Wprowadzenie do p5.js

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Animacje w p5.js

W p5.js mamy możliwość pisania programów z animacją. Jest to bardzo proste i nie musimy się martwić o mechanizm podwójnego buforowania, bo p5.js stosuje go automatycznie.

Domyślna konstrukcja programu w p5.js, tzn. podział na funkcje `setup` i `draw`, oraz działanie tych funkcji (uruchamianie funkcji `draw` w pętli) sprawiają, że mechanizm animacji mamy dostępny od razu. W funkcji `draw` umieszczamy kod rysujący zawartość klatki, a na końcu dokonujemy aktualizacji, zależnej od przyjętego algorytmu animacji, parametrów modelów znajdujących się na scenie.

p5.js oprócz samego mechanizmu animacji (funkcje `setup`, `draw`) posiada kilka funkcji mających wpływ na działanie animacji.

Pierwszą taką funkcją jest funkcja określająca maksymalną liczbę klatek na sekundę:

```
frameRate( fps )
```

gdzie `fps` to maksymalna liczba klatek na sekundę.

Aktualną liczbę klatek na sekundę możemy odczytać wywołując funkcję `frameRate` bez argumentów. Zaś liczbę klatek wyświetlonych od momentu uruchomienia programu ze zmiennej `frameCount`.

p5.js umożliwia jeszcze pobranie różnicy czasu (w milisekundach) pomiędzy początkiem rysowania bieżącej i poprzedniej klatki animacji. Do tego celu służy zmienna `deltaTime`. Taka informacja może być przydatna przy pisaniu animacji wrażliwych na czas, które powinny pozostać stałe niezależnie od wartości fps, np. obliczenia fizyki.

Kolejną funkcją jest funkcja pozwalająca zatrzymać proces uruchamiania funkcji `draw` w pętli:

```
noLoop()
```

Jeśli chcemy wznowić proces uruchamiania funkcji `draw` w pętli, to musimy skorzystać z funkcji:

```
loop()
```

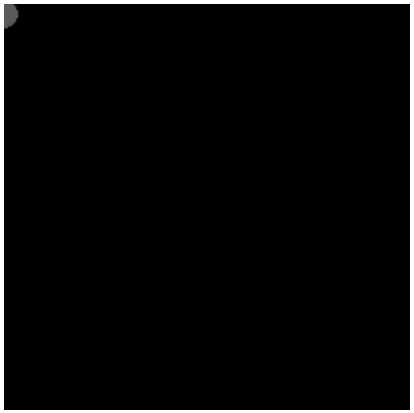
Do sprawdzania aktualnego stanu działania funkcji `draw` (tzn. czy działa w pętli) służy funkcja:

```
isLooping()
```

Ostatnią dostępną funkcją jest funkcja powiadamiająca p5.js, że chcemy odrysować zawartość okna czyli, że chcemy uruchomić funkcję `draw`:

```
redraw()
```

```
1   let x = 0;
2
3   function setup()
4   {
5     createCanvas( 200, 200 );
6
7     frameRate( 15 );
8     background( 0 );
9   }
10
11  function draw()
12  {
13    fill( 0, 36 );
14    rect( 0, 0, width, height );
15
16    fill( 255, 90 );
17    ellipse( x, x + 5, 14, 14 );
18
19    x += 10;
20    if( x > width )
21    {
22      x = 0;
23      background( 0 );
24    }
25  }
```



Obsługa myszki w p5.js

Obsłużenie myszki w p5.js jest bardzo proste. Zaczniemy od możliwości jakie mamy dostępne w dowolnym miejscu naszego programu.

W każdej chwili możemy odczytać współrzędne kursora myszki w oknie programu. Służą do tego zmienne `mouseX` (współrzędna X), `mouseY` (współrzędna Y).

Oprócz bieżących współrzędnych możemy również odczytać współrzędne poprzedniej pozycji kursora. W tym celu używamy zmiennych `pmouseX` (współrzędna X), `pmouseY` (współrzędna Y).

Wszystkie współrzędne podawane są w układzie współrzędnych jaki przedstawiliśmy przy omawianiu grafiki 2D.

