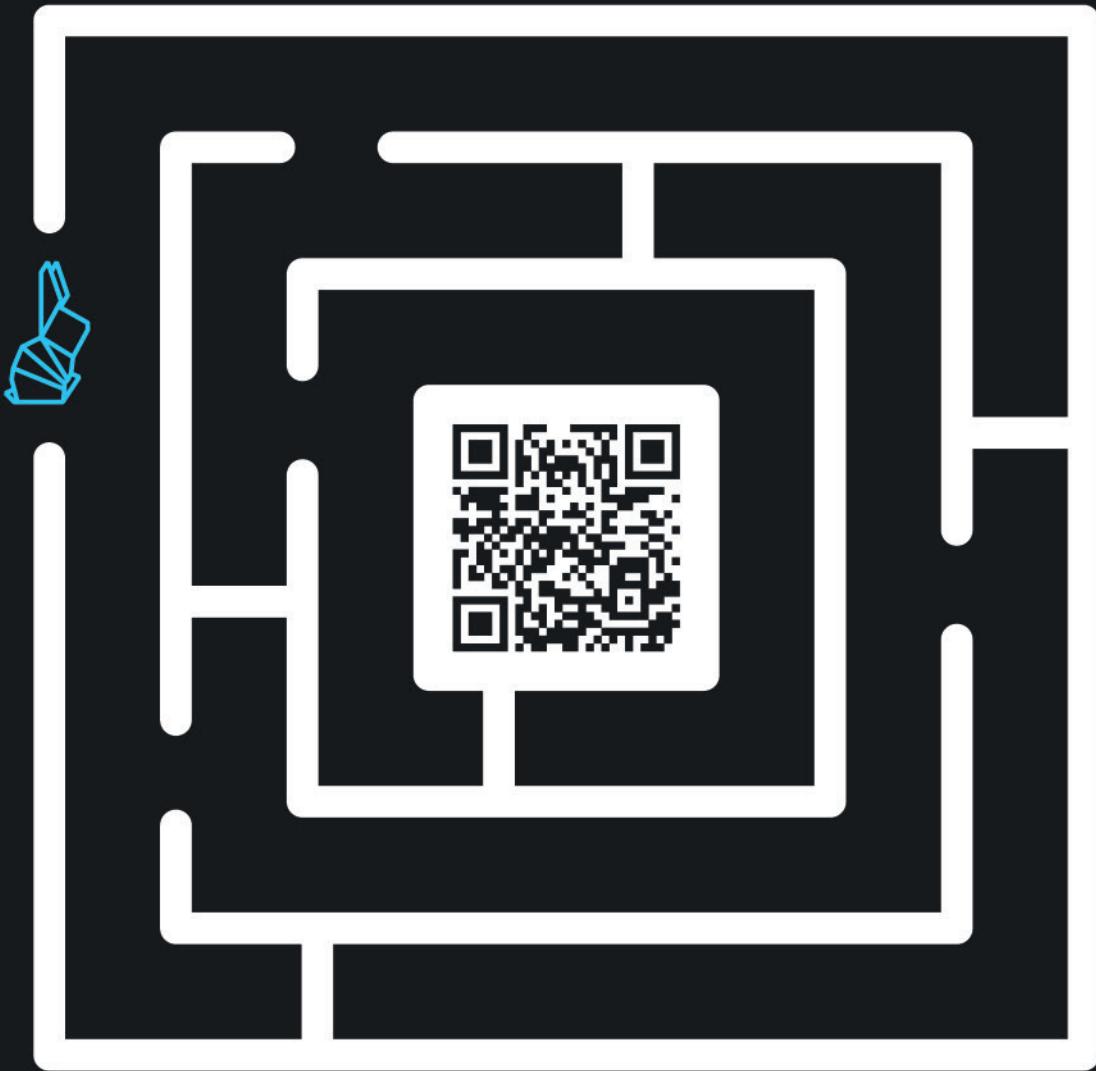
Miałem swoje obawy i zastanawiałem się nad tym, czy ta zmiana jest mnie i mojemu zespołowi potrzebna. Przecież sobie radzimy.

OBAWY

Obawiałem się utraty wpływu na zespół

Bieżące rozdzielanie zadań, ich odbieranie i monitorowanie przebiegu prac, choć zabierało sporo czasu, to jednak dawało poczucie sprawstwa. Miałem kontrolę nad tym, co robią moi pracow-

follow the blue rabbit...



Simple?
More riddles is waiting for you!

coming soon

nicy, jak to robią, na kiedy itd. W agilowej rzeczywistości pojawia się ktoś z zewnątrz, zwany Product Ownerem, który to zamiast mnie definiuje priorytety, oczekiwania i terminy. De facto zarządza pracą moich ludzi, a ja się muszę temu przyglądać z boku. Nie przeszkadzać, tylko ich wspierać i grzecznie prosić, jak coś od nich chce.

