Univerza
v Ljubljani
Fakulteta
za gradbeništvo
in geodezijo

Jamova 2
1000 Ljubljana, Slovenija
telefon (01) 47 68 500
faks (01) 42 50 681
fgg@fgg.uni-lj.si

Univerzitetni program Geodezija,
smer Geodezija

Kandidat:
Jure Kop

Spletno orodje za učenje kot uporabni GIS

Diplomska naloga št.: 719

Mentor:
izr. prof. dr. Radoš Šumrada

Ljubljana, 28. 9. 2007

IZJAVA O AVTORSTVU

Podpisani JURE KOP izjavljam, da sem avtor diplomske naloge z naslovom »SPLETNO ORODJE ZA UČENJE KOT UPORABNI GIS«.

Izjavljam, da prenašam vse materialne avtorske pravice v zvezi z diplomsko nalogo na UL, Fakulteto za gradbeništvo in geodezijo.

Ljubljana, 19. 9. 2007 Jure Kop

IZJAVE O PREGLEDU NALOGE

BIBLIOGRAFSKO–DOKUMENTACIJSKA STRAN IN IZVLEČEK

UDK:	**004.6:004.738.5:659.2:91(043.2)**
Avtor:	**Jure Kop**
Mentor	**izr. prof. dr. Radoš Šumrada**
Naslov:	**Spletno orodje za učenje kot uporabni GIS**
Obseg in oprema:	**42 str., 29 sl., 9 pril.**
Ključne besede:	**GIS, algoritem, wiki**

Izvleček:

Z vedno večjim kopičenjem podatkov se vse bolj soočamo s problemom, kje te podatke shraniti, da bodo dostopni širšemu krogu ljudi. V začetku informacijske družbe je imela vsaka informacija svojo ceno. Želja kopice ljudi po brezplačnem znanju je pripeljala do tega, da lahko vsakdo aktivno sodeluje pri ustvarjanju podatkov, ki so lahko še vedno avtorsko zaščiteni, vendar prosto in brezplačno dostopni komur koli.

Vsako novejše področje, ki ima za sabo vsaj nekaj desetletij razvoja, kot na primer genska tehnologija ali informacijski sistemi, potrebuje dobro organiziran sistem shranjevanja in prikazovanja podatkov. Vedno več ljudi se ukvarja s čedalje zahtevnejšimi področji, kar posledično pripelje do tega, da je na trgu vedno več podatkov, ki pa so lahko težko dostopni, plačljivi ali pretirano zaščiteni ter zato manj uporabni.

V diplomski nalogi je predstavljeno okolje, ki omogoča ponujanje podatkov na svetovnem spletu, hkrati pa jih lahko udeleženci sami dodajajo, spreminjajo in komentirajo. Za predstavitev uporabnosti okolja so dodani ter razloženi tudi najpogostejši vektorski algoritmi, ki se uporabljajo pri geografskih informacijskih sistemih.

BIBLIOGRAPHIC–DOCUMENTALISTIC INFORMATION

UDC:	**004.6:004.738.5:659.2:91(043.2)**
Author:	**Jure Kop**
Supervisor:	**Assoc. Prof. dr. Radoš Šumrada**
Title:	**Web based help tools for practical GIS**
Notes:	**42 pag., 29 fig., 9 add.**
Key words:	**GIS, algorithms, wiki**

Abstract:

As more and more data is being accumulated over the internet, the problem occurs of how to save this data so that it would be accessible to as many people as possible. At the beginning of the information age all data had its price, but the enthusiasm of several individuals has lead us to a point where everyone can make and publish data over the internet. This data may still be copyrighted but is freely available to anyone.

Every new field of research with at least several decades of existance, i.e. gene technology or information systems, needs a well organized system of saving and publishing the data. More and more people are involved in research, which leads to large amounts of data generated, which can be difficult to access or may come with a high price tag or restrictive copyright protection.

This Master Thesis shows an environment for publishing the data over the internet where users are able to add, modify or comment certain articles. To show the usability of the environment most common vector algorithms used in Geographical Information Systems are added and explained.

