Primjena staničnih automata u generativnoj umjetnosti

Senkić, Ivana

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ivana Senkić

PRIMJENA STANIČNIH AUTOMATA U GENERATIVNOJ UMJETNOSTI

Diplomski rad

Voditelj rada:
prof. dr. sc. Ivica Nakić

Zagreb, rujan 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Generativna umjetnost je vrsta suvremene umjetnosti u kojoj je nastanak djela većim dijelom automatiziran, odnosno nastaje uz pomoć računala. Umjetnik, stvaralac djela, najčešće programskim kodom zadaje pravila po kojima će se stvoriti njegovo djelo, a zatim u cijelu stvar unosi određeni dio slučajnosti. Na taj način, svako umjetničko djelo se ne može ponoviti, već ostaje jedinstveno. Pravila koja umjetnik zadaje potječu iz različitih znanosti: matematike, kemije, biologije, robotike... Ono što je bitno kod generativne umjetnosti jest da cijeli proces nastanka djela ne može biti pod kontrolom autora i treba biti u nekoj mjeri nepredvidljiv.

Jedan od popularnih, ali i jednostavnih alata za kreiranje takvih djela je programski jezik Processing. U ovom radu, opisat ćemo osnove Javascript inačice, p5.js, tog jezika te na kraju implementirati par primjera djela generativne umjetnosti u čijoj pozadini "žive" stanični automati.

Poglavlje 1

Biblioteka p5.js

1.1 Uvodno o p5.js

p5.js je JavaScript inačica programskog jezika Processing, nastala s ciljem da programiranje približi umjetnicima, dizajnerima i početnicima te ih prenese na internet. Kao i u JavaScriptu, u p5.js se crta na *canvasu*. Međutim, crtanje nije ograničeno samo taj dio, već se može koristiti i cijela web-stranica. Kako bi olakšali takvo korištenje p5.js, on nudi razne dodatne biblioteke pomoću kojih se može manipulirati drugim HTML5 elementima, poput teksta, videa, kamere i zvuka.

Glavna karakteristika ove biblioteke jest što se na vrlo jednostavan način, u malo linija koda mogu postići odlični crteži.

p5.js je **FLOOS** projekt (free/libre/open source software) i može se preuzeti s [7]. Programi za p5.js se mogu pisati u najobičnijem uređivaču teksta (službeni uređivač je još u razvoju).

1.2 Osnovna sintaksa

Sintaksa p5.js je skoro identična JavaScriptu:

- razlikuje mala i velika slova
- komentari: jednolinijski počinju s //, a blokovski su unutar /* */
- naredbe završavaju s ;
- blokovi unutar naredbi se nalaze između

Međutim, p5.js ima dodatnih mogućnosti za grafiku i interakciju s objektima. Varijable se deklariraju s ključnom riječi `var` ili `let`. Doseg varijable deklarirane s `var` je globalni

ili funkcijski, dok je doseg varijable deklarirane s `let` blokovski. p5.js ima nekoliko specijalnih varijabli poput `width` i `height` (širina i visina *canvasa*), `PI`, `mouseX`, `mouseY` (x i y koordinata kursora miša), `windowsWidth` i `windowsHeight` (širina i visina prozora preglednika) i druge.

Primitivni tipovi su kao u JavaScriptu: `undefined`, `null`, `Boolean`, `Number`, `String` i `Symbol`. Složeniji tipovi podataka su `Object`, funkcije i `Array`. Operatori su također isti, osim operatora == i !=, koji su kao u jeziku C. `if, switch, while, do...while, for, for...in, break, continue, throw, try, catch` su isti kao u JavaScriptu.

Nakon što smo preuzeli p5.js biblioteku sa službene stranice, otvorimo mapu naziva empty-example. U njoj se nalaze dvije datoteke, index.html i sketch.js. U datoteci index.html je u <script> tagu uključena biblioteka p5.js, ali i datoteka sketch.js. Upravo u toj datoteci se pišu p5.js programi, koji se još nazivaju *sketchevi*. Kad je otvorimo njen sadržaj je sljedeći:

```
function setup() {

}

function draw() {

}
```

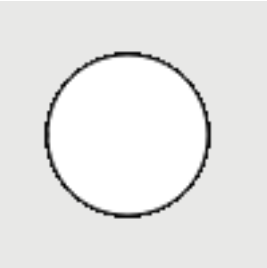
Glavne funkcije u kojima se piše program su `setup()` i `draw()`. Dio programa u kojem se najčešće inicijaliziraju globalne varijable, definira *canvas*, brzina crtanja (`frameRate`) i učitavaju podaci pripadaju `setup()` bloku. Drugi blok, kako mu i samo ime kaže, služi za crtanje i animaciju objekata. Razlika ovih dviju funkcija, osim njihove namjene, jest što se funkcija `setup()` poziva samo jednom na početku programa, dok se funkcija `draw()` poziva u beskonačnoj petlji. Zadano je da se izvodi 60 puta u sekundi. Naravno, postoje funkcije kojima možemo zaustaviti neprekidno izvođenje `draw()`, ali je i ponovno pokrenuti. Svaki p5.js *sketch* treba imati barem jednu od ove dvije funkcije *ili* se može koristiti tzv. *instance mode* u kojem imamo veću kontrolu nad učitavanjem *sketcha*.

Područje web-stranice se mjeri u pikselima pa je mjerna jedinica u p5.js piksel. Prostor gdje se crta p5.js tretira kao Kartezijev koordinatni sustav, gdje svaki piksel odgovara jednoj točki s x i y koordinatama. Koordinatni sustav u p5.js ima ishodište u gornjem lijevom kutu web-stranice. x-os je usmjerena prema desno, a y-os prema dolje.

1.3 Hello World!

Prvi primjer programa kojim demonstriramo osnovnu sintaksu je „Hello World" program. U programskim jezicima poput C, C++ ili Jave ovakav program ispisuje na standardni izlaz poruku „Hello World!". U p5.js, on izgleda malo drugačije:

```
function setup() {
  createCanvas();
}

function draw() {
  ellipse(50, 50, 60, 60);
}
```

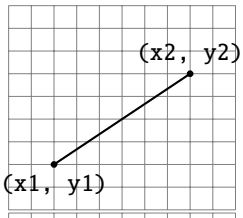


Slika 1.1: Kružnica

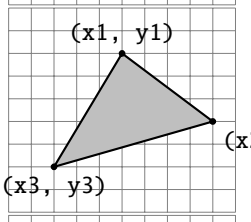
Ovaj program će nacrtati kružnicu na sredini ekrana. Pojasnimo funkcije koje smo koristili: funkcija `createCanvas()` stvara JavaScript *canvas*, zadane veličine 100x100 px i pridodaje ga na tijelo web-stranice. Ona kao argumente prima širinu i visinu *canvasa*. Obično se ta funkcija poziva prva u `setup()` funkciji.
Funkcija `ellipse(200, 200, 100, 100)` će nacrtati elipsu na koordinatama (200, 200), širine 100 px i visine 100 px (velika poluos i mala poluos iznose 50), odnosno nacrtat će kružnicu radijusa 50. Program pokrenemo tako da datoteku index.html otvorimo u bilo kojem web-pregledniku (npr. Firefox, Opera, Chrome).

1.4 Osnovni grafički elementi

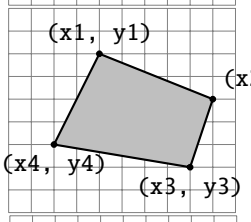
p5.js pruža nekoliko funkcija za crtanje osnovnih grafičkih elemenata: dužina, kružni luk, trokut, elipsa, pravokutnik i četverokut.



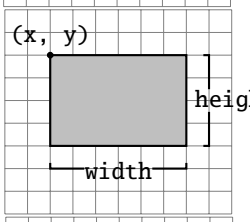

```
line(x1, y1, x2, y2)
```



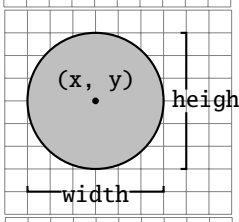

```
triangle(x1, y1, x2, y2, x3, y3)
```



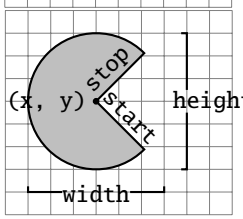

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```




```
rect(x, y, width, height)
```




```
ellipse(x, y, width, height)
```

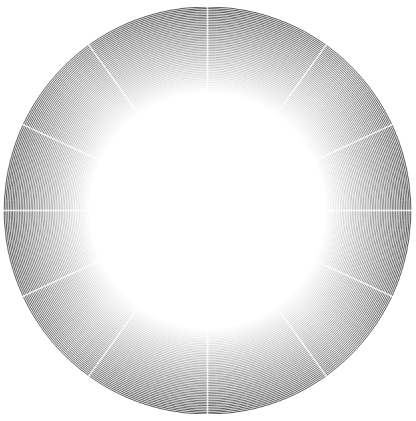



```
arc(x, y, width, height, start, stop)
```

Slika 1.2: Funkcije za crtanje osnovnih grafičkih elemenata

Pogledajmo par primjera korištenja ovih funkcija.

