Programowanie (nie tylko) dla artystów

Agnieszka BOROWIECKA

We współczesnym cyfrowym świecie korzystanie z technologii informacyjno-komunikacyjnych stało się koniecznością. Dotyczy to zarówno życia codziennego (programowane telewizory, lodówki, pralki czy nawet odkurzacze), jak i zawodowego. Z komputerów muszą obecnie korzystać także artyści: graficy, projektanci, muzycy. Jednak czy mają je traktować jako zwykłe narzędzie pracy, takie jak pędzel lub ołówek, czy może podejść do tego nieco inaczej?

Cytując Pavla V. Orlova: *„Зачем художнику нужно программировать? Почему нельзя просто взять и нарисовать от руки? А что, фотошоп отменили? Неужели у художников есть математическая логика?" Мне эти вопросы кажутся замечательными и напоминают художника в момент появления фотографии, когда художники спорили с фотографами примерно теми же словами, разве что слова фотошоп еще не было*. W wolnym tłumaczeniu można to ująć: *„Po co artyście programowanie? Dlaczego po prostu nie może wziąć i narysować czegoś odręcznie? A co, Photoshopa skasowali? Czy artyści znają logikę matematyczną?". Pytania te wydają mi się cudowne i przypominają moment pojawienia się fotografii, gdy artyści spierali się z fotografami w mniej więcej tych samych słowach, z wyjątkiem tego, że Photoshopa jeszcze nie było.*

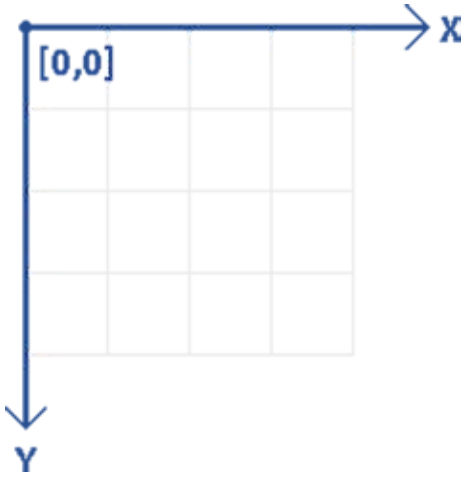
Programowanie i artyści to wydawałoby się dwa odmienne światy. Jednak warto zwrócić uwagę, że wszystko zależy od tego, jakie środowisko pracy im zaproponujemy i w jaki sposób będą z niego korzystać.

W 2001 roku Casey Reas i Benjamin Fry zaczęli pracę nad językiem i środowiskiem Processing dla grupy Aesthetics and Computation w Massachusetts Institute of Technology Media Lab. Korzenie tego środowiska tkwią w języku DBN (*Design by Numbers*) opracowanym przez profesora Johna Maeda z MIT. Czym Processing wyróżnia się wśród innych języków programowania? Cóż, właściwie wszystkim. Celem jego powstania miały być wizualizacja danych, sztuka mediów, sztuka elektroniczna i nauczanie. Powstał jako narzędzie do uczenia artystów programowania. Tak naprawdę nie tworzymy w nim aplikacji, ale serię szkiców (sketch) – które trafiają do szkicownika (sketchbook), co jest naturalnym stylem pracy dla artystów. Processing jest oparty na licencji open source, wzbogacają go liczne biblioteki tworzone przez niezależnych twórców i rozszerzające znacznie jego możliwości. Możemy pracować, korzystając z różnych systemów operacyjnych (MacOS, Windows, Linux). Tworząc programy, w łatwy sposób uzyskujemy konkretne efekty wizualne, podobnie jak pracując z grafiką żółwia w Logo. Możemy także rozwiązywać skomplikowane problemy, nie tylko dla artystów.

Jeśli w swojej klasie mamy uczniów, którzy pięknie rysują, projektują gry lub tworzą komiksy, spróbujmy wprowadzić ich w świat programowania, korzystając właśnie z Processingu.

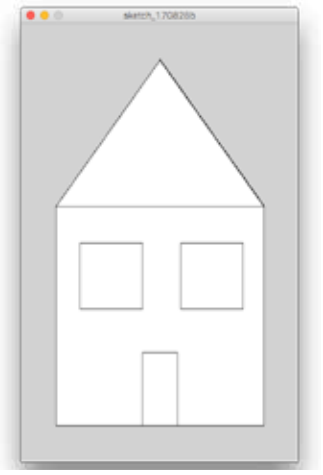
Logika dla artystów, a może zaczniemy od „zwykłej" matematyki?