SEZNAM KRATIC

CASE – Computer Assisted Software Engineering; računalniško podprto inženirstvo programja

CSS – Cascading Style Sheets; predloge ki določajo izgled spletnih strani

GIS – Geografski informacijski sistem

GNU – GNU is not Unix; operacijski sistem, ki temelji na odprti kodi

GPL – General Public Licence; splošno javno dovoljenje

HTML – Hypertext Markup Language; jezik za označevanje nadbesedila

JSP – Java Server Pages; tehnologija na osnovi jave, ki omogoča dinamično generiranje HTML ter XML dokumentov

LGPL – Lesser General Public Licence; manj obsežni GPL

PHP – PHP: Hypertext Preprocessor; predprocesor nadbesedila

RAD – Rapid Application Development; hiter razvoj aplikacij

RSS – Rich Site Summary; zgoščeni povzetek strani

SQL - Structured Query Language; sestavljeni jezik za poizvedbe

XML – Extensible Markup Language; razširljivi označevalni jezik

KAZALO VSEBINE

KAZALO SLIK

KAZALO PRILOG

1 UVOD

Študentje in tudi drugi uporabniki geografskih informacijskih sistemov (v nadaljevanju GIS) se večkrat soočamo s problemom iskanja vsebin, povezanih z GIS. Na spletu so vsebine razpršene in le redke so strani, ki vsebujejo koristne informacije, prikazane študentu prijazno. Še manj je strani, kjer bi bili študentje oziroma drugi uporabniki udeleženi pri oblikovanju vsebin. Cilj te naloge je vzpostaviti programsko okolje, ki bi bilo uporabnikom prijazno, hkrati pa bi omogočalo njihovo sooblikovanje vsebin.

GIS so računalniško podprti informacijski sistemi, ki jih uporabljamo za zajem, modeliranje, shranjevanje, analiziranje, manipuliranje ter predstavljanje geografskih podatkov (Burrough et al., 1998). Geografski informacijski sistemi imajo za sabo pol stoletja razvoja in jih lahko prištevamo k bolje razvitim informacijskim sistemom. Z razvojem so GIS postali vse bolj obsežni ter dodelani, njihova uporabnost pa praktično ne pozna meja. Ena temeljnih vsebin GIS so GIS-algoritmi.

GIS-algoritme lahko v grobem delimo na rastrske, vektorske, mrežne ter topološke. Kot povedo že imena sama, se rastrski algoritmi uporabljajo za manipuliranje z rastrskimi vsebinami, npr. senčenje (ang. hill shading), pri čemer je osnovna enota slikovna pika. Vektorski se uporabljajo za vektorske vsebine, ki temeljijo na koordinatah, npr. razdalja od točke do premice. Mrežni algoritmi so prav tako pomembni v GIS, recimo tam, kjer so pomembne razdalje med vozlišči. Eno takšnih okolij je transport. Znan algoritem, ki izračuna najkrajšo pot med vozlišči, je Dijsktrin algoritem.

Orodja, v katerih lahko uporabniki dodajajo vsebine na spletnih portalih, že obstajajo. Eno takšnih orodij je wiki. To orodje med drugim omogoča sodelovanje javnosti, dodajanje komentarjev in drugo. Zadnjih nekaj let so se razvila še druga wiki orodja, ki temeljijo na različnih konceptih in filozofijah – tako obstajajo prosto dostopna, plačljiva, naročniška orodja, kar se tiče cenovne politike. Glede na jedro, t.j. programski jezik, s katerim so bila razvita, se delijo na PHP, Java, C, C++, ASP in JSP wiki orodja, če naštejemo samo nekatere. Tudi glede podpore se zelo razlikujejo med sabo. Nekatera podpirajo druge programske jezike

s pomočjo dodatkov (ang. plug-in), druge pa so okleščene vsega in podpirajo samo osnovne funkcije.

Če naštejemo nekaj najbolj znanih wiki okolij, ne moremo mimo Wikipedije, ki ponuja več kot dva milijona člankov v različnih jezikih. Wikipedijo poganja orodje mediawiki, napisano v programskem jeziku PHP. Poleg običajne enciklopedije ponuja Wikipedija še wiki slovar, wiki knjižnico, wiki univerzo ter drugo.