Obawiałem się „marnotrawstwa czasu” (czytaj „Man Day’i”) na spotkaniach

W trakcie dwutygodniowych sprintów scrumowych zespół powinien się spotykać codziennie na tzw. daily (do 15 minut), na refinementach (raz/dwa razy w tygodniu po 1-2 h), na review (raz na dwa tygodnie na 1 lub 2 h), na retro (raz na dwa tygodnie na 1-2 h). Sumując te wszystkie godziny w trakcie dwutygodniowego sprintu, można stwierdzić, że developer ma wyrwane z życiorysu na spotkania (bez których do tej pory sprawnie funkcjonował) od 5,5 godziny do 8,5 godzin w trakcie dwóch tygodni. Co przy pięcioosobowym zespole oznacza od 3,5 do 5,5 MD developerskich mniej w okresie 2 tygodni. To oznacza, że zrobimy kilka zadań mniej i nie zrealizujemy iluś projektów w ciągu roku. Biznes nie będzie zadowolony. Scrum nie jest tani.

Obawiałem się Agile Coacha

Co to za dziwna rola – Agile Coach? Na pokładzie w organizacji pojawiła się grupa osób mających nieść kaganek agile’owej oświaty. Mieli stymulować rozwój zwanego rozwoju oprogramowania w zespołach, usuwać przeszkode, rozmawiać z zespołami, z ich managerami, zachęcać do innego myślenia o rozwijaniu softu i zniechęcać do starych metod. Zaczęli rozmawiać z moimi ludźmi, wypytywać ich o różne tematy przy kawie i bez kawy. Siedzieli razem z nimi na open space i obserwowali, co się dzieje. O czym oni rozmawiają z moimi ludźmi? Albo gorzej, co i komu donoszą? Przemykali ze spotkania na spotkanie w zespołach agilowych jak tajemniczy Don Pedro. O co tu chodzi?

Obawiałem się zmian

Po co coś zmieniać, skoro działa? Nie widziałem potrzeby wprowadzania Scruma, ponieważ mój zespół osiągał stawiane przed nim cele.

Franciszek Smuda jako trener polskiej reprezentacji podczas meczu z Grecją na Euro 2012 nie przeprowadził żadnej zmiany osobowej. Na pytanie dziennikarzy, czemu nie wprowadził „świeżej krwi” na boisko, stwierdził, że „nie chciał niczego zepsuć”. Lepsze jest wrogiem dobrego...

Obawiałem się, że nie będę w stanie wydzielić produktu

W zespole miałem indywidualistów, którzy przez lata pracy wytworzyli tony softu – raporty, zasilania systemów, aplikacje, zasilania hurtowni, systemy informacyjne, data marty itp.

Mając wiedzę o tym, że Scrum skupia się w swej istocie na iteracyjnym rozwoju produktu, zastanawiałem się, jak w tym otoczeniu znaleźć jeden produkt, na którym powinny skupić się pojedyncze zespoły? My tych systemów i systemów mieliśmy kilkadziesiąt sztuk. Nie ma szansy na to, by dać zespołowi komfort zajmowania się jednym dużym tematem. Możemy jedynie próbować pogrupować systemy i zajmować się kategoriami tematów. Czy warto zatem wokół takich zbiorów systemów próbować tworzyć zespoły?

Obawiałem się geografii

Może lepiej nie walczyć z „geografią”? Jak stworzyć zespoły z pracowników pracujących w Łodzi i w Warszawie? Oczywiście jest to możliwe. Najlepiej niech ci z Łodzi pracują z łodzianami, a ci z Warszawy niech tworzą zespoły z warszawiakami. A gdy tak nie

można? Bo może w Excelu i by się dało, ale w rzeczywistości jest to trudne.

Obawiałem się braku doświadczonych Scrum Masterów na pokładzie

Załóżmy, że ten Scrum wystartuje. Założymy, że wydzielony zostanie produkt, osobowo się to uda poskładać w zespoły, a ja pogodzę się z utratą pełni kontroli nad ludźmi i ich zadaniami. Skąd jednak mam wziąć Scrum Masterów dla każdego z zespołów? Skąd wziąć ludzi, którzy będą pilnowali stosowania reguł Scrum'a w poszczególnych zespołach? Etaty zablokowane. Nikogo nowego nie zatrudnię.

Z drugiej strony miałem pewne nadzieję.