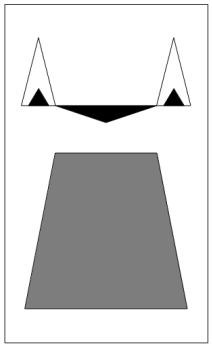
```
function setup() {
  createCanvas(windowWidth, windowHeight);
}

function draw() {
  background(255);

  strokeWeight(1);
  for(let i = 0; i <= 100; ++i){
    stroke(i*5);
    ellipse(width/2, height/2, height - i*5, height - i*5);
  }

  strokeWeight(2);
  line(width/2, 0, width/2, height);
  line(0, height/2, width, height/2);
  line(0, 0, width, height);
  line(width, 0, 0, height);
  line(width/3, 0, 2*width/3, height);
  line(2*width/3, 0, width/3, height);
}
```



Slika 1.3: Primjer crtanja kružnice i dužina

Funkcijom `strokeWeight()` određujemo debljinu ruba oblika, a funkcijom `stroke()` njegovu boju. Vrijednosti njenih argumenata mogu biti između 0 i 255. Ako joj proslijedimo jedan argument, oblici koje se crtaju nakon poziva te funkcije će biti ispunjeni nijansom sive boje. Ako želimo rubove likova bojati različitim bojama, ovoj funkciji ćemo proslijediti tri argumenta, odnosno RGB vrijednosti željene boje. Na sličan način, mijenjamo boju pozadine, pozivajući funkciju `background()`. Također, možemo postaviti prozirnost boje, tako da postavimo četvrti argument ovih funkcija, čija je vrijednost također između 0 i 255 (ako je vrijednost 0, boja je prozirna). Ako ne želimo da oblici imaju rub, pozovemo funkciju `noStroke()`.

```
function setup() {
  createCanvas(windowWidth, windowHeight);
}

function draw() {
  background(255);

  // body
  fill(255);
  rect(100, 100, 300, 500);

  // eyes
  triangle(150, 150, 175, 250, 125, 250);
  triangle(350, 150, 375, 250, 325, 250);
  fill(0);
  triangle(150, 225, 135, 250, 165, 250);
  triangle(350, 225, 335, 250, 365, 250);

  // beak
  triangle(175, 250, 325, 250, 250, 275);

  // fur
  fill(125);
  quad(175, 320, 325, 320, 370, 550, 130, 550);
}
```



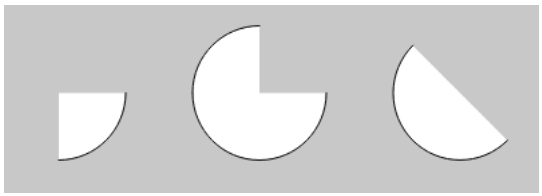
Slika 1.4: Primjer crtanja trokuta, pravokutnika i četverokuta

Funkcija `fill()` mijenja boju ispune nacrtanih oblika. Njeni parametri su posve isti kao i za funkcije `background()` i `stroke()`. Ako ne želimo da oblici imaju ispunu možemo pozvati funkciju `noFill()`. Ono što je bitno napomenuti kod spomenutih funkcija jest da se njihove vrijednosti primjenjuju na oblike koji se crtaju nakon njihovog poziva.

U ovom primjeru naglasimo da je bitan redoslijed crtanja likova: najprije se nacrtao pravokutnik, a zatim ostali likovi. Da su se na primjer, bijeli trokuti nacrtali poslije crnih, crni se više ne bi vidjeli. Naime, p5.js crta likove u onom redoslijedu u kojem su napisani u programu tako da onaj lik koji je zadnji u programu se nalazi na vrhu svih ostalih na *canvasu*.

Kod crtanja pravokutnika, prva dva argumenta označavaju koordinate gornjeg lijevog vrha. Međutim, koristeći funkciju `rectMode()` možemo mijenjati mjesto početka crtanja pravokutnika: na primjer da crtamo od središta pravokutnika ili da zadamo koordinate nasuprotnih vrhova. Za elipsu, postoji funkcija `ellipseMode()`.

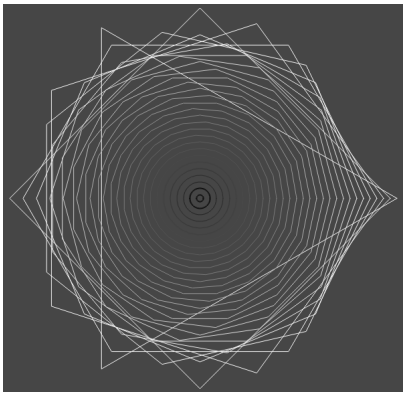
```
function setup() {
  createCanvas(windowWidth, windowHeight);
}

function draw() {
  background(200);

  arc(width/3, height/2, 100, 100, 0, HALF_PI);
  arc(width/3 + 150, height/2, 100, 100, 0, PI + HALF_PI);
  arc(width/3 + 300, height/2, 100, 100, QUARTER_PI, PI + QUARTER_PI);
}
```



Slika 1.5: Primjer crtanja kružnog luka

Prilikom crtanja kružnih lukova, prva dva argumenta određuju koordinate mjesta, treći i četvrti širinu i visinu. Peti argument je kut od kojeg se luk počinje crtati, a šesti argument je kut do kojeg se luk crta. Obje vrijednosti se zadaju u radijanima i rastu u smjeru kazaljke na satu. Pri tome mogu pomoći ugrađene konstante: `QUARTER_PI, HALF_PI, PI` i `TWO_PI`. Ako pak želimo pisati kuteve u stupnjevima, to možemo pomoću funkcije `radians()` koja pretvara stupnjeve u radijane. Drugi način jest da zadamo da cijeli *sketch* koristi stupnjeve. To postižemo funkcijom `angleMode()` koja kao argumente uzima ključne riječi `DEGREES` ili `RADIANS`. Ona se najčešće poziva u funkciji `setup()`.

Pored ovih osnovnih oblika, u p5.js možemo crtati proizvoljne oblike. Takve oblike crtamo između poziva funkcija `beginShape()` i `endShape()`, a pojedine vrhove željenog oblika pozivom funkcije `vertex()`.



Slika 1.6: Primjer korištenja funkcija `beginShape(), endShape() i vertex()`

```
var x, y, n, r;

function setup() {
  createCanvas(windowWidth, windowHeight);
  background(70);
  noFill();

  x = width / 2;
  y = height / 2;
  n = 3;
  r = height / 2;
}

function draw() {

  if(r <= 0)
    noLoop();

  let angle = TWO_PI / n;
  r -= 10;
  n += 1;
  stroke(r);
  beginShape();
  for(let a = 0; a < TWO_PI; a += angle){
    let x1 = x + r*cos(a);
    let y1 = y + r*sin(a);
    vertex(x1, y1);
  }
  endShape(CLOSE);
}
```

Funkciji `beginShape()` možemo proslijediti različite argumente poput `POINTS`, `LINES`, `TRIANGLES`, `QUADS`, `QUAD_STRIP` ovisno koje oblike želimo nacrtati. Ako joj ne proslijedimo argumente, lik može biti proizvoljan nepravilni poligon. Funkcija `endShape()` se može pozvati samo nakon `beginShape()`. Ako joj proslijedimo argument `CLOSE`, vrhovi definirani s funkcijom `vertex()` će biti spojeni u onom redoslijedu kojim su navedeni u kodu. Transformacije poput translacije, rotacije i skaliranja nemaju utjecaj unutar `beginShape()`. Osim toga, pozivanje funkcija poput `ellipse()` i `rect()` neće funkcionirati unutar `beginShape()`.

1.5 Transformacije

Zadani koordinatni sustav u p5.js ima ishodište u gornjem lijevom kutu te x-os umjerenu prema desno, a y-os prema dolje. Ovaj sustav možemo modificirati transformacijama. Koordinatni sustav se može translatirati, rotirati i skalirati tako da likovi koje crtamo mogu imati drugačije pozicije, orijentaciju i dimenzije.