Pričujoče delo je razdeljeno na dva sklopa. Prvi, programski del, je opis konceptov vzpostavitve programskega okolja. Ker se pri takšnih delih srečujemo z vrsto vprašanj, poskuša ta del odgovoriti na nekatera ključna. Pri tem ni toliko poudarka na sami programski kodi, temveč se osredotočimo na zasnovo in uporabnost okolja.

Drugi, vsebinski del, je dodajanje vsebin v to programsko okolje. Vsebine, ki bi najbolje izkoristile programsko okolje, v smislu uporabnosti okolja samega, so vektorski algoritmi, zato so opisani v večjem številu. Ker pa to niso edine aplikacije v GIS, je delno opisano poglavje o rastrskih algoritmih s primeri.

Sledi še uporabnost okolja pri drugih, nealgoritemskih delih ter priloga, ki zajema podatke, dodane v okolje. Te podatke smo razvili v programskem jeziku java, za katere potrebujemo ustrezno programsko opremo, če jih želimo uporabiti.

2 PROGRAMSKO OKOLJE

S pojmom programsko okolje ne pojmujemo samo programa v strogem pogledu zbirke navodil za računalnik v binarni kodi, ampak celotno okolje, ki lahko vsebuje več programov in je lahko izveden v več instancah hkrati, se pravi, da ga lahko uporablja več uporabnikov, kar je predvsem značilnost spletnih aplikacij. V programsko okolje štejemo poleg osnovne kode še mnogo drugih pomožnih programov. Tako imamo strežnik, ki skrbi za izvajanje kode in posredovanje podatkov uporabnikom, na uporabniški strani imamo odjemalca, kot npr. Internet Explorer ali Mozilla Firefox, ki mora imeti ustrezno dodatno opremo, tako da lahko uporabnik nemoteno dela.

Ker so v našem primeru algoritmi napisani v programskem jeziku java, mora uporabnik za pravilno delovanje namestiti še javino navidezno okolje (ang. Java Virtual Machine). V nasprotnem primeru bo uporabnik videl samo prazna okna znotraj svojega brskalnika.

Programsko okolje, ki deluje preko spleta, imenujemo spletno okolje oz. spletni servis, v kolikor ponuja različne informacije in orodja.

2.1 Razvoj programskih okolij

Koncept razvoja programskega okolja se ne razlikuje od razvoja programske opreme oz. informacijskih sistemov, čeprav obstajajo razlike pri sami implementaciji.

Med pomembnejše koncepte razvoje spadajo (Šumrada 2005a):

- zaporedni oz. kaskadni razvoj,
- postopni razvoj,
- ciklični razvoj,
- hiter razvoj prototipov ter
- objektno usmerjeni razvoj.

Vsak model razvoja programske opreme, od kompleksnejših do preprostejših programov, načeloma vsebuje naslednje korake:

- **analiza** pomeni definiranje namena in lastnosti okolja,
- **načrtovanje** je nadgradnja analize ter pomeni razširitev določitev v postopku analize,
- **razvoj** okolja,
- **izvedba** je sestavljena iz zagona in testiranja okolja ter
- **vzdrževanje,** ki pomeni podporo sistemu ter njegov nadaljnji razvoj.

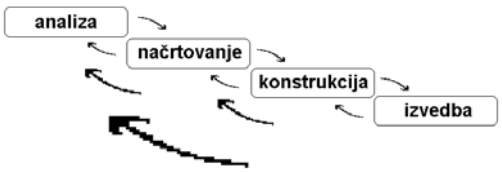
2.1.1. Zaporedni pristop

Zaporedni pristop, včasih poimenovan tudi kaskadni (ang. Waterfall), spada med najstarejše pristope. Glavna značilnost tega pristopa je dokončanje posamezne faze pred nadaljevanjem na naslednjo fazo. Ta proces bi lahko v grobem primerjali z gradnjo hiše. Ko so enkrat dokončani, se temeljev (načeloma) ne spreminja več. Enako velja za zidove, streho. Glavna prednost tega pristopa je, da imamo lahko zaposlene različne strokovnjake na različnih fazah. Tako so na primer pri analizah zaposleni ekonomisti, če gre za tržno aplikacijo, pri načrtovanju imamo lahko zaposlene strokovnjake s posameznega področja, npr. geodete ter pri razvoju programerje. Tako kot pri aplikacijah je tudi pri gradnji hiše značilno, da se krovci ne ubadajo s temelji, ampak samo s streho in podpornimi tramovi.