Miałem nadzieję na więcej oddechu

Chciałem, żeby tzw. bieżączka pochłaniała mnie mojego czasu. Zajmowanie się rozdzieleniem zadań, ustalaniem priorytetów, weryfikowanie postępów prac, reagowanie na eskalacje... to był chleb powszedni. Zadanie ważne, doniosłe i potrzebne. Pytanie, czy musiałe to robić od początku do końca ja? Nie było nigdy czasu na pracę nad wizją zespołu, rozwojem pracowników itp. Zawsze robota była ważniejsza, tak jakby rozwój pracownika i zespołu to nie była praca dla managera. No jest, ale zaraz po „prawdziwej robocie”.

Miałem nadzieję na wprowadzenie zastępowałości

Gdy pracownik jest na urlopie, to w 99% można stwierdzić, że coś się wydarzy. Nie wszystkie role w zespole miały obsadzone podwójnie. To zwiększało ryzyko i irytovalo, gdy pojawiały się awarie. Trzeba było sobie jakoś radzić i wykonywać telefony do przyjaciela...na wakacjach. Miałem nadzieję, że zespołoowość wprowadzi wymienność pozycji i ludzie będą skutecznie mogli zastępować się w pracy.

Miałem nadzieję, że zbudujemy prawdziwe zespoły

Jak już wspominałem, kierowałem zespołem indywidualistów. Nadarżała się dobra okazja do tego, żeby coś zmienić. Żeby ludzie zaczęli ze sobą współpracować, przekazywać wiedzę, uczyć się od siebie nawzajem, dostrzegać lepsze rozwiązania u swoich kolegów. Dzięki temu moglibyśmy produkować lepszy soft, mniej awaryjny, bardziej optymalnie napisany, za który odpowiedzialność bierze grupa, a nie jednostka.

Miałem nadzieję, że Biznes weźmie na siebie ustalanie priorytetów

Przyjemnie jest, gdy klienci wewnętrzni dzwonią i proszą o delegowanie developera do realizacji ich zadania. Czujesz się jako manager ważny i potrzebny. Mniej przyjemnie się robi, gdy proszą, by ten developer zrobił to o połowę szybciej niż w wycenie, którą przekazał. Jeszcze mniej przyjemnie robi się, gdy developerów brak, a interesariusze zaczynają ci grozić karami tudzież znajomością z prezesem. To oczywiście korporacyjna szarość dnia i nie ma w tym nic niezwykłego, bo popyt na MD zazwyczaj przewyższa zasobową podaż. Z drugiej strony kuszące jest rzucenie tego w ... diabły i pojechanie w agilowe Bieszczady – Biznesowy Product Owner, który rozumie IT i zna swój produkt i biznes, to takie agilowe marzenie.

Obawy czy nadzieje?

Rozstrzygnięcie tego, czy moje obawy czy też nadzieje wezmą górę, trochę trwały. Wiedziałem jednak, że trzeba zaryzykować, po-

święcić trochę czasu na przygotowania, uzyskać kilka odpowiedzi, a reszta pójdzie do przodu.

Od czego zacząłem?

Od spotkań z praktykami, z ludźmi mądrzejszymi ode mnie, którzy kiedyś pracowali w Agile. Którzy byli w stanie mi pokazać, jak może wyglądać ten proces przejścia od Waterfall'a do Agile'a. Zadawałem pytania, wątpliwości powoli się rozwiewały.

W kolejnym kroku wybrałem obszar, który spróbujemy „uagliować” wspólnie z nowym zespołem i Agile Cochem. Padło na obszar raportowy. Odbył się kick-off meeting i się zaczęło. Najpierw wystartował jeden zespół, a następnie co kilka miesięcy startowały kolejne zespoły w innych obszarach hurtowni danych. W sumie uruchomiliśmy 4 zespoły.

Jeden z zespołów przeszedł drogę od Scrum'a do Kanbana. Po-zostałe trzy pracowały w Scrumie cały czas. Każdy z nich miał swoją specyfikę, swoje problemy, swoje atuty. Jeden się rozpadł, po to by później wrócić w innej konfiguracji i być jednym z lepszych zespołów na pokładzie.

CO DLA MNIE BYŁO NAJTRUDNIEJSZE?

Pieprzony Scrum

Kiedyś kolega, Product Owner, jednego z moich zespołów powiedział do mnie po retrospektywie zespołu:

» Tomek, pieprzony Scrum. Kiedyś bym po prostu kazał, a teraz muszę prosić.

W Scrumie jeśli chcesz zbudować solidne podstawy do tego, by zespół się usamodzielniał, nie możesz kazać im coś zrobić. Musisz prosić i argumentować. To oczywiście zabiera czas, z drugiej jednak strony uczy ludzi odpowiedzialności. Było to dla mnie sporym wyzwaniem, kiedy musiałem prosić Product Ownera, człowieka z biznesu, żeby zechciał pozwolić moim ludziom zrealizować dla mnie ważne zadanie...co łączy się z kolejnym punktem...

