Izrada web stranice za Processing

Babok, Veronika

Sveučilište
Sjever

Završni rad br. 645/MM/2019

Izrada web stranice za Processing

Veronika Babok, 1631/336

Varaždin, rujan 2019. godine

Sveučilište Sjever

Multimedija, oblikovanje i primjena

Završni rad br. 645/MM/2019

Izrada web stranice za Processing

Student

Veronika Babok, 1631/336

Mentor

mr. sc. Vladimir Stanisavljević

Varaždin, rujan 2019. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ preddiplomski stručni studij Multimedija, oblikovanje i primjena

PRISTUPNIK Veronika Babok | MATIČNI BROJ 1631/336

DATUM 16.09.2019. | KOLEGIJ Programski alati 2

NASLOV RADA Izrada web stranice za Processing

NASLOV RADA NA ENGL. JEZIKU Developing web page for Processing

MENTOR mr.sc. Vladimir Stanisavljević | ZVANJE Viši predavač

ČLANOVI POVJERENSTVA

1. doc.dr.sc. Dean Valdec - predsjednik
2. doc.dr.sc. Andrija Bernik, pred. - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. pred. Snježana Ivančić Valenko, dipl.graf.ing. - rezervni član
5.

Zadatak završnog rada

BROJ 645/MM/2019

OPIS

Processing je programerska platforma kojoj je osnovni cilj olakšati programiranje za početnike za primjene bliske multimedijalnim umjetnicima te drugim ne-programerskim zanimanjima. Pomoću njega se jednostavnije nego u drugim programskim jezicima može se ostvariti grafički prikaz i interakcija sa okolinom što ga čini pogodnim za primjene u multimedijskih instalacija. Sastoji se od programskog jezika temeljenog na Java-i te razvojnog okruženja.
U radu je potrebno:
* ukratko prikazati povijest, razvoj, mogućnosti i primjere korištenja platforme Processing,
* pomoću Processinga izvesti jednostavnije i složene samostalne primjere i opisati korake i primitive korištene u izradi,
* osmisliti i izraditi demo web stranicu na kojoj će biti prikazane mogućnosti Processinga u obliku primjera i tutorijala
Detaljno opisati sve korištene tehnologije i korake u radu potrebne da bi se ostvarilo traženo te detaljno opisati stečena iskustva i postignute rezultate.

ZADATAK URUČEN 13.09.2019 | POTPIS MENTORA

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

Predgovor

Kroz izradu ovog završnog rada dobila sam želju i motivaciju za daljnjim upoznavanjem Processinga te naprednijom izradom web stranica. Nadam se da će mi cijelokupni projekt pomoći s poboljšanjem u programiranju te pronalaskom posla u odabranoj struci.

Zahvaljujem se profesoru mr.sc.Vladimiru Stanisavljeviću, koji je pristao biti mentor mojeg završnog rada te me kroz godine studiranja vodio kroz programiranje i ukazao mi na neke nove i zanimljive stvari poput Processinga. Također zahvaljujem svojoj obitelji, dečku i prijateljici koji su mi bili podrška kroz studiranje i izradu završnog rada.

Sažetak

U ovom radu objašnjena je izrada web stranice kojoj je glavna tema Processing. Samim time opisano je korištenje kodnim jezikom HTML-om te stilskim jezikom CSS-om, ali najbitnije programiranje koristeći Processingovo integrirano razvojno okruženje.

U praktičnom dijelu ovog rada, objašnjena je samostalna izrada web stranicu koja objašnjava što je Processing te kako se njime koristiti. Na stranici se nalazi nekoliko riječi o samom programiranju Processingom, primjeri koji uključuju objašnjenja te njihovi izvorni kodovi. Za kraj, napravljena je stranica s uputama koja uključuje nekoliko glavnih segmenata Processinga, a vođena je primjerima s kodovima.

Ključne riječi: Processing, HTML, CSS, programiranje

Abstract

The idea of this work is to explain the development of a web page based on Processing. It describes the use of HTML code language and CSS style language, but most importantly programming using Processing's integrated development environment.

In the practical part of this work, it is explained how to create a web page on its own and how to implement Processing into it and how to use it. The web page is filled with information about Processing but also examples with explanations and their source codes. At the very end of the web page, you can find tutorials on how to create your own Processing guided by code examples.

Keywords: Processing, HTML, CSS, programming

Popis korištenih kratica

HTML HyperText Markup Language
Sintaksa za obilježavanje hipertektualnih dokumenata.

CSS Cascading Style Sheets
Stilski jezik za oblikovanje web stranica

P2D/P3D Processing 2D / Processing 3D

PDE Processing Development Environment

Sadržaj

1. Uvod

Processing promovira softversku pismenost, osobito u području vizualnih umjetnosti i vizualne pismenosti unutar tehnologije. U početku je kreiran s namjerom da služi kao programski „blok za skiciranje“ i podučava osnove programiranja unutar vizualnog konteksta, a također se razvio u alat za profesionalce. Ovaj softver je besplatan i otvoren izvor te se pokreće na Mac, Windows i GNU / Linux platformama.

Inspiriran je ranijim jezicima poput BASIC-a i Logo-a. Processing je usmjeren prema stvaranju vizualnih, interaktivnih medija, tako da prvi programi započinju nekakvom vrstom crteža. [1]

Inovacije u poučavanju Processinga prilagođene su računalnim tutorijalima Khan akademije te su ponuđene na internetu besplatno. Tutoriali počinju crtežom, koristeći većinu funkcija koje Processing nudi za crtanje. Pristup Processingu također je primijenjen na elektroniku kroz Arduino i Wiring projekte. Arduino koristi sintaksu nadahnutu onom koja se koristi s Processingom i nastavlja koristiti modificiranu verziju programskog okruženja Processinga kako bi studenti lakše učili kako programirati robote i bezbroj drugih elektroničkih projekata. [1]

U ovom radu opisana je izrada web stranice pomoću HTML i CSS jezika te nekih pomoćnih framework-ova poput Bootstrapa. Tema web stranice je Processing, a na samoj stranici nalaze se osnovne informacije o istome, te su potkrijepljene primjerima i tutorialima kako napisati kod te dobiti interaktivnu „skicu“.

Processing je fleksibilan softverski „blok za crtanje“, ali i jezik za učenje kodiranja u kontekstu vizualne umjetnosti i smatra se zanimljivim dodatkom na web stranicama iz razloga što može biti interaktivan ili pak može predstavljati neku animaciju ili slično.

2. HTML – HyperText Markup Language

HTML je programski jezik koji ima kodne riječi i sintaksu kao i svaki drugi jezik, a koristi se za kreiranje dokumenata (stranica) na internetu. Koristi se za stvaranje hipertekstualnih datoteka, odnosno, datoteka koje sadržavaju linkove.

HyperText je način na koji se kreće na webu - klikom na tekst koji je posebno označen kao hiperveza i koji vodi na sljedeću stranicu ili na neku drugu lokaciju. Hyper znači da nije linearna, odnosno, može odvesti na bilo koje mjesto na internetu bilo kad klikom na hiperverze.

Markup je ono što HTML oznake rade s tekstom unutar njih. Oni ga označavaju kao određenu vrstu teksta (npr. kurzivni tekst).

HTML se sastoji od niza kratkih kodova koji se unose u tekstualnu datoteku - to su oznake (tags). Tekst se zatim sprema kao html datoteka (.html) i pregledava putem web preglednika. Odabrani preglednik zatim čita datoteku i prevodi tekst u vidljivi oblik, uz pravilno kodiranje, točno onako kako je zamišljena. [2]

Oznake su ono što odvaja normalni tekst od HTML koda. Njima se stvaraju stvari poput slika, tablica i ostalog, na način da pregledniku „govore“ što će prikazati na stranici. Različite oznake obavljaju različite funkcije. Neke od jednostavnijih oznaka ne čine ništa više od primjene oblikovanja na neki tekst:

Primjerice oznaka označava podebljani ili „boldani(bold)“ tekst.
* Ovaj tekst je podebljan. *

Prikaz na stranici:
Ovaj tekst je podebljan.

Zagrade u kojima se pišu oznake su šiljaste (< >) i obično dolaze u paru. Kada je oznaka napisana u ovim zagradama (< >), to znači da je oznaka započeta. Za kraj oznake koristimo kosu crtu između zagrada i ona se piše prije oznake (</ >). Kombinacijom početne i završne oznake i sadržaja između, dobiven je element. [2]

Atributi djeluju kao dodatne informacije koje se vežu uz HTML element i pišu se unutar oznake i iz toga razloga nisu vidljive u pregledniku. Na primjer: [2]

Atribut *href* koristi se za definiranje cilja poveznice u oznaci < a >.

< a href = „arwen.unin.hr/~vebabok/processing2019/HTML/index.html“ >

*Poveznica na Processing *

Neki elementi imaju obavezne atribute. Prilikom umetanja slike u dokument, obavezno je unijeti lokaciju slike pomoću atributa *src*. [2]

<img src="#" alt="Opis slike">

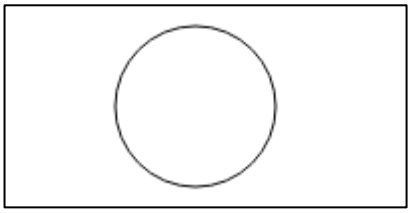
Komentari u HTML-u pišu se ovako:

<!--komentar -->

Njih će preglednik ignorirati, a oni mogu biti vrlo korisni samom korisniku tijekom pisanja i mijenjanja koda jer se uz njih lakše snalazi, pogotovo ako ima mnogo sličnog koda. [2]

HTML `<canvas>` element koristi se za crtanje grafike na web stranici, a on je zapravo spremnik za grafiku. Za stvarno crtanje grafike mora se koristiti JavaScript. Canvas ima nekoliko metoda za crtanje putanja, okvira, krugova, teksta i dodavanja slika. Canvas(„platno“) je pravokutno područje na HTML stranici. Prema zadanim postavkama, platno nema obruba i nema sadržaja. Oznaka izgleda ovako: [3]

```
<canvas id = "myCanvas" width = "200" height = "100"> </canvas>
```



Slika 2.1.1: Primjer <canvas> elementa

```
<canvas id="myCanvas" width="200" height="100" style="border:1px
solid #d3d3d3;"> </canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
</script>
```

3. CSS – Cascading Style Sheets

Dok se HTML koristi za strukturiranje web-dokumenta (definiranje stvari kao što su naslovi i paragrafi, te ugradnja slika, videozapisa i drugih medija), CSS se kodira usput i određuje stil dokumenta - izgled stranice, boja i fontova. HTML je temelj, a CSS je estetika.

CSS štedi vrijeme - CSS može se pisati jednom, a zatim ponovno koristiti isti dokument na više HTML stranica. Može se definirati stil za svaki HTML element i primijeniti se na onoliko web stranica koliko želite. Stranice se brže učitavaju – nije potrebno svaki put pisati atribute HTML oznaka. Jednostavnije je pisanje CSS oznake koja se može primijeniti na više mjesta. Manje koda znači brže vrijeme preuzimanja. Jednostavno održavanje – kod promjene stila, svi elementi na svim web stranicama automatski će se ažurirati. CSS ima mnogo širi raspon atributa nego HTML, tako da može dati daleko bolji izgled HTML stranici u usporedbi s HTML atributima. [4]

4. Processing

Processing koriste tisuće vizualnih dizajnera, umjetnika i arhitekata za izradu svojih djela. Projekti izrađeni s Processingom prikazani su u Muzeju moderne umjetnosti u New Yorku, muzeju Victoria and Albert u Londonu, Centru Georges Pompidou u Parizu i mnogim drugim istaknutim mjestima. Processing se koristi za izradu projektiranih scenografija za plesne i glazbene izvedbe; za generiranje slika za glazbene videozapise i filmove; izvoz slika za plakate, časopise i knjige; stvaranje interaktivnih instalacija u galerijama, muzejima i na ulici. Neki od istaknutih projekata uključuju video za House of Card za Radiohead, generativni logotip MIT Media Lab i softverski mural Chronograph koji je dizajniran za novi svjetski centar Frank Gehry u Miamiju. Ali najvažnija stvar u Processingu i kulturi je način na koji je softver angažirao novu generaciju vizualnih umjetnika da razmotre programiranje kao bitan dio njihove kreativne prakse. [5]

U proteklih šesnaest godina, Processing je promovirao softversku pismenost, osobito unutar vizualne umjetnosti i vizualne pismenosti u tehnologijama. U početku je kreiran kao programska skica i da poducava osnove programiranja unutar vizualnog konteksta, Processing se također razvio u razvojni alat za profesionalce. Softver za obradu je besplatan i otvoreni izvor te može biti pokrenut na Mac, Windows i GNU / Linux platformama. Processing je i dalje alternativa softverskim alatima s ograničenim i skupim licencama, što ga čini pristupačnim školama i individualnim studentima. Njegov status otvorenog koda potiče sudjelovanje zajednice i suradnju koja je od vitalnog značaja za razvoj samog programa. Suradnici dijele programe, doprinose kodu i grade biblioteke, alate i načine kako bi proširili mogućnosti softvera. [1]

Processing su pokrenuli Ben Fry i Casey Reas u proljeće 2001. godine, dok su obojica diplomirali na MIT Media Labu u sklopu istraživačke skupine za estetiku i računanje Johna Maede. Razvoj se nastavio u slobodno vrijeme dok je Casey nastavio svoju umjetničku i nastavnu karijeru, a Ben je pohađao doktorat i osnovao Fathom Information Design. Mnoge ideje iz Processinga vraćaju se na radionicu vizualnog jezika Muriel Cooper, a ona je nastala izravno iz projekta Maeda, Design By Numbers, koji je razvijen u Media Labu i objavljen 1999. godine. Projekti Wiring i Arduino nastali su iz Processinga dok je Casey predavao na Institutu za interakcijski dizajn Ivrea u Italiji. [5]

4.1. Razvojno okruženje

Razvojno okruženje Processinga (PDE) olakšava pisanje programa. Programi se pišu u uređivaču teksta i pokreću pritiskom na gumb Pokreni. U Processingu se računalni program naziva skica. Skice su pohranjene u Sketchbooku, a to je ustvari je mapa na vašem računalu.

Skice mogu crtati dvodimenzionalnu i trodimenzionalnu grafiku. Zadani renderer je za crtanje dvodimenzionalne grafike. P3D renderer omogućuje crtanje trodimenzionalne grafike, što uključuje upravljanje fotoaparatom, rasvjetom i materijalima. P2D renderer je brz, ali manje točan prikaz za crtanje dvodimenzionalne grafike. Oba P2D i P3D prikazivača ubrzavaju se ako korišteno računalo ima OpenGL kompatibilnu grafičku karticu. [6]

Mogućnosti Processinga su proširene pomoću knjižnica i alata. Knjižnice omogućuju da skice rade nešto izvan osnovnog koda. Postoje stotine knjižnica koje su pridonijele zajednici Processinga i koje se mogu dodati skicama kako bi se omogućile nove stvari kao što su zvukovi, rad na računalnom vidu i rad s naprednom 3D geometrijom. Alati proširuju PDE kako bi olakšali stvaranje skica lakšim pružanjem sučelja za zadatke poput odabira boja. [6]

Processing je korišten za stotine vrhunskih projekata u širokom rasponu polja, od multimedijskih instalacija do vizualizacije informacija. Osnovni aplikacijski okvir pojednostavljuje većine multimedijskih potreba (OpenGL, Quicktime, izvoz u PDF, snimanje fotoaparatom), uklanjajući projekte koji su uključeni u dosadan zadatak postavljanja osnovnih aplikacija. Koristi se proširiva struktura koda koja je omogućila stvaranje desetaka korisnih knjižnice za sve, od uvoza / izvoza 3D do složene sinteze geometrije.[7]

5. Izrada web stranice

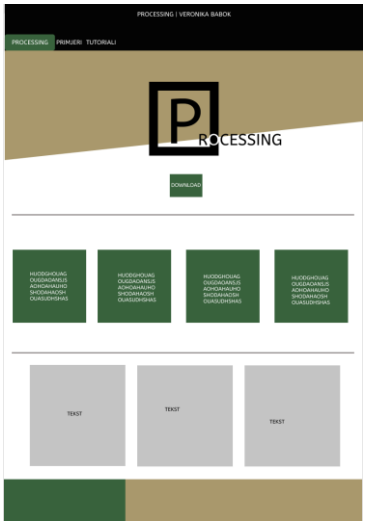
Glavni razlog izrade web stranice je to što su web stranice gotovo svima dostupni besplatni izvori literature te se njima lako kreće i lako se dolazi do željenih informacija te su vrlo pregledne i korisne jer one same mogu dovesti posjetitelje do ostalih traženih podataka. Također, na hrvatskom jeziku nema mnogo literature što se tiče programiranja, a gotovo Processinga pa bi ova stranica mogla doprinijeti učenicima i studentima koji su zainteresirano za ovo razvojno okruženje.

5.1. Definiranje izgleda stranice

Prije stvaranja bilo koje stranice, glavni korak je odrediti kako će stranica izgledati i što će sadržavati. Stranica bi uvijek trebala biti pregledna i privlačna kako bi zadržala pažnju posjetitelja, a stranice koje objašnjavaju nešto poput programiranja, trebale bi biti potkrijepljene slikama i primjerima kako bi oni mogli što više naučiti. Vrlo su bitne boje koje se koriste jer one ne smiju biti prenapadne i ne smiju preusmjeravati pažnju sa sadržaja, također trebale bi biti pažljivo odabrane zbog osoba koje imaju neku vrstu oštećenja vida. U ovom završnom radu odabrane glavne boje su bijela, crna, zelena i svijetlo smeđa jer one djeluju neutralno.

Također treba paziti na tekst, on ne smije biti premalen, ne smije ga biti previše na jednom mjestu bez pauze i mora biti čitak. U praktičnom dijelu ovog završnog rada nalaze se dva fonta: jedan je običan, san-serifni, a drugi podsjeća na izgled koda kako bi se uklopio u temu stranice.

Koristeći web aplikaciju, odnosno sučelje za dizajniranje izgleda stranica, Figmu (www.figma.com), izarđuje se dizajn prema kojem je zatim realizirana stranicu pomoću HTML i CSS jezika.



Slika 5.1.1: Dizajn web stranice u Figmi

Sljedeća bitna stvar je određivanje sadržaja koji se prikazuje na stranici te raspoređivanje istoga da bi bilo što preglednije. Prije izrade, prikupljeno je nekoliko informacija s kojima se kasnije popunjavala stranica. Kako se nebi kršila neka korisnička prava, izrađeni su logotipi stranice i favicona[1], a izrađeni su u programu Adobe Illustrator.

Slika 5.1.2: Logotip izrađen u Adobe Illustratoru

Slika 5.1.3: Favicon

[1] favicon – datoteka koja sadrži jednu ili više malih ikona

5.2. Realizacija dizajna web stranice

Prema željenom izgledu koji je napravljen u Figmi, kreće se sa izradom same stranice. Za početak pomoću oznaka odvojen je prostor na stranici na način da su postavljene određene dimenzije, boje te pozicija samih elemenata. Na samom vrhu nalazi se izbornik koji vodi na druge stranice, a ispod toga nalazit će se informacije o Processingu.

Dio HTML koda:

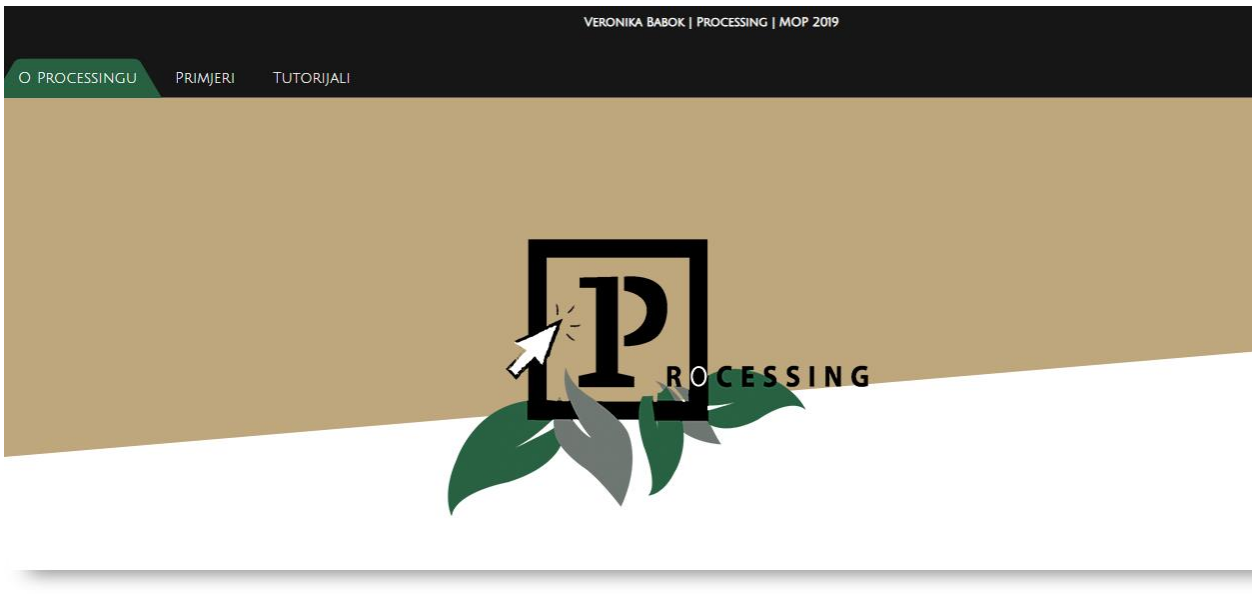
```
  <div id="traka">
              <center>
                   <p   id="gore">   </br>Veronika   Babok      |
Processing  |  MOP 2019 </p>
              </center>
         </div>


     <div id='cssmenu'>
          <ul>
               <li      class='active'><a      href='index.html'>O
Processingu</a></li>
              <li><a href='primjeri.html'>Primjeri</a></li>
              <li><a href='tutorial.html'>Tutorijali</a>
              </li>
          </ul>
     </div>



     <div id="shape">
     </div>


     <div id="slika">
          <img src="slika.png">
     </div>
```

Ovdje vidimo kako s oznakama možemo podijeliti prostor na koliko god dijelova trebamo te oni uz pomoć CSS oblikovanja mogu biti bilo koje veličine, boje, oblika, kao i pozicije na ekranu. „ID“ i „CLASS“ su selektori s kojima određujemo koji stil će kojem elementu biti dodijeljen u CSS-u.

Dio CSS koda:

```
   #shape {
      -webkit-clip-path: polygon(0 100%, 100% 55%, 100% 55%, 0%
55%);
      clip-path: polygon(0 100%, 100% 85%, 100% 55%, 0% 55%);
      width:110%;
      height:800px;
      background-color: #BEA67D;
      margin-top:-459px;
      margin-left: -10px;
      position:relative;
   }


   #traka {
      width:110%;
      height:50px;
      background-color: #161616;
      margin-top:-20px;
      margin-left: -8px;
   }


   #slika {
      width:200px;
      height:200px;
      position: absolute;
      margin-top:-220px;
      margin-left:33%;
   }

   #gore {
      font-family: 'Julius Sans One', sans-serif;
      color:white;
      text-align:center;
      font-size: 9pt;
   }

   #cssmenu,
   #cssmenu ul,
   #cssmenu ul li,
   #cssmenu ul li a {
      margin-left: -4px;
      margin-top:auto;
      padding: 0;
      border: 0;
      list-style: none;
      line-height: 1;
```

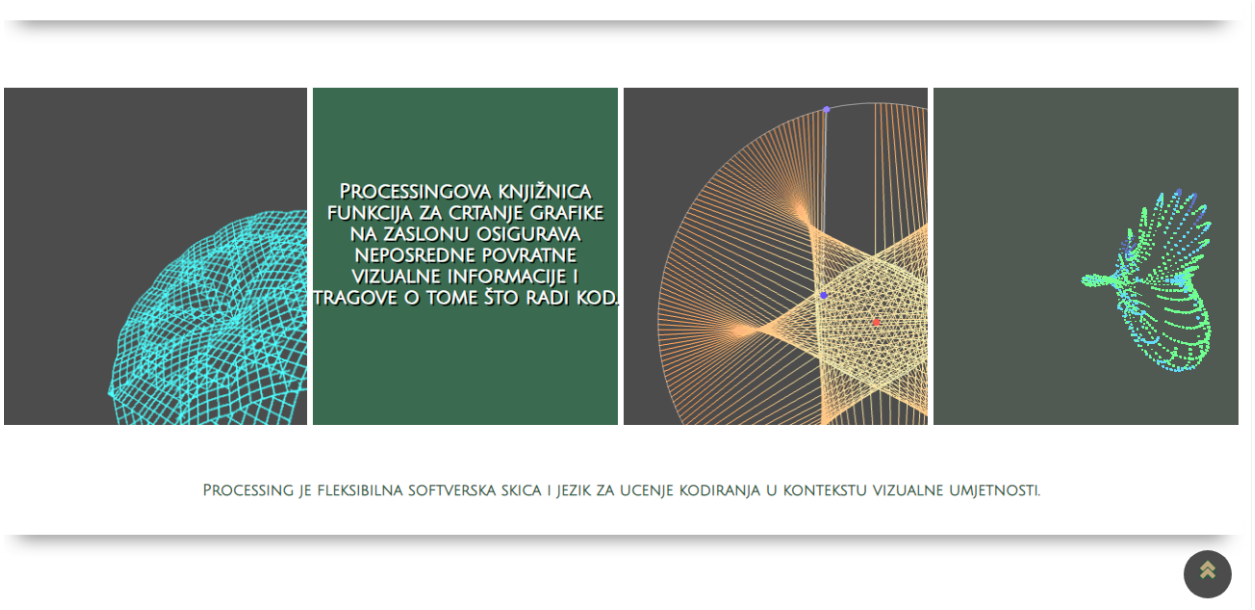
U CSS-u se zadaje određeni stil elementima na stranici. Poput visine, širine, boje pozadine i teksta, oblikovanja teksta, margina, pozicije te ostalog.



Slika 5.2.1: Početni izgled stranice

Uz gore napisani HTML i CSS kod, stranica za početak izgleda ovako. Umetnuta je slika logotipa, kreiran je izbornik, koji vodi na druge stranice koje su za sada istog oblikovanja kao i početna i bez sadržaja.

Na isti način kao i prethodni, uređuje se ostatak stranice, raspoređuju se elementi u njoj te se dodaje sadržaj, u ovome slučaju, tekst koji objašnjava što je Processing.



Slika 5.2.2: Umetanje sadržaja na stranicu

Stavljeno je nekoliko animiranih slika koje prelaskom miša preko njih otkrivaju dodatne informacije o Processingu. Takve stvari posjetiteljima mogu biti zanimljive i zabavne te ih potaknuti da dalje istražuju stranicu.

Također, u donjem desnom rubu, dodan je gumb koji služi za automatsko „skrolanje“ na vrh stranice.

HTML:

```
<a href="javascript:" id="return-to-top">
        <img src="arrow.png" width="20px" height="20px"
        style="margin-left:15px; margin-top:10px;">
</a>
        <h3><i class="icon-arrow-down"></i></h3>
```

CSS:

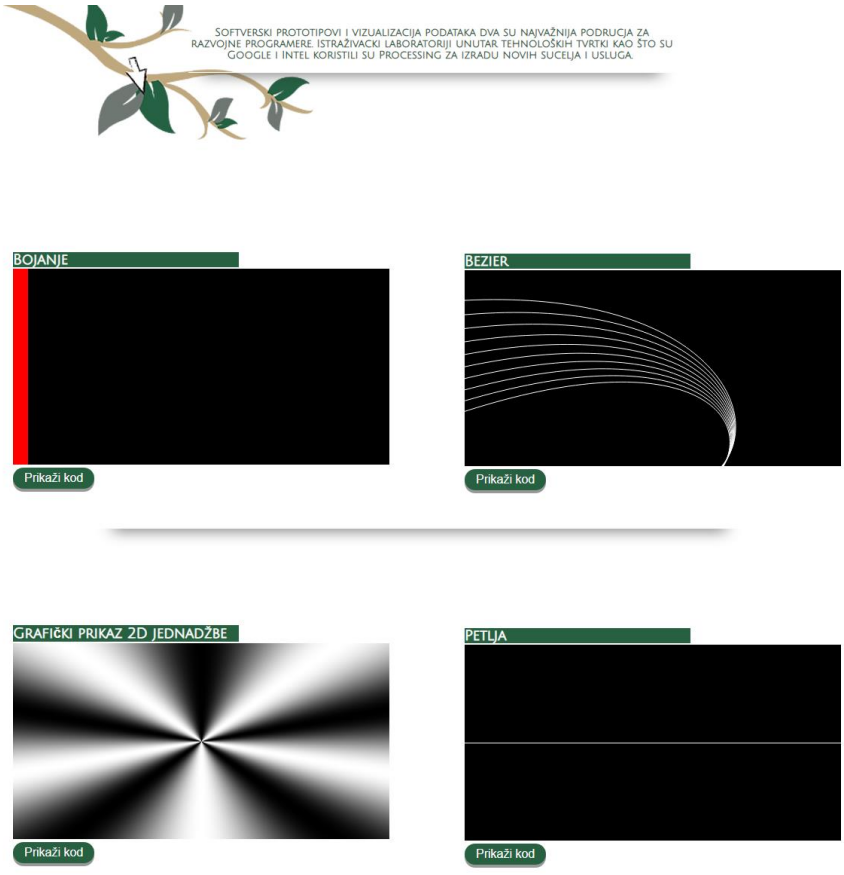
```
#return-to-top i {
    color: #fff;
    margin: 0;
    position: relative;
    left: 16px;
    top: 13px;
    font-size: 19px;
    -webkit-transition: all 0.3s ease;
    -moz-transition: all 0.3s ease;
    -ms-transition: all 0.3s ease;
    -o-transition: all 0.3s ease;
    transition: all 0.3s ease;
}
#return-to-top:hover {
    background: rgba(0, 0, 0, 0.9);
}
#return-to-top:hover i {
    color: #fff;
    top: 5px;
}
```

Nakon završetka popunjavanja sa sadržajem, ostatak stranice izgleda ovako:

Slika 5.2.3: Srednji dio stranice koji prikazuje reference

Slika 5.2.4: Dno stranice s informacijama

Na ostale dvije stranice nalaze se primjeri i tutoriali Processinga.



Slika 5.2.5: Prikaz stranice s primjerima s implementiranim Processingom

Ovdje se nalaze prozori koji zasebno prikazuju skice nastale u Processingu. Skice su interaktivne bilo prelaskom ili pritiskom miša. Svaka skica ima mogućnost prikaza koda te objašnjenje svoje interaktivnosti. Detaljnije objašnjenje nalazi se opisano dalje u radu.

Posljednja stranica nudi nekoliko područja Processinga koja su objašnjena slikovnim i kodnim primjerima. Svako područje vodi na zasebnu stranicu gdje je detaljno opisan postupak programiranja Processingom.

Slika 5.2.6: Stranica s tutorijalima

5.3. Korištenje Processingom

Prvi korak je instalacija programa i odabir platforme, a zatim nekoliko koraka ovisno o platformi koja se koristi. Kod korištenja Windows platforme, prate se sljedeći koraci:

Nastaje .zip datoteka. Dvoklikom se otvara te se zatim povlači na željenu lokaciju na tvrdom disku. Važno je da se mapa Processing izvuče iz te .zip datoteke. Zatim slijedi dvoklik na datoteku processing.exe kako bi se pokrenula instalacija. [8]

Kod korištenja Mac OS X platforme: Također postoji .zip datoteka. Dvoklikom se otvara ikona Processing u mapi Aplikacije. Zatim slijedi dvoklik na datoteku processing.exe kako bi se pokrenula instalacija. [8]

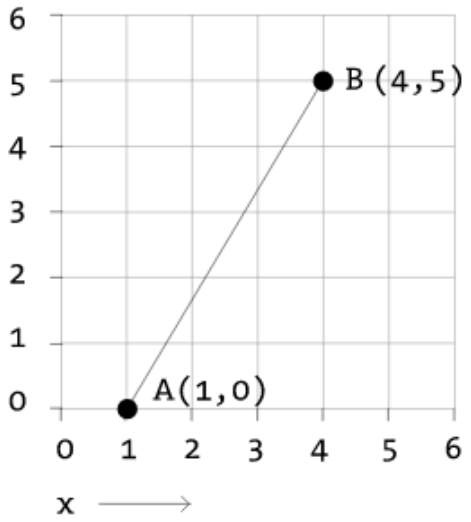
Kod Linux platforma : Ovdje se nalazi.tar.gz datoteka, koja bi trebala biti poznata većini korisnika Linuxa. Potrebno je preuzeti datoteku u kućni direktorij, a zatim otvoriti prozor terminala i upisati: tar xvfz obrada-xxxx.tgz(xxxx je potrebno zamijeniti s ostatkom imena datoteke, što je broj inačice.) Ovo stvara mapu pod nazivom processing-2.0 ili nešto slično. Zatim je potrebno promijeniti u direktorij: cd processing-xxxx i pokrenuti ga:./processing. [8]

U programu se nalazi: veliko područje koje je uređivač teksta, a na vrhu je niz gumba; odnosno alatna traka. Ispod uređivača nalazi se područje poruke, a ispod njega konzola. Područje za poruke koristi se za kodove napisane u jednoj liniji, a konzola se koristi za više tehničkih pojedinosti. [6]

Jedna od najtežih stvari u vezi s početkom programiranja je zahtijevanje specifičnosti što se tiče sintakse. Processingov softver nije uvijek dovoljno pametan da bi znao na što korisnik misli i prilično je "strog" glede postavljanja interpunkcijskih znakova i upotrebe velikih i malih slova. [8]

5.3.1. Osnovni oblici

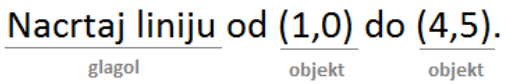
Najkraća udaljenost između dvije točke je dobra staromodna crta, a prije programiranja, s njome se i počinje. [6]



Slika 5.3.1.1: Crta između točke A i točke B

Gornja slika prikazuje crtu između točke A (1,0) i točke B (4,5). Da bi računalo pokazalo takvu liniju na zaslonu, naredba mora izgledati ovako: [6]
line (1,0,4,5);

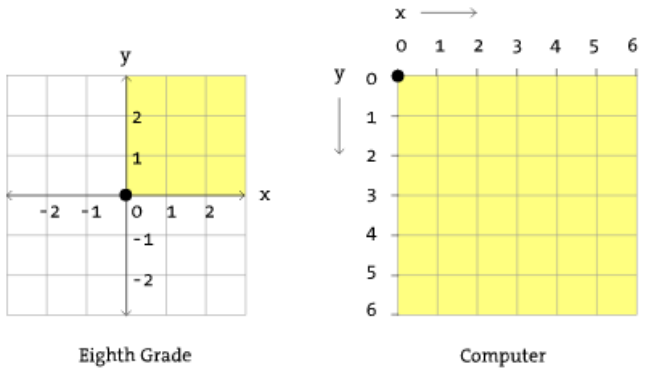
Zadaje se naredba (nazvana "funkcija") koju će računalo slijediti pod nazivom "line(linija)". Osim toga, navode se neki argumenti za nacrt te linije, od točke A (1,0) do točke B (4,5).
Ako o toj liniji mislite kao rečenicu, funkcija je glagol, a argumenti su objekti rečenice.
Izraz koda također se završava točkom-zarezom umjesto točkom. [6]



Slika 5.3.1.2: Funckije i argumenti

Zaslon računala jako je sličan komadu papira s kockicama. [6]
Svaki piksel zaslona je koordinata - dva broja, x" (vodoravno) i "y" (okomito) - koja određuje mjesto točke u prostoru. Zadatak je odrediti koji oblici i boje bi se trebali pojaviti na tim koordinatama piksela.. [6]

Na tom papiru("kartezijanski koordinatni sustav") postavljeno je (0,0) u središte s y-osi okrenutom prema gore, a x-os je usmjerena udesno (u pozitivnom smjeru, negativno prema dolje i lijevo). Međutim, koordinatni sustav za piksele u računalnom prozoru je obrnut duž y-osi. (0,0) može se naći na vrhu lijevo s pozitivnim smjerom u desno vodoravno i dolje okomito. [6]



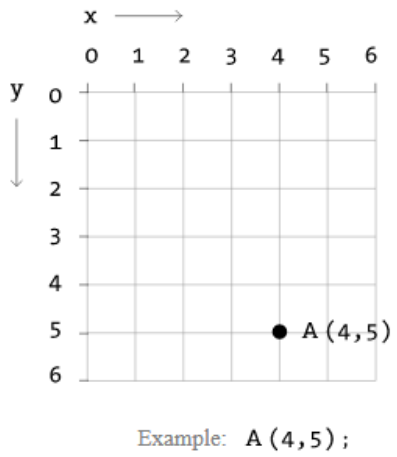
Slika 5.3.1.3: Papirnati i računalni koordinatni sustav

Velika većina primjera programiranja u Processingu vizualne su prirode.
Ti primjeri, u svojoj osnovi, uključuju crtanje oblika i postavljanje piksela. Promatraju se četiri primitivna oblika. [9]

Slika 5.3.1.4: Primitivni oblici

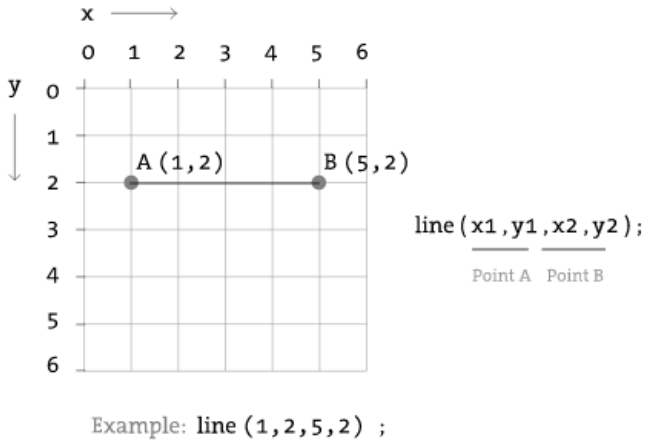
Za svaki oblik potrebno je zapitati se koje su informacije potrebne za određivanje lokacije i veličine (a kasnije i boje) tog oblika te saznati kako Processing očekuje primanje tih informacija. U svakom od dijagrama u nastavku pretpostavlja se prozor s širinom od 10 piksela i visinom od 10 piksela(za demonstraciju). [9]

point()(točka) je najlakši oblik za početak. Za crtanje točke, potrebne su samo koordinate x i y.



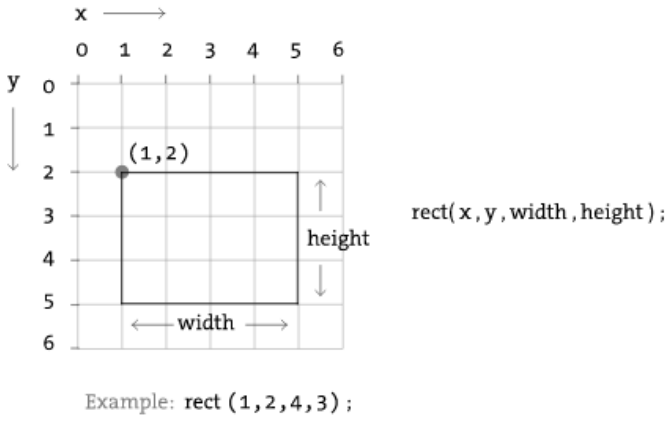
Slika 5.3.1.5: point()

Ni line()(linija) nije teška te zahtijeva dvije točke: (x1, y1) i (x2, y2):



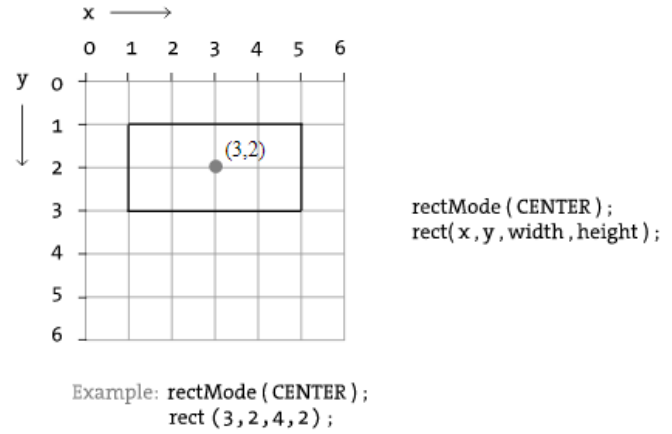
Slika 5.3.1.6: line()

Kod crtanja pravokutnika rect(), stvari postaju malo složenije.
U Processingu, pravokutnik je određen koordinatom za gornji lijevi kut pravokutnika, kao i širinom i visinom. [9]



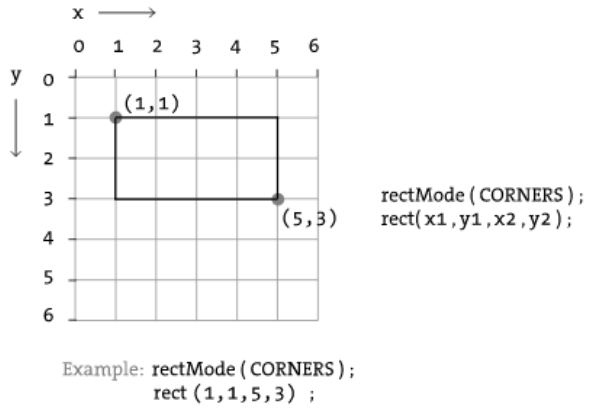
Slika 5.3.1.7: rect()

Drugi način crtanja pravokutnika uključuje određivanje središnje točke, zajedno sa širinom i visinom. Kod ove metode, prvo se naznačuje da korištenja "CENTER" prije argumenata za sam pravokutnik. Processing je osjetljiv na velika i mala slova. [9]



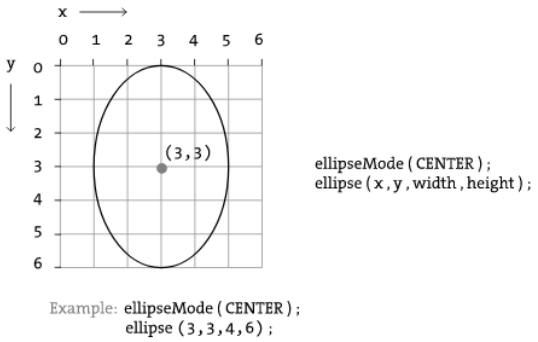
Slika 5.3.1.8: rect() sa središnjom točkom

Također je moguće nacrtati pravokutnik s dvije točke (gornji lijevi i donji desni kut) koristeći "CORNERS". [9]

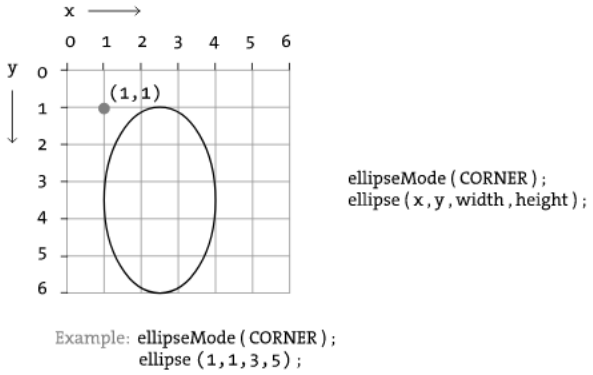


Slika 5.3.1.9: rect() s dvije točke

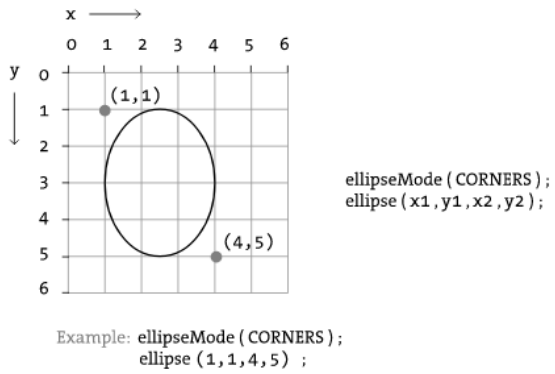
Elipsa je gotovo identična pravokutniku s razlikom što je elipsa nacrtana tamo gdje bi bio granični okvir pravokutnika. Zadani način za elipse() je "CENTER", a ne "CORNER". [9]



Slika 5.3.1.10: elipse() sa središnjom točkom



Slika 5.3.1.11: elipse() s vanjskom točkom



Slika 5.3.1.12: elipse() s dvije točke

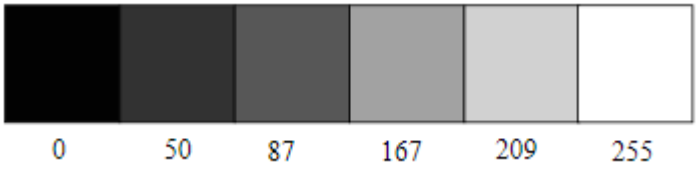
Primjer sa svim oblicima zajedno:

Slika 5.3.1.13: Primjer sa svim oblicima zajedno

```
size(630,400);
background(203,164,113);
stroke(0);
noFill();
strokeWeight(30);
rect(100, 100, 200,
200);
strokeWeight(10);
fill(0);
rect(160,150,15,120);
rect(190, 150, 55, 70,
3, 90, 90, 18);
fill(203,164,113);
fill(0);
stroke(0);
rect(135,150, 25, 6);
rect(140, 264, 50, 6);
textSize(45);
text("R", 230, 274);
fill(255);
textSize(45);
text("O", 282, 274);
fill(0);
textSize(45);
text("C E S S I N G", 325,
274);
```

5.3.2. Boje

Boja je definirana kao raspon brojeva. Najjednostavniji slučaj su crno-bijeli ili sivi tonovi. 0 znači crno, 255 znači bijelo. U međuvremenu, svaki drugi broj - 50, 87, 162, 209 i tako dalje - je nijansa sive boje, od crne do bijele. [10]



Slika 5.3.2.1: Raspon sive boje

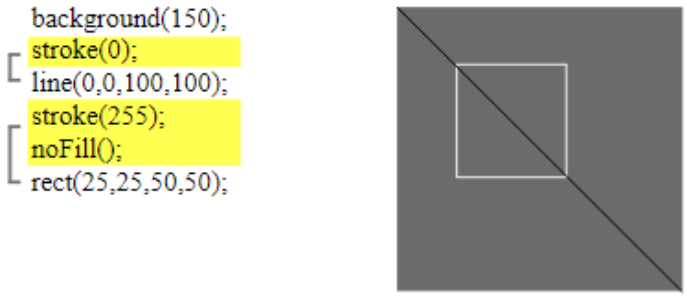
Dodavanjem funkcija stroke () i fill () prije nego što se nešto nacrta, može se postaviti boja bilo kojeg danog oblika. Tu je i funkcija background (), koja postavlja boju pozadine prozora.

Primjerice: [10]

```
size(200, 200);
background(255); // Postavljanje pozadine u bijelu boju.
stroke(0); // Postavljanje ruba u crnu boju.
fill(150); // Postavljanje unutrašnjosti oblika u sivu boju.
rect(50,50,75,100); // Crtanje pravokutnika.
```

Stroke ili fill mogu se eliminirati funkcijama: noStroke () i noFill (). Ako se ne želi imati obojeni rub, pravilan kod nije "stroke(0)", međutim, važno je zapamtiti da 0 nije "ništa", već označava crnu boju. Isto tako, ne smije se zaboraviti da se uklanjanjem obruba i ispune- sa noStroke () i noFill (), ništa neće pojaviti. [10]

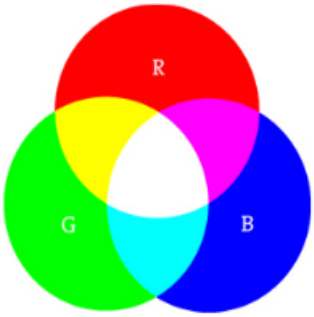
Crtanjem dvaju oblika, Processing će uvijek koristiti zadnje zadane crte i ispune, čitajući kod odozgo prema dolje. [10]



Slika 5.3.2.2: Boje i ispune

Digitalne boje također se izrađuju miješanjem triju osnovnih boja, ali djeluju drugačije od prave boje. Prvo, osnovne su razlike: crvena, zelena i plava (tj. "RGB" boja). [11]

S bojom na zaslonu miješa se svjetlost, a ne boja, pa su i pravila miješanja različita. [11]



Slika 5.3.2.3: RGB miješanja

Crvena + Zelena = Žuta

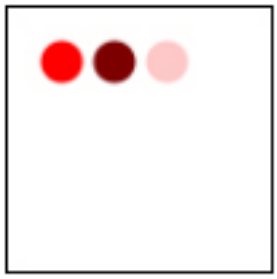
Crvena + Plava = Ljubičasta

Zelena + Plava = Cyan (plavo-zelena)

Crvena + Zelena + Plava = Bijela

Bez boje = Crna

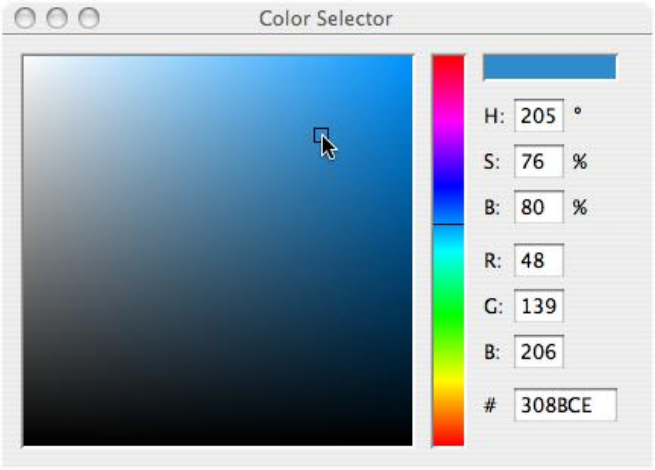
To podrazumijeva da su sve boje što je moguće svijetlije, ali naravno tu je raspon boja na raspolaganju. Crvena plus zelena plus plava daju sivu, a malo crvene plus malo plave daju tamno ljubičastu. Kao i kod sivih tonova, pojedinačni elementi boje izraženi su u rasponu od 0 (nijedna od tih boja) do 255 (koliko god je to moguće), a navedeni su redoslijedom R, G i B. [11]



Slika 5.3.2.4: Primjer RGB boja

```
Primjer: RGB boja
background(255);
noStroke();
// Svijetlo crvena
fill(255,0,0);
ellipse(20,20,16,16);
// Tamno crvena
fill(127,0,0);
ellipse(40,20,16,16);
// Roza (blijeda crvena)
fill(255,200,200);
ellipse(60,20,16,16);
```

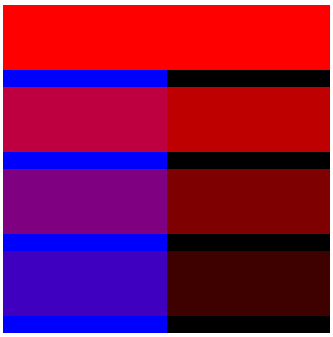
Processing također ima izbornik boja koji pomaže u odabiru. Ovom alatu pristupa se putem Alata (iz trake izbornika) → Odabir boje. [11]



Slika 5.3.2.5: Izbornik boja

Osim R, G i B komponente svake boje, postoji i dodatna neobvezna četvrta komponenta, koja se naziva "alfa" boja. Alfa znači transparentnost i osobito je korisno kod crtanja elemenata koji su djelomično prozirni. Alfa vrijednosti za sliku ponekad se zajednički nazivaju "alfa kanal" slike. Važno je shvatiti da pikseli nisu doslovno transparentni, to je jednostavno prikladna iluzija koja se postiže miješanjem boja.Iza kulisa, Processing uzima brojeve boja i dodaje postotak od jednog do postotka drugog, stvarajući optičku percepciju miješanja. [11]

Vrijednosti alfa također se kreću od 0 do 255, pri čemu je 0 potpuno prozirno (tj. 0% neprozirno) i 255 potpuno neprozirno (tj. 100% neprozirno). [11]



Slika 5.3.2.6: Primjer alfa transparentnos

```
Primjer:Alfa
transparentnost
size(200,200);
background(0);
noStroke();

// Ako nema četvrtog argumenta,
neprozirnost je 100%.
fill(0,0,255);
rect(0,0,100,200);

// 55% neprozirnosti.
fill(255,0,0,127);
rect(0,100,200,40);

// 25% neprozirnosti.
fill(255,0,0,63);
rect(0,150,200,40);

// 255 znači 100%
neprozirnosti.
fill(255,0,0,255);
rect(0,0,200,40);
```

RGB boja s rasponom od 0 do 255 nije jedini način bojanja u Processingu. U memoriji računala, o boji se uvijek govori kao o nizu od 24 bita (ili 32 u slučaju alfa boja). Međutim, Processing omogućuje pretvaranje vrijednosti u brojeve koje računalo razumije. To možete Boju u rasponu od 0 do 100, ali u postotcima može se gledati navođenjem prilagođenog colorMode (). [11]

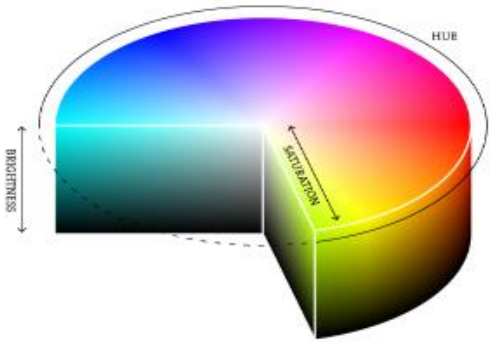
colorMode (RGB, 100);

Gore navedena funkcija kaže: "Želimo razmišljati o boji u smislu crvene, zelene i plave. Raspon RGB vrijednosti bit će od 0 do 100." Iako je to rijetko pogodno, moguće je imati različite raspone za svaku komponentu boje: [11]

colorMode (RGB, 100,500,10,255);

"Crvene vrijednosti idu od 0 do 100, zelene od 0 do 500, plave od 0 do 10, a alfa od 0 do 255". Također je moguće odrediti boje u načinu HSB (nijansa, zasićenost i svjetlina).

HSB boja radi na sljedeći način: [11]



Slika 5.3.2.7: Hue, saturation i brightness

Hue (Nijansa) - tip boje je u rasponu od 0 do 255 prema zadanim postavkama.

Saturation (Zasićenost) - živost boje, od 0 do 255 prema zadanim postavkama.

Svjetlina - Svjetlina boje, od 0 do 255 prema zadanim postavkama.

Pomoću funkcije colorMode () mogu se postaviti vlastiti rasponi za te vrijednosti. Neki preferiraju raspon od 0-360 za nijansu (360 stupnjeva na kotaču boja) i 0-100 za zasićenje i svjetlinu (0-100%). [10]

5.3.3. Interaktivnost

Elementi se kontroliraju na zaslonu putem raznih uređaja kao što su jastučići za dodir, trackball i joysticks, ali tipkovnica i miš ostaju najčešći ulazni uređaji za stolna računala. Računalni miš datira iz kasnih 1960-ih, kada je Douglas Engelbart predstavio uređaj kao element oN-Line sustava (NLS),jednog od prvih računalnih sustava s video prikazom. Koncept miša dodatno je razvijen u istraživačkom centru Xerox Palo Alto (PARC), ali njegovo uvođenje u Apple Macintosh 1984. bio je katalizator njegove sadašnje sveprisutnosti. Dizajn miša prošao je kroz mnoge revizije u posljednjih četrdeset godina, ali njegova je funkcija ostala ista. U Engelbartovoj izvornoj patentnoj prijavi 1970. godine miš je nazvao "pokazateljem položaja X-Y", a to još uvijek točno. [12]

Miš se koristi za kontrolu položaja pokazivača na zaslonu i za odabir elemenata sučelja. Položaj pokazivača čitaju računalni programi kao dva broja, x-koordinatu i y-koordinatu. Ovi brojevi mogu se koristiti za kontrolu atributa elemenata na zaslonu. Ako se te koordinate prikupljaju i analiziraju, one se mogu koristiti za izdvajanje informacija više razine, kao što su brzina i smjer miša. Ovi se podaci mogu koristiti za prepoznavanje gesta i uzoraka. [12]

Tipkovnice se obično koriste za unos znakova za sastavljanje dokumenata, e-pošte i trenutnih poruka, ali tipkovnica ima potencijal za korištenje izvan svoje izvorne namjere. Migracija tipkovnice s pisaćeg stroja na računalo proširila je njezinu funkciju kako bi omogućila pokretanje softvera, kretanje kroz izbornike softverskih aplikacija i navigaciju 3D okruženjima u igrama.
Prilikom pisanja vlastitog softvera imate slobodu koristiti podatke tipkovnice na bilo koji način.

Na primjer, osnovne informacije kao što su brzina i ritam prstiju mogu se odrediti brzinom kojom se tipke pritišću. Ove informacije mogu kontrolirati brzinu događaja ili kvalitetu kretanja. Također je moguće zanemariti znakove ispisane na samoj tipkovnici i koristiti položaj svakog ključa u odnosu na mrežu tipkovnice kao brojčanu poziciju. [12]

Moderna računalna tipkovnica je izravni potomak pisaćeg stroja.Položaj tipki na tipkovnici na engleskom jeziku je naslijeđen od ranih pisaćih strojeva. Ovaj raspored se naziva QWERTY zbog redoslijeda gornjeg retka tipki slova. Ovo više od stogodišnje mehaničko nasljeđe još uvijek utječe na način na koji danas pišemo softver. [12]

Varijable Processinga mouseX i mouseY pohranjuju x-koordinatu i y-koordinatu kursora u odnosu na podrijetlo u gornjem lijevom kutu zaslona. Kako bi se vidjele stvarne vrijednosti proizvedene tijekom pomicanja miša, potrebno je pokrenuti ovaj program da bi se ispisale vrijednosti u konzolu: [12]

```
void draw () {
FRAMERATE (12);
println (mouseX + ":" + mouseY);
}
```

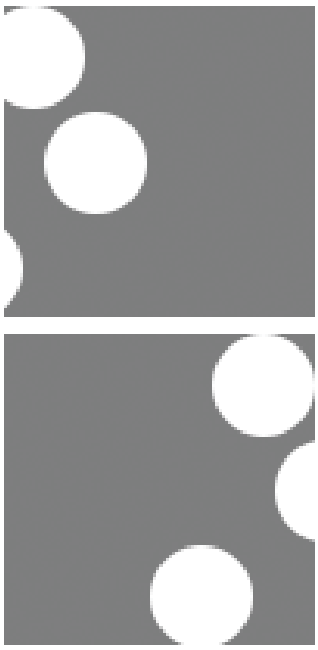
Kada se program pokrene, vrijednosti mouseX i mouseY su 0. Ako se kursor pomakne u prozor prikaza, vrijednosti se postavljaju na trenutni položaj pokazivača. Ako se pokazivač nalazi na lijevoj strani, vrijednost mouseX je 0, a vrijednost se povećava kako se kursor pomiče udesno. Ako je pokazivač na vrhu, vrijednost miša je 0, a vrijednost se povećava kako se kursor pomiče prema dolje. Ako se mouseX i mouseY koriste u programima bez draw-a (ili ako se noLoop () izvodi u setup-u (), vrijednosti će uvijek biti 0). [11]

Položaj miša najčešće se koristi za kontrolu položaja vizualnih elemenata na zaslonu. Zanimljiviji odnosi se stvaraju kada se vizualni elementi razlikuju od vrijednosti miša, umjesto da jednostavno oponašaju trenutnu poziciju. Dodavanje i oduzimanje vrijednosti od položaja miša stvara odnose koji ostaju konstantni, dok množenje i dijeljenje tih vrijednosti stvara promjenu vizualnih odnosa između položaja miša i elemenata na zaslonu. U prvom od sljedećih primjera, krug se izravno mapira na pokazivač, u drugom se brojevi dodaju i oduzimaju od položaja pokazivača kako bi se stvorili pomaci, a u trećem se množenje i dijeljenje koriste za mjerenje pomaka. [11]

```
void setup() {
size(100, 100);
noStroke();
}
void draw() {
background(126)
;
ellipse(mouseX,
mouseY, 33,
33);
}
```

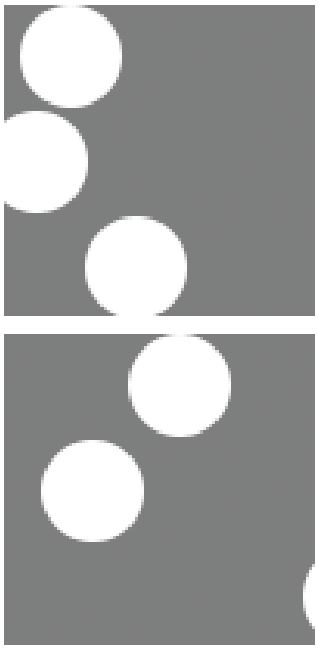
Slika 5.3.3.1: Izravno mapirar

kruga na pokazivač



```
void setup() {
size(100, 100);
noStroke();
}

void draw() {
background(126);
ellipse(mouseX, 16, 33, 33); //
Gornji krug
ellipse(mouseX+20, 50, 33, 33); //
Srednji krug
ellipse(mouseX-20, 84, 33, 33); //
Donji krug
}
```

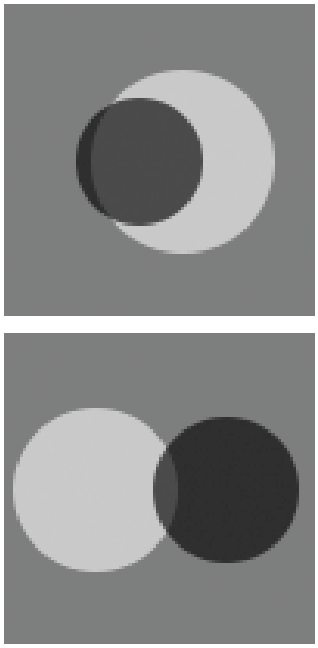
Slika 5.3.3.2: Stvaranje i mjerenje pomaka



```
void setup() {
size(100, 100);
noStroke();
}

void draw() {
background(126);
ellipse(mouseX, 16, 33, 33); //
Gornji krug
ellipse(mouseX/2, 50, 33, 33); //
Srednji krug
ellipse(mouseX*2, 84, 33, 33); //
Donji krug
}
```

Slika 5.3.3.3: Micanje krugova

Da bi se invertirala vrijednost miša, treba oduzeti vrijednost mouseX iz širine prozora i oduzeti vrijednost mouseY od visine zaslona. [11]



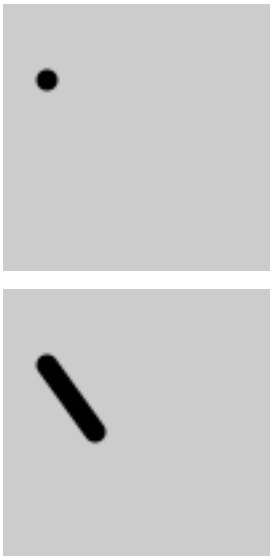
```
void setup() {
size(100, 100);
noStroke();
}
void draw() {
float x = mouseX;
float y = mouseY;
float ix = width - mouseX; // Obrnuto
X
float iy = height - mouseY; //
Obrnuto Y
background(126);
fill(255, 150);
ellipse(x, height/2, y, y);
fill(0, 159);
ellipse(ix, height/2, iy, iy);
}
```

Slika 5.3.3.4: Invertiranje vrijednosti pomoću miša

Varijable Processinga pmouseX i pmouseY pohranjuju vrijednosti miša iz prethodnog okvira. Ako se miš ne pomakne, vrijednosti će biti iste, ali ako se miš brzo kreće, može doći do velikih razlika između vrijednosti. Kod ovog primjera, da bi se vidjela razlika, potrebno je pomicati kursor različitim brzinama. [11]

```
void draw() {
frameRate(12);
println(pmouseX - mouseX);
}
```

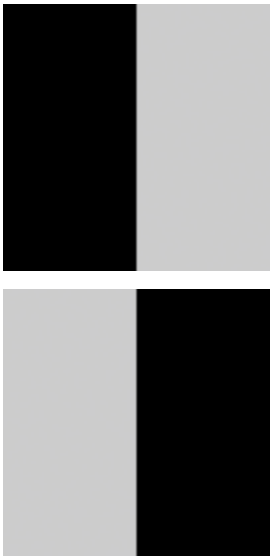
Crta se linija od prethodnog položaja miša do trenutne pozicije kako bi se prikazala promjena položaja u jednom okviru i otkrila brzina i smjer miša. Kada se miš ne pomiče, povlači se točka, ali brzi pokreti miša stvaraju duge crte. [11]



```
void setup() {
size(100, 100);
strokeWeight(8);
}

void draw() {
background(204);
line(mouseX,mouseY,
pmouseX, pmouseY);
}
```

Slika 5.3.3.5: Crtanje linije pomoću miša

Varijable mouseX i mouseY koriste se sa if strukturom kako bi se pokazivaču omogućio odabir područja na zaslonu. Sljedeći primjeri pokazuju pokazivač koji vrši odabir između različitih područja prozora zaslona. Prvi dijeli zaslon na polovice, a drugi dijeli zaslon na trećine. [11]



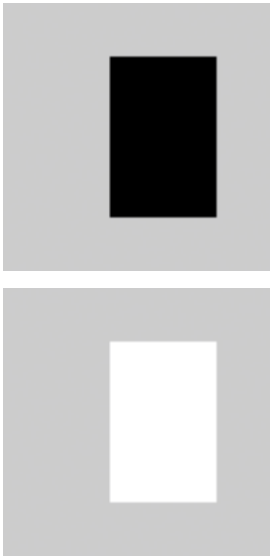
```
void setup() {
size(100, 100);
noStroke();
fill(0);
}
void draw() {
background(204);
if (mouseX < 50) {
rect(0, 0, 50, 100); //
Lijevo
} else {
rect(50, 0, 50, 100); //
Desno
}
}
```

Slika 5.3.3.6: Dijeljenje zaslona na polovice

```
void setup() {
size(100, 100);
noStroke();
fill(0);
}

void draw() {
background(204);
if (mouseX < 33) {
rect(0, 0, 33, 100); // Lijevo
} else if (mouseX < 66) {
rect(33, 0, 33, 100); // Sredina
} else {
rect(66, 0, 33, 100); // Desno
}
}
```

Slika 5.3.3.7: Dijeljenje zaslona na trećine

Logički operator && sa strukturom if koristi se za odabir pravokutnog područja zaslona. Kao što je prikazano u sljedećem primjeru, kada je izrađen relacijski izraz za testiranje svakog ruba pravokutnika(lijevo, desno, gore, dolje) i oni su spojeni s logičkim AND, cijeli relacijski izraz je istinit samo kada je pokazivač unutar pravokutnika. [12]



```
void setup() {
size(100, 100);
noStroke();
fill(0);
}
void draw() {
background(204);
if ((mouseX > 40) && (mouseX <
80) &&
(mouseY > 20) && (mouseY <
80)) {
fill(255);
} else {
fill(0);
}
rect(40, 20, 40, 60);
}
```

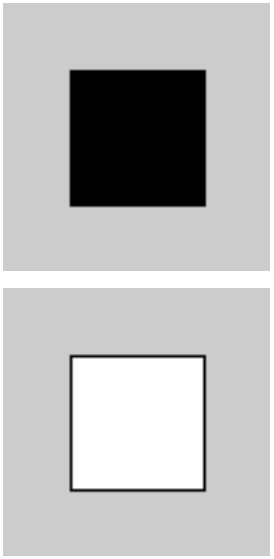
Slika 5.3.3.8: Relacijski izraz

Ovaj kod pita:"Je li pokazivač desno od lijevog ruba, lijevo od desnog ruba, iznad gornjeg ruba i da li je pokazivač iznad dna?" i skup sličnih pitanja i kombinira ih s ključnom riječi else kako bi se odredilo koje od definiranih područja sadrži pokazivač. [12]

```
void setup() {
size(100, 100);
noStroke();
fill(0);
}

void draw() {
background(204);
if ((mouseX <= 50) && (mouseY <= 50)) {
rect(0, 0, 50, 50); // Upper-left
} else if ((mouseX <= 50) && (mouseY > 50))
{
rect(0, 50, 50, 50); // Lower-left
} else if ((mouseX > 50) && (mouseY <= 50))
{
rect(50, 0, 50, 50); // Upper-right
} else {
rect(50, 50, 50, 50); // Lower-right
}
}
```

Slika 5.3.3.9: Kombiniranje skupa pitanja sa „else“

Računalni miševi i drugi srodni ulazni uređaji obično imaju jedan do tri gumba; Processing može otkriti kada su pritisnute ove tipke s varijablama mousePressed i mouseButton. Koristi se sa statusom gumba, položaj pokazivača omogućuje mišu da izvodi različite radnje. Na primjer, pritiskom na tipku kada je miš iznad ikone, može se odabrati, tako da se ikona može premjestiti na drugo mjesto na zaslonu. Varijabla mousePressed je istinita ako se pritisne bilo koja tipka miša i ako nije pritisnuta nijedna tipka miša. Varijabla mouseButton je LIJEVA, CENTRALNA ili DESNA ovisno o nedavno pritisnutoj tipki miša. Varijabla mousePressed vraća se na false čim se gumb oslobodi, ali varijabla mouseButton zadržava svoju vrijednost dok se ne pritisne drugi gumb. Ove varijable mogu se koristiti samostalno ili u kombinaciji za kontrolu softvera. [12]



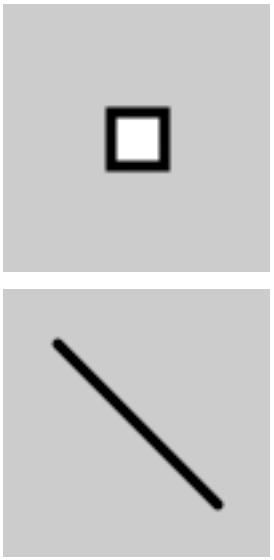
```
void setup() {
size(100, 100);
}
void draw() {
background(204);
if (mousePressed == true) {
fill(255); // Bijelo
} else {
fill(0); // Crno
}
rect(25, 25, 50, 50);
}
```

Slika 5.3.3.10: Korištenje varijable mousePressed

```
void setup() {
size(100, 100);
}
void draw() {
if (mouseButton == LEFT) {
fill(0); // Crno
} else if (mouseButton == RIGHT) {
fill(255); // Bijelo
} else {
fill(126); // Sivo
}
rect(25, 25, 50, 50);
}
```

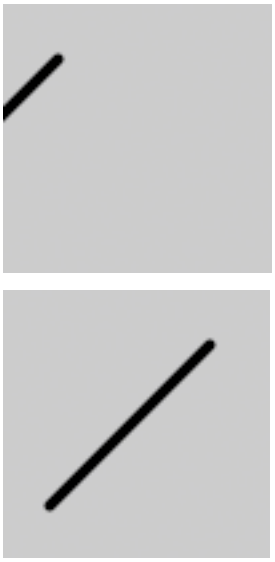
Slika 5.3.3.11: Korištenje varijable mouseButton

Processing registrira posljednju pritisnutu tipku i je li koja tipka trenutno pritisnuta. Boolean varijabla keyPressed je istinita ako je tipka pritisnuta, a lažna je ako nije. Ova varijabla uključuje se u test strukture if da bi se omogućilo pokretanje linija koda samo ako se pritisne tipka. Varijabla keyPressed ostaje istinita dok se tipka drži pritisnutom i postaje lažna samo kada se tipka oslobodi. [11]



```
void setup() {
size(100, 100);
strokeWeight(4);
}
void draw() {
background(204);
if (keyPressed == true) { // Ako je
tipkapritisnuta,
line(20, 20, 80, 80); // nacrtaj liniju
} else { // Ako nije pritisnuta,
rect(40, 40, 20, 20); // nacrtaj
pravokutnik.
} }
```

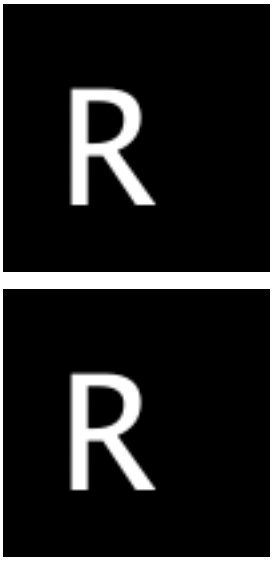
Slika 5.3.3.12: Utvrđivanje istinitosti varijable keyPressed



```
int x = 20;
void setup() {
size(100, 100);
strokeWeight(4);
}
void draw() {
background(204);
if (keyPressed == true) { // Ako je
tipka pritisnuta,
x++; // dodaj 1 varijabli x.
}
line(x, 20, x-60, 80);
}
```

Slika 5.3.3.13: Crtanje s varijablom keyPressed

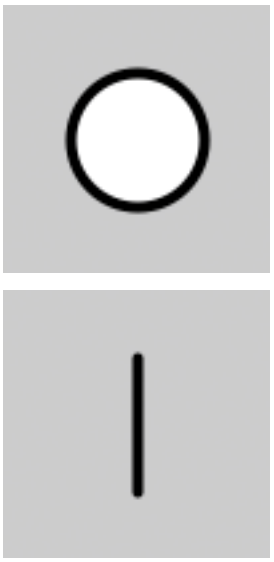
Ključna varijabla pohranjuje jedan alfanumerički znak. Konkretno, on sadrži nedavno pritisnutu tipku. Tipka se može prikazati na zaslonu pomoću funkcije text (). [11]




```
void setup() {
size(100, 100);
textSize(60);
}
void draw() {
background(0);
text(key, 20, 75); // Crtaj po
koordinatama(20,75)
}
```

Slika 5.3.3.14: Pohranjeni alfanumerički znak

Ključna varijabla može se koristiti za određivanje pritiska na određenu tipku. Sljedeći primjer koristi tipku izraza == 'A' za testiranje ako je pritisnuta tipka A. Pojedinačni navodnici označavaju A kao znak podataka tipa char. Ključ izraza == "A" uzrokovat će pogrešku jer dvostruki navodnici označavaju znak A kao niz i nije moguće usporediti niz sa znakom. Logički AND simbol, && operator, koristi se za povezivanje izraza s varijablom keyPressed kako bi se uvjerilo da je pritisnuta tipka veliko slovo A. [11]



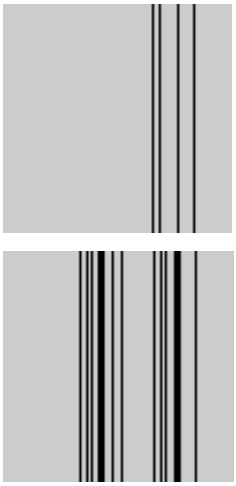
```
void setup() {
size(100, 100);
strokeWeight(4);
}
void draw() {
background(204);
// Ako je pritisnuto 'A' nacrtaj
liniju
if ((keyPressed == true) && (key
== 'A')) {
line(50, 25, 50, 75);
} else { // Ako nije, nacrtaj
elipsu
ellipse(50, 50, 50, 50);
}
}
```

Slika 5.3.3.15: keyPressed izraz

Prethodni primjer radi s velikim slovom A, ali ne i ako se pritisne malo slovo. Za provjeru velikih i malih slova, relacijski izraz proširuje se s logičkim OR, || relacijski operatorom. Redak 9 u prethodnom programu mijenja se u: [11]

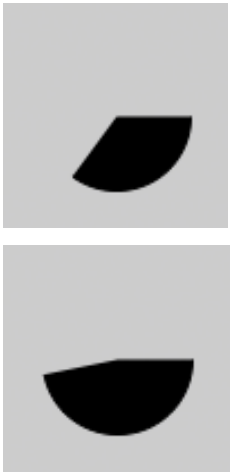
if ((keyPressed == true) && ((key == 'a') || (key == 'A'))) {

Budući da svaki znak ima brojčanu vrijednost definiranu ASCII tablicom, vrijednost varijable ključa može se koristiti kao i svaki drugi broj za kontrolu vizualnih atributa kao što su položaj i boja elemenata oblika. Na primjer, ASCII tablica definira veliko slovo A kao broj 65, a brojka 1 je definirana kao 49. [11]



```
void setup() {
size(100, 100);
stroke(0);
}
void draw() {
if (keyPressed == true)
{
int x = key - 32;
line(x, 0, x, height);
}
}
```

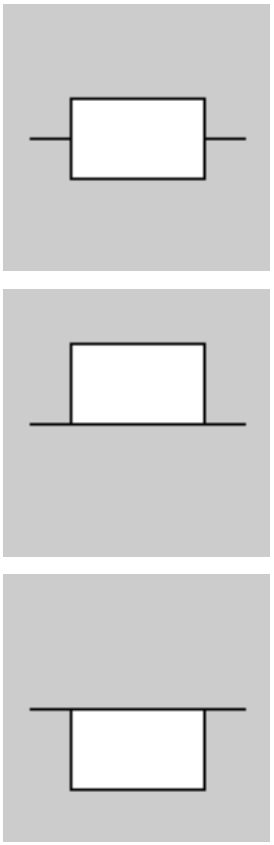
Slika 5.3.3.16: Crtanje pomoću ASCII tablice



```
float angle = 0;
void setup() {
size(100, 100);
fill(0);
}
void draw() {
background(204);
if (keyPressed ==
true) {
if ((key >= 32) &&
(key <= 126)) {
// Ako je ključ
alfanumerički,
}
// koristi njegovu
vrijednost kao kut
angle = (key - 32)
* 3;
}
}
arc(50, 50, 66,
66, 0,
radians(angle));
```

Slika 5.3.3.17: Korišt vrijednosti kao kut

Osim čitanja ključnih vrijednosti za brojeve, slova i simbole, Processing može čitati i vrijednosti drugih tipaka, uključujući tipke sa strelicama i tipke Alt, Control, Shift, Backspace, Tab, Enter, Return, Escape i Delete. Varijabla keyCode pohranjuje tipke ALT, CONTROL, SHIFT, GORE, DOLJE, LIJEVO i DESNO kao konstante. Prije utvrđivanja kodiranog ključa potrebno je prvo provjeriti je li ključ kodiran. Tipka izraza == CODED je istinita ako je ključ kodiran i pogrešan u suprotnom. Iako nisu alfanumerički, tipke uključene u ASCII specifikaciju (BACKSPACE, TAB, ENTER, RETURN, ESC i DELETE) neće biti identificirane kao kodirani ključ. [11]



```
int y = 35;

void setup() {
size(100, 100);
}

void draw() {
background(204);
line(10, 50, 90, 50);
if (key == CODED) {
if (keyCode == UP) {
y = 20;
} else if (keyCode == DOWN) {
y = 50;
}
} else {
y = 35;
}
rect(25, y, 50, 30);
}
```

Slika 5.3.3.18: Kodiranje tipaka UP i DOWN

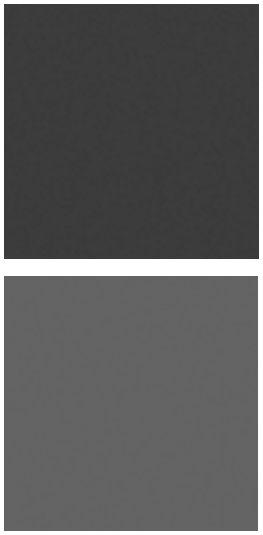
Kategorija funkcija koje se nazivaju događaji mijenjaju normalan tijek programa kada se dogodi akcija poput pritiska tipke ili miša. Događaj je pristojan prekid uobičajenog tijeka programa. Pritisci tipki i pokreti miša pohranjuju se do kraja draw (), gdje mogu poduzeti radnju koja ne ometa crtanje koje je trenutno u tijeku. Kod unutar funkcije događaja pokreće se svaki put kad se dogodi odgovarajući događaj. Na primjer, ako se pritisne tipka miša, kod unutar funkcije mousePressed() će se jednom pokrenuti i neće se ponovno prikazivati dok se ponovno

ne pritisne gumb. To omogućuje da se podaci proizvedeni od strane miša i tipkovnice čitaju nezavisno od onoga što se događa u ostatku programa. [12]

Događaji miša - funkcije događaja miša su mousePressed (), mouseReleased (), mouseMoved () i mouseDragged (): [12]

mousePressed() *Kod unutar ovog bloka se pokreće jednom kada se pritisne gumb miša*
mouseReleased() *Kod unutar ovog bloka se pokreće jednom kada se otpusti gumb miša*
mouseMoved() *Kod unutar ovog bloka se pokreće jednom kada se miš pomakne*
mouseDragged () *Kod unutar ovog bloka se pokreće jednom kada se miš pomakne dok se pritisne gumb miša*

Funkcija mousePressed() radi drugačije od varijable mousePressed. Vrijednost varijable mousePressed je istinita dok se gumb miša ne oslobodi. Stoga se može koristiti unutar draw() da bi se crta koda izvodila dok je miš pritisnut. Nasuprot tome, kod unutar funkcije mousePressed() izvodi se samo jednom kada se pritisne gumb. To ga čini korisnim kada se klikom miša pokrene radnja, kao što je čišćenje zaslona. U sljedećem primjeru, vrijednost pozadine postaje svjetlija svaki put kada se pritisne gumb miša. [12]

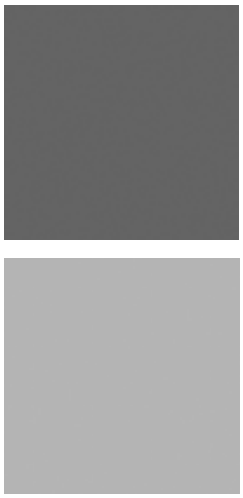


```
int gray = 0;
void setup() {
size(100, 100);
}
void draw() {
background(gray);
}
void mousePressed() {
gray += 20;
}
```

Slika 5.3.3.19: Mijenjanje pozadine pritiskom tipke miša

Sljedeći primjer je isti kao gore, ali siva varijabla postavljena je u funkciji događaja mouseReleased(), koja se zove jednom svaki put kada se gumb oslobodi. Ta se razlika može

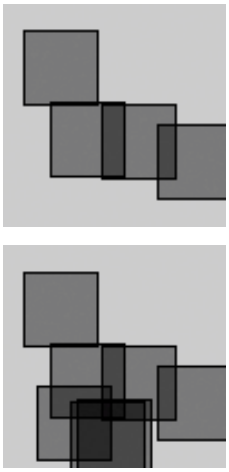
vidjeti samo pokretanjem programa i klikom na gumb miša Vrijednost pozadine mijenja se samo kada je gumb otpušten. [12]



```
int gray = 0;
void setup() {
size(100, 100);
}
void draw() {
background(gray);
}
void mouseReleased() {
gray += 20;
}
```

Slika 5.3.3.20: Mijenjanje pozadine otpuštanjem tipke miša

Općenito nije dobra ideja crtati unutar funkcije događaja, ali to se može učiniti pod određenim uvjetima. Prije crtanja unutar tih funkcija, važno je razmisliti o tijeku programa. U ovom primjeru, kvadrati se crtaju unutar mousePressed() i ostaju na zaslonu jer nema background() unutar draw(). Ali ako se koristi background(), vizualni elementi nacrtani unutar jedne od funkcija događaja miša pojavit će se na zaslonu samo za jedan okvir, ili, po zadanom, 1/60 sekunde. Zapravo, primijetit ćete da ovaj primjer nema ništa unutar draw(), ali mora biti tu da prisili Processing da nastavi slušati događaje. Ako je unutar draw() izvođena funkcija background(), pravokutnici će treptati na zaslonu i nestati. [11]



```
void setup() {
size(100, 100);
fill(0, 102);
}
void draw() {
} // Prazno draw() pokreće program
void mousePressed() {
rect(mouseX, mouseY, 33, 33);
}
```

Slika 5.3.3.21:Crtanje oblika unutar draw() funckije

Kod unutar funkcija događaja mouseMoved() i mouseDragged() izvodi se kada se promijeni položaj miša. Kod u mouseMoved() bloku se pokreće na kraju svakog pomaka miša i nije potrebno pritisnuti nijednu tipku. Kod u bloku mouseDragged() radi isto kada se pritisne tipka na mišu. Ako miš ostaje u istom položaju od okvira do okvira, kod unutar tih funkcija ne radi. U ovom primjeru sivi krug slijedi kursor kada gumb nije pritisnut, a crni krug slijedi kursor kada se pritisne tipka na mišu. [11]

Slika 5.3.3.22:Izvođenje različitih radnja uz pritiske tipke miša

```
int dragX, dragY, moveX, moveY;
void setup() {
size(100, 100);
noStroke();
}
void draw() {
background(204);
fill(0);
ellipse(dragX, dragY, 33, 33); // Crni
krug
fill(153);
ellipse(moveX, moveY, 33, 33); // Sivi
krug
}
void mouseMoved() { // Pomakni sivi
krug
moveX = mouseX;
moveY = mouseY;
}
void mouseDragged() { // Pomakni crni
krug
dragX = mouseX;
dragY = mouseY;
}
```

Događaji tipaka - Svaki pritisak tipke registrira se preko funkcija događaja na tipkovnici keyPressed() i keyReleased(): [12]

keyPressed() *Kod unutar ovog bloka se pokreće jednom kada se pritisne bilo koja tipka*
keyReleased() *Kod unutar ovog bloka se pokreće jednom kada se otpusti bilo koji ključ*

Svaki put kada se pritisne tipka, kod unutar bloka keyPressed() se pokreće jednom. Unutar ovog bloka moguće je testirati koji je ključ pritisnut i koristiti tu vrijednost za bilo koju svrhu. Ako se ključ zadržava dulje vrijeme, kod unutar bloka keyPressed() može se izvoditi mnogo puta u brzom slijedu jer će većina operativnih sustava preuzeti i opetovano pozivati funkciju keyPressed(). Vrijeme potrebno za početak ponavljanja i brzina ponavljanja razlikovat će se od računala do računala, ovisno o postavkama tipkovnice. U ovom primjeru, vrijednost boolean varijable drawT postavlja se iz false na true kada se pritisne T ključ; ovo uzrokuje da linije koda prikažu pravokutnike u draw(). [12]


```
boolean drawT = false;
void setup() {
size(100, 100);
noStroke();
}
void draw() {
background(204);
if (drawT == true) {
rect(20, 20, 60, 20);
rect(39, 40, 22, 45);
}
}
void keyPressed() {
if ((key == 'T') || (key == 't'))
{
drawT = true;
}
}
```

Slika 5.3.3.23: Mijenjanje vrijednosti varijable drawT

Svaki put kada se tipka otpusti, kod unutar bloka keyReleased() se pokreće jednom. Sljedeći primjer temelji se na prethodnom kodu; svaki put kada se tipka otpusti, boolean varijabla drawT vraća se u false kako bi se zaustavio prikaz u okviru draw (). [12]


```
boolean drawT = false;
void setup() {
size(100, 100);
noStroke();
}
void draw() {
background(204);
if (drawT == true) {
rect(20, 20, 60, 20);
rect(39, 40, 22, 45);
}
}
void keyPressed() {
if ((key == 'T') || (key == 't'))
{
drawT = true;
}
}
void keyReleased() {
drawT = false;
}
```

Slika 5.3.3:24: Vraćanje boolean varijable drawT u false

5.3.4. Tipografija

Slova na zaslonu nastaju postavljanjem boje piksela. Kvaliteta tipografije ograničena je razlučivošću zaslona. Jer, povijesno gledano, ekrani imaju nisku rezoluciju u usporedbi s papirom, ali razvijene su tehnike kako bi se poboljšao izgled tipa na zaslonu. Fontovi na najranijim Apple Macintosh računalima sastojali su se od malih bitmap slika stvorenih pri određenim veličinama poput 10, 12 i 24 točaka. Koristeći ovu tehnologiju, varijacija svakog fonta je dizajnirana za svaku veličinu određenog slova. Na primjer, znak A u fontu San Francisca koristio je drugu sliku za prikaz znaka veličine 12 i 18. Kada je pisač LaserWriter uveden 1985., PostScript tehnologija je definirala fontove s matematičkim opisom obrisa svakog znaka. To je omogućilo da tip na ekranu razmjera do velikih veličina i dalje izgleda glatko. Apple i Microsoft kasnije razvili su TrueType, drugi oblik fonta. U novije vrijeme ove su se tehnologije spojile u format OpenType. U međuvremenu su uvedene metode za ujednačavanje teksta na zaslonu. Ove anti-aliasing tehnike koriste sive piksele na rubu znakova kako bi kompenzirali nisku razlučivost zaslona. [13]

Funkcija text() se koristi za crtanje slova, riječi i odlomaka na zaslonu. U najjednostavnijoj uporabi, prvi parametar može biti string, char, int ili float. Drugi i treći parametar postavljaju položaj teksta. Prema zadanim postavkama drugi parametar definira udaljenost od lijevog ruba prozora; treći parametar definira udaljenost od osnovne linije teksta do vrha prozora. Funkcija textSize() definira veličinu slova koja će se crtati u jedinicama piksela. Broj koji se koristi za određivanje veličine teksta neće biti precizna visina svakog slova, a razlika ovisi o dizajnu svakog fonta. Funkcija fill() kontrolira boju i prozirnost teksta. Ova funkcija utječe na tekst na isti način na koji utječe na oblike kao što su rect() i ellipse(), ali tekst ne utječe na stroke(). [13]


```
fill(0);
text("LAX", 0, 40); // Napiši "LAX" na
koordinatama (0,40)
text("AMS", 0, 70); // Napiši "AMS" na
koordinatama (0,70)
text("FRA", 0, 100); // Napiši "FRA"
na koordinatama (0,100)
```

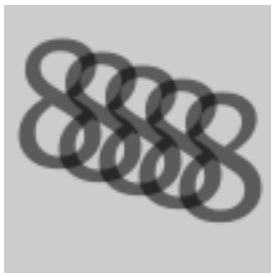
Slika 5.3.4.1: Crtanje teksta na koordinatama


```
textSize(32); // Postavi veličinu
teksta na 32
fill(0);
text("LAX", 0, 40);
text("ORD", 0, 70);
text("DAY", 0, 100);
```

Slika 5.3.4.2: Postavljanje veličine teksta


```
textSize(32);
fill(0); // Ispuni crnom bojom
text("LAX", 0, 40);
fill(126); // Ispuni sivom bojom
text("HKG", 0, 70);
fill(255); // Ispuni bijelom
bojom
text("PVG", 0, 100);
```

Slika 5.3.4.3: Ispun teksta bojom



```
textSize(64);
fill(0, 140); // Ispuni crnom
bojom s malom neprozirnošću
text("8", 0, 60);
text("8", 15, 65);
text("8", 30, 70);
text("8", 45, 75);
text("8", 60, 80);
```

Slika 5.3.4.4: Ispuna teksta bojom s neprozirnošću

Druga verzija text() crta znakove unutar pravokutnika. U tom korištenju drugi i treći parametri određuju položaj gornjeg lijevog kuta okvira, a četvrti i peti parametri određuju širinu i visinu okvira. Ako duljina teksta premašuje dimenzije definiranog okvira, tekst se neće prikazati. [13]


```
String s = "Five hexing wizard
bots jump quickly.";
fill(0);
text(s, 10, 10, 60, 80);
```

Slika 5.3.4.5: Dovoljno prostora za prikaz teksta


```
String s = "Five hexing
wizard bots jump quickly.";
fill(0);
text(s, 10, 10, 60, 55); //
Premalo prostora
```

Slika 5.3.4.6: Premalo prostora za prikaz teksta

Vektorski fontovi :

Za rad s različitim fontovima od zadanih, više je funkcija potrebno za pripremu fonta koji će se koristiti.Funkcija createFont() koristi se za pretvaranje TrueType fonta (.ttf) ili fonta OpenType (.otf) tako da se može prikazati kroz text(). Funkcija textFont() se koristi za definiranje trenutnog fonta za prikaz. Može se koristiti bilo koji kompatibilan font instaliran na računalu ili pohranjen u mapi podataka skica. Sljedeći kratki program koristi se za ispis popisa dostupnih instaliranih fontova u konzolu: [13]

```
String[ ] fontList = PFont.list( );
printArray(fontList);
```

Funkcija printArray() koristi se za pisanje svakog fonta u novom retku. Prvih nekoliko opcija ispisanih na konzoli su opće tipografske klasifikacije kao što su Serif, SansSerif i Monospaced. Pomoću ovih opcija definira se stil, ali ne i određeni font. Računalo će proizvesti različite rezultate ovisno o instaliranom operativnom sustavu i prilagođenim fontovima. Popis počinje s općim kategorijama fontova koje će funkcionirati na različitim platformama, a zatim nastaviti s određenim imenima fontova. [13]

[0] "Serif"
[1] "SansSerif"
[2] "Monospaced"
[3] "Dialog"
[4] "DialogInput"
[5] "ACaslonPro-Bold"
.
.
.
[567] "ZXX-Sans"
[568] "ZXX-Xed"
[569] "ZapfDingbatsITC"
[570] "Zapfino"
[571] "Ziggurat-Black"
[572] "Ziggurat-BlackItalic"

Prije nego što se font koristi u programu, mora se pretvoriti i postaviti kao trenutni font. Processing ima jedinstveni tip podataka koji se naziva PFont za pohranu podataka o fontu. Napravite novu varijablu tipa PFont i koristite funkciju createFont() za pretvaranje fonta. Prvi parametar za createFont() je naziv fonta za pretvorbu, a drugi parametar definira osnovnu veličinu fonta. Funkcija textFont() se tada mora koristiti za postavljanje trenutnog fonta. [13]

Slika 5.3.4.7:Korištenje drugog fonta

```
PFont zigBlack;
void setup() {
size(100, 100);
zigBlack = createFont("Ziggurat-
Black", 32);
textFont(zigBlack);
fill(0);
}
void draw() {
background(204);
text("LAX", 0, 40);
text("LHR", 0, 70);
text("TXL", 0, 100);
}
```

Da bi se osiguralo učitavanje fonta na sva računala, bez obzira na to je li font instaliran, potrebno je dodati datoteku u mapu podataka skice. (Fontovi u mapi s podacima ne ispisuju se na popisu konzole.) Kada se koriste fontovi unutar mape s podacima, cijelo ime datoteke, uključujući proširenje tipa podataka, mora biti zapisano kao parametar za createFont(). Sljedeći primjer sličan je prethodnom primjeru, ali koristi font OpenType unutar mape s podacima. Ona koristi izvorni kod Pro, open source font koji se može naći online i preuzeti putem web preglednika. [13]

Slika 5.3.4.8: Korištenj OpenType fonta unutar s podacima

```
PFont sourceLight;
void setup() {
size(100, 100);
sourceLight=
createFont("SourceCodePro-
Light.otf", 34);
textFont(sourceLight);
fill(0);
}
void draw() {
background(204);
text("LAX", 0, 40);
text("LHR", 0, 70);
text("TXL", 0, 100);
}
```

Za korištenje dva fonta u jednom programu, potrebno je stvoriti dvije PFont varijable i koristiti funkciju textFont() za promjenu trenutnog fonta. Na temelju prethodna dva primjera, Ziggurat-Black font učitava se sa svog mjesta na lokalnom računalu i učitava izvorni kod Pro iz mape s podacima. [13]

Slika 5.3.4.9:Učitavanje fonta s lokalnog računala

```
PFont zigBlack, sourceLight;
void setup() {
size(100, 100);
zigBlack = createFont("Ziggurat-
Black", 24);
sourceLight =
createFont("SourceCodePro-
Light.otf", 34);
fill(0);
}
void draw() {
background(204);
textFont(zigBlack);
text("LAX", 0, 40);
textFont(sourceLight);
text("LHR", 0, 70);
textFont(zigBlack);
text("TXL", 0, 100);
}
```

Pikselski fontovi:

Processing također može raditi s fontovima koje pretvara u teksture malih slika. Ovi fontovi nisu toliko fleksibilni i jasni kao fontovi konvertirani za Processing s createFont () i korišteni sa zadanim rendererom, ali su optimiziraniji za upotrebu s P2D i P3D prikazivačima. Format fonta piksela koji se koristi u Processingu razvijen je u MIT Media Labu sredinom 1990-ih u radionici Visual Language Workshop (VLW). VLW format sprema svaki alfanumerički znak kao mrežu piksela. To je brz način prikaza teksta i omogućuje uključivanje fonta sa skicom bez uključivanja vektorskih podataka. [13]

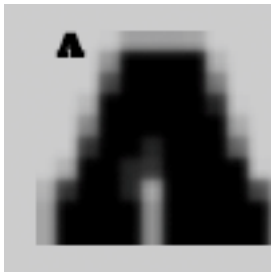
Za pretvaranje fonta u VLW format, odabire se "Stvori font" u izborniku Alati. Otvara se prozor u kojem su prikazana imena kompatibilnih fontova instaliranih na računalu. Odabire se font s popisa i potvrđuje se sa "OK". Font se generira i kopira u mapu podataka trenutne skice. Alat Create Font nudi mogućnost postavljanja veličine fonta i odabira ima li glatke, antialiased rubove. Ovaj alat također nudi mogućnost izvoza "Svi znakovi", što znači da će svaki znak u fontu biti uključen i stoga će povećati veličinu datoteke. [13]

Sljedeći primjer koristi isti font kao prethodni primer createFont(). Jedina razlika je zamjena te funkcije s loadFont(). Za pokretanje ovih primjera, potrebno je koristiti alat "Stvori font" kako bi se font pretvorio u VLW datoteku. Treba promijeniti naziv parametra u loadFont() tako da odgovara nazivu VLW datoteke koja je stvorena. [13]

Slika 5.3.4.10:Korištenje funkcije loadFont()

```
PFont zigBlack;
void setup() {
size(100, 100);
zigBlack = loadFont("Ziggurat-
Black-32.vlw");
textFont(zigBlack);
fill(0);
}
void draw() {
background(204);
text("LAX", 0, 40);
text("LHR", 0, 70);
text("TXL", 0, 100);
}
```

Kada je font nacrtan u različitoj veličini od veličine na kojoj je stvoren, on se umanjuje i stoga ne izgleda uvijek tako jasno i glatko. Na primjer, ako je font stvoren s 12 piksela i prikazuje se u 96 piksela, pojavit će se mutno. [13]



Slika 5.3.4.11:Mutna pojava fonta

```
PFont zigBlack;
void setup() {
size(100, 100);
zigBlack =
loadFont("Ziggura
t-Black-12.vlw");
textFont(zigBlack
);
fill(0);
}
void draw() {
background(204);
textSize(12);
text("A", 20, 20);
textSize(96);
text("A", 20, 90);
}
```

Processing uključuje funkcije koje kontroliraju leading (razmak između redaka teksta) i poravnanje. Processing može također izračunati širinu bilo kojeg znaka ili grupe znakova, što je korisna tehnika za raspoređivanje oblika i tipografskih elemenata. Funkcija textLeading() postavlja razmak između redaka teksta. Ima jedan parametar koji definira taj prostor u jedinicama piksela. [13]

Slika 5.3.4.12: Postavljanje razmaka između teksta

```
String lines = "L1 L2 L3";
textSize(12);
fill(0);
textLeading(10);
text(lines, 10, 15, 30,
100);
textLeading(20);
text(lines, 40, 15, 30,
100);
textLeading(30);
text(lines, 70, 15, 30,
100);
```

Slova i riječi mogu se izvući iz njihovih središnjih, lijevih i desnih rubova. Funkcija textAlign() postavlja poravnanje za crtanje teksta kroz svoj parametar, koji može biti LEFT, CENTER ili RIGHT. On postavlja karakteristike prikaza slova u odnosu na x-koordinatu navedenu u funkciji text(). [13]

Slika 5.3.4.13: Izvlačenje teksta iz središta

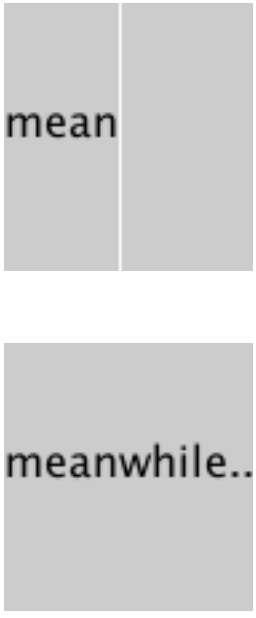
```
fill(0);
textSize(12);
line(50, 0, 50, 100);
textAlign(LEFT);
text("Left", 50, 20);
textAlign(RIGHT);
text("Right", 50, 40);
textAlign(CENTER);
text("Center", 50, 80);
```

Postavke za textSize(), textLeading() i textAlign() koristit će se za sve naknadne pozive na text() funkciju. Međutim, funkcija textSize() poništiti će vođenje teksta, a funkcija textFont() će resetirati veličinu i leading. Funkcija textWidth() izračunava i vraća širinu piksela bilo kojeg znaka ili tekstualnog niza. Taj se broj izračunava iz trenutnog fonta i veličine definirane funkcijama textFont() i textSize(). Budući da su slova svakog fonta različite veličine, a slova unutar mnogih fontova imaju različite širine, ova funkcija je jedini način da se sazna koliko je dugačak niz znakova prikazan na zaslonu. Iz tog razloga uvijek se koristi textWidth() za pozicioniranje elemenata u odnosu na tekst. [13]


```
String s = "AEIOU";
float tw; // Text width
fill(0);
textSize(14);
tw = textWidth(s);
text(s, 4, 40);
rect(4, 42, tw, 5);
textSize(28);
tw = textWidth(s);
text(s, 4, 76);
rect(4, 78, tw, 5);
```

Slika 5.3.4.14: Mijenjanje veličine teksta

Crtanje slova na zaslonu postaje privlačnije kada se koristi u kombinaciji s tipkovnicom. Funkcija događaja keyPressed() može se koristiti za snimanje svakog slova kako je upisano. Sljedeća dva primjera koriste ovu funkciju za čitanje i analiziranje ulaza s tipkovnice pomoću metoda String. Svako upisano slovo dodaje se na kraj niza. Prvi primjer prikazuje niz dok raste kako se tipke pritisnu i uklanja slova od kraja kada se pritisne tipka Backspace. Drugi primjer temelji se na prvom - kada se pritisne tipka Return ili Enter, program provjerava je li upisana riječ “siva” ili “crna”. Ako je unesena jedna od ovih riječi, pozadina se mijenja u tu vrijednost. [13]




```
String letters = "";
void setup() {
size(100, 100);
stroke(255);
fill(0);
textSize(16);
}
void draw() {
background(204);
float cursorPosition =
textWidth(letters);
line(cursorPosition,
0, cursorPosition,
100);
text(letters, 0, 50);
}
```

```
void keyPressed() {
if (key == BACKSPACE)
{
if (letters.length() >
0) {
letters =
letters.substring(0,
letters.length()-1);
}
} else if
(textWidth(letters+key
) < width) {
letters = letters +
key;
}
}
```

Slika 5.3.4.15: Analiziranje ulaza s tipkovnice pomoću metoda String


```
String letters = "";
int back = 102;
void setup() {
size(100, 100);
textSize(16);
textAlign(CENTER);
}
void draw() {
background(back);
text(letters, 50, 50);
}
void keyPressed() {
if ((key == ENTER) ||
(key == RETURN)) {
letters =
letters.toLowerCase();
println(letters); //
Print to console to
see input
if
(letters.equals("black
")) {
back = 0;
} else if
(letters.equals("gray"
)) {
back = 204;
}

letters = ""; // Clear
the variable
} else if ((key > 31)
&& (key != CODED)) {
// If the key is
alphanumeric, add it
to the String
letters = letters +
key;
}
}
```

Slika 5.3.4.16: Provjera upisanih riječi

Tipografskim elementima mogu se dodijeliti ponašanja koja definiraju osobnost u odnosu na miš ili tipkovnicu. Riječ može izraziti „agresivnost“ ili „plahost“ tako što se brzo kreće prema kursoru ili se polagano udaljava od kursora. Sljedeći primjeri pokazuju osnovne primjene ovog područja. U prvom, riječ „avoid“ ostaje udaljena od kursora jer je njezin položaj postavljen na obrnuto od položaja pokazivača. U drugom, riječ škakljanje treperi kada pokazivač lebdi nad svojim položajem. [13]


```
void setup() {
size(100, 100);
textSize(24);
textAlign(CENTER);
}

void draw() {
background(204);
text("avoid", width-mouseX,
height-mouseY);
}
```

Slika 5.3.4.17: Dodjela ponašanja tekstu


```
float x = 33;
float y = 60;
void setup() {
size(100, 100);
textSize(24);
noStroke();
}
void draw() {
fill(204, 120);
rect(0, 0, width, height);
fill(0);
// Ako je pokazivač na tekstu,
promijeni poziciju
if ((mouseX >= x) && (mouseX <=
x+55) &&
(mouseY >= y-24) && (mouseY <= y))
{
x += random(-2, 2);
y += random(-2, 2);
}
text("tickle", x, y);
}
```

Slika 5.3.4.18:Primjer treperenja riječi

5.3.5. Slike i pikseli

Za početak, ispituje se PImage, klass za učitavanje i prikazivanje slike, kao i za gledanje njezinih piksela: [14]

```
PImage img;     // Deklaracija varijable tipa PImage

void setup() {
size(320,240);
// Stvaranje nove instance PImage učitavanjem slikovne datoteke
img = loadImage("mysummervacation.jpg");
}

void draw() {
background(0);
// Crtaj sliku na zaslon po koordinatama (0,0)
image(img,0,0);
}
```

Korištenje instance PImage objekta se ne razlikuje od korištenja korisnički definirane klase. Prvo, deklarirana je varijabla tipa PImage, nazvana "img". Drugo, nova instanca PImage objekta kreira se pomoću metode loadImage(). loadImage() uzima jedan argument, niz koji označava ime datoteke i učitava tu datoteku u memoriju. loadImage() traži slikovne datoteke pohranjene u mapi "data" u skici. U gornjem primjeru, može se činiti pomalo neobičnim to što nije pozvan "konstruktor" za instanciranje PImage objekta, pozivanjem "new PImage()". Uostalom, u većini primjera koji se odnose na objekte, konstruktor je nužan za stvaranje instance objekta. [14]

```
Spaceship ss = new Spaceship();
Flower flr = new Flower(25);
```

već:

```
PImage img = loadImage("file.jpg");
```

Zapravo, funkcija loadImage() obavlja rad konstruktora, vraćajući potpuno novu instancu Pimage objekta generiranog iz navedenog naziva datoteke. To je PImage konstruktor za učitavanje slika iz datoteke. Za stvaranje prazne slike koristi se funkcija createimage(). [14]

```
// Kreirajte praznu sliku, 200 x 200 piksela s RGB bojom
PImage img = createImage (200, 200, RGB);
```

Također treba napomenuti da je proces učitavanja slike sa tvrdog diska u memoriju spor, a trebase pobrinuti da program to učini samo jednom, u setupu(). Učitavanje slika u draw() može rezultirati sporim performansama kao i pogreškama "Out of Memory". Kad je slika učitana, prikazuje se s funkcijom image(). Funkcija image() mora sadržavati 3 argumenta - sliku koja se prikazuje, x lokaciju i lokaciju y. Moguće je dodati dva argumenta za promjenu veličine slike na određenu širinu i visinu. [14]

```
image(img,10,20,90,60);
```

Kod prikaza slika, moguća je promjena njezina izgled. Moguće je postići da se slika pojavi tamnija, prozirnija, plavija, itd. Ova vrsta jednostavnog filtriranja slika postiže se funkcijomProcessing tint(). tint() je u suštini ekvivalentna slici fill(), koja postavlja boju i alfa transparentnost za prikazivanje slike na zaslonu. Slika, ipak, obično ne sadrži sve boje. Argumenti za tint() jednostavno određuju koliki dio zadane boje treba koristiti za svaki piksel te slike, kao i koliko bi se ti pikseli trebali pojaviti. [14]

Za sljedeće primjere pretpostavlja se da su dvije slike (suncokret i pas) učitane i pas je prikazan kao pozadina (što će nam omogućiti da pokažemo transparentnost). [6]

```
PImage sunflower = loadImage ("sunflower.jpg");
PImage dog = loadImage ("dog.jpg");
background (dog);
```

Ako ton () primi jedan argument, to utječe samo na svjetlinu slike. [6]

```
// Slika ostaje u
originalnom stanju.
tint(255);
image(sunflower,0,0);
```

```
//Slika izgleda
tamnije.
tint(100);
image(sunflower,0,0);
```

Slika 5.3.5.1: Mijenjanje tona slike

Drugi argument mijenja alfa transparentnost slike. [6]

```
// Slika je na 50%
neprozirnosti.
tint(255,127);
image(sunflower,0,0);
```

Slika 5.3.5.2: Mijenjanje alfa transparentnosti slike

Tri argumenta utječu na svjetlinu crvene, zelene i plave komponente svake boje. [6]

```
// Nema crvene,
većina zelene i
cijela plava.
tint(0,200,255);
image(sunflower,0,0
);
```

Slika 5.3.5.3: Utjecaj na svjetlinu RGB komponenata

Konačno, dodavanjem četvrtog argumenta metodi manipulira s alfom (isto kao s 2). Usput, raspon vrijednosti za tint() može se odrediti pomoću colorMode(). [6]

```
// Slika je crvene boje
te je prozirna.
tint(255,0,0,100);
image(sunflower,0,0);
```

Slika 5.3.5.4: Mijenjanje boje i neprozirnosti slike

Linija se ne pojavljuje samo pozivanjem line (), ona se pojavljuje jer se boji svaki piksel duž linearne putanje između dvije točke. Svaki piksel na zaslonu ima X i Y poziciju u dvodimenzionalnom prozoru. Međutim, pikseli niza imaju samo jednu dimenziju, čuvajući vrijednosti boja u linearnom nizu. [6]

Slika 5.3.5.5: Izgled i pohrana piksela

Ovaj program postavlja svaki piksel u prozoru na slučajnu vrijednost sivih tonova. Niz piksela je poput drugog niza, jedina razlika je što ga nije potrebno deklarirati jer je to ugrađena varijabla Processinga. [6]

```
size(200, 200);
// Prije rada s pikselima
loadPixels();
// Petlja kroz svaki piksel
for (int i = 0; i < pixels.length; i++) {
// Izaberi nasumičan broj, 0 do 255
float rand = random(255);
// Kreiraj boju u sivim tonovima baziranu na nasumičnom broju
color c = color(rand);
// Postavi piksel na toj lokaciji u tu boju
pixels[i] = c;
}
// Kada smo gotovi s radom s pikselima
updatePixels();
```

Prvo, treba istaknuti nešto važno u gornjem primjeru. Kad god se pristupa pikselima prozora Processinga, mora se obavijestiti Processing za ovu aktivnost. To se postiže s dvije funkcije: [6]

loadPixels() - ova funkcija se poziva prije nego što pristupite nizu piksela
updatePixels () - ova funkcija se zove nakon što završite s nizom piksela

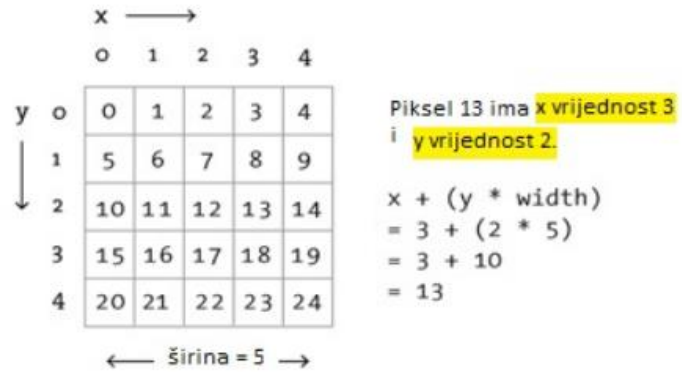
U gornjem primjeru, budući da su boje nasumično postavljene, nije de trebalo brinuti o tome gdje su pikseli na zaslonu dok im se pristupa, jer se jednostavno postavljaju svi piksele bez obzira na njihovu relativnu lokaciju. Međutim, u mnogim aplikacijama za obradu slika, XY mjesto piksela je ključna informacija. Jednostavan primjer za to bi mogao biti, postavljanje svakog parnog stupca piksela na bijelo, a svakog neparanog na crno. [14]

1) Pretpostavlja se prozor ili slika s danom širinom i visinom.

*2) Skup piksela ima ukupan broj elemenata jednakih WIDTH * HEIGHT(širina*visina).*

3) Za bilo koju X, Y točku u prozoru, lokacija u dvodimenzionalnom nizu piksela je:

*LOCATION = X + Y * WIDTH*



Slika 5.3.5.6: Vrijednosti piksela

```
size(200, 200);
loadPixels();
// Petlja kroz svaki stupac piksela
for (int x = 0; x < width; x++) {
// Petlja kroz svaki redak piksela
for (int y = 0; y < height; y++) {
// Koristite formulu da bi pronašli 1D lokaciju
int loc = x + y * width;
if (x % 2 == 0) { // Ako je stupac paran
pixels[loc] = color(255);
} else { // Ako je stupac neparan
pixels[loc] = color(0);
} } }
updatePixels();
```

U prethodnom odjeljku razmatrani su primjeri koji postavljaju vrijednosti piksela prema proizvoljnom izračunu. Postavljanje piksela prema onima koji se nalaze u postojećem PImage objektu radi se na sljedeći način. [14]

1)Umeće se slikovna datoteka u PImage objekt

2)Za svaki piksel u PImage, dohvaća se boja piksela i postavlja se piksel na tu boju.

Klasa PImage uključuje neka korisna polja koja pohranjuju podatke vezane uz sliku - širinu, visinu i piksele. Baš kao i kod naših korisnički definiranih klasa, može se pristupiti tim poljima preko sintakse točaka. [14]

```
PImage img = createImage(320,240,RGB); // Izradi PImage objekt
println(img.width); // 320
println(img.height); // 240
img.pixels[0] = color(255,0,0); // Postavlja prvi piksel slike u
crveno
```

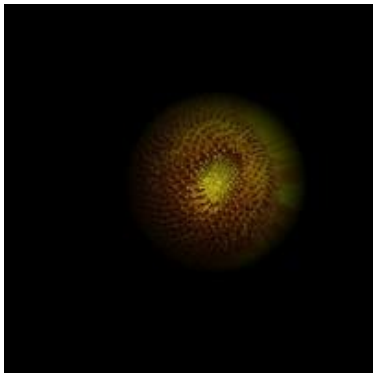
Pristup tim poljima omogućuje prolaz kroz sve piksele slike i prikazivanje istih na zaslonu.

```
PImage img;
void setup() {
size(200, 200);
img = loadImage("sunflower.jpg");
}
void draw() {
loadPixels();
img.loadPixels();
for (int y = 0; y < height; y++) {
for (int x = 0; x < width; x++) {
int loc = x + y*width;
// Funkcije red(), green(), i blue() izlaze 3 komponente boja iz
piksela.
float r = red(img.pixels[loc]);
float g = green(img.pixels[loc]);
float b = blue(img.pixels[loc]);
// Postavi piksel zaslona na piksel slike
pixels[loc] = color(r,g,b);
}
}
updatePixels();
}
```

Metoda piksel po piksel, omogućit će razvijanje prilagođenih algoritama za matematičko mijenjanje boja slike. Svjetlije boje imaju veće vrijednosti za crvene, zelene i plave komponente. Prirodno slijedi da se može promijeniti svjetlina slike povećanjem ili smanjivanjem komponenti boje svakog piksela. U sljedećem primjeru dinamički se povećavaju ili smanjuju te vrijednosti na temelju horizontalnog položaja miša. [14]

```
for (int x = 0; x < img.width; x++) {
for (int y = 0; y < img.height; y++ ) {
// Izračunaj lokaciju 1D piksela
int loc = x + y*img.width;
// Izvuci R,G,B vrijednosti iz slike
float r = red (img.pixels[loc]);
float g = green (img.pixels[loc]);
float b = blue (img.pixels[loc]);
// Promijeni svjetlinu prema položaju miša
float adjustBrightness = ((float) mouseX / width) * 8.0;
r *= adjustBrightness;
g *= adjustBrightness;
b *= adjustBrightness;
// Ograniči RGB između 0-255
r = constrain(r,0,255);
g = constrain(g,0,255);
b = constrain(b,0,255);
// Napravi novu boju i postavi piksel na zaslon
color c = color(r,g,b);
pixels[loc] = c;
}
}
```

Budući da se mijenja slika po osnovi piksela, sve piksele ne treba tretirati jednako. Na primjer, može se promijeniti svjetlina svakog piksela u skladu s njegovom udaljenosti od miša. [14]



Slika 5.3.5.7: Mijenjanje svjetline svakog piksela u skladu s udaljenosti od miša

```
for (int x = 0; x < img.width; x++) {
for (int y = 0; y < img.height; y++ ) {
// Izračunaj lokaciju 1D piksela
int loc = x + y*img.width;
// Izvuci R,G,B vrijednosti iz slike
float r = red (img.pixels[loc]);
float g = green (img.pixels[loc]);
float b = blue (img.pixels[loc]);
// Izračunajte iznos za promjenu svjetline
// na temelju blizine miša
float distance = dist(x,y,mouseX,mouseY);
float adjustBrightness = (50-distance)/50;
r *= adjustBrightness;
g *= adjustBrightness;
b *= adjustBrightness;
// Ograniči RGB između 0-255
r = constrain(r,0,255);
g = constrain(g,0,255);
b = constrain(b,0,255);
// Napravi novu boju i postavi piksel u prozoru
color c = color(r,g,b);
pixels[loc] = c;
}
}
```

Svi primjeri obrade slike čitaju svaki piksel iz izvorne slike i izravno pišu novi piksel. Međutim, često je prikladnije pisati nove piksele na odredišnu sliku (koja se zatim prikazuje pomoću funkcije image()). Još jedna jednostavna radnja piksela je threshold(prag). Threshold filtar prikazuje svaki piksel slike samo u jednom od dva stanja, crnom ili bijelom. To je stanje postavljeno prema određenoj vrijednosti praga. Ako je svjetlina piksela veća od praga, piksel se boja u bijelo, ako je manja, u crno. U donjem kodu koristi se proizvoljni prag od 100. [14]

Slika 5.3.5.8: Postavljanje slike u crno bijelo

```
PImage source; // Izvorna slika
PImage destination; // Odredišna slika
void setup() {
size(200, 200);
source = loadImage("sunflower.jpg");
// Odredišna slika je kreirana kao prazna slika iste veličine kao i
izvorna.
destination = createImage(source.width, source.height, RGB);
}
void draw() {
float threshold = 127;
// Gledamo piksele slika
source.loadPixels();
destination.loadPixels();
for (int x = 0; x < source.width; x++) {
for (int y = 0; y < source.height; y++ ) {
int loc = x + y*source.width;
if (brightness(source.pixels[loc]) > threshold) {
destination.pixels[loc] = color(255); // Bijelo
} else {
destination.pixels[loc] = color(0); // Crno
}
}
}
// Mijenjamo piksele u odredišnoj
destination.updatePixels();
// Prikaži odredišnu
image(destination,0,0);
}
```

U prethodnim primjerima prikazan je odnos jedan-na-jedan između izvornih piksela i odredišnih piksela. Da bi se povećala svjetlina slike, uzima se jedan piksel iz izvorne slike, poveća se RGB vrijednosti i prikazuje se jedan piksel u izlaznom prozoru. Da bi se izvršile naprednije funkcije obrade slike, potrebno je pomaknuti se izvan paradigme jedan-na-jedan u obradu pikselnih grupa. Počinje se stvaranjem novog piksela od dva piksela iz izvorne slike - piksela i njegovog susjeda na lijevoj strani. [14]

Ako se piksel nalazi na (x, y):

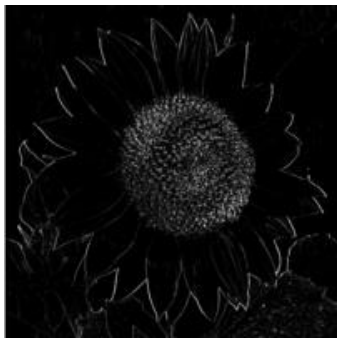
```
int loc = x + y*img.width;
color pix = img.pixels[loc];
```

Tada se njegov lijevi susjed nalazi na (x-1, y):

```
int leftLoc = (x-1) + y*img.width;
color leftPix = img.pixels[leftLoc];
```

Tada se može napraviti nova boja od razlike između piksela i njegovog susjeda na lijevoj strani. [14]

```
float diff = abs(brightness(pix) - brightness(leftPix));
pixels[loc] = color(diff);
```



Slika 5.3.5.9: Razlika između piksela i „lijevih“ susjeda

```
// Budući da gledamo lijeve "susjede",
// preskačemo prvi stupac
for (int x = 1; x < width; x++) {
for (int y = 0; y < height; y++ ) {
// Lokacija i boja piksela
int loc = x + y*img.width;
color pix = img.pixels[loc];
// Lokacija i boja lijevog piksela
int leftLoc = (x-1) + y*img.width;
color leftPix = img.pixels[leftLoc];
// Nova boja je razlika između piksela i
lijevog "susjeda"
float diff = abs(brightness(pix) -
brightness(leftPix));
pixels[loc] = color(diff);
}
}
```

Slijedi primjer koji izvodi konvoluciju koristeći 2D niz za spremanje težina piksela 3x3 matrice. Ovaj je primjer vjerojatno najnapredniji primjer spomenut do sada. [14]

Slika 5.3.5.10: Primjer koji izvodi konvoluciju koristeći 2D niz

```
PImage img;
int w = 80;
float[][] matrix = { { -1, -1, -1 },
                     { -1, 9, -1 },
                     { -1, -1, -1 } };
void setup() {
size(200, 200);
frameRate(30);
img = loadImage("sunflower.jpg");
}

void draw() {
// Prvo ćemo obraditi samo dio slike
// pogledajmo cijeli sliku kao pozadinu
image(img,0,0);
// mali pravokutnik kojeg ćemo obraditi
int xstart = constrain(mouseX-w/2,0,img.width);
int ystart = constrain(mouseY-w/2,0,img.height);
int xend = constrain(mouseX+w/2,0,img.width);
int yend = constrain(mouseY+w/2,0,img.height);
int matrixsize = 3;
loadPixels();
// počinje petlja za svaki piksel
for (int x = xstart; x < xend; x++) {
for (int y = ystart; y < yend; y++ ) {
// svaka lokacija piksela (x,y) prelazi u funkciju convolution()
// koja vraća novu vrijednost boje na zaslon.
color c = convolution(x,y,matrix,matrixsize,img);
int loc = x + y*img.width;
pixels[loc] = c;
}
}
updatePixels();

stroke(0);
```

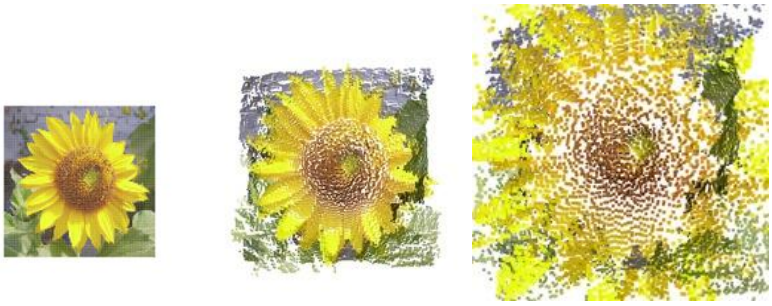
```
noFill();
rect(xstart,ystart,w,w);
}

color convolution(int x, int y, float[][] matrix, int matrixsize,
PImage img) {
float rtotal = 0.0;
float gtotal = 0.0;
float btotal = 0.0;
int offset = matrixsize / 2;
// petlja kroz matricu konvolucije
for (int i = 0; i < matrixsize; i++){
for (int j= 0; j < matrixsize; j++){
// piksel koji testiramo
int xloc = x+i-offset;
int yloc = y+j-offset;
int loc = xloc + img.width*yloc;
loc = constrain(loc,0,img.pixels.length-1);
// izračun konvolucije
// zbrajamo sve susjedne piksele pomnožene s vrijednostima iz
konvolucijske matrice
rtotal += (red(img.pixels[loc]) * matrix[i][j]);
gtotal += (green(img.pixels[loc]) * matrix[i][j]);
btotal += (blue(img.pixels[loc]) * matrix[i][j]);
}
}
// RGB mora biti unutar opsega
rtotal = constrain(rtotal,0,255);
gtotal = constrain(gtotal,0,255);
btotal = constrain(btotal,0,255);
// vratu dobivenu boju
return color(rtotal,gtotal,btotal);
}
```

Slijede dva primjera algoritama za crtanje oblika. Umjesto nasumičnog bojanja oblika ili tvrdih kodiranih vrijednosti kao što smo to činili u prošlosti, odabiremo boje od piksela unutar Pimage objekta. Sama slika se nikada ne prikazuje; radije služi kao baza podataka koju možemo iskoristiti za mnoštvo kreativnih potraga. U ovom prvom primjeru, za svaki ciklus kroz draw(), popunjavamo jednu elipsu na slučajnom mjestu na zaslonu s bojom preuzetom iz odgovarajućeg mjesta u izvornoj slici. Rezultat je osnovni "pointilistički nalik" efekt: [14]

Slika 5.3.5.11:Pointilistički efekt

```
PImage img;
int pointillize = 16;
void setup() {
size(200,200);
img =
loadImage("sunflower.jpg"
);
background(0);
smooth();
}
void draw() {
// odaberi nasumičnu točku
int x = int(random(img.width));
int y = int(random(img.height));
int loc = x + y*img.width;
// potraži RGB boju u izvornoj
slici
loadPixels();
float r = red(img.pixels[loc]);
float g = green(img.pixels[loc]);
float b = blue(img.pixels[loc]);
noStroke();// nacrtaj elipsu na
toj lokaciji te boje
fill(r,g,b,100);
ellipse(x,y,pointillize,pointilli
ze);
}
```

U sljedećem primjeru uzimaju se podaci iz dvodimenzionalne slike i koristeći tehnike 3D prevođenja, prikazuje se pravokutnik za svaki piksel u trodimenzionalnom prostoru. Položaj z određen je svjetlinom boje. Svjetlije boje izgledaju bliže gledatelju, a tamnije su dalje. [14]



Slika 5.3.5.12: Uzimanje podataka iz 2D slike i korištenje 3D tehnika prevođenja

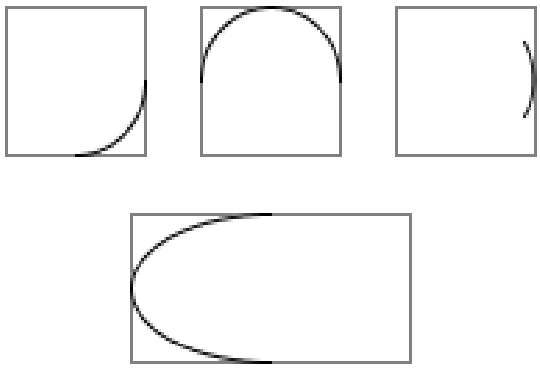
```
PImage img; // izvorna slika
int cellsize = 2; // dimenzije svakog polja na mreži
int cols, rows; // broj stupaca i redaka u našem sistemu
void setup() {
size(200, 200, P3D);
img = loadImage("sunflower.jpg"); // učitaj sliku
cols = width/cellsize; // izračunaj # stupaca
rows = height/cellsize; // izračunaj # redaka }
void draw() {
background(0);
loadPixels();
// započni petlju za stupce
for ( int i = 0; i < cols;i++) {
// započni petlju za redove
for ( int j = 0; j < rows;j++) {
int x = i*cellsize + cellsize/2; // x position
int y = j*cellsize + cellsize/2; // y position
int loc = x + y*width; // Pixel array location
color c = img.pixels[loc]; // Grab the color
// izračunaj z poziciju kao funkciju mouseX i svjetline piksela
float z = (mouseX/(float)width) * brightness(img.pixels[loc]) -
100.0;
// prevedi na lokaciju, dodaj ispunu i obrub, nacrtaj pravokutnik
pushMatrix();
translate(x,y,z);
fill(c);
noStroke();
rectMode(CENTER);
rect(0,0,cellsize,cellsize);
popMatrix();
} } }
```

5.3.6. Krivulje

Lukovi su najjednostavnije krivulje za crtanje. Processing definira luk kao dio elipse. Funkcija se poziva s ovim parametrima: [15]

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

Prva četiri parametra jednaka su onima za elipsu (); oni definiraju rubni okvir za luk. Zadnja dva parametra su početni i završni kut za luk. Ovi kutovi, kao i svi ostali kutovi u Processingu, dati su u radijanima. Kutovi se mjere u smjeru kazaljke na satu, s nula stupnjeva usmjereni prema istoku. Koristeći činjenicu da je PI radijan jednak 180 °, ovdje su neki primjeri lukova. [15]



```
void setup() {
size(300, 200);
background(255);
smooth();

rectMode(CENTER); // granični
okviri
stroke(128);
rect(35, 35, 50, 50);
rect(105, 35, 50, 50);
rect(175, 35, 50, 50);
rect(105, 105, 100, 50);
```

Slika 5.3.6.1: Krivulje

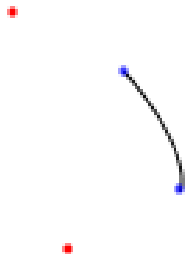
```
stroke(0);
arc(35, 35, 50, 50, 0, PI / 2.0); // donja
četvrtina kruga
arc(105, 35, 50, 50, -PI, 0); // gornja četvrtina
kruga
arc(175, 35, 50, 50, -PI / 6, PI / 6); // 60
stupnjeva
arc(105, 105, 100, 50, PI / 2, 3 * PI / 2); // 180
stupnjeva
}
```

Sljedeća funkcija, curve(), omogućuje crtanje krivulja koje nisu nužno dio luka. Ova funkcija crpi ono što se tehnički naziva Rom-Catmull Spline. Za crtanje krivulje potrebno je odrediti (x, y) koordinate točaka u kojima počinje i završava krivulja. Također je potrebno navesti dvije kontrolne točke koje određuju smjer i količinu zakrivljenosti. Poziv na curve() koristi ove parametre: [15]

```
curve(cpx1, cpy1, x1, y1, x2, y2, cpx2, cpy2);
```

cpx1, cpy1 *Koordinate prve kontrolne točke*
x1, y1 *Koordinate početne točke krivulje*
x2, y2 *Koordinate završne točke krivulje*
cpx2, cpy2 *Koordinate druge kontrolne točke*

Ovdje je primjer koji prikazuje krivulju (). Kontrolne točke prikazane su crvenom bojom, a krivulja je plava. [15]

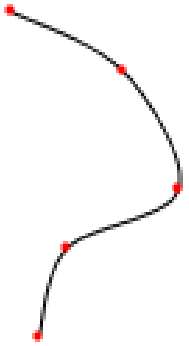


Slika 5.3.6.2: Prikaz krivulje

```
void setup() {
size(200, 200);
background(255);
smooth();
stroke(0);
curve(40, 40, 80, 60, 100,
100, 60, 120);
noStroke();
fill(255, 0, 0);
ellipse(40, 40, 3, 3);
fill(0, 0, 255, 192);
ellipse(100, 100, 3, 3);
ellipse(80, 60, 3, 3);
fill(255, 0, 0);
ellipse(60, 120, 3, 3);
}
```

Tangenta na krivulju na početnoj točki paralelna je s linijom između kontrolne točke jedan i kraja krivulje. Tangenta na krivulju na krajnjoj točki je paralelna s linijom između početne točke i kontrolne točke 2. [15]

Za crtanje kontinuirane krivulje kroz nekoliko točaka bolje je koristiti funkciju curveVertex (). Ova funkcija može se koristiti samo kod stvaranja oblika s funkcijama beginShape () i endShape (). Ovdje je krivulja koja povezuje točke (40, 40), (80, 60), (100, 100), (60, 120) i (50, 150). U uobičajenoj upotrebi, ljudi koriste prvu točku krivulje kao prvu kontrolnu točku, a posljednju točku krivulje kao zadnju kontrolnu točku. [15]



Slika 5.3.6.3: Krivulja s nekoliko točaka

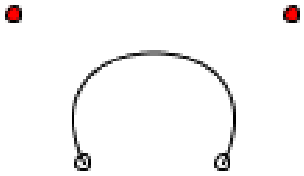
```
int[] coords = {
40, 40, 80, 60, 100,
100, 60, 120, 50, 150
};
void setup() {
size(200, 200);
background(255);
smooth();
noFill();
stroke(0);
beginShape();
curveVertex(40,   40);   //   prva
kontrolna točka
curveVertex(40, 40); // je također
početna točka krivulje
curveVertex(80, 60);
curveVertex(100, 100);
curveVertex(60, 120);
curveVertex(50,  150);  //  zadnja
točka krivulje
curveVertex(50, 150); // je također
zadnja kontrolna točka
endShape();
fill(255, 0, 0);
noStroke();
for (int i = 0; i < coords.length;
i += 2) {
ellipse(coords[i],  coords[i  +  1],
3, 3);
} }
```

Bezierove krivulje - funkcija bezier () ima osam parametara, ali redoslijed je različit: [15]

bezier(x1, y1, cpx1, cpy1, cpx2, cpy2, x2, y2);

x1, y1	*Koordinate početne točke krivulje*
cpx1, cpy1	*Koordinate prve kontrolne točke*
cpx2, cpy2	*Koordinate druge kontrolne točke*
x2, y2	*Koordinate završne točke krivulje*

Prikaz Bézierove krivulje i njezinih kontrolnih točaka. [15]

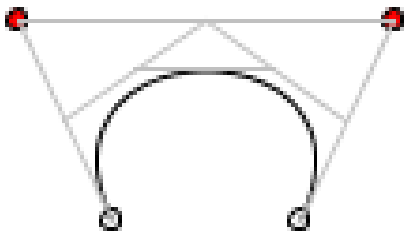


Slika 5.3.6.4: Bezierova krivulja

```
void setup() {
size(150, 150);
background(255);
ellipse(50, 75, 5, 5); //
završne točke
ellipse(100, 75, 5, 5);
fill(255, 0, 0);
ellipse(25, 25, 5, 5); //
kontrolne točke
ellipse(125, 25, 5, 5);
noFill();
stroke(0);
bezier(50, 75, 25, 25, 125,
25, 100, 75);
}
```

Iako je teško zamisliti kako kontrolne točke utječu na krivulju (), malo je lakše vidjeti kako kontrolne točke utječu na Bézier-ove krivulje. Zamišljena su dva stupa i nekoliko gumenih traka. Polovi povezuju kontrolne točke s krajnjim točkama krivulje. Gumena vrpca povezuje vrhove stupova. Još dvije gumene vrpce povezuju središnje točke polova s središtem prve gumene trake. Još jedan gumeni pojas povezuje njihove sredine. Središte te posljednje gumene trake je vezano za krivulju. [15]

Ovaj dijagram pomaže objasniti: [15]



Slika 5.3.6.5: Dijagram Bezierove krivulje

Baš kao što curveVertex() omogućuje stvaranje kontinuiranih krivulja, bezierVertex() omogućuje stvaranje kontinuiranih krivulja Béziera. Mora se biti unutar beginShape() / endShape() slijeda. Mora se koristiti vrh (startX, startY) za određivanje početne točke krivulje.

Sljedeće točke se navode s pozivom na: [15]

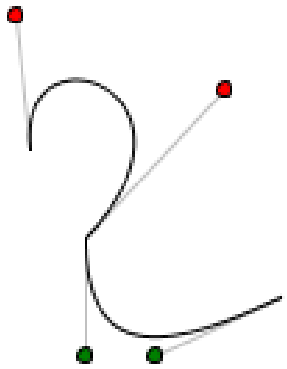
bezierVertex(cpx1, cpy1, cpx2, cpy2, x, y);

cpx1, cpy1	*Koordinate prve kontrolne točke*
cpx2, cpy2	*Koordinate druge kontrolne točke*
x, y	*Sljedeća točka na krivulji*

Dakle, da bi se nacrtao prethodni primjer koristeći bezierVertex (), potrebno je to učiniti ovako: [15]

```
void setup() {
size(150, 150);
background(255);
smooth();
noFill();
stroke(0);
beginShape();
vertex(50, 75); // prva točka
bezierVertex(25, 25, 125, 25, 100, 75);
endShape();
}
```

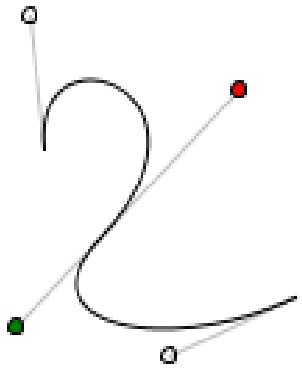
Ovdje je kontinuirana Bézierova krivulja, ali se ne uklapa glatko. Dijagram pokazuje kontrolne točke, ali ovdje je samo relevantan kod za crtanje krivulje. [15]



```
size(200, 200);
background(255);
noFill();
beginShape();
vertex(30,  70);  //
prva točka
bezierVertex(25,  25,
100, 50, 50, 100);
bezierVertex(50, 140,
75, 140, 120, 120);
endShape();
```

Slika 5.3.6.6: Kontinuirana Bezierova krivulja

Da bi dvije krivulje A i B bile glatko kontinuirane, zadnja kontrolna točka A, zadnja točka A i prva kontrolna točka B moraju biti na pravoj liniji. Evo primjera koji ispunjava te uvjete. [15]



```
size(200, 200);
background(255);
beginShape();
vertex(30,  70);  //
prva točka
bezierVertex(25,  25,
100, 50, 50, 100);
bezierVertex(20, 130,
75, 140, 120, 120);
endShape();
```

Slika 5.3.6.7: Glatko kontinuirane krivulje A i B

6. Zaključak

Processing promovira softversku pismenost, osobito u području vizualnih umjetnosti i vizualne pismenosti unutar tehnologije. Usmjeren je prema stvaranju vizualnih, interaktivnih medija, tako da prvi programi započinju nekakvom vrstom crteža. Pristup Processingu također je primijenjen na elektroniku kroz Arduino i Wiring projekte.

U ovom radu opisana je izrada web stranice putem HTML i CSS jezika, a služi za upoznavanje s Processingom, vođena je tutorijalima i primjerima koji su objašnjeni na samoj stranici, ali i u ovome radu. Napredovanjem tehnologije, ljudi teže za nečim zanimljivijim i interaktivnijim, a Processing nudi baš to. Od jednostavnijeg pa do kompliciranijeg programskog koda, čovjek sam može stvoriti svoju interaktivnu skicu koja može služiti u vlastite svrhe kao vježbanje programiranja, za poboljšanje web stranica na internetu ili pak kao nekakva projekcija u stvarnome svijetu.

U Varaždinu, _____

_____

7. Literatura

[1] https://processing.org/ (15.07.2019.)

[2] https://marksheet.io/html-syntax.html (15.07.2019.)

[3] https://www.w3schools.com/html/html5_canvas.asp (15.07.2019.)

[4] https://www.theserverside.com/definition/cascading-style-sheet-CSS (15.07.2019.)

[5] https://processing.org/overview/ (15.07.2019.)

[6] Learning Processing, A Beginner's Guide to Programming Images, Animation and Interaction - Daniel Shiffman

[7] https://stackoverflow.com/questions/2918289/what-is-the-processing-programming-language-used-for (17.07.2019.)

[8] https://processing.org/tutorials/gettingstarted/ (17.07.2019.)

[9] https://processing.org/tutorials/drawing/ (17.07.2019.)

[10] https://processing.org/tutorials/color/ (18.07.2019.)

[11] Processing, A Programming Hanbook for Visual Designers and Artists - Casey Reas, Ben Fry

[12] https://processing.org/tutorials/interactivity/ (26.07.2019.)

[13] https://processing.org/tutorials/typography/ (29.07.2019.)

[14] https://processing.org/tutorials/pixels/ (01.08.2019.)

[15] https://processing.org/tutorials/curves/ (01.08.2019.)

Popis slika

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Veronika Babok *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Izrada web stranice za Processing *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Veronika Babok
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Veronika Babok *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Izrada web stranice za Processing *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Veronika Babok
(vlastoručni potpis)