Slika 1: Zaporedni razvojni pristop

2.1.2 Postopni razvojni pristop

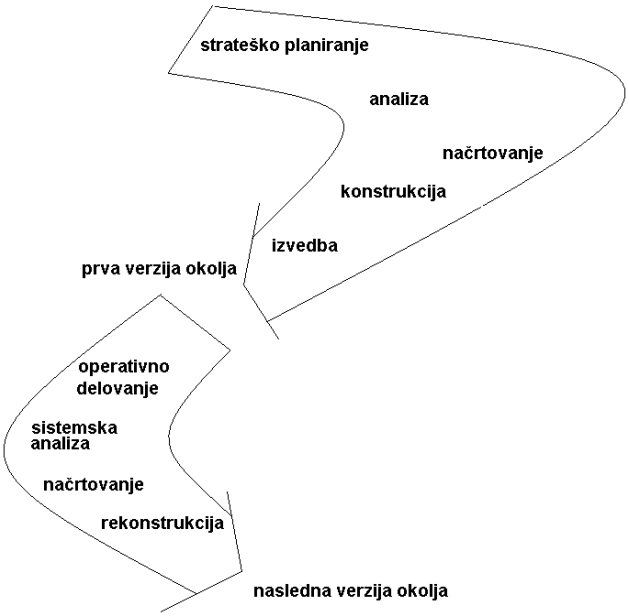
Pristop, pri katerem se vse faze začnejo istočasno ter se razvijajo delno ali povsem vzporedno, se imenuje ciklični, krožni. Glavna prednost tega pristopa je prekrivanje faz, s tem sprotno popravljanje ter izboljševanje aplikacije že med razvojem. Za razliko od kaskadnega pristopa se tukaj posamezne faze ne končajo v celoti, preden se začne naslednja.



Slika 2: ***Postopni razvojni pristop***

2.1.3 Ciklični pristop

Z velikostjo postajajo programska okolja vse bolj zapletena. Prednosti, ki jih takšen pristop omogoča, so predvsem povratne informacije uporabnikov, ki tako sodelujejo pri razvoju (lahko skozi beta testiranje ali kako drugače). Prednost je tudi ta, da se lahko razvijalci kot tudi uporabniki učijo hkrati z razvojnim procesom. Veliko napak pri razvoju se lahko odkrije sproti, vendar pa to še ne pomeni, da je na koncu razvoja programsko okolje brezhibno.



Slika 3: Krožni razvojni pristop

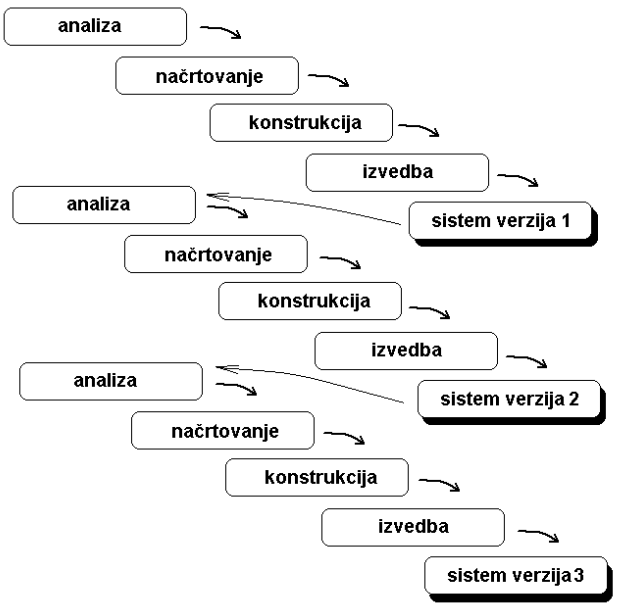
2.1.4 Hiter razvoj aplikacij

Hiter razvoj aplikacij (Rapid Application Development - RAD) je konec osemdesetih let 20. stoletja razvil James Martin. Metode vključujejo iterativni pristop, razvoj prototipov ter uporabo orodij CASE (Computer Aided Software Engineering).