Translacija

Funkcija `translate()` translatira početno ishodište u drugu točku. Prima dva argumenta, prvi je pomak (*offset*) x koordinate, a drugi pomak y koordinate. Vrijednosti argumenata se zbrajaju s koordinatama oblika koji se crtaju nakon poziva funkcije. Npr. ako smo pozvali `translate(15, 20)` i nakon toga `rect(0, 5, 30, 40)` gornji lijevi vrh pravokutnika će imati koordinate (15, 25).

Funkcija `translate()` je aditivna, tj. ako je pozovemo `translate(15, 20)` dva puta, to je ekvivalentno jednom pozivu `translate(30, 40)`. Važno je napomenuti da, iako se sve transformacije unutar `draw()` funkcije akumuliraju, kada se dođe do kraja `draw()`, sve se transformacije resetiraju na zadane vrijednosti.

Kompozicija svih primijenjenih transformacija se pamti unutar matrice transformacije. Kako bi pažljivije i lakše vodili računa o transformacijama koristimo `push()` i `pop()` funkcije. `push()` pamti trenutnu matricu transformacije i postavke stila poput ispune likova, boje, debljine ruba i slično. `pop()` funkcija vraća ove postavke. One nam omogućuju da pojedine promjene „izoliramo" unutar programa kako ne bi utjecale na cijeli *sketch*. `push()` i `pop()` dolaze u paru: za svaki `push()` mora postojati `pop()`.

Rotacija

Funkcija `rotate()` rotira koordinatni sustav. Ona prima jedan argument, a to je kut rotacije. Uvijek se rotira oko (0, 0) (tj. ishodišta) u smjeru kazaljke na satu. Kao i kod translacije, rotacija je također aditivna funkcija. Pokažimo primjer u kojem koristimo translaciju i rotaciju kako bi rotirali zvijezdu oko njenog središta.

```
function setup() {
  createCanvas(windowWidth, windowHeight);
}

function draw() {

  background(102);
```

```
  push();
  translate(width/2, height/2);
  rotate(frameCount / -100);
  star(0, 0, 50, 125, 6);
  pop();

}

function star(x, y, radius1, radius2, npoints) {
  var angle = TWO_PI / npoints;
  var halfAngle = angle/2.0;
  beginShape();
  for (var a = 0; a < TWO_PI; a += angle) {
    var sx = x + cos(a) * radius2;
    var sy = y + sin(a) * radius2;
    vertex(sx, sy);
    sx = x + cos(a+halfAngle) * radius1;
    sy = y + sin(a+halfAngle) * radius1;
    vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

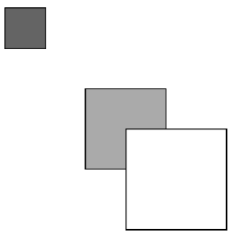
Slika 1.7: Primjer rotacije i translacije s [2]

Najprije translatiramo ishodište na željeno mjesto na *canvasu*, zatim rotiramo koordinatni sustav za određeni kut i na kraju nacrtamo željeni lik tako da je njegovo središte na koordinatama (0, 0). Na početku `draw()` pozivamo funkciju `background()` kako bi „očistila” *canvas* i dala iluziju da se „jedna” zvijezda okreće oko svoje osi.

Skaliranje

Funkcija `scale()` skalira svaki sljedeći nacrtani oblik. Prima jedan argument koji se tretira kao decimalni postotak. Naime, poziv `scale(2.0)` će povećati veličinu sljedećih oblika na 200%, a `scale(0.3)` će smanjiti dimenziju na 30%. Pozovemo li najprije `scale(1.5)`, a zatim `scale(2.0)` to je isto kao da smo pozvali `scale(3.0)`, odnosno primijenjene vrijednosti skaliranja se množe.

```
function setup() {
  createCanvas(windowWidth, windowHeight);
  background(255);
}

function draw() {
  fill(170);
  rect(200, 200, 100, 100);
  scale(0.5);
  fill(100);
  rect(200, 200, 100, 100);
  scale(2.5);
  fill(255);
  rect(200, 200, 100, 100);
}
```



Slika 1.8: Primjer skaliranja pravokutnika

1.6 Random

Nepredvidive događaje u svijetu, poput opadanja listova, leta leptira, padanja kapljica itd., možemo simulirati generiranjem slučajnih brojeva. U p5.js s funkcijom `random()` možemo dobiti slučajne brojeve ili na slučajan način izabrati element polja. Pozovemo li `random()`, funkcija će vratiti slučajan broj u intervalu $[0, 1\rangle$, a ako pozovemo `random(5)` kao rezultat ćemo dobiti broj u intervalu $[0, 5\rangle$.

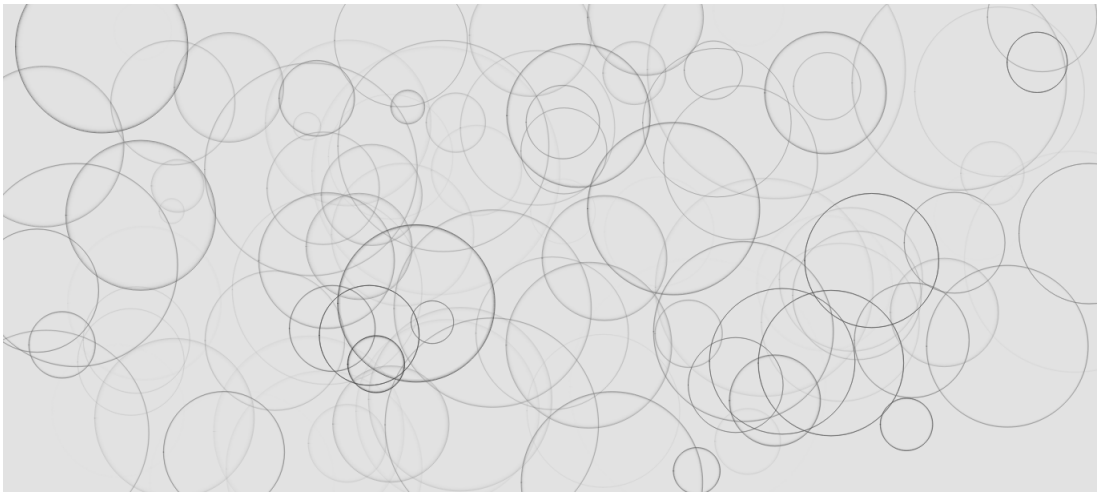
`random(['lubenica', 'smokva', 'kivi', 'breskva'])` će vratiti slučajan element iz danog polja, a `random(-5, 5)` će vratiti broj iz intervala $[-5, 5\rangle$.

Samom `random()` funkcijom, nećemo postići *oku ugodne* rezultate. No, kombinirajući je s trigonometrijskim funkcijama i/ili `for` petljom dolazimo do zanimljivih crteža.

U sljedećem primjeru ćemo simulirati padanje kapi kiše. Kapi ćemo nacrtati pomoću kružnica, a njihove položaje, radijus, boju i brzinu širenja odabrat ćemo na slučajan način.

```
var N = 100;
var drops = new Array();

function setup(){
  createCanvas(windowWidth, windowHeight);
  noFill();

  for(let i = 0; i < N; ++i){
    let x = random(width);
    let y = random(height);
    let d = random(200);
    let speed = random(2);
    drops.push(new drop(x, y, speed, d));
  }
}

function draw(){
  background(225, 120);
  for(let i = 0; i < N; ++i)
    drops[i].display();
}

function drop(x, y, speed, diameter){
  this.x = x;
  this.y = y;
```

```
  this.speed = speed;
  this.diameter = diameter;
  this.fade = random(200);
  this.color = random(30, 150);

  this.display = function(){
    stroke(this.color, this.fade);
    ellipse(this.x, this.y, this.diameter, this.diameter);
    this.diameter += this.speed;
    this.fade -= this.speed;
    if(this.diameter > height){
      this.diameter = random(200);
      this.fade = 0;
    }
    if(this.fade < 0){
      this.diameter = random(200);
      this.fade = random(200);
    }
  }
}
```



Slika 1.9: Simulacija padanja kapi kiše

Kap kiše je implementirana u klasi `drop`. Njen konstruktor za argumente prima x i y koordinatu središta kružnice, promjer i brzinu širenja. U metodi `display()` crtamo

kružnicu i ažuriramo vrijednosti njenog promjera i boje ruba. Ako te vrijednosti prijeđu određenu granicu, na slučajan način ih resetiramo. U funkciji `setup()` inicijaliziramo polje `drops`. U njemu pohranjujemo `N` varijabli tipa `drop`. U funkciji `draw()` prolazimo kroz petlju i pozivamo na svakom elementu polja `display()` metodu.

1.7 Interakcija

Dosadašnji programski primjeri su bili statični, tj. pokrenuli smo program i čekali da se izvrši te nisu zahtjevali neki unos (eng. *input*) od samog korisnika. U p5.js korisnik može mišem, unosom s tipkovnice ili čak dodirom na ekran mijenjati boju, položaj ili veličinu oblika.

Miš

Neki podaci o kursoru miša koje p5.js ima ugrađene su: `mouseX`, `mouseY`, `pmouseY`, `pmouseY`, `mouseIsPressed`, `mouseButton`... `pmouseY` i `pmouseY` sadrže x i y koordinatu prethodne pozicije kursora miša, `mouseIsPressed` je `boolean` sistemska varijabla koja je `true` ako smo pritisnuli tipku miša, a `mouseButton` je sistemska varijabla koja vraća `LEFT`, `MIDDLE` ili `RIGHT` ovisno o tome koju tipku miša smo pritisnuli.

Korištenjem funkcija poput `mouseMoved()`, `mouseDragged()`, `mousePressed()`, `mouseReleased()`, `mouseClicked()` ili `mouseWheel()` možemo kontrolirati razne događaje s mišem.