Oddanie pola Product Ownerowi (PO)

Czyli pozwolenie PO, by to on ustalał priorytety, rozmawiał z interesariuszami, był twarzą mojego zespołu. Ja schodziłem na dalszy plan i miałem inne zadania (rozwój i integracja pracowników, stymulowanie rozwoju zespołów, wizja i strategia zespołu, prowadzenie dużych projektów, budżetowanie itp.).

Połączenie roli managera i PO

Z teorii i z doświadczenia już wiem, że zdrowiej dla psychiki i pracy zespołowej jest, gdy manager i PO są w dwóch, a nie w jednej osobie.

Jednym z niestandardowych zadań, jakie musiałem pełnić, była rolą PO w moim zespole. Tak się zdarzyło, że PO musiał udać się na długotrwałe zwolnienie lekarskie (kilka miesięcy) i nie znajdując alternatywy, zgodziłem się go zastąpić. Przez pierwsze tygodnie było ok, później jednak zaczęły wkradać się w moją pracę niewłaściwe zachowania – mikro management, wywieranie niepotrzebnej presji na zespół itp. W efekcie moi podwładni poprosili mnie, żeby nie przychodził na daily, a kilka retro zrobili bezem mnie. To nie był łatwy czas dla nich i dla mnie.

Scrum Master potrzebuje czasu

To było jak objawienie. Okazuje się, że człowiek, który pilnuje reguł Scrum'a, potrzebuje połowy swojego czasu, czyli około 5 MD

w dwutygodniowym sprincie na pracę scrum masterską. Ponieważ w naszych zespołach nie było zawodowych Scrum Masterów – tę rolę pełnili developerzy – oznaczało to, że w zespole było o pół człowieka mniej „do pracy”. Najbardziej było to dla mnie bolesne, gdy pełniłem rolę jednocześnie PO i managera, który musiał zmagąć się z presją interesariuszy na drogocenne MD.

EFEKTY?

Trudności, obawy, nadzieję....czyli jak to wszystko się skończyło i jakie były efekty tych zmian, jakie zaszły?

Na scenie w hurtowni danych pojawiły się 4 zespoły, z trzema Scrum Masterami i jednym Kanban Masterem. Utraciłem trochę władzy, bo to PO rozdzielały zadania. Z drugiej strony odzyskałem czas na inne aktywności (rozwój i integrację pracowników, stymulowanie rozwoju zespołów, prowadzenie dużych projektów, kształcenie następców).

Moja obawa o to, że czas przeznaczany na spotkania, zwłaszcza w Scrumie, zabierze MD developerskie, w pewien sposób się ziszcza. Z drugiej strony organizacja uzyskała coś, co trudno skwantyfikować – rozpropagowanie wiedzy o systemach, bardziej przemyślane rozwiązania informatyczne – bo dyskutowane w grupie, a błędy stanowią materiał do nauki, a nie do chowania pod dywan. Jestem przekonany, że ten czas, który zainwestowaliśmy, odebraliśmy już z nawiązką we wdrożonych projektach i ich utrzymaniu. Pojawiła się zastępcość w zespołach i ryzyko operacyjne w zakresie wąskich gardeł w zasadzie przestało istnieć.

Agile Coache, okazało się, że nie donosiły, tylko byli przemiłymi ludźmi, którzy chcieli nam pomóc...bo nasz sukces był ich sukcesem. Zmian nie należało się obawiać, bo choć powodowały trochę zamieszania, to jednak wprowadziły nową jakość i dodatkową energię do pracy, a do starych metod nie chce się już wracać.

Nie wszystko udało się zrealizować tak, jak byśmy tego chcieli. Produktu per zespół nie udało się nam do końca wydzielić i nadal nad tym pracujemy.

Zespoły łódzko-warszawskie funkcjonują. Idealnie, gdyby były zlokalizowane w jednym miejscu, ale jak się nie ma, co się lubi, to się lubi, co się ma. Skype, telefon, samochód i PKP załatwiają resztę.

Po jakimś czasie (około 2,5 roku) pojawiły się następcy, managerowie, którzy chcieli wprowadzić dodatkowe zmiany, pozwolić nam interesarzom z biznesu pogrupować produkty, dać szansę developerom, żeby zmienili zespół, w którym pracują. Zmiany nadeżą zmiany...ale to już zupełnie inną historią.

Nie trzeba się zmieniać, przetrwanie nie jest obowiązkowe.

William Edwards Deming

TOMASZ DUTKOWSKI

tomasz.dutkowski@mbank.pl

Doświadczony manager zarządzający Hurtownią Danych mBanku obszaru detalicznego i korpo. Posiada 14 letnią praktykę w bankowości, zarówno po stronie biznesu, jak i IT. Ma na swoim koncie wiele z sukcesem przeprowadzonych dużych projektów integracyjnych w obszarze hurtowni danych. Lider transformacji zwanego wytwarzania oprogramowania na poziomie zespołów. Entuzjasta innowacji w pracy zespołowej. Prywatnie mają jednej żony i ojciec dwójki dzieci.