RAD vključuje uporabnike že pri samem začetku razvoja oz. takoj, ko je to mogoče. Ti uporabniki nato preizkušajo določene modele oz. prototipe, ki jih razvijalci razvijajo tekom razvoja.

Poznamo dva glavna razvojna pristopa pri hitrem razvoju aplikacij (Šumrada, 2005a). To sta fazni razvoj verzij ter hiter razvoj prototipov.

Fazni razvoj verzij se veliko uporablja pri razvoju programske opreme. Verzije sistemov oz. okolij so velikokrat oštevilčene tako, da lahko uporabniki že po številki vidijo, ali je okolje stabilno, t.j. namenjeno končnim uporabnikom, ali pa je še v fazi razvoja.



Slika 4: Hiter razvoj aplikacij

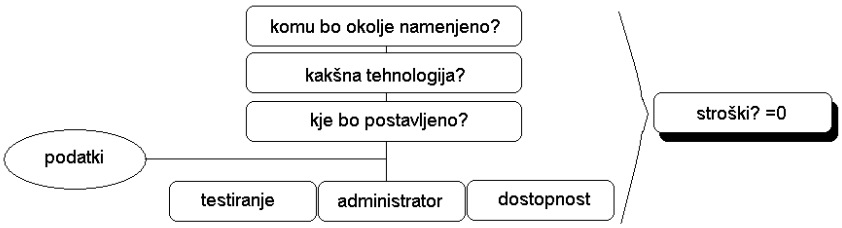
Lep primer takšnega razvoja je GNU/Linux, kjer je njegovo jedro oz. kernel oštevilčen z dvema poddeliklama. Na primer, kernel 1.2.34 pomeni 34. razvojno fazo 2. (sodega = nestabilnega) jedra. Za vsako večjo spremembo se spremeni tudi osnovna različica, torej prva številka.

2.2 Načrtovanje našega spletnega okolja

Vedno imamo možnost, da se lotimo razvoja okolja od samega začetka, vendar je smotrno pogledati, če morda obstaja možnost, da je pod ugodno licenco že razvit del ali morda celotno okolje, ki ga moramo nato samo prilagoditi svojim potrebam.

Ugodna licenca za nas je GPL (General Public Licence), ki na kratko pomeni to, da lahko določeno delo uporabimo, spremenimo ali mu dodamo svojo kodo, pri tem pa moramo navesti izvirnega avtorja ter to našo kodo omogočiti drugim uporabnikom pod enakimi pogoji. Veliko izdelkov pod to licenco je brezplačno dosegljivih.

Seveda velja omeniti, da je bila za razvoj našega programskega okolja na voljo samo ena oseba, ki je morala skrbeti tako za načrtovanje, finance, programiranje ter ostale stvari, kar je privedlo do tega, da je to okolje veliko bolj preprosto, kot bi bilo, če bi na njem delalo več oseb.



Slika 5: Načrtovanje našega programskega okolja

Zaradi enostavnosti in varčevanja, tako s časom in denarjem, smo se odločili, da postavimo okolje s pomočjo wiki orodja.