Pisząc programy w Processingu, czy może raczej tworząc szkice, nie unikniemy odrobiny matematyki. Jest to w zasadzie nieuniknione dla każdego, kto chce tworzyć rysunki nie za pomocą myszki, ale wydając odpowiednie polecenia dla komputera. Processing, uruchamiając program, wyświetla małe okienko, w którym pojawią się zdefiniowane przez nas obiekty: punkty, linie, elipsy, trójkąty, czworokąty, napisy. Aby cokolwiek pojawiło się na ekranie, musimy określić położenie i wielkość rysowanego obiektu. Warto najpierw zorientować się, ile mamy miejsca na rysunek i jak określamy współrzędne ekranowe.



Domyślnie dostępny do rysowania obszar ma wymiary 100 na 100 pikseli. Punkt o współrzędnych (0, 0) znajduje się w lewym górnym narożniku okna, współrzędne x-owe rosną w prawo, zaś współrzędne y-owe w dół. Rozmiar okna można zmienić za pomocą polecenia `size()`. Tworząc dowolny rysunek, musimy odpowiednio wyliczać współrzędne wierzchołków rysowanych figur oraz dbać o kolejność rysowania obiektów (figury kreślone z wypełnionym wnętrzem mogą się zasłaniać). Warto wykorzystać kartkę w kratkę do zaprojektowania wyglądu okna aplikacji, a następnie zaprogramować rysunek na komputerze. Tworząc szkice w Processingu, definiujemy zawsze dwie standardowe funkcje – `setup()` do ustalania początkowych wartości parametrów oraz `draw()`, odpowiedzialną za zmianę zawartości okna aplikacji. Poniżej prezentujemy przykład programu rysującego dom z trójkątnym dachem, dwoma oknami i drzwiami.

```
void setup()
{
   size(400, 600);
}
void draw()
{
   // dom
   rect(50, 250, 300, 300);
   // dach
   triangle(50, 250, 350, 250,
            200, 50);
   // drzwi
   rect(175, 450, 50, 100);
   // okna
   rect(85, 300, 90, 90);
   rect(230, 300, 90, 90);
}
```



Olbrzymią zaletą Processingu jest łatwość tworzenia interakcji z użytkownikiem. Dostępne są specjalne zmienne i funkcje pozwalające komunikować się z myszką i klawiaturą. Projekt, który chcielibyśmy opisać, to prosty edytor graficzny. Przesuwając kursorem myszki po ekranie, będziemy tworzyć rysunek, do przygotowania ciekawszych efektów wizualnych wykorzystamy symetrię i funkcje matematyczne. Uczniowie, tworząc taki program, muszą zastanowić się, jak wyliczać współrzędne odpowiednich punktów w oknie aplikacji, by uzyskać symetryczne odbicie kreślonych linii względem środka okna. Jeśli będą chcieli uzależnić sposób rysowania (np. kolor linii, kształt pisaka) od pewnych dodatkowych warunków, pojawi się konieczność skorzystania z logiki matematycznej i instrukcji warunkowej. Na przykład budując w aplikacji przycisk pozwalający określić kolor pędzla, musimy zbadać położenie kursora myszki

względem tego przycisku – czyli zapisać złożony warunek, korzystając z koniunkcji i alternatywy. Im bardziej złożone projekty będziemy przygotowywać, tym bardziej znajomość logiki matematycznej będzie nam potrzebna.

Projekt Malowanka[1]

Najprostsza wersja projektu wymaga od nas napisania zaledwie kilku instrukcji. Rysunki będziemy tworzyć za pomocą pędzla w kształcie koła, korzystając z polecenia `ellipse()`. Należy podać cztery liczby określające współrzędne środka elipsy oraz długości jej małej i wielkiej osi. Jeśli osie mają taką samą długość, na ekranie zostanie narysowane koło. Możemy skorzystać ze zmiennych systemowych `mouseX` i `mouseY`, by środek rysowanej elipsy pokrywał się ze współrzędnymi kursora myszki.