```
var c = 0;
var x = 200;
var y = 200;

function setup() {
  createCanvas(windowWidth, windowHeight);
  rectMode(CENTER);
  noStroke();
}

function draw() {
  background(255);

  fill(100);
  rect(x, y, 300, 300);
```

```
  fill(c);
  if(dist(mouseX, mouseY, x, y) <= 75)
    rect(x, y, 150, 150);
  else
    ellipse(x, y, 150, 150);
}

function mousePressed(){
  if(dist(mouseX, mouseY, x, y) <= 75)
    c = 255 - c;

  return false;
}

function mouseDragged(){
  x = mouseX;
  y = mouseY;

  return false;
}
```

Slika 1.10: Primjer interakcije s mišem, ako se kursor nalazi van kružnice, onda se ona pretvori u kvadrat

U ovom primjeru, ako pritisnemo tipku miša na središte velikog kvadrata i vučemo kursor po ekranu, oba lika će se pomicati zajedno s mišem. Pritisnemo li tipkom miša unutar manjeg lika, njegova ispuna će se promijeniti u bijelu. Koji će se manji lik nacrtati, kvadrat ili kružnica, ovisi o tome nalazi li se kursor miša izvan ili unutar manjeg lika.

U funkcijama `mousePressed()` i `mouseDragged()` smo vraćali vrijednost `false` zbog toga što preglednici imaju različita zadana ponašanja kod događaja s mišem. Kako bi spriječili zadana ponašanja, dodamo `return false;` na kraju ovih metoda.

Tipkovnica

p5.js osim događaja s mišem prati i događaje na tipkovnici, poput koja je tipka zadnja pritisnuta. Kao i `mouseIsPressed` varijabla, `keyIsPressed` je `boolean` varijabla koja je `true` ako smo pritisnuli tipkovnicu, a inače je `false`. Varijabla `key` sadrži informaciju koja je tipka zadnja pritisnuta. Međutim, ona ne razlikuje velika i mala slova. Zbog toga, preporuča se korištenje funkcije `keyTyped()`. Ako smo pritisnuli neku od specijalnih tipki poput `ENTER, RETURN` ili `SHIFT`, varijabla `keyCode` će sadržavati tu informaciju. U sljedećem primjeru pokazat ćemo kako se u p5.js može pisati tekst i to preko unosa s tipkovnice.

```
var x = 100;

function setup() {
  createCanvas(windowWidth, windowHeight);
  background(255);

  textSize(32);
  textAlign(CENTER);
  fill(0);
}

function draw() {
  text(key, x, height/2);
}

function keyPressed(){
  x += 32;
  return false;
}
```

Funkcijom `text()` pišemo tekst u p5.js. Prvi argument je `String, Object, Array, Number` ili `Boolean` varijabla koju želimo ispisati, a drugi i treći parametar su x i y koordinate točke od koje ćemo početi pisati tekst. Funkcija `keyPressed()` se poziva svaki put

kad se pritisne tipka na tipkovnici. Kao i kod funkcije `mousePressed()`, na kraj dodajemo liniju `return false;` kako bi spriječili zadano ponašanje pregledinka (npr. da se pritiskom tipke F5 osvježava web-stranica).

Slika 1.11: Primjer interakcije s tipkovnicom

Dodir

Kod *touch-screen* uređaja, p5.js pamti je li ekran dodirnut i njegovo mjesto dodira. Kao i kod prethodnih, „isPressed" varijabli, varijabla `touchIsDown` je `true` ako smo dodirnuli ekran, a inače je `false`. Varijable `touchX` i `touchY` sadrže x i y koordinate mjesta gdje smo dodirnuli ekran. Ako smo ekran dodirnuli na više mjesta istovremeno, u varijabli `touches` će biti pohranjena sva trenutna mjesta dodira s ekranom. `touches` je varijabla tipa `Array`, u kojoj svaki element ima x, y i id podatke. U sljedećem primjeru, svaki put kad dodirnemo ekran, nacrtat će se kružnica na mjestu dodira.

```
function setup() {
  createCanvas(windowWidth, windowHeight);
  fill(0, 102);
  noStroke();
}

function draw() {
  ellipse(touchX, touchY, 15, 15);
}
```

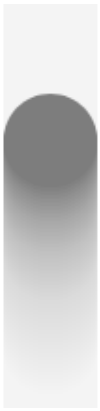
1.8 Animacija

Za sada smo likove mogli pomaknuti pomoću miša ili tipkovnice. Ono što možemo napraviti je da ih pustimo da se sami kreću po ekranu, neovisno o tome gdje smo postavili kursor miša. Likovima možemo zadati brzinu i smjer kretanja. Počnimo s nečim jednostavnim: krug koji se odbija od rub ekrana.

```
var centerX, centerY;
var diameter;
var speed;
var direction = -1;

function setup(){
  createCanvas(windowWidth, windowHeight);
  background(255);
  noStroke();

  diameter = 50;
  centerX = width/2 - diameter/2;
  centerY = height - diameter/2;

  speed = 2;
}

function draw(){
  background(255);

  if(centerY < diameter/2 || centerY > height - diameter/2)
    direction *= -1;

  fill(125);
  ellipse(centerX, centerY, diameter, diameter);

  centerY += speed*direction;
}
```

Na početku programa deklariramo varijable za x i y koordinatu središta kruga, njegov promjer te brzinu i smjer, a zatim ih inicijaliziramo u funkciji `setup()`. U funkciji `draw()` provjeravamo je li krug došao do ruba ekrana i ako jest, promijenimo mu smjer.

Na kraju ove funkcije izračunamo novi položaj kruga tako da y koordinati središta dodamo vrijednost brzine kretanja kruga.



Slika 1.12: Krug koji ostavlja trag za sobom

Ako želimo da krug ostavlja trag iza sebe, samo promijenimo liniju `background(255)` u `background(255, 10)`. Što je veća vrijednost drugog argumenta, trag iza kruga je manji.

Sljedeći korak je: više krugova koji se odbijaju.

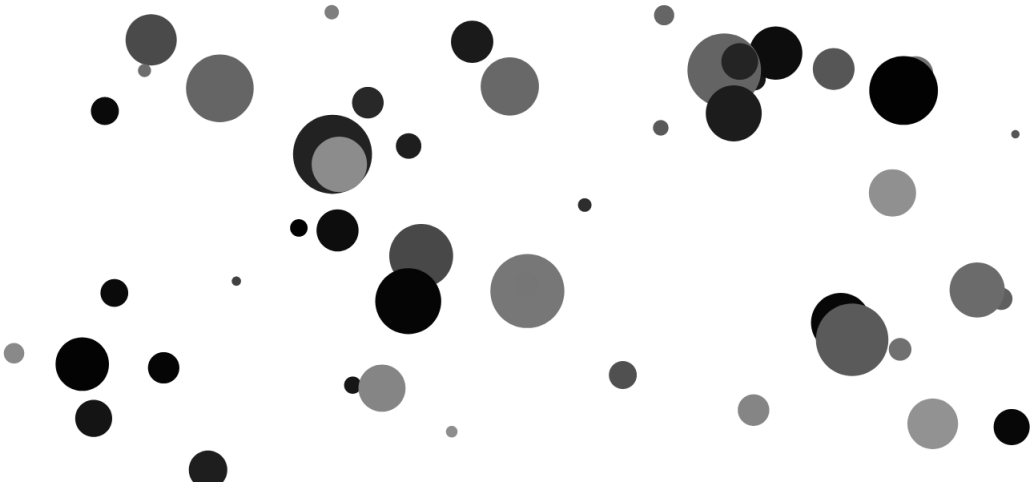
```
var N = 50;
var disks = new Array(N);