2.2.1 Wiki okolje

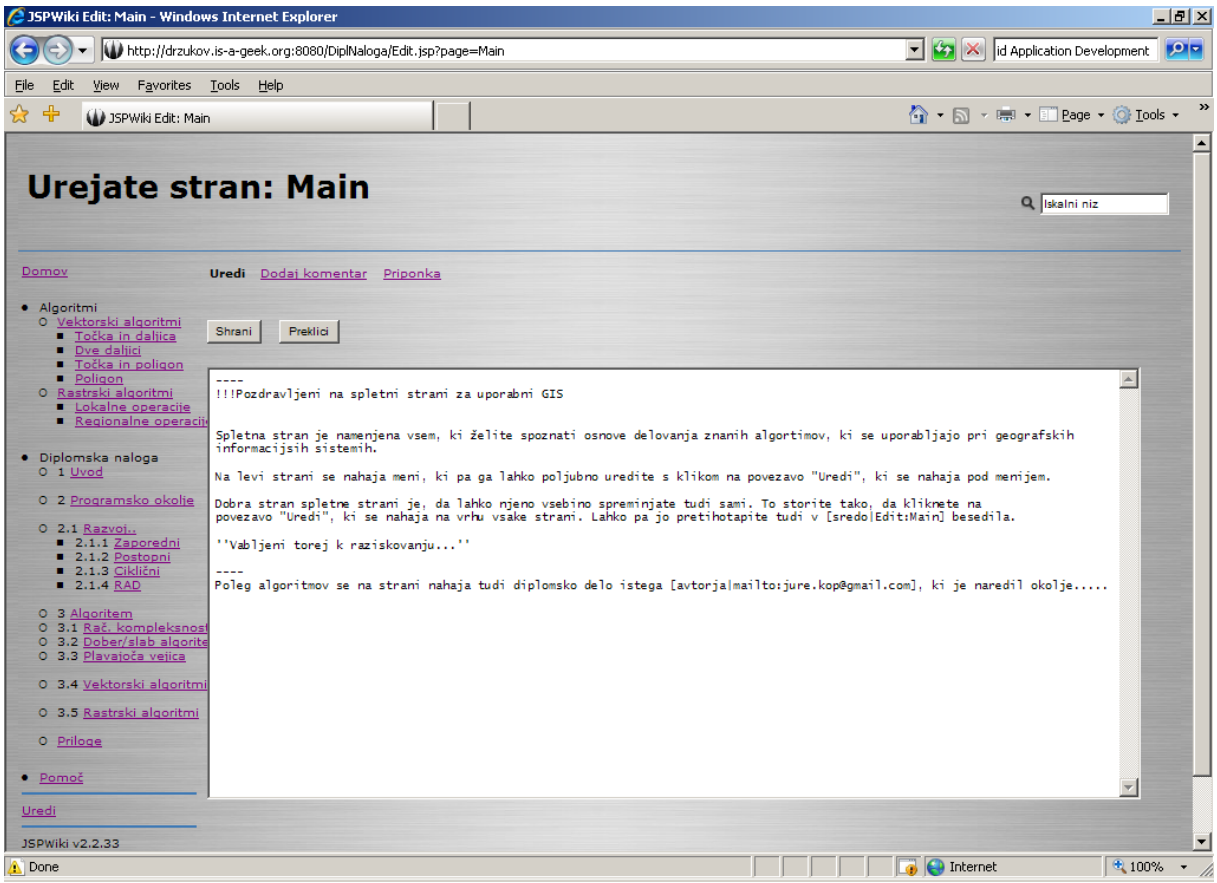
Wiki je spletno orodje, ki je zasnovano tako, da lahko obiskovalci strani dodajajo, spreminjajo ter ustvarjajo nove vsebine. Obstaja več vrst wiki orodij, ki se razlikujejo predvsem glede na namen, ceno ter podporo, ki jo nudijo.

Prvo wiki orodje je razvil Ward Cunningham leta 1994 ter ga namestil na strežnik c2.com. Ime wiki je okrajšava za wiki-wiki, ime avtobusa, ki vozi med terminali na enem od havajskih letališč. Beseda wiki pa v havajščini pomeni hitro.

Pri načrtovanju wiki okolja smo morali najprej opraviti temeljito raziskavo trga ponudnikov wiki storitev. Ker obstajajo wiki okolja tudi v brezplačni obliki, smo se zaradi varčevanja odločili za takšno obliko.