Confidence 2018 – KrkAnalytica

Confidence to jedna z największych imprez o tematyce security/infosec w Polsce. Samej konferencji często towarzyszy konkurs w stylu capture-the-flag. W tym roku zadanie z cyklu CTF pojawiło się przed samą konferencją i zostało przygotowane przez firmę SecuRing. Osadzone ono było na bieżących wydarzeniach związanych z Facebookiem i firmą Cambridge Analytica, co sprawiało, że było ono szczególnie ciekawe.



O ZADANIU¹

W zadaniu mamy do czynienia z atakiem hakerskim na firmę Never-Leaks. Firma otrzymała groźby od hakerów, którzy próbują ją szantażować. Atak można powstrzymać, uzyskując dostęp do tajnych plików zlokalizowanych kontenerze AWS S3 „krkanalytica-confidential”.

REKONESANS

Zadanie to miało ciekawą historię osadzoną w realiach ostatnich wydarzeń związanych z Facebookiem i firmą Cambridge Analytica. Stąd jawne odwołanie do afery już w samym tytule zadania.

Na samym początku takich zadań zawsze warto przeprowadzić mały rekonesans nie tylko samej treści zadania, ale także np. jego autorów. Sam opis przynosi nam kilka informacji odnośnie firmy i jej CEO. Przegląd google pod kątem tych fraz daje trochę rezultatów, ale chyba wszystkie to (niezwiązane z zadaniem) faktyczne firmy i profile.

W tym przypadku mieliśmy nie tylko opis zadania, ale także dostępny profil na Twitterze (@KrkAnalytica). Najciekawsze w tym profilu było to, kogo śledził. Nie spodziewałem się tutaj rewelacji z rodzaju tych, jakie odkrył Mike Edgette² (analizując profil KFC), ale zawsze jest to dobry punkt startu.

I w tym przypadku okazało się to nawet opłacalne. Jeden z obserwowanych profili należał do @Rzepski, który to właśnie omawia błędy w konfiguracji oraz problemy z AWS S3. Bardzo ciekawa prezentacja³ na ten temat dała podstawy informacji oraz warsztat (narzędzia) przydatny przy pracy z S3.

AMAZON S3

W opisie samego zadania mamy informację o potrzebie uzyskania dostępu do folderu AWS S3. Z kolei przeglądając prezentację @Rzepski, wiemy, że często może być więcej niż jeden taki folder S3, które dana firma posiada. Może podobnie jest w tym przypadku?

Aby to sprawdzić, możemy posłużyć się jednym z kilku narzędzi. Pierwszym z nich jest Internet Archive. Jest to moduł do Metasploita i za jego pomocą otrzymamy listę otwartych kontenerów AWS S3,

które są w tym archiwum trzymane. Niestety w tym przypadku żaden z kilku tysięcy linków do potencjalnie otwartych kontenerów S3 nie miał powiązania z krkanalytica. A więc nie tedy droga.

Drugim narzędziem, które okazało się skuteczne, jest lazys⁴.

```
> ruby lazys3.rb krkanalytica
```

Jest to prosty skrypt Ruby, który dla danej frazy/firmy sprawdza, czy istnieją otwarte buckety. I faktycznie dla krkanalytica narzędzie zwróci nam informację, iż takowy istnieje pod nazwą krkanalytica-backup.

BACKUP

Uruchomienie narzędzia dało nam informację o kolejnym kroku, który może nas przybliżyć do rozwiązania zadania. Wiedząc, że istnieje dodatkowy kontener z backupem, ściągamy go lokalnie, aby szybciej analizować jego zawartość.

Uruchamiamy polecenie:

```
> aws s3 sync s3://krkanalytica-backup/ ./krkanalytica-backup
```

Archiwum nie jest małe, bo zawiera około 1000 plików. Jest tam sporo potencjalnych nowych wektorów ataku. Mamy więc: kopię wordpressa z ciekawym plikiem users.sql, kopię plików z danymi klientów, archiwum zip z hasłem (prostym!), jak również kopie maili.



Rysunek 1. Zawartość kontenera AWS S3 krkanalytica-backup

Wszystkie dają potencjalną opcję dalszego atakowania, ale strzałem w 10-tkę okazała się analiza tych ostatnich, gdyż w jednym z nich znajdujemy informację, iż został stworzony snapshot z narzędziami testującymi AWS S3. Zlokalizowany jest pod nazwą: snap-0e08c8dc58d80f99e.