Za pomocą poleceń `fill()` i `stroke()` ustalamy kolor wypełnienia i obwódki elipsy, taki sam podczas całego działania aplikacji. Możemy także zmienić domyślny rozmiar i kolor tła okna aplikacji – polecenie `background()`.

```
void setup()
{
   size(500, 500);
   fill(255);
   stroke(255);
   background(0);
}
void draw()
{
   ellipse(mouseX, mouseY, 30, 30);
}
```

Funkcja `draw()` jest wywoływana automatycznie 60 razy na sekundę, jeśli w tym czasie zmieni się położenie kursora myszki, to na ekranie zostaną narysowane kolejne elipsy.

Pierwsza wersja aplikacji jest gotowa. Jednak łatwo można zauważyć jej niedoskonałość – nowe elipsy są rysowane przy każdej zmianie położenia kursora myszki, niezależnie od tego, czy w danym miejscu chcieliśmy coś narysować, czy nie. Użytkownik ma ograniczoną kontrolę nad powstającym rysunkiem. Lepszym rozwiązaniem byłoby rysowanie nie przy każdym ruchu kursora myszki, lecz jedynie wtedy, gdy sobie tego życzymy – czyli po wciśnięciu przycisku myszki.

Należy wprowadzić dwie zmiany w programie. Dodajemy nową funkcję `mouseDragged()`. Jest ona wywoływana automatycznie, gdy przytrzymamy wciśnięty dowolny przycisk myszki i zaczniemy przesuwać jej kursor po ekranie. Instrukcję rysującą elipsę przenosimy z funkcji `draw()` do funkcji `mouseDragged()`. Poprawiony program wygląda następująco:

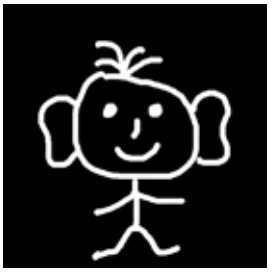
```
void setup()
{
   size(500, 500);
   fill(255);
   stroke(255);
   background(0);
}

void draw()
{

}

void mouseDragged()
{
   ellipse(mouseX, mouseY, 30, 30);
}
```

Tworzenie czarno-białych rysunków może się szybko znudzić. Uzyskane efekty byłyby ciekawsze, gdyby można było zmieniać kolor, grubość i kształt pędzla. Bardziej energiczny użytkownik zauważy również, że przy szybszym przemieszczaniu kursora myszy pojawiają się przerwy w kreślonych liniach – widać wyraźnie, w jaki sposób one powstają. Zamiast tego możemy rysować łamaną,

[1] Opisany scenariusz zajęć pochodzi z projektu Warszawa programuje!, realizowanego od 2016 roku przez OEIiZK oraz WCIES.

wykorzystując polecenie `line()`, a nie stawiać dużą liczbę kropek poleceniem `ellipse()`. Kreśląc linie, będziemy łączyć punkt wskazywany w danej chwili przez kursor myszy z poprzednią pozycją kursora (zmienne systemowe `pmouseX` i `pmouseY`). Aby kreślone linie były wyraźniejsze, a punkty łączenia łamanej mniej „kanciaste", warto zmienić grubość rysowanych krawędzi, korzystając z polecenia `strokeWeight()`. Możemy także usunąć definicję koloru wypełnienia, gdyż nie będzie już nam potrzebna.

```
void setup()
{
  size(500, 500);
  stroke(255);
  strokeWeight(10);
  background(0);
}
void draw()
{
}
void mouseDragged()
{
   line(pmouseX, pmouseY, mouseX, mouseY);
}
```



Wprowadzone zmiany znacznie poprawiły jakość tworzonych rysunków. W następnym kroku dodamy do aplikacji możliwość zmiany koloru rysowanych linii. Użytkownik może określać kolor rysowania, wciskając odpowiedni klawisz na klawiaturze lub klikając myszką w wybrany obszar okna projektu. To drugie rozwiązanie ograniczyłoby obszar rysowania przez konieczność zdefiniowania na ekranie palety barw, czyli paska narzędziowego dla użytkownika aplikacji. W tym celu należałoby narysować kolorowe prostokąty np. w górnej części okna aplikacji. Po kliknięciu w prostokąt zmieniany byłby odpowiednio kolor pędzla. W podobny sposób można dodać możliwość wyboru grubości kreślonych linii. Wykorzystanie palety barw polecamy dla aplikacji, którą chcielibyśmy uruchamiać na urządzeniach mobilnych – rozwiąże to problem z dostępem do klawiatury ekranowej.