Slika 6: Uvodna stran spletnega okolja

Običajno je wiki stran sestavljena iz menija, ki je praviloma na levi strani ter glavne strani, kjer se izpisujejo podatki. Dodatno sta tu še naslovni del, kjer se prikazujejo naslovi trenutne strani in del z raznimi gumbi (kot so uredi, priponke, iskanje ipd.).



Slika 7: Stran za urejanje začetne strani

2.2.2 JSPWiki

JSPWiki je wiki okolje, ki temelji na programskih jezikih java ter JSP. Dostopno je na spletu in je izdano pod licenco LGPL. Licenca LGPL je manj obsežna različica licence GPL. Razlika med njima je naslednja: če se v programu uporabi del kode, ki je licenciran z licenco GPL, mora biti tudi končni izdelek v celoti izdan pod to isto licenco. Če pa se v programu uporabi del kode, ki je licenciran z licenco LGPL, pa to ni zahtevano.

Orodje JSPWiki je razvil Janne Jalkanen. Orodje ponuja poleg običajnih še mnoge druge funkcije ter dodatke. Če naštejemo samo nekaj najpomembnejših, so to:

- internacionalizacija (uporaba UTF-8 nabora znakov),
- uporaba javinih dodatnih programov (plug-inov),
- šablone (templates),
- integracija CSS neposredno v wiki jezik,
- iskanje strani ter mnogo drugih.

JSPWiki uporablja veliko posameznikov, univerz in podjetij, med drugim tudi Sun Microsystems, ki je okolje integriralo v svoj strežniški portal.

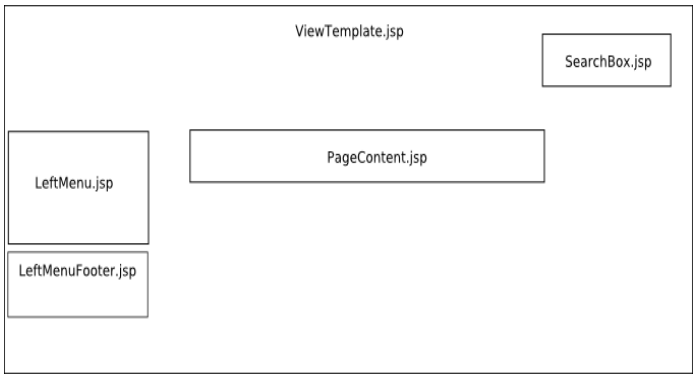
Seznam pomembnejših datotek orodja JSPWiki.

Glavne datoteke	Datoteke, ki skrbijo za izgled strani
Wiki.jsp	PageContent.jsp
Edit.jsp	DiffContent.jsp
Search.jsp	InfoContent.jsp
Comment.jsp	PreviewContent.jsp
PageInfo.jsp	ConflictContent.jsp
	FindContent.jsp
	PreferencesContent.jsp
	DisplayMessage.jsp
	EditContent.jsp
	CommentContent.jsp

Velja omeniti, da je zgornji seznam nepopoln, saj je datotek še veliko, dosti pa jih dobimo tudi s šablonami ter dodatki.

Glavne datoteke so jedro programa in so odgovorne za to, kako se bodo procesi izvajali, npr. iskanje, urejevanje ipd. Ostale datoteke pa služijo temu, da je vse na svojem mestu in da je oblika strani takšna, kot smo si jo zamislili. Obstajajo še CSS datoteke, s katerimi spreminjamo privzete odmike in barve besedila ter okvirjev, ki se pojavijo na naši spletni strani.

V grobem izgleda naše okolje nekako tako, kot na sliki 8. Seveda lahko z uporabo šablon ali malce programiranja izgled strani spremenimo.



Slika 8: Razporeditev datotek po zaslonu

2.2.2.1 Wiki jezik za oblikovanje

Na kratko je predstavljen jezik za urejevanje v okolju wiki. Omeniti velja, da se vsako wiki orodje razlikuje tudi po jeziku, v katerem uporabniki urejajo strani. Podrobnejši pregled jezika za urejevanje se nahaja v prilogi (priloga C).