function setup(){
  createCanvas(windowWidth, windowHeight);
  background(255);
  noStroke();

  for(let i = 0; i < disks.length; ++i){
    let speed = random(2, 10);
    let diameter = random(10, 100);
    let x = random(width - diameter);
    let y = random(diameter/2, height - diameter/2);

    disks[i] = new Disk(x, y, diameter, speed);
  }
}
```

```
function draw(){
  background(255);

  for(let i = 0; i < disks.length; ++i)
    disks[i].display();
}

function Disk(centerX, centerY, diameter, speed){
  this.centerX = centerX;
  this.centerY = centerY;
  this.diameter = diameter;
  this.speed = speed;
  this.direction = random([-1, 1]);
  this.fill = random(150);

  this.display = function() {
    if(this.centerY < this.diameter/2 ||
       this.centerY > height - this.diameter/2)
        this.direction *= -1;

    fill(this.fill);
    ellipse(this.centerX, this.centerY, this.diameter, this.diameter);
    this.centerY += this.speed * this.direction;
  }
}
```

Definiramo klasu `Disk`, čiji konstruktor prima x i y koordinatu središta kruga, njenu brzinu i promjer. Početni smjer kretanja i boja kruga će se odabrati na slučajan način. U metodi `display()` klase `Disk` najprije provjeramo je li krug došao do ruba ekrana i ako jest, promijenimo njegov smjer kretanja. Zatim crtamo krug i ažuriramo koordinate njegovog središta. U polju `disks` ćemo pohraniti sve krugove koje ćemo crtati na *canvasu*. Elemente polja inicijaliziramo u funkciji `setup()`, a u funkciji `draw()` u petlji prolazimo kroz polje i na svakom njegovom elementu pozivamo funkciju `display()` kako bi ih nacrtali na *canvasu*.

Osim običnog odbijanja, vrlo jednostavno možemo simulirati i elastično gibanje:

```
var M = 0.8;  // mass
var K = 0.2;  // spring constant
```



Slika 1.13: Više krugova koji se odbijaju od rubove prozora

```
var D = 0.93; // damping
var R;        // rest position
var ypos;     // position
var v = 0.0;  // velocity
var a = 0;    // acceleration
var f = 0;    // force
var released = false;

function setup(){
  createCanvas(windowWidth, windowHeight);

  ypos = height/2;
  R = ypos;
}

function draw(){
  background(255);

  if(released) move();
  ellipse(width/2, ypos, 100, 100);
}

function move(){
```

```
  f = -K * (ypos - R);
  a = f / M;
  v = D * (v + a);
  ypos += v;

  if(abs(v) < 0.01){
    v = 0.0;
    released = false;
  }
}

function mouseMoved(){
  let distance = dist(mouseX, mouseY, width/2, ypos);
  if(distance < 100)
    stroke(255, 0, 0);
  else
    stroke(0);
}

function mouseDragged(){
  released = false;
  ypos = mouseY;
}

function mouseReleased(){
  released = true;
}
```

Na početku programa incijaliziramo varijable nužne za elastično gibanje, poput mase i početnog položaja kugle i konstante elastične opruge. U funkciji `move()` simuliramo gibanje kugle na elastičnoj opruzi: prema formuli izračunamo silu, akceleraciju, brzinu i novi položaj kugle. Kako bi pomaknuli kuglu iz početnog položaja, dovedemo kursor miša unutar nje i povučemo je gore ili dolje. Nakon što pustimo tipku miša, kugla će se gibati kao da se nalazi na elastičnoj opruzi.

U p5.js možemo simulirati i kompliciranija fizikalna gibanja, a primjeri se mogu pronaći u [5] i [6].

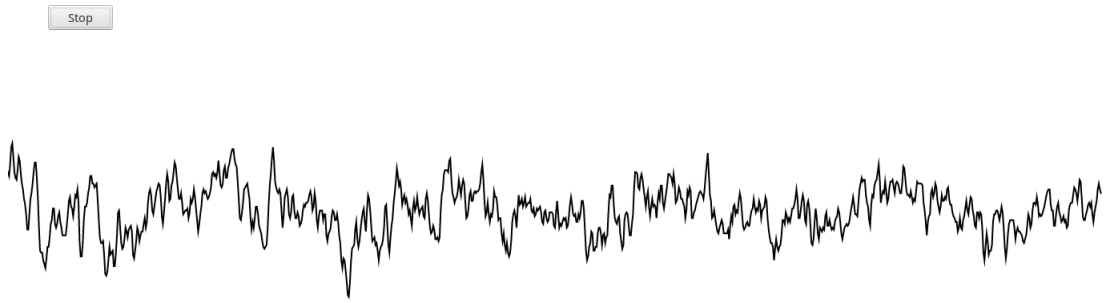
1.9 Dodatno

U radu smo obradili osnovne funkcije za rad u biblioteci p5.js, koja nudi još puno više toga kao što je rad sa DOM elementima, tabličnim i JSON podacima, slikama, zvukom i web-kamerom. Potrebne biblioteke se mogu preuzeti s [7], a zatim se dodaju u datoteku index.html unutar *taga* <script>.

Pokazat ćemo vrlo jednostavni primjer kako učitati zvuk, pokrenuti i zaustaviti ga s klikom na gumb te crtati zvučne valove.

```
var button;

var sound, fft;

function preload(){
  sound = loadSound("data/sample.mp3");
}

function setup(){
  createCanvas(windowWidth, windowHeight);

  button = createButton("Start");
  button.position(50, 50);
  button.mousePressed(playSound);

  sound.setVolume(1.0);
  fft = new p5.FFT();
}

function draw(){
  background(255);

  let wave = fft.waveform();
  noFill();
  beginShape();
  stroke(0);
  strokeWeight(2);
  for(let i = 0; i < wave.length; ++i){
    let x = map(i, 0, wave.length, 0, width);
    let y = map(wave[i], -1, 1, 0, height);
```

```
    vertex(x, y);
  }
  endShape();
}

function playSound(){

  if(button.html() == "Start"){
    sound.loop();
    button.html("Stop");
  }
  else{
    sound.stop();
    button.html("Start");
  }

}
```



Slika 1.14: Zvučni valovi na canvasu

Funkcija `preload()` u p5.js služi za učitavanje podataka. Preporučeno je nju koristiti jer se ona uvijek poziva prije funkcija `setup()` i `draw()` i tek po njenom završetku se ove dvije funkcije pozivaju. Na taj način smo se osigurali da su naši podaci u cjelosti učitani prije nego što krenemo s njima raditi.

Funkcija `loadSound()` učitava zvučnu datoteku i njoj prosljeđujemo put do željene datoteke. U funkciji `setup()` najprije crtamo DOM gumb kojim ćemo pokrenuti i zaustaviti zvuk. Taj gumb se dodaje na kraj tijela web-stranice i nije dio *canvasa*. Funkciji `mousePressed()` prosljeđujemo ime funkcije koja će se pozvati kada pritisnemo na gumb. `fft` je `p5.FFT` objekt iz koje možemo dobiti amplitude valova u frekvencijskoj i vremenskoj domeni.

U `draw()` funkciji ćemo pomoću `waveform()` dobiti upravo polje vrijednosti amplituda u vremenskoj domeni. S `map()` preslikavamo vrijednosti iz jednog raspona u drugi. Prvi argument je vrijednost koju želimo konvertirati, sljedeća dva su granice koje ta vrijednost može imati, a zadnja dva argumenta su granice ciljane vrijednosti. Pomoću `loop()` i `stop()` pokrenemo i zaustavimo datoteku. Umjesto `loop()` možemo koristiti i `start()`. U tom slučaju će se učitana datoteka samo jednom izvesti.

Poglavlje 2

Stanični automati. Igra života

Stanični automat (eng. *cellular automaton*) je autonomni sustav koji se sastoji od stanica i pravila koja određuju ponašanje pojedine stanice u odnosu na njene susjede. John von Neumann bio je među prvima koji je promatrao stanične automate 40-tih godina prošlog stoljeća, a postali su popularni kasnije, 1970. godine, kada je Martin Gardner objavio članak o staničnom automatu Igra života (*Game of Life*) Johna Conwaya. Automati su svoju popularnost stekli jer su mogli simulirati događaje u raznim područjima: biologiji, ekonomiiji, sociologiji, neuroznanosti i drugima.

Stanični automat se sastoji od mreže stanica proizvoljnih dimenzija. Svaka od stanica može biti u jednom od konačno mnogo stanja, na primjer živom ili neživom; *on*, *dying* ili *off*. Susjedstvo stanice definiramo kao skup stanica koje je okružuju. Početno stanje automata se zadaje tako da odredimo stanje svake stanice. Nova generacija stanica nastaje prema pravilima automata. Pravila automata određuju stanje stanice u odnosu na stanja njenih susjeda. Tipično, pravila se ne mijenjaju i primjenjuju se istovremeno na sve stanice u mreži.

Conwayeva Igra života je dvodimenzionalni stanični automat, čije stanice mogu biti u jednom od moguća dva stanja: živom ili neživom. Pravila automata su sljedeća:

- Ako živa stanica ima dva ili tri živa susjeda, ostaje živa. Inače, postaje neživa.
- Ako neživa stanica ima točno tri živa susjeda, postaje živom.

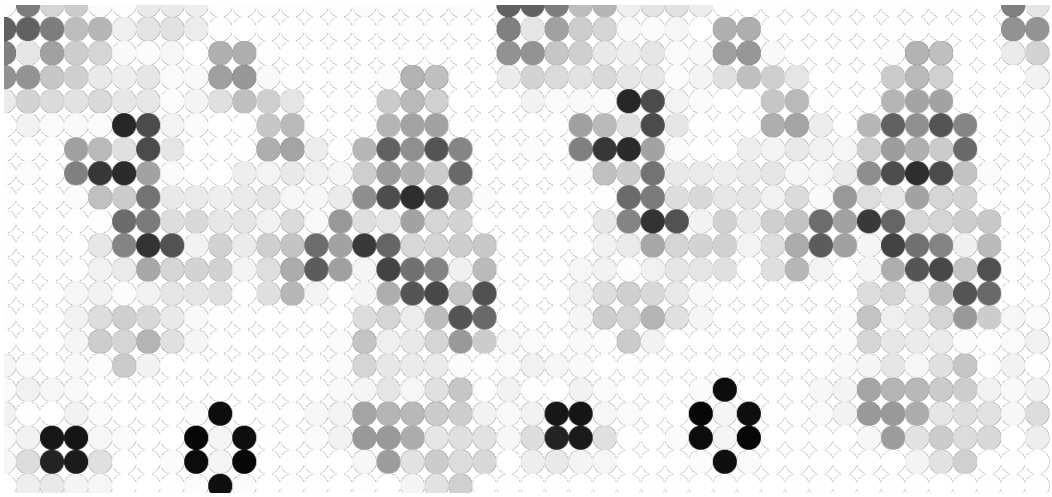
Najčešće vrste uzoraka (eng. *pattern*) ovih automata su: *still life*, oscilatori i *spaceship*. Uzorak *still life* se ne mijenja iz generacije u generaciju, oscilator uzorak dolazi do početnog stanja nakon konačno mnogo generacija, a *spaceship* je uzorak koji također dolazi do početnog stanja nakon konačnog broja generacija, ali se nalazi na drugom mjestu u mreži. Kod posljednja dva uzorka, najmanji broj generacija potreban da se uzorak opet vrati u početno stanje zove se period.

U sljedećem primjeru implementirat ćemo Igru života. Mreža automata će biti cijela web-stranica, a svaka stanica će biti kružnica, obojana u crno ili bijelo. Početno stanje mreže zadat ćemo na slučajan način.

```
var automaton;
var diametar = 30;
var M, N;

function setup(){

  createCanvas(windowWidth, windowHeight);
  noStroke();
  frameRate(15);

  M = int(height/diametar) + 1;
  N = int(width/diametar) + 1;

  automaton = new Array(M*N);
  for(let i = 0; i < M*N; ++i)
    automaton[i] = random([0, 1]);
}

function draw(){

  for(let i = 0; i < M; ++i)
    for(let j = 0; j < N; ++j){
      if(automaton[i*N + j])
        fill(0, 50);
      else
        fill(255, 50);

      let x = (j)*diametar;
      let y = (i)*diametar;

      ellipse(x, y, diametar, diametar);
    }

  nextGeneration();
}
```

```
function nextGeneration(){
  let temp = new Array(M*N);

  for(let i = 0; i < M; ++i)
    for(let j = 0; j < N; ++j){
      let neighbours = numNeighbours(i, j);
      let cell = automaton[i*N + j];

      if(cell && (neighbours == 2 || neighbours == 3))
        temp[i*N + j] = 1;
      else if(!cell && neighbours == 3)
        temp[i*N + j] = 1;
      else
        temp[i*N + j] = 0;
    }

  automaton = temp;
}

function numNeighbours(x, y){
  let num = 0;

  for(let i = -1; i < 2; ++i){
    for(let j = -1; j < 2; ++j){
      if(x+i >= 0 && y+j >= 0 &&
         x+i < M  && y+j < N  &&
         automaton[(x+i)*N + y+j])
          ++num;
    }
  }

  return (automaton[x*N + y] > 0) ? num-1 : num;
}
```

Na početku zadajemo promjer kružnice i izračunamo broj stanica po recima i stupcima mreže. Broj redaka pohranjujemo u varijablu `M`, a broj stupaca u `N`. Stanični automat ćemo reprezentirati varijablom `automaton` tipa `Array`. Svaki element polja će biti inicijaliziran na slučajan način na vrijednost 0 ili 1. U funkciji `draw()` u `for` petlji prolazimo kroz polje



Slika 2.1: Stanični automat, Igra života

`automaton` i ovisno o tome je li element polja 0 ili 1 boja kružnice će biti crna ili bijela. Na kraju `draw()` pozivamo funkciju `nextGeneration()` koja računa sljedeću generaciju stanica primjenjujući pravila Igre života. Funkcija `numNeighbours(x, y)` se koristi za računanje broja susjeda stanice koja se u polju nalazi na mjestu `x*N + y`.
U sljedećem poglavlju ćemo vidjeti kako možemo generirati zanimljive likove pomoću staničnih automata.

Poglavlje 3

Primjeri

U ovom poglavlju pokazat ćemo dva kompliciranija primjera korištenja biblioteke p5.js i staničnih automata kako bi nacrtali maslačke i lava lampu. U oba primjera, stanični automat „živi" u pozadini i generira likove na *canvasu*. Uz pomoć `random()` funkcije i raznih svojstava stanica poput starosti, broja susjeda, broja susjeda od susjeda i njihovom kombinacijom možemo kontrolirati, između ostalog, boju, veličinu, kut i položaj naših likova. Na taj način uspješno kombiniramo stroga pravila automata i nepredviljivost `random()` funkcije.

3.1 Maslačak

U ovom primjeru na *canvasu* će se nacrtati maslačak u pahuljastoj formi. U gornjem lijevom kutu ekrana postoji padajući izbornik u kojem možemo odabrati koliko maslačaka želimo da se nacrtaju te *slider* kojim možemo povećavati i smanjivati brzinu generiranja crteža.

Maslačak je implementiran kao klasa `Dandelion` čiji konstruktor kao argumente prima x i y koordinatu središta kružnice unutar koje će se crtati latice. Klasa ima varijable članice petals, x0, y0. `petals` je polje varijabli tipa `Petal` koje predstavljaju latice cvijeta. Metoda `addPetal` na osnovu svojih argumenata inicijalizira jednu varijablu tipa `Petal` i dodaje je na kraj polja `petals`.

Svaka latica maslačka će biti na *canvasu* nacrtana kao pravilni peterokut, unutar kojeg smo nacrtali pet parabola tehnikom šivanja krivulja. Svakoj latici pridružena je jedna živa stanica automata. Konstruktor klase `Petal` kao argumente prima x i y koordinatu središta peterokuta, radijus opisane njemu opisane kružnice, starost pripadne stanice, broj rupica potrebnih za šivanje parabole, indeks i indeks cvijeta kojem latica pripada. Klasa ima metodu `drawPetal` koja crta lik na *canvasu*. Boja latice ovisi o zbroju linearnog indeksa stanice u automatu i šuma. Položaj latice se određuje na slučajan način, radijus

opisane kružnice je jednak umnošku starosti stanice i kvadrata broja susjeda od susjeda. Broj rupica za šivanje parabole je jednak umnošku starosti i broja susjeda stanice. Metoda `myCell(i, j)` vraća `true` ako latica pripada stanici koja se nalazi na mjestu (i,j), a inače vraća `false`.

Klasa `GameOL` predstavlja stanični automat. Na početku programa iz datoteke se učita početno stanje automata. Zvjezdice u datoteci označavaju žive stanice, a crtice nežive. Klasa `GameOL` ima jednu varijablu članicu `GOL`. `GOL` je dvodimenzionalno polje u koje pohranjujemo elemente tipa `Cell`. Klasa `Cell` kao varijable članice ima i, j koordinate u polju i `life` koja označava starost ćelije. Prilikom učitavanja iz datoteke, ako smo učitali znak *, starost ćelije postavljamo na vrijednost 1. Klasa `GameOL` ima metodu `update()` kojom prelazimo u sljedeću generaciju automata. Kao argument uzima indeks cvijeta kojem automat pripada. Svaki put kad se pojavi živa stanica u automatu ili jedna stanica ostari, cvijetu dodajemo po jednu laticu. Metoda `numOfAliveCells` računa broj živih stanica, `size` vraća veličinu polja `GOL`, `numNeighbours(x, y)` računa broj živih susjeda stanice na mjestu (x, y) u polju, a `numNeighboursx2(x, y)` računa broj susjeda od susjeda stanice na mjestu (x, y).

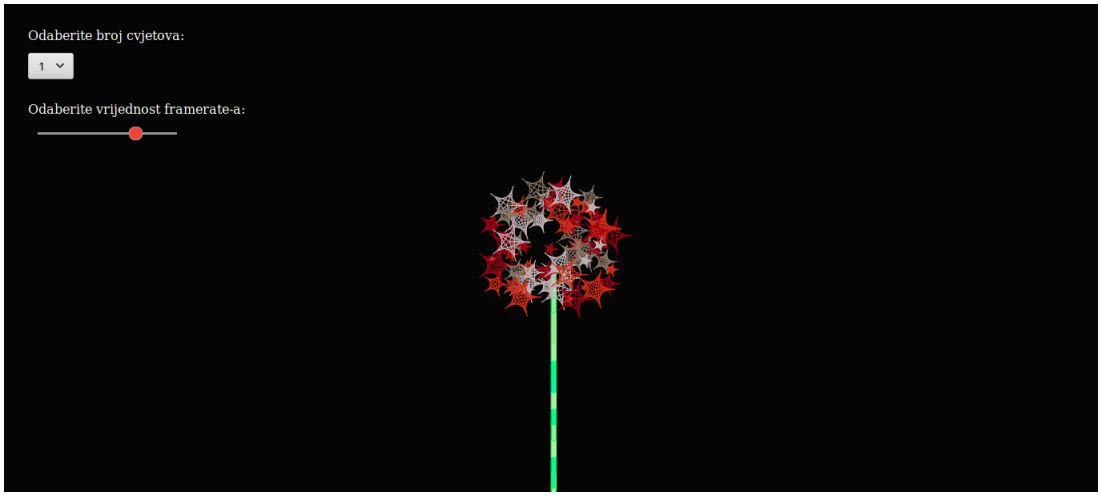
U glavnom programu, najprije se u funkciji `preload()` učitaju imena datoteka u kojima se nalaze automati u tekstovnom obliku. Imena se spreme u polje `automataNames`. Zatim se na slučajan način iz tog polja odabere jedno ime datoteke, učita se njen sadržaj i instancira jedan objekt tipa `GameOL`. To sve događa za vrijeme poziva funkcije `initializeValues()`. Funkcijom `drawControls()` nacrtamo padajući izbornik u kojem biramo broj maslačaka koje ćemo nacrtati. Funkcija `draw()` ima jedan `switch case` blok. Ako se radi o `case:0`, crta se stabljika cvijeta. Stabljika cvijeta je prikazana kao konačni niz pravokutnika koje se crtaju do središta cvijeta. Zatim se prelazi u `case:1`, u kojem automat prelazi u novo stanje i crta se cvijet.

```
// class Petal

function Petal( myX,
                myY,
                radius,
                age,
                nstitchPoints,
                myIdx,
                dandelionIdx){

    this.myX = myX;
    this.myY = myY;
    this.radius = radius;
```

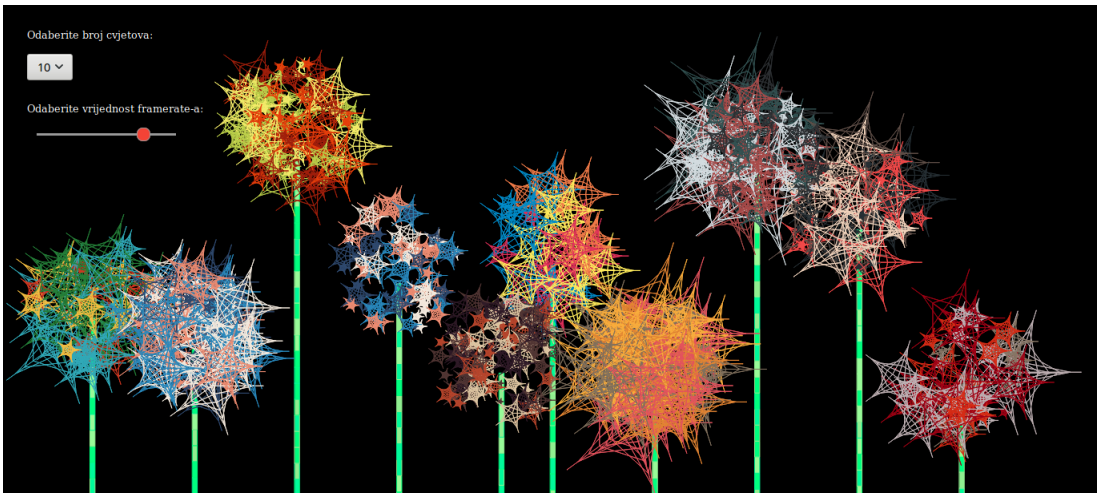
```
    this.age = age;
    this.nstitchPoints = nstitchPoints;
    this.myIdx = myIdx;
    this.dandelionIdx = dandelionIdx;

    this.drawPetal = function(){
        let numOfVertices = 5;

        poligonWithStitches(this.myX,
                            this.myY,
                            this.dandelionIdx,
                            this.radius,
                            numOfVertices,
                            this.nstitchPoints,
                            this.myIdx);
    };

    this.myCell = function(i, j){
        if(this.myIdx[0] == i && this.myIdx[1] == j)
            return true;
        return false;
    }
}
```

```
//class Dandelion

function Dandelion(myX, myY){
  this.petals  = new Array();
  this.x0      = myX;
  this.y0      = myY;

  this.addPetal = function(numOfNeighbours,
                           numOfNeighboursx2,
                           age,
                           petalIdx,
                           dandelionIdx){
    //r is radius of circumscribed circle of pentagon
    let r = (numOfNeighboursx2*numOfNeighboursx2)*age;
    let nstitchPoints = numOfNeighbours*age;
```

```
    if(nstitchPoints < 5)
      nstitchPoints *= 2;

    // (x1, y1) is the center of pentagon
    let x1 = random(this.x0 - radius, this.x0 + radius);
    let y1 = random(this.y0 - radius, this.y0 + radius);

    while(dist(x1, y1, this.x0, this.y0) > radius){
      x1 = random(this.x0 - radius, this.x0 + radius);
      y1 = random(this.y0 - radius, this.y0 + radius);
    }

    let newPetal = new Petal(x1, y1, r,
                             age,
                             nstitchPoints,
                             petalIdx,
                             dandelionIdx);
    this.petals.push(newPetal);
  }
}
```



Slika 3.1: Jedan maslačak



Slika 3.2: Više maslačaka

3.2 Lava lampa

U ovom primjeru pokazat ćemo kako možemo crtati u trodimenzionalnom *canvasu* u p5.js. Kugle u lavi lampi će biti povezane sa stanicama automata, a slijeve strane možemo odabrati hoćemo li prikazati lavu lampu ili njen stanični automat. Pomičemo li točkić na mišu možemo napraviti *zoom-in* ili *zoom-out*. Pritisnemo li mišem na *canvas* možemo rotirati lampu. Kada pustimo tipku miša lampa se vraća u početni položaj. Kao i u prethodnom primjeru, sa *sliderom* određujemo brzinu generiranja crteža.

Lava lampa je implementirana kao polje `lamp`. Elementi polja su varijable tipa `Bubble`. Konstruktor klase `Bubble` kao argumente prima x i z koordinatu kugle, brzinu, radijus, smjer i indeks kugle. Metoda `drawBubble()` crta kuglu na *canvasu*. Najprije se provjeri je li kugla došla do ruba kvadra i ako jest, promijeni se njen smjer kretanja. Sljedeće se provjerava je li stanica automata koja je pridružena ovoj kugli postala neživa. Ako je postala neživa, varijabla `deathFlag` se postavi na vrijednost 2, što znači da će kugla kada dođe do vrha kvadra, spuštajući se nazad, postupno smanjivati svoj radijus i kada on padne ispod određene granice, kugla će se ukloniti iz polja `lamp`. Sljedeće metode u funkciji `drawBubble` služe za osvjetljivanje kugle. Na kraju, nacrtamo kuglu i izračunamo njenu novu poziciju.

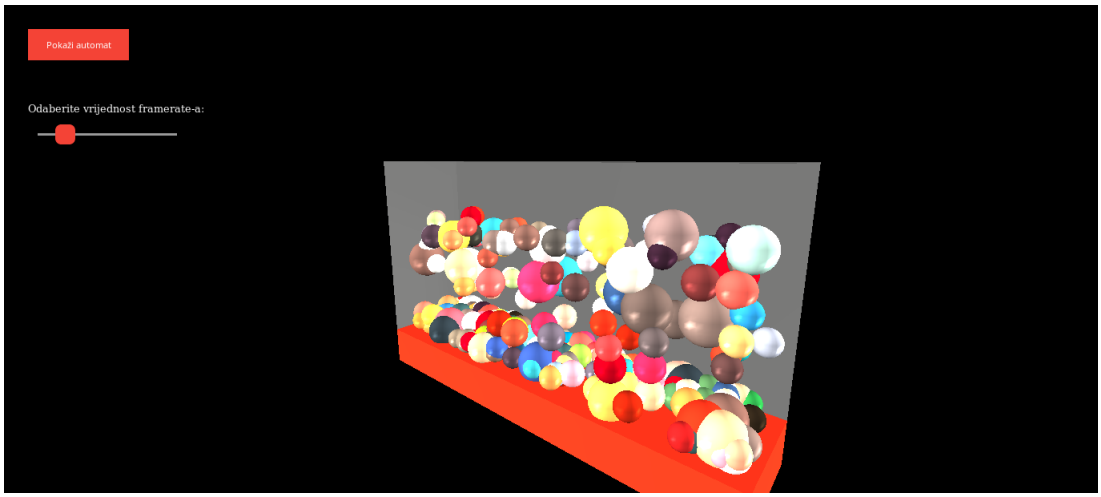
Klasa `GameOL` je skoro identična kao i u prethodnom primjeru. Ovdje, prilikom računanja nove generacije automata, dodatno provjeravamo je li neka stanica postala neživa kako bismo odgovarajućoj kugli mogli promijeniti varijablu `deathFlag`. Ako je neka stanica postala živa instanciramo novi objekt tipa `Bubble` i dodajemo ga na kraj polja `lamp`. Radijus kugle se dobiva kombiniranjem svojstava stanica poput broja susjeda, broja

susjeda od susjeda i prosječnim brojem živih stanica. Brzina kugle ovisi o životu stanice, trenutnom broju frameova ili vrijednosti `frameRate()`. Položaj kugle se određuje na slučajan način.

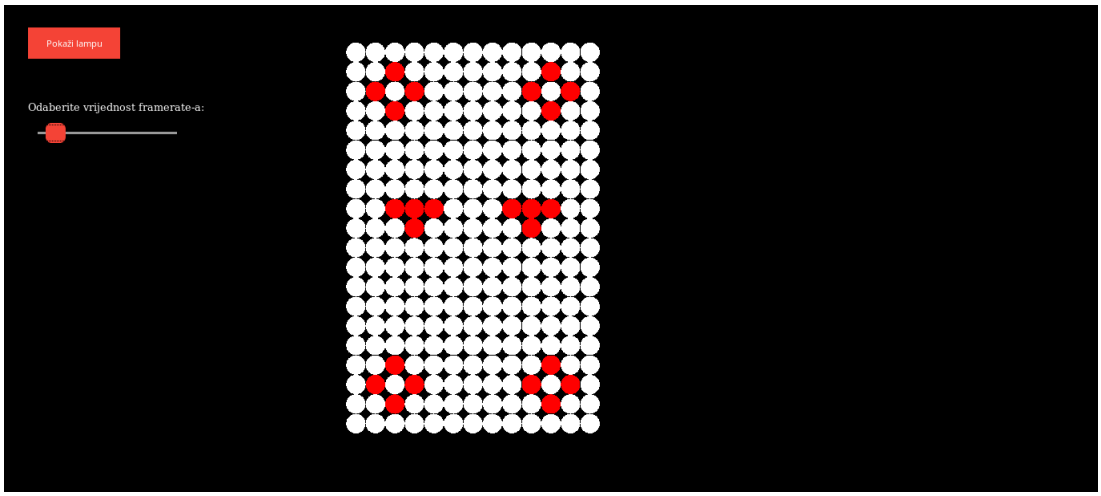
U glavnom programu, učitamo iz jedne datoteke (njeno ime se nalazi u kodu) automat i instanciramo jedan objekt tipa `GameOL`. Odredimo dimenzije kvadra u kojem će se kugle gibati te nacrtamo *slidere*, gumb i jedan padajući izbornik. U funkciji `draw()` ovisno o vrijednosti varijable `showAutomata` prikazujemo ili automat ili lampu. Zatim računamo sljedeću generaciju automata, izbacujemo malene kuglice iz polja i crtamo lampu i njene kugle.

```
// class Bubble

function Bubble(myX, myZ, myShift, myRadius, mySign, myIdx){
  this.myX = myX;
  this.myZ = myZ;
  this.myY = bubbleY;
  this.myShift = myShift;
  this.myRadius = myRadius;
  this.mySign = mySign;
  this.myIdx = myIdx;
  this.deathFlag = 0;
  this.color = random(colors);

  // function for drawing a bubble
  this.drawBubble = function(){
    // if the bubble is at the top/bottom of lamp, bounce it back
    if(Math.abs(this.myY) + this.myRadius >= lampHeight/2)
       this.mySign *= -1;
    if(this.deathFlag == 1 && this.mySign == 1){
      if(this.myRadius < 5)
        this.deathFlag = 2;
      else
        this.myRadius -= 0.5;
    }

    push();
    translate(this.myX, this.myY, this.myZ);
    // illumination for bubbles
    specularMaterial(this.color);
```

```
    sphere(this.myRadius);
    pop();
    this.myY += this.mySign*this.myShift;
  }

  // function returns true if automata's cell
  // belongs to this bubble
  this.myCell = function(i, j){
    if(this.myIdx[0] == i && this.myIdx[1] == j)
      return true;
    return false;
  }

}
```



Slika 3.3: Lava lampa



Slika 3.4: Stanični automat lave lampe

Bibliografija

[1] A. Adamatzky i G.J. Martinez, *Designing Beauty - The Art of Cellular Automata*, Springer, 2016.

[2] https://processing.org/examples/star.html (01.09.2018.).

[3] Lauren McCarthy, Casey Reas i Ben Fry, *Getting Started with P5. js: Making Interactive Graphics in JavaScript and Processing*, Maker Media, Inc., 2015.

[4] Matt Pearson, *Generative Art: A Practical Guide Using Processing,(2011)*, Manning Publications, Shelter Island, NY.

[5] Casey Reas i Ben Fry, *Processing: a programming handbook for visual designers and artists*, br. 6812, Mit Press, 2007.

[6] Kostas Terzidis, *Algorithms for visual design using the processing language*, John Wiley & Sons, 2009.

[7] www.p5js.org (01.09.2018.).

Sažetak

U ovom radu smo izložili osnove JavaScript inačice jezika Processinga, p5.js. U prvom poglavlju smo kroz primjere opisali kako se crtaju osnovni grafički elementi, rade transformacije na njima te kako možemo napraviti interaktivne *sketcheve*.

U drugom poglavlju smo ukratko predstavili stanične automate koji mogu služiti kao podloga za generiranje geometrijskih likova u p5.js programima. Na kraju smo implmentirali dva kompliciranija primjera korištenja ove biblioteke u čijoj pozadini se upravo nalaze stanični automati.

Summary

In this diploma thesis we have outlined the basics of the JavaScript version of the Processing language, p5.js. In the first part we have, through examples, described how to draw basic graphical elements, make transformations on them and create interactive sketches.

In the second part we briefly introduced cellular automata that can serve as a basis for generating geometric characters in p5.js programs. Lastly, we implemented two more complicated examples of using this library in the background of which cellular automata are currently in place.

Životopis

Ivana Senkić je rođena 15.11.1994. u Mostaru gdje je pohađala osnovnu školu i gimnaziju. Nakon završetka gimnazije, 2013. godine upisuje preddiplomski studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Nakon završetka, 2016. godine upisuje diplomski studij Računarstva i matematike na istom fakultetu.