I've just started to create an AWS Sec instance, where I put some tools to test our AWS resources. Soon I'll add more tools there. If you'd like to test it by yourself I made a snapshot with ID: snap-0e08c8dc58d80f99e. Please let me know how you'd like my idea :)

W samych mailach też był dodatkowy mylący trop, gdyż jeden z nich był informacją o otrzymaniu tajnej wiadomości z linkiem do odpowiedniego serwisu. Niestety link nie działał, ale mógł spowodować próbę podążania tą ścieżką w celu rozwiązania zadania.

1. <http://confidence-conference.org/krkanalytica.html>

2. <https://mashable.com/2017/11/09/kfc-sents-gift-to-man-who-had-viral-tweet/>

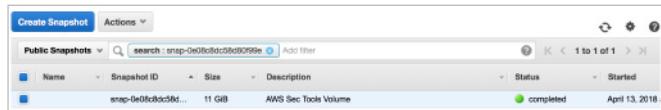
3. <https://medium.com/securing/exploring-25k-aws-s3-buckets-f22ec87c3f2a>

4. <https://github.com/nahamsec/lazys3>

AWS S3 CONSOLE

Znając ID snapshotu, możemy go wyszukać. Aby to zrobić, musimy zalogować się do naszej konsoli AWS i w zakładce *snapshots* odszukać ten z wiadomości e-mail. Faktycznie jest on dostępny. Mając go, możemy na jego bazie utworzyć wolumen, a ten podpisać pod instancję maszyny hostowanej w chmurze Amazon.

Po takiej serii kroków będziemy mogli zalogować się na naszą maszynę ze snapshotem wspomnianym w mailach NeverLeaks.



Rysunek 2. Wspomniany snapshot publicznie dostępny

ANALIZA

Mającinstancję EC2 ze wspomnianym snapshotsem, możemy się na daną maszynę zalogować. Jedyne, co widzimy, to folder AWS Tools, a w nim 3 narzędzia:

- » Scout2,
- » prowler,
- » KrkAnalyticaHarvester.

oraz plik README.txt.

```
[ec2-user@ip-172-31-22-75 AWS Tools]$ ll
total 16
drwxr-xr-x 3 root root 4096 Apr 13 08:45 KrkAnalyticaHarvester
-rw-r--r-- 1 root root 266 Apr 13 09:06 README.txt
drwxr-xr-x 6 root root 4096 Apr 13 09:03 Scout2
drwxr-xr-x 5 root root 4096 Apr 13 09:03 prowler
[ec2-user@ip-172-31-22-75 AWS Tools]$
```

Rysunek 3. Zawartość folderu AWS Tools

Narzędzia te to, tak jak sugerował mail, przydatne aplikacje do analizowania zasobów umieszczanych w S3. Ostatnie z nich sugeruje specjalne przygotowanie na potrzeby zadania, jednak przegląd plików nie zawiera informacji o niezbędnych kluczach autoryzacyjnych. Postęp następuje, gdy wyświetlimy wszystkie pliki w tym ostatnim projekcie, także te ukryte. Wtedy naszym oczom ukaże się folder .git.

GIT HISTORY

Wiedząc, że mamy do czynienia z repozytorium kodu (w tym przypadku git), możemy przejrzeć jego historię, sprawdzając, czy przypadkiem programiście nie omsknął się palec przy pushu i nie zostały do repozytorium przesłane dane wrażliwe.

```
> git log

[ec2-user@ip-172-31-22-75 KrkAnalyticaHarvester]$ git log
commit edd5e9665d13ed75e5d425d85835a0442afcc48 (HEAD -> master)
Author: root <root@ip-172-31-31-192.eu-west-1.compute.internal>
Date:   Fri Apr 13 09:00:42 2018 +0000

    Ooops forgot to remove access keys

commit b15890b7723c12d5b819ac49237872edb8004378
Author: root <root@ip-172-31-31-192.eu-west-1.compute.internal>
Date:   Fri Apr 13 08:49:15 2018 +0000

    init commit
[ec2-user@ip-172-31-22-75 KrkAnalyticaHarvester]$
```

Rysunek 4. Historia projektu KrkAnalyticaHarvester

Okazuje się, że tak. Zobaczmy zatem, co zostało wprowadzone w zmianie oznaczonej identyfikatorem zaczynającym się od edd5e96, wykonując poniższe polecenie, aby uzyskać wiedzę na temat danych autoryzacyjnych.

```
> git log -p -2 edd5e96
```

I wśród informacji o zmianach otrzymujemy wynik:

```
-#keys for reading our bucket's content
-AWS_ACCESS_KEY_ID = 'AKIAJK4WQAVSYATSWLKQ'
-AWS_SECRET_ACCESS_KEY = '1xRV/uiC4knZQzyIZxSS1Q2xN1ZMjo4kn+LnjNif'
##Here you have to put your access keys
+AWS_ACCESS_KEY_ID =
+AWS_SECRET_ACCESS_KEY = ''
```

Boom! Mamy nasze dane.