Jeśli chcemy sprawdzić czy został naciśnięty jakikolwiek przycisk myszki, to możemy to zrobić korzystając ze zmiennej boolowskiej `mouseIsPressed`. Przyjmie ona wartość `true` jeśli jest naciśnięty jakikolwiek przycisk, a `false` w przeciwnym razie.

Jeśli jesteśmy ciekawi, który przycisk dokładnie został naciśnięty, to sprawdzamy to używając zmiennej `mouseButton`. Przyjmuje ona jedną z trzech wartości: `LEFT` (lewy przycisk myszki), `RIGHT` (prawy przycisk myszki), `CENTER`.

```
1  function setup()
2  {
3    createCanvas( 400, 400 );
4  }
5
6  function draw()
7  {
8    background( 0 );
9
10   if( mouseIsPressed && mouseButton == LEFT )
11     background( 255 );
12 }
```

Oprócz przedstawionych możliwości, z których możemy korzystać w każdym momencie naszego programu, p5.js udostępnia możliwość obsłużenia podstawowych zdarzeń związanych z myszką.

Obsługa zdarzenia wiąże się z napisaniem odpowiedniej funkcji. W funkcji umieszczamy kod, który ma być wykonany w momencie wystąpienia danego zdarzenia.

Dostępne zdarzenia i ich funkcje przedstawione są na następnych slajdach.

- Naciśnięcie przycisku myszki

```
1  function mousePressed()
2  {
3    // instrukcje
4  }
```

- Zwolnienie przycisku myszki

```
1  function mouseReleased()
2  {
3    // instrukcje
4  }
```

- Kliknięcie

```
1  function mouseClicked()
2  {
3    // instrukcje
4  }
```

- Podwójne kliknięcie

```
1  function doubleClicked()
2  {
3    // instrukcje
4  }
```

- Poruszenie kółkiem myszki

```
1  function mouseWheel()
2  {
3    // instrukcje
4  }
```

- Ruch kursora myszki

```
1  function mouseMoved()
2  {
3    // instrukcje
4  }
```

- Ruch kursora z naciśniętym przyciskiem myszki

```
1  function mouseDragged()
2  {
3    // instrukcje
4  }
```

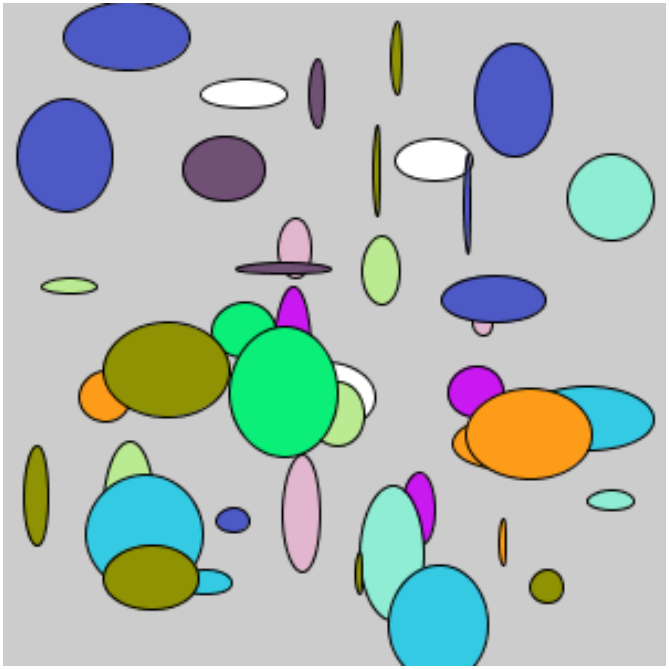
Wewnątrz wymienionych funkcji możemy sprawdzać czy został naciśnięty klawisz myszki, który klawisz został naciśnięty za pomocą zmiennych przedstawionych wcześniej.

Wszystkie funkcje obsługujące zdarzenia mają drugą wersję z opcjonalnym argumentem `event`.

W przypadku obsługi kółka myszki za pomocą tego argumentu możemy odczytać w którą stronę kółko zostało obrócone oraz jak szybko. Do tego celu służy `event.delta`. Zmienna ta przyjmuje wartości dodatnie i ujemne w zależności od tego, w którą stronę kręci się kółko.