Ustalanie koloru rysowania za pomocą klawiatury lub myszki wymaga dodania do projektu funkcji `keyPressed()` lub `mousePressed()` oraz skorzystania z instrukcji warunkowej. Zamiast tego proponujemy wprowadzenie automatycznego określania koloru linii. Definiując kolor podajemy zwykle trzy składowe RGB – czerwoną, zieloną i niebieską. Każda składowa przyjmuje domyślnie wartości z zakresu od 0 do 255. W projekcie uzależnimy wartość składowej od współrzędnych kursora myszy. Na przykład składowa czerwona może zależeć od współrzędnej x, składowa zielona od współrzędnej y, zaś składowa niebieska od odległości między kursorem myszy a środkiem okna programu. Do obliczeń wykorzystamy funkcje `map()` i `dist()`. Pierwsza z nich pozwala przeliczać wartości z jednego zakresu na drugi, druga oblicza odległość między dwoma punktami.

Współrzędna x-owa kursora myszy może przyjmować wartości od zera (lewa krawędź okna aplikacji) do szerokości okna zdefiniowanej w funkcji `setup()` (zmienna systemowa `width`). Musimy znaleźć liczbę, jaka odpowiadałaby położeniu pierwszej współrzędnej kursora myszki, po przeliczeniu na zakres od 0 do 255, czyli wartości odpowiadające składowej RGB. Definiujemy pomocniczą zmienną `red`:

```
float red = map(mouseX, 0, width, 0, 255);
```

W analogiczny sposób wyliczamy składową zieloną, tym razem odwołując się do zmiennej systemowej `height`, określającej wysokość okna aplikacji:

```
float green = map(mouseY, 0, height, 0, 255);
```

Ostatnią składową wyliczamy jako odległość kursora myszki od środka ekranu:

```
float blue = dist(mouseX, mouseY,
                  width / 2, height / 2);
```

Zmienna `blue` powinna przyjmować wartości z zakresu od 0 do 255, tymczasem istnieją punkty w oknie aplikacji, których odległość od środka okna jest większa od 255. Możemy ponownie skorzystać z funkcji `map()` lub użyć funkcji `constrain()` do przycięcia wartości zmiennej do właściwego zakresu.

```
blue = constrain(blue, 0, 255);
```

Na podstawie wyliczonych wartości definiujemy kolor kreślonych linii:

```
stroke(red, green, blue);
```

Możemy przedyskutować z uczniami, jaki efekt spowodują wprowadzane przez nas zmiany. Warto definiować najpierw jedną składową, pozostałym przypisując wartość zero. Pozwoli to zauważyć, jak zmiana położenia kursora myszki wpływa na zmianę koloru kreślonych linii, zarówno na ich barwę, jak i jaskrawość. Warto również zastanowić się, jak zmieni się wygląd rysunku, gdy zamienimy rodzaj wyliczeń przypisanych poszczególnym składowym.

Ostatnia wersja programu pozwala uzyskać naprawdę interesujące efekty. Warto dodać do projektu dwie nowe funkcjonalności – możliwość zapisywania rysunku oraz czyszczenia ekranu bez konieczności ponownego uruchomienia aplikacji. Znaki wprowadzone za pomocą klawiatury są przechowywane w zmiennej `key`. Przyjmijmy, że naciśnięcie klawisza z literą *c* spowoduje wyczyszczenie ekranu, klawisz *s* pozwoli zapisać gotowy rysunek pod podaną nazwą.

```
void keyPressed()
{
   if (key == 'c') background(0);
   if (key == 's') saveFrame("rysunek.png");
}
```

Wróćmy na chwilę do matematyki. Chcielibyśmy dodać do tworzonych rysunków odbicie lustrzane, czyli symetrię względem pionowej osi przechodzącej przez środek okna. Jakie zmiany należałoby wprowadzić w programie? Warto zacząć od dyskusji z uczniami, ilustrując zachowanie współrzędnych punktów w kartezjańskim układzie współrzędnych. Następnie próbujemy przekształcić otrzymane wzory tak, by pasowały do rozmiarów okna i położenia układu współrzędnych w aplikacji tworzonej w Processingu.

Po wprowadzeniu ostatnich poprawek funkcja `mouseDragged()` przyjmie następującą postać:

```
void mouseDragged()
{
  float red = map(mouseX, 0, width, 0, 255);
  float green = map(mouseY, 0, height, 0, 255);
  float blue = dist(mouseX, mouseY,
                    width / 2, height / 2);
  blue = constrain(blue, 0, 255);
  stroke(red, green, blue);
  line(pmouseX, pmouseY, mouseX, mouseY);
  line(width - pmouseX, pmouseY,
       width - mouseX, mouseY);
}
```

Opisany projekt pozwala na stopniowe wprowadzanie wielu pojęć, łączy tworzenie interesującej grafiki z nauką pisania programów, rozwija także umiejętności matematyczne uczniów. Tworząc program, zapisujemy w języku formalnym – przy wykorzystaniu metod z matematyki i informatyki – to, co rozumiemy intuicyjnie. Aplikacja **Malowanka** może stanowić bazę do tworzenia wielu interesujących modyfikacji: kreślenia symetrycznych modeli – projektów naczyń, dodania efektu „zanikania" rysunku, dodania kolejnych symetrii oraz możliwości włączania i wyłączania wybranych funkcjonalności za pomocą klawiatury itd.

Trochę więcej matematyki

Bawiąc się rysowaniem, możemy testować działanie różnych poleceń i wprowadzać skomplikowane pojęcia. Przypuśćmy, że będziemy rysować proste kwiatki złożone z żółtego środka i kilku niebieskich płatków. Do rysowania płatków wykorzystamy iterację – pętlę `for`. Wyliczenie położenia kolejnych płatków może się wydawać zbyt skomplikowane dla naszych uczniów, wymaga bowiem znajomości funkcji trygonometrycznych, które w nowej podstawie programowej są wprowadzane dopiero w liceum. Jednak Processing przychodzi nam z pomocą. Wystarczy wyliczyć położenie jednego płatka, a pozostałe zostaną narysowane dzięki funkcji `rotate()`. Załóżmy, że środek kwiatka rysowany jest w środku układu współrzędnych. Wykorzystamy do tego poniższe dwie instrukcje:

```
fill(250, 250, 0);

ellipse(0, 0, 20, 20);
```

Przyjęliśmy, że średnica koła będącego środkiem kwiatka wynosi 20. Zastanówmy się, jak wyliczyć współrzędne środka dowolnego płatka. Najprościej jest narysować płatek, przesuwając się wzdłuż jednej z osi układu współrzędnych o sumę promienia środka kwiatka i promienia płatka, np.

```
ellipse(25, 0, 30, 30);
```

Pozostałe płatki są tak samo odległe od środka układu współrzędnych, jednak zamiast wyliczać ich współrzędne wykonamy „sztuczkę" – obrócimy cały układ współrzędnych o odpowiedni kąt, a następnie narysujemy identyczną elipsę. Funkcja `rotate()` obraca układ współrzędnych o podany kąt wyrażony w radianach. Możemy jednak podać wartość kąta w stopniach, a następnie przeliczyć go za pomocą funkcji `radians()`. Do rysowania pięciu płatków moglibyśmy użyć następującej pętli `for`:

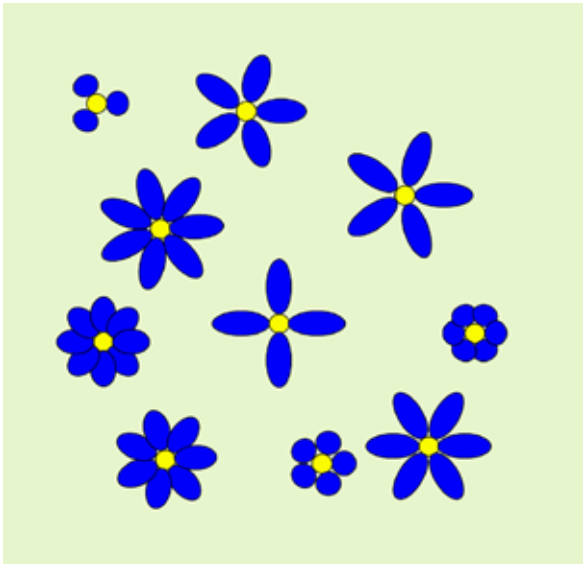
```
fill(0,0,250);
for (int i=0; i<5; i++)
{
   rotate(radians(360 / 5));
   ellipse(25, 0, 30, 30);
}
```

Kwiatek zostanie narysowany, jednak nie będziemy pewni, czy uzyskany efekt jest prawidłowy – w oknie aplikacji widoczny jest tylko jego fragment. Pamiętajmy, że środek układu współrzędnych, będący jednocześnie środkiem kwiatka, leży w górnym lewym rogu okna aplikacji. W Processingu jest dostępna także funkcja `translate()`, pozwalająca przesunąć środek układu współrzędnych w inne miejsce, np. do punktu wskazywanego przez kursor myszy. Do naszego programu dodamy jeszcze dwie funkcje: `pushMatrix()`, zapamiętującą bieżące położenie układu współrzędnych oraz `popMatrix()`, przenoszącą układ współrzędnych do zapamiętanego położenia. Dzięki temu można na przykład dodać do projektu wypisywanie licznika narysowanych kwiatków bez konieczności zastanawiania się, w jakim punkcie ekranu powinien być rysowany.

```
void setup() {
   size(600, 600);
   background(#E7F5CD);
}

void draw() {
}

void mousePressed() {
   pushMatrix();
   translate(mouseX,mouseY);
   fill(250, 250, 0);
   ellipse(0, 0, 20, 20);
   fill(0, 0, 250);
   for (int i=0; i<5; i++) {
      rotate(radians(360 / 5));
      ellipse(25, 0, 30, 30);
   }
   popMatrix();
}
```

Program możemy wzbogacić o możliwość rysowania kwiatków o zmiennej liczbie płatków. Do ustalania liczby płatków wykorzystamy funkcję `random()`, losującą liczbę rzeczywistą z zakresu od zera do podanej wartości. Wynik musimy przekonwertować na wartość całkowitą i zapamiętać w pomocniczej zmiennej. Uzyskaną wartość użyjemy do określenia liczby powtórzeń pętli oraz wyliczenia kąta, o jaki obracamy układ współrzędnych.

Każdy rysowany kwiatek może mieć od trzech do ośmiu płatków.



```
void mousePressed() {
   pushMatrix();
   translate(mouseX, mouseY);
   fill(250, 250, 0);
   ellipse(0, 0, 20, 20);
   fill(0, 0, 250);
   int n = 3 + int(random(6));
   for (int i=0; i<n; i++) {
      rotate(radians(360 / n));
      ellipse(25, 0, 30, 30);
   }
   popMatrix();
}
```

Uczniowie mogą eksperymentować w celu uzyskania innego kształtu płatków, dodać losowy kolor kwiatków itp. Możemy także zastanowić się, jak utworzyć bukiet kwiatków, dorysowując zielone łodyżki wychodzące z jednego punktu.

Programowanie w szkole

Programowanie nie jest łatwym zagadnieniem dla uczniów. Powinniśmy się zastanowić jak ich zachęcić do nauki. Ucząc programowania, warto dobrać właściwe przykłady, aby wyjaśnić uczniom sposób działania poszczególnych instrukcji i rozwijać prawidłowe intuicje. Podczas pracy nad projektem **Kwiaty** możemy zwrócić uwagę uczniów na łatwość automatyzowania pewnych czynności w celu uzyskania regularnych wzorów. Podobny efekt byłby bardzo trudny do uzyskania podczas odręcznego tworzenia grafiki. W obu projektach warto zauważyć, że praca koncepcyjna, planowanie wyglądu i działania aplikacji jest bardzo ważna i poświęcamy jej więcej uwagi niż pracy „manualnej" – zaprogramowaniu aplikacji. Przejście od planu do realizacji jest płynne, polega bardziej na sformalizowaniu wiedzy posiadanej intuicyjnie niż na zapamiętywaniu gotowych formułek i instrukcji.

Równie istotnym zagadnieniem jest właściwy dobór języka programowania i środowiska, w którym będziemy pracować. Zachęcamy do korzystania ze środowiska Processing, dostępnego na różne systemy operacyjne i umożliwiającego m.in. tworzenie aplikacji mobilnych dla systemu Android. Język, jakim się posługujemy, to uproszczona wersja Javy. W opisanych projektach w sposób naturalny wprowadzamy pojęcie zmiennej, instrukcję warunkową, pętlę oraz losowość. Jednak możliwości środowiska są dużo większe, pozwalają definiować proste gry logiczne i planszowe, symulacje zjawisk czy przygotowywać wygodne narzędzia do podstawowej obróbki zdjęć. Warto także dodać, że Processing jest wykorzystywany na kursach dotyczących programowania, oferowanych przez bardzo popularny portal Akademia Khana.

Agnieszka BOROWIECKA jest nauczycielem konsultantem w Ośrodku Edukacji Informatycznej i Zastosowań Komputerów w Warszawie.