FINAŁ

Mając dane autoryzacyjne, nie pozostaje nam nic innego jak dostać się do zabezpieczonego repozytorium i pobrać poszukiwany plik: secure_codes.txt.

```
> aws s3 cp s3://krkanalytica-confidential/secret_codes.txt .
> cat secret_codes.txt
```



Rysunek 5. Zawartość pliku secret_codes.txt

QR code? Hmm... czyżby nie koniec zadania? Po sprawdzeniu, gdzie linkuje dany kod QR, jednak okazuje się, że to już finał.

PODSUMOWANIE

Opisywane zadanie obrazowało popularne ostatnimi czasy błędy związane z konfiguracją AWS oraz te dotyczące historii repozytorium kodu. Zadanie posiadało kilka potencjalnych wektorów ataku, które sprawiły, że w trakcie jego rozwiązywania trzeba było wykonać krok w tył i spróbować innego podejścia, co sprawiło, że było ono niezwykle ciekawe.

Podziękowania dla firmy SecuRing za przygotowanie zadania.



PAWEŁ ŁUKASIK

biuro@octal.pl

Programista .NET z 15-letnim doświadczeniem. Ostatnimi czasy bardziej niż programowaniem zaciękwiony tematyką security/infosec. Fan zadań CTF oraz narzędzi radare2. Swoje zmagania z tymi dwoma opisuje na blogu – <http://ctfs.ghost.io>.

Czysta architektura



Długo czekałem na to, aby w ramach Klubu Dobréj Książki opisać *Czystą architekturę*: trzecią książkę z popularnej serii *Clean*, autorstwa znanego i lubianego Wujka Boba (Roberta C. Martina), poprzedzoną dwoma świetnymi tytułami – *Czysty kod* (ang. *Clean Code*) i *Mistrz czystego kodu* (ang. *Clean Coder*). Patrząc z szerszej perspektywy, w *Czystym kodzie* omawiano przede wszystkim idiomy oraz techniki pracy z kodem – na stosunkowo niskim poziomie abstrakcji. *Mistrz czystego kodu* skupiał się na bardziej miękkich aspektach pracy programisty i na całej otocze związanej z programowaniem (praca zespołowa, szacowanie, komunikacja z klientem itp.). W *Czystej architekturze* – stonującą w pewnym sensie domknięcie cyklu – omówiono tematy związane z projektowaniem architektury oprogramowania. W tym kontekście można śmiało stwierdzić, iż poszczególne odcinki serii prowadzą czytelnika przez kolejne etapy kariery programisty-rzemieślnika: poczynając od ucznia (który uczy się możolnej pracy z kodem), przez czeladnika (który posiada już wiele podstawowej i zdobywa doświadczenie, pracując w projektach), aż do mistrza (który występuje w roli architekta i uczy innych). Przekonajmy się, co oferuje nam Wujek Bob w ramach swojego najnowszego opracowania.

Zawartość *Czystej architektury* skonstruowana jest na zasadzie warstw: autor omawia kolejne tematy – poczynając od kwestii najbardziej fundamentalnych, przy czym każda kolejna część książki (warstwa) opiera się na poprzedniej. Struktura ta determinuje porządek czytania książki, który zdecydowanie powinien mieć charakter chronologiczny.

Pierwsza, bardzo krótka część książki – *Wprowadzenie* – to próba znalezienia odpowiedzi na fundamentalne pytania: czym jest architektura oprogramowania i na podstawie jakich kryteriów można określić, czy jest ona dobra bądź zła. Znając te odpowiedzi (bardzo celnie i dobrinie wyartykułowane przez autora), możemy zagłębić się w kolejne rozdziały, które próbują odpowiedzieć na pytanie, jak projektować oprogramowanie, aby jego architektura była *dobra*. Tutaj właśnie zaczynamy przekopywać się przez warstwy wiedzy, o których wspomniałem wyżej. Na początek – paradigmaty programowania: strukturalne, obiektowe i funkcyjne. Trzeba w tym miejscu powiedzieć, że autor ma niezwykły dar przekazywania wiedzy w sposób prosty, klarowny i dobitny – *Czysta architektura* potwierdza w stu procentach tę tezę. W trzeciej części książki omówiono pięć fundamentalnych zasad projektowania oprogramowania, znanych pod kryptonimem SOLID. Ta część *Czystej architektury* jest nieco redundantna w stosunku do poprzednich opracowań autora (który o SOLID pisał już nieraz w swoich wcześniejszych publikacjach), jednakże muszę przyznać, że ich omówienie jest bardzo dokładne i celne, świetnie też komponuje się z pozostałymi częściami tej publikacji i zdecydowanie nie mogło go tu zabraknąć. Czwarta część książki przybliża koncepcję komponentów jako wysokopoziomowych składników architektury, zasady ich spójnego projektowania oraz