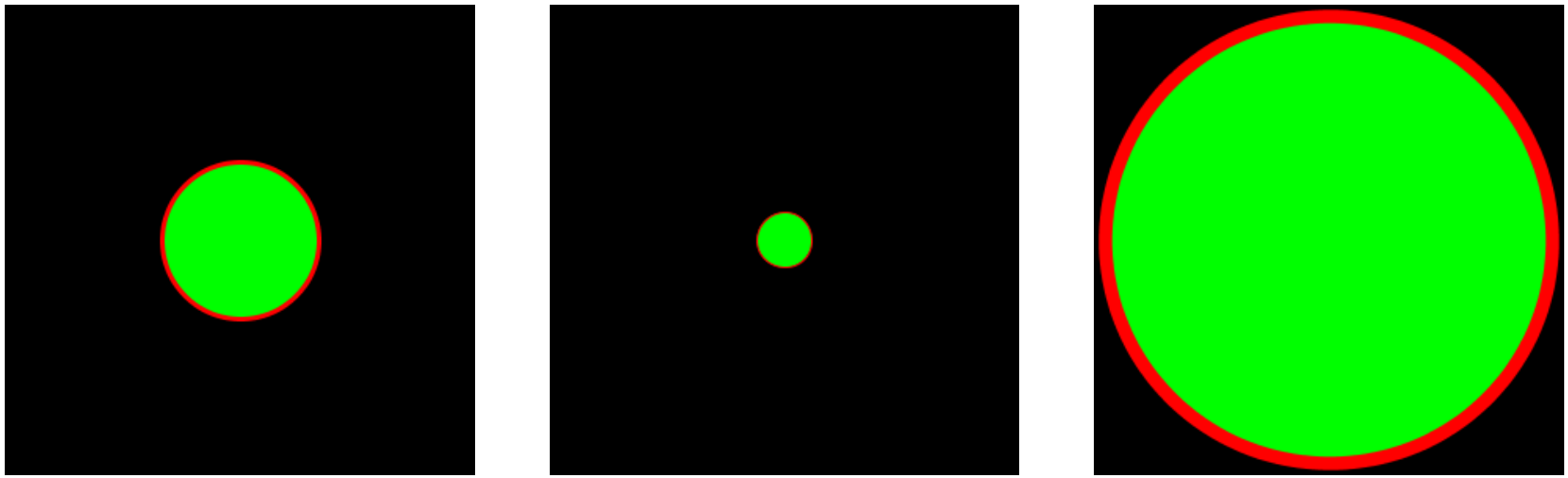
```
1   function setup()
2   {
3     createCanvas(400, 400);
4
5     ellipseMode( RADIUS );
6     background( 220 );
7   }
8
9   function draw()
10  {
11
12  }
13
14  function mousePressed()
15  {
16    switch( mouseButton )
17    {
18      case LEFT:
19        ellipse( mouseX, mouseY, random( 30 ) + 1, random( 30 ) + 1 );
20        break;
21
22      case RIGHT:
23        fill( random( 256 ), random( 256 ), random( 256 ) );
24        break;
25    }
26  }
```

Przykładowa zawartość okna podczas użytkowania programu.



```
1   let s = 1.0;
2
3   function setup()
4   {
5     createCanvas( 300, 300 );
6
7     ellipseMode( RADIUS );
8   }
9
10  function draw()
11  {
12    background( 0 );
13    fill( 0, 255, 0 );
14    stroke( 255, 0, 0 );
15    strokeWeight( 3 );
16
17    push();
18    translate( width / 2, height / 2 );
19    scale( s );
20    ellipse( 0, 0, 50, 50 );
21    pop();
22  }
23
24  function mouseWheel(event)
25  {
26    if( event.delta > 0.0 )
27      s *= 1.1;
28    else
29      s /= 1.1;
30  }
```

Przykładowa zawartość okna (od lewej): po uruchomieniu, po ruchu kółkiem od siebie, po ruchu kółkiem do siebie.

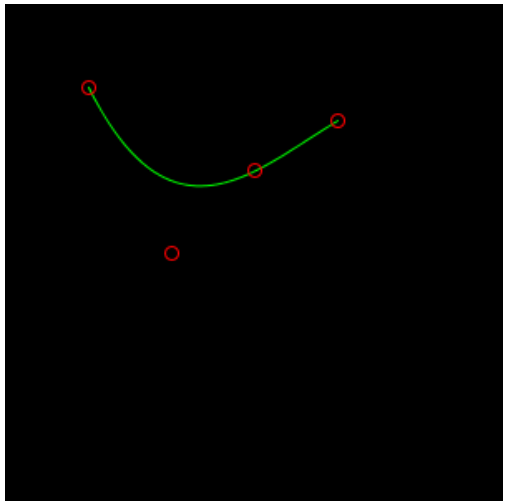


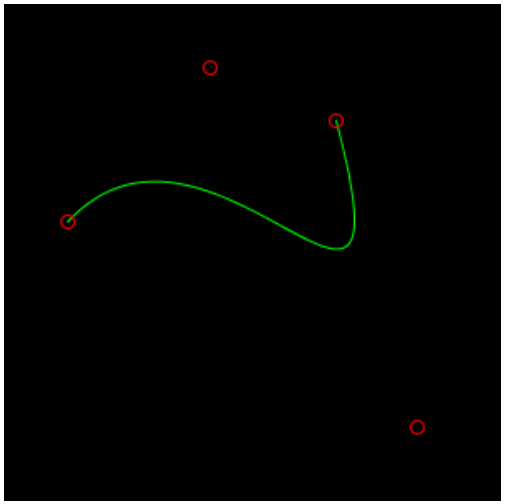
Zauważmy, że brzeg naszego kształtu również poddawany jest skalowaniu. Jeśli chcemy, aby był stałej grubości, to musimy odpowiednio go skalować używając funkcji `strokeWeight`.

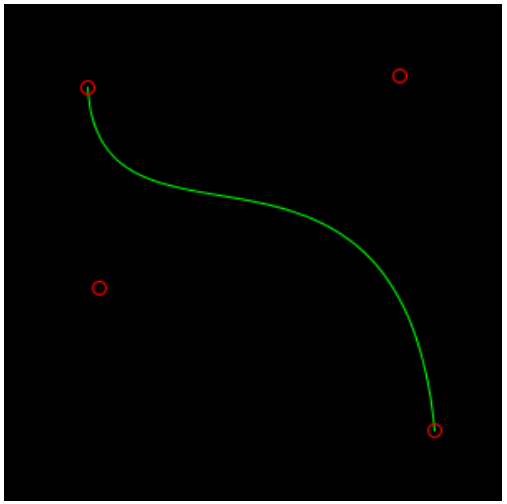
```
1   let cp = []; // punkty kontrolne krzywej
2   let n = -1; // numer wybranego punktu kontrolnego (-1 -- zaden punkt nie jest
        wybrany)
3
4   function setup()
5   {
6     createCanvas(300, 300);
7
8     cp.push( createVector( 50, 50 ) );
9     cp.push( createVector( 100, 150 ) );
10    cp.push( createVector( 150, 100 ) );
11    cp.push( createVector( 200, 70 ) );
12  }
13
14  function draw()
15  {
16    background( 0 );
17
18    noFill();
19    stroke( 0, 255, 0 );
20    bezier( cp[0].x, cp[0].y, cp[1].x, cp[1].y, cp[2].x, cp[2].y, cp[3].x,
          cp[3].y );
21
22    stroke( 255, 0, 0 );
23    for( let i = 0; i < cp.length; ++i )
24      ellipse( cp[i].x, cp[i].y, 8, 8 );
25  }
26
27  function mousePressed()
```

```
28  {
29    for( let i = 0; i < cp.length; ++i )
30    {
31      if( dist( cp[i].x, cp[i].y, mouseX, mouseY ) < 5 )
32      {
33        n = i;
34        break;
35      }
36    }
37  }
38
39  function mouseReleased()
40  {
41    n = -1;
42  }
43
44  function mouseDragged()
45  {
46    if( n > -1 )
47    {
48      cp[n].x = mouseX;
49      cp[n].y = mouseY;
50    }
51  }
```

Przykładowa zawartość okna dla kilku różnych położeń punktów kontrolnych.







Obsługa klawiatury w p5.js

Podobnie jak w przypadku myszki tak i w przypadku klawiatury część obsługi dostępna jest w każdym miejscu naszego programu, a część w specjalnych funkcjach do obsługi zdarzeń.

Zaczniemy od sprawdzenia czy został naciśnięty jakikolwiek klawisz na klawiaturze. Do tego celu używamy zmiennej boolowskiej `keyIsPressed`, która przyjmuje wartość `true` jeśli naciśnięto jakikolwiek klawisz, a `false` w przeciwnym przypadku.

Klawisze w p5.js dzielą się na dwa rodzaje. Pierwszym są klawisze ASCII tzn. klawisze, których symbol znajduje się w tablicy kodów ASCII. Drugim rodzajem jak nietrudno się domyślić są klawisze nie-ASCII.

W zależności od rodzaju klawisza sprawdzenie jaki klawisz został naciśnięty wygląda trochę inaczej.

Dla klawiszy ASCII sprawdzamy wartość zmiennej `key`. Możemy ją porównać do dowolnej stałej znakowej, np. `'s'`.

W przypadku klawiszy nie-ASCII porównujemy zmienną `keyCode` do odpowiedniego kodu klawisza. Dla niektórych klawiszy w p5.js zdefiniowano stałe reprezentujące odpowiednie kody. Są to:

`BACKSPACE`, `DELETE`, `ENTER`, `RETURN`, `TAB`, `ESCAPE`, `SHIFT`, `CONTROL`, `OPTION`, `ALT`, `UP_ARROW`, `DOWN_ARROW`, `LEFT_ARROW`, `RIGHT_ARROW`.

Dla klawiszy, których nie ma liście powyżej odpowiednie kody możemy znaleźć na stronie:

http://keycode.info/

Sposób ze zmienną `keyCode` działa również dla klawiszy ASCII.

```
1   function setup()
2   {
3     createCanvas(300, 300);
4   }
5
6   function draw()
7   {
8     background( 0 );
9
10    if( keyIsPressed )
11    {
12      switch( keyCode )
13      {
14        case 87: // w
15          background( 255 );
16          break;
17
18        case UP_ARROW:
19          background( 255, 0, 0 );
20          break;
21
22        case 112: // F1
23          background( 0, 255, 0 );
24          break;
25      }
26    }
27  }
```

Dostępne zdarzenia związane z klawiaturą:

- naciśnięcie klawisza

```
1  function keyPressed()
2  {
3    // instrukcje
4  }
```

- zwolnienie naciśniętego klawisza

```
1  function keyReleased()
2  {
3    // instrukcje
4  }
```

- naciśnięcie i przytrzymanie klawisza (ignorowane są klawisze specjalne np. Ctrl, Shift, Alt)

```
1  function keyTyped()
2  {
3    // instrukcje
4  }
```

W p5.js mamy jeszcze jedną funkcję związaną z obsługą klawiatury, a mianowicie

```
keyIsDown( code )
```

Funkcja sprawdza czy podany klawisz `code` jest naciśnięty i zwraca wartość boolowską. Funkcję tę możemy wykorzystać przy poruszających się obiektach.

```
1  let x = 10;
2  let y = 10;
3  let c;
4
5  function setup()
6  {
7    createCanvas( 300, 300 );
8
9    c = color( 255 );
10 }
11
12 function draw()
13 {
14   background( 0 );
15   fill( c );
16
17   ellipse( x, y, 20, 20 );
18
19   update();
20 }
21
```

```
22  function update()
23  {
24    const delta = 5;
25
26    if( keyIsDown( LEFT_ARROW ) )
27      x -= delta;
28    else if( keyIsDown( RIGHT_ARROW ) )
29      x += delta;
30    else if( keyIsDown( UP_ARROW ) )
31      y -= delta;
32    else if( keyIsDown( DOWN_ARROW ) )
33      y += delta;
34  }
35
36  function keyPressed()
37  {
38    if( key == 'c' )
39      c = color( random( 256 ), random( 256 ), random( 256 )
          );
40  }
```