Verjetno najpomembnejša uporabnost orodja je vnos besedila. Besedilo preprosto napišemo, kot bi pisali v katerem koli drugem urejevalniku besedil. Pri oblikovanju besedila pa moramo upoštevati naslednja pravila:

----	**vodoravna črta**
\\	prelom vrste
\\\	prelom vrste z dodatnim preskokom vrstic
[povezava]	ustvari povezavo na interno wiki stran imenovano 'Povezava'
[to je drevo]	ustvari povezavo na interno wiki stran imenovano 'ToJeDrevo'
[klikni|povezava]	ustvari povezavo na interno wiki stran imenovano 'Povezava', vendar izpiše na zaslon besedilo 'klikni' namesto 'povezava'
[1]	ustvari referenco na opombo imenovano 1
[#1]	označi opombo imenovano 1
[[oglato]	izpiše besedilo '[oglato]'
!velikost	manjši naslov z besedilom 'velikost'
!!velikost	srednje velik naslov z besedilom 'velikost'
!!!velikost	velik naslov z besedilom 'velikost'
''poševno''	izpiše besedilo 'poševno' poševno
__krepko__	izpiše besedilo 'krepko' krepko
{{isto}}	izpiše besedilo 'isto', pri čemer so vse črke enako široke
*** oznaka**	ustvari označeno besedilo 'oznaka'
# numeral	ustvari oštevilčeno besedilo 'numeral'
;definicija:razlaga	ustvari definicijo imenovano 'definicija' ter razlago imenovano 'razlaga', ki je eno vrstico nižje ter premaknjena v desno

Pri načrtovanju tega programskega okolja je bilo potrebno poleg ostalih razvojnih korakov upoštevati naslednje pogoje:

- kdo je odgovoren za vzdrževanje vsebine,
- kakšne prioritete imajo uporabniki ter
- kakšna bo zaščita okolja tako s strani uporabnika kot tudi ponudnika aplikacije.

Osnovno vodilo pri vzpostavitvi okolja je bilo, da ima uporabnik možnost dodajanja ter spreminjanja vsebin. Na ta račun se je treba odpovedati določenim varnostnim elementom, saj vsak dostop do strežnika pomeni določeno grožnjo.

Nadaljnji korak je bil, da smo pretehtali možnosti, kje bo naše wiki okolje nameščeno. Na medmrežju obstajajo ponudniki gostitelji, ki nam za določeno ceno ali pa tudi zastonj omogočijo postavitev okolja. Velja omeniti, da smo pri takšni izbiri dokaj omejeni, vendar pa je dobra stran to, da lahko okolje urejamo od kjer koli.

2.2.3 Strežnik

Strežnik, okrajšano za strežniški program, je program, ki poganja strežniške aplikacije. Ko pride zahtevek od uporabnika do ponudnika, ga strežnik obdela ter vrne zahtevan podatek.

Uporaba strežnika je obvezna, saj določena, predvsem cenejša wiki orodja ne vsebujejo t.i. vgrajenih strežnikov. Na trgu obstaja velika izbira strežnikov, ki so lahko nameščeni na različne operacijske sisteme. Za to nalogo je bil izbran Apache Tomcat 5.5, saj je namestitev preprosta, uporaba dokaj enostavna in brezplačna.

2.3.4 Varnost

Preprečevanje zlorab je čedalje zahtevnejše, saj so takšna okolja pogosto tarča nezaželenih vdorov. Velikokrat se tudi zgodi, da kdo namerno zbriše vsebino kakšne strani ali jo spremeni tako, da ni več uporabna oz. ne prikazuje pravilnih podatkov.

Varnost lahko razdelimo na tri področja. Vdor v omrežje ter s tem izbris vseh podatkov, tudi wiki okolja, lahko označimo kot najnevarnejše, vendar se proti temu ne moremo boriti z boljšim oz. slabšim wiki orodjem. Zaščita strežnika, na katerem je nameščeno okolje, je pri tem najpomembnejši dejavnik.

Drugo področje so nevarne vsebine, ki pridejo v poštev predvsem z uporabