Wprowadzenie

Wprowadzenie

Na tych zajęciach będziemy implementować część algorytmów omawianych na wykładach. Idealnie byłoby zaimplementować wszystko, ale ze względu na ograniczenia czasowe, skupimy się tylko na najważniejszych tematach.

Pierwsze kilka ćwiczeń będzie miało charakter programistyczny i językiem, jakim będziemy się posługiwać będzie JavaScript. Do rysowania użyjemy bardzo wygodnej biblioteki **p5.js**, która jest rozszerzeniem języka do tworzenia aplikacji graficznych o nazwie Processing. Istotą ćwiczeń jest implementacja algorytmów, więc wybrano język, który jest najprostszy i najmniej wymagający do ustawienia i zastosowania.

Jeśli ktoś preferuje pracę we własnym edytorze lub IDE, jest to jak najbardziej dozwolone. Wszystkim innym polecamy pracę w środowisku webowym do programowania w JS, na przykład:

- **https://jsbin.com**
- **http://js.do/**
- **https://jsfiddle.net/**
- **https://repl.it**

UWAGA! Proszę zapisywać swoje prace w trakcie i pod koniec zajęć. Część z powyższych rozwiązań oferuje, po załogowaniu, zapisywanie skryptów w "chmurze", ale w innym wypadku warto koniecznie kopiować zawartość wykonanych ćwiczeń do plików i zapisywać je w bezpiecznym miejscu. Posiadanie zapisanych ćwiczeń pomoże rozwiać wątpliwości dotyczące ocen z przedmiotu pod koniec semestru (jeśli się takie pojawią).

Pierwsze kroki

Po wejściu na stronę jedną z powyższych stron prawdopodobnie zobaczysz jakiś kod wpisany w polu tekstowym. Ten kod można skasować i zostawić na razie pustą stronę. W niektórych stronach, dostępne będzie nowe jedno pole tekstowe. Do zrobienia testu działania programu wystarczy napisać polecenie, do pola kodu HTML:

```
<script type="text/javascript">console.log('Hello world!');</script>
```

Albo prościej, do pola JavaScript:

```
console.log('Hello world!');
```

Aby zobaczyć postępy swojej pracy kliknij na przycisk *Run*. Jeśli strona nie ma opcji pokazania zawartości konsoli, wciśnij na klawiaturze przycisk F12, żeby pokazać konsolę wbudowaną w przeglądarkę.

Pierwszym krokiem będzie wczytanie biblioteki do p5.js używając następującej linii (w polu HTML):

```
<script src="/cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.7/p5.js"></script>
```

Potem dodajemy poniższy kod do pola JavaScript (albo wewnątrz tagów `<script>...</script>` w polu HTML):

```
function setup() {
}
function draw() {
}
```

Powyższy kod zawiera 2 metody: *setup* i *draw*. Pierwsza metoda jest wykonywana tylko raz na początku programu i służy do konfiguracji środowiska i ustawienie globalnych wartości. Druga metoda jest metodą do rysowania i zostanie wykonana raz (w przypadku rysunku) lub wiele razy na sekundę (w przypadku animacji).

Setup

Pierwsza i najważniejszą rzeczą jaką należy wykonać w setup to ustawienie rozmiaru canvasu:

```
createCanvas(800,600);
```

Polecenie to stworzy na stronie element języka HTML służący do operacji rysowania. Spróbuj zmienić rozmiary elementów na stronie do pisania kodu, tak żeby było widać cały canvas.

Drugą rzeczą jaką warto to zrobić jest zdecydować czy mamy rysować pojedynczy obraz, czy animację. W przypadku obrazu wykonujemy następującą metodę, co spowoduje jednokrotne wykonanie metody *draw*:

```
noLoop();
```

W przypadku animacji, można wykonać następującą metodę, która spowoduje odtwarzanie metody *draw* 25 razy na sekundę:

```
frameRate(25);
```

Do tego ćwiczenia wykorzystajmy na razie pierwszą z tych dwóch opcji, czyli *noLoop*.

Draw

Jeśli chodzi o rysowanie, pierwszą rzeczą, jaką warto zrobić, jest skasowanie obrazu, a raczej ustawienie koloru tła:

```
background(100);
```

Metoda ta może przyjąć zarówno pojedynczą wartość, co spowoduje określenie koloru na podstawie odcieni szarości, albo za pomocą 3 liczb, co pozwoli na ustalenie wartości jako sumy barw R, G i B.

Żeby wpisać jakiś napisztosy tekst, można użyć zwykłego polecenia:

```
text("Hello world!",20,20);
```

Gdzie pierwszy argument to tekst który chcemy wpisać, a 2 kolejne argumenty to współrzędne X,Y określające jego położenie (możesz sprawdzić, jak to wygląda klikając *Run Code*).

Żeby zmienić czcionkę, należy ją przed wykonaniem powyższego polecenia najpierw ustawić razem z rozmiarem używając odpowiedniego polecenia:

```
textFont("Courier", 24);
```

W tym miejscu w kodzie można rysować dowolne rzeczy używając funkcji dostępnych w języku Processing według dokumentacji poniżej:

https://p5js.org/reference/

Ale my się skupimy na nieco innych zagadnieniach.

Piksele

(zadanie na ocenę 3)

Chociaż język Processing zawiera sporo zaimplementowanych algorytmów rysowania, na lekcji GRK będziemy ich świadomie unikać. Większość zadań będziemy wykonywać bezpośrednio na poszczególnych pikselach obrazu.

Skasuj zawartość funkcji draw z poprzedniego zadania i zostaw w niej tylko ustawienie koloru tła na czarny (wartość 0). Żeby zmienić kolor wszystkich pikseli po kolei, musimy je przeiterować używając podwójnej pętli *for* (zmienne *height* i *width* są nam udostępnione przez bibliotekę) :

```
for(y=0; y<height; y++)
  for(x=0; x<width; x++) {
  }
```

Wewnętrzna pętla będzie iterować poszczególne piksele od lewej do prawej, a zewnętrzna poszczególne wiersze od góry do dołu. Do ustawiania pixeli, użyjemy następującej funkcji:

```
set(x, y, color(100));
```

Trzeci parametr funkcji `set` może przyjąć różne wartości. Jeśli podamy jedną liczbę, będzie ona traktowana jako kolor w odcieniach szarości. Podając trzy liczby, będą traktowane jako wartości R, G B. Dodatkowo można podać również czwartą wartość reprezentującą kanał Alpha. Po zmodyfikowaniu pikseli, należy wykonać następującą funkcję, żeby zaktualizować wygląd canvasu:

```
updatePixels();
```

Po uruchomieniu powyższego kodu, może się okazać, że nie zostanie on narysowany do końca. To jest dlatego, że funkcja ta działa dosyć powolnie i uruchamiany jest mechanizm zabezpieczający przed zawieszeniem programu (spójrz do konsoli). Żeby go wyłączyć, wystarczy na początku funkcji *draw* dopisać komentarz:

```
//noprotect
```

Na zaliczenie tego zadania ustaw w pętli wszystkie piksele na kolor fioletowy (magenta) o dowolnej jasności.

Gradienty

(zadanie na ocenę 3.5)

Zanim przejdziesz do rozwiązywania tego zadania, zapisz napisane poprzednie rozwiązanie albo otwórz nowe okno programu/strony i skopiuj niezbędny kod z poprzedniego rozwiązania.

Do realizacji tego zadania należy narysować gradient w odcieniach szarości, biegnący płynnie (bez przerw) od lewej do prawej krawędzi ekranu.

Zadanie to można łatwo zrealizować podstawiając za wartość koloru (w odcieniach szarości) współrzędną *x*, ale w ten sposób tylko część obrazu zostanie zamalowana gradientem. Cała prawa strona obrazu będzie zamalowana na biało.

Powodem tego jest to, że canvas ma 800 pikseli szerokości, a odcieni szarości mamy tylko 256. Po narysowaniu ostatniego koloru białego (o wartości 255), kolejne liczby zostną ustawione na maksymalną dozwoloną liczbę. Tak samo, wartości poniżej 0, zostałyby ustawione na 0 (kolor czarny).

Aby skorygować obraz, należy tak zmieniać wartość koloru, żeby:

- jeśli x = 0, odcień = 0
- jeśli x = 799, odcień = 255
- wartości między tymi dwoma skrajnymi wartościami są równomiernie rozłożone

Wyliczenie takiej wartości jest dosyć proste:

1. najpierw należy podzielić wartość współrzędnej *x* przez szerokość (*width*) - w ten sposób otrzymamy wartość od =0..1) zmieniającą się od lewej do prawej
2. otrzymaną wartość należy pomnożyć przez 256 (ilość odcieni barw) żeby otrzymać liczbę w zakresie =0..255=

(zadanie na ocenę 4)

Jak otrzymywać gradient powyżej, zachowaj to rozwiązanie i zmodyfikuj go, żeby otrzymać gradient kolorowy, ale w tym przypadku chcemy, żeby się zmieniał w sposób trochę inny:

- gradient liniowy ma biec po przekątnej od górnej lewej krawędzi do dolnej prawej i zmieniać *ilość* składowej niebieskiej od 0 do maksimum
- gradient kolowy ma biec od środka ekranu na zewnątrz i zmieniać się od koloru czerwonego (w środku) do zielonego (na zewnątrz)

Ma to razem wyglądać mniej więcej tak:

Aby wykonać pierwszą część zadania, należy policzyć dla koloru niebieskiego podobny gradient do poprzedniego, ale tym razem uwzględniając sumę współrzędnych *x* i *y* znormalizowaną o sumę wysokości i szerokości ekranu. Pozostałe kolory (R i G) możesz na razie ustawić na 0.

Druga część zadania jest trochę bardziej skomplikowana:

1. należy policzyć odległość współrzędnej *x* od środka ekranu - różnicę między *x* i połową szerokości ekranu
2. należy to samo policzyć dla współrzędnej *y*
3. należy policzyć pierwiastek sumy kwadratów (patrz wzór Pitagorasa) powyższych dwóch wartości:

```
d=sqrt(dx*dx+dy*dy)
```

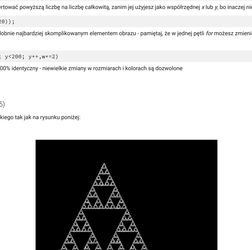
Otrzymaną wartość należy podstawić odpowiednio do kanałów R i G żeby otrzymać obrazek powyżej. Jedna z wartości ma wzrastać od 0..255, a druga malec od 255..0.

Obrazek

(zadanie na ocenę 4.5)

Przed wykonaniem tego zadania, zapisz poprzednie.

W tym zadaniu musisz odtworzyć obrazek poniżej, ale tylko używając dostepu do pikseli i pętli for!



Wskazówki

- uważaj na kolejność rysowanych obiektów - polecana kolejność:
  1. niebo
  2. trawa
  3. kwiaty
  4. fasada
  5. dach
- kwiaty na łące to 1000 losowo wybranych współrzędnych *x* i *y* o losowym kolorze
- żeby przykładowo wylczyć losową liczbę od 10 do 20 użyj funkcji:

```
random(10,20);
```

- uważaj, żeby skonwertować powyższą liczbę na liczbę całkowitą, zanim jej użyjesz jako współrzędnej *x* lub *y*, bo inaczej nie zadziała:

```
floor(random(10,20));
```

- dach jest prawdopodobnie najbardziej skomplikowanym elementem obrazu - pamiętaj, że w jednej pętli *for* możesz zmieniać więcej niż jedną zmienną:

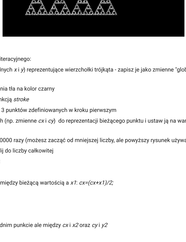
```
for(var y=50,w=0; y<200; y++,w+=2)
```

- obraz nie musi być 100% identyczny - niewielkie zmiany w rozmiarach i kolorach są dozwolone

Fraktale

(zadanie na ocenę 5)

Narysuj trójkąt Sierpińskiego tak jak na rysunku poniżej:



Do tego rysunku użyjemy *algorytmu iteracyjnego*:

1. zdefiniuj 3 punkty (pary współrzędnych *x* i *y*) reprezentujące wierzchołki trójkąta - zapisz je jako zmienne "globalne" na początku skryptu processingowego
2. w funkcji draw zacznij od wymazania tła na kolor czarny
3. ustaw kolor rysowania na biały funkcją *stroke*
4. użyj funkcji *point* do zamalowania 3 punktów zdefiniowanych w kroku pierwszym
5. zdefiniuj nową parę współrzędnych (np. zmienne *cx* i *cy*) do reprezentacji bieżącego punktu i ustaw ją na wartość pierwszej pary punktów (*x1* i *y1*)
6. w pętli *for* zrób co następuje ok. 30000 razy (możesz zacząć od mniejszej liczby, ale powyższy rysunek używał 30k punktów)
   1. wylosuj liczbę od 0..3 i zaokrąglij do liczby całkowitej
   2. sprawdź switchem 3 przypadki:
      - jeśli wylosowano 0:
        - ustaw *cx* na połowę drogi między bieżącą wartością a *x1*: *cx=(cx+x1)/2*;
        - ustaw to samo dla *cy* i *y1*
        - zamaluj punkt *cx,cy*
      - jeśli wylosowano 1:
        - zrób to samo co w poprzednim punkcie ale między *cx* i *x2* oraz *cy* i *y2*
      - w innym wypadku (default):
        - zrób to samo co w poprzednim punkcie ale między *cx* i *x3* oraz *cy* i *y3*

Powyższy algorytm generuje trójkąt Sierpińskiego pomiędzy dowolnymi 3 punktami (*x1,y1*, *x2,y2*, *x3,y3*) na ekranie w sposób iteracyjny. Im więcej iteracji liczymy, tym dokładniejszy jest rysunek. Algorytm działa poniewaz z definicji matematycznej, dlaczego trójkątu Sierpińskiego wynika, że każdy punkt należący do tego zbioru, znajduje się w połowie linii łączącej jakiś punkt z tego zbioru i jeden z wierzchołków trójkąta

Jeśli masz czas i ochotę, możesz zanimować rysowanie tego fraktalu, zmieniając pozycje wierzchołków w każdej klatce animacji. W setupie zmień funkcję *noLoop()* na *frameRate(25)*, a na początku funkcji draw zmieniaj wartości wierzchołków trójkąta (*x1,y1*, *x2,y2*, *x3,y3*) o jakieś losowe wektory (*dx1,dy1*, *dx2,dy2*, *dx3,dy3*). Możesz też zaimplementować odbicie punktów od krawędzi ekranu, opdowiednio zmieniając znaki wektorów jeśli punkt jest blisko odpowiedniej krawędzi ekranu (tak jak w prostej animacji odbijającej się piłki). Do wydajnego wykonywania tej animacji, warto zmienić sposób rysowania pikseli z metody `set` na bezpośrednią modyfikację tablicy `pixels`. Więcej o tym **tutaj**.