sposoby ich łączenia. Wreszcie piąta – najbardziej obszerna część opracowania – tłumaczy pojęcie architektury systemów oprogramowania. Autor otwiera ten fragment książki dokładnym omówieniem znaczenia tego pojęcia, następnie przedstawia po kolejne aspekty, które należy uwzględnić przy projektowaniu architektury, także konsekwencje podejmowanych na tym poziomie decyzji. W tej części przedstawione są też koncepcje „Krzyczącej Architektury” (ang. *Screaming Architecture*) oraz tytuowej „Czystej Architektury” (ang. *Clean Architecture*), jako przykłady praktycznego zastosowania opisanych w książce zasad projektowania. Książkę zamknięta jest zatytułowana *Szczegóły*, gdzie autor stara się wytłumaczyć, jak połączyć wyidealizowaną wizję czystej architektury z takimi „drobiazgami” implementacyjnymi jak bazy danych, sieć WWW czy framework. Czytelnik znajdzie tu również ciekawe studium przypadku projektowego.

Tyle jeśli chodzi o zawartość... Teraz kilka słów na temat odczuć po lekturze najnowszej pozycji Wujka Boba. Moje ogólne wrażenie było zdecydowanie pozytywne, przy czym kilka kwestii rzuciło mi się w oczy. Po pierwsze, bardziej zaawansowani/doświadczeni czytelnicy (a oni są głównymi adresatami *Czystej architektury*) mogą czuć się łatwo zawiedzeni, jeśli oczekiwali po tej publikacji jakiejś rewolucji. Rzecz w tym, że jeśli ktoś na bieżąco śledził prace autora książki umieszczone na przestrzeni ostatnich lat na blogach <https://8thlight.com/blog/uncle-bob/> i <https://blog.cleancoder.com/>, może odnieść wrażenie, że autor trochę się powtarza. Dla jasności – nie uważam wcale, że jest to wada, chociażby z racji tego, że w książce materiał ten jest uporządkowany, uzupełniony i tworzy chronologiczną, spójną całość. Druga kwestia jest taka, że ze względu na merytoryczną wagę przedstawionego materiału książki tej, w porównaniu z *Czystym kodem* czy *Mistrzem czystego kodu*, nie da się czytać do poduszki – ponieważ wymaga od odbiorcy dużo skupienia i uwagi. Mam też wrażenie, że u niektórych osób, szczególnie tych, którzy mają mocno spolaryzowane poglądy na temat konstruowania systemów oprogramowania, może wywoływać skrajne uczucia, jako że autor dość autorytarnie przedstawia swoje poglądy w tym zakresie. Podsumowując: nie jest to książka dla każdego i wymaga odpowiedniego nastawienia od czytelnika. Ja osobiste bardzo dużo z niej wyniosłem i szczerze polecam ją każdemu doświadczonemu twórcy oprogramowania: daje ona bardzo szeroki i przekrojowy pogląd na tematy związane z projektowaniem architektury oprogramowania.

Rafał Kocisz

Tytuł:	<i>Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów</i>
Autor:	Robert C. Martin
Stron:	386
Wydawnictwo:	Helion
Data wydania:	2018-05-11

redakcja

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: Mariusz „maryush” Witkowski (mariuszwitkowski@programistamag.pl).

Korekta: Tomasz Łopuśński. **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 Bartylek, Mariusz Sierakiewicz, Dawid Kaliszewski, Marek Sawerwain, Łukasz Mazur, Łukasz Łopuśński, Jacek Matulewski, Sławomir Sobótka, Dawid Borycki, Gynvael Coldwind, Bartosz Chrabski, Rafał Kocisz, Michał Sajdak, Michał Bentkowski, Paweł „KraZQ” Zajączkowski.

Adres wydawcy: Derenowa 4/47, 02-776 Warszawa.

Druk: <http://www.moduss.waw.pl/>, Nakład: 4500 egz.

Nota prawna

Redakcja zastrzega sobie prawo do skrótów i opracowania tekstu 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 bezpośrednie i pośrednie, 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**



Dołącz do Ericsson i mają wpływ na to,
w którym kierunku zmierza postęp
technologiczny. Nasze zespoły biorą udział
w prestiżowych projektach ICT, które
pozwala nam wszystkim swobodniej
pracować, studiować i żyć w zrównoważonych
społecznościach na całym świecie.

Obecnie ponad 40% światowego ruchu
mobilnego przechodzi przez sieci firmy
Ericsson, z których korzysta ponad
2,5 miliarda abonentów.

Aplikuj już dziś!

[ericsson.com
/careerspoland](http://ericsson.com/careerspoland)



Zmieniaj
z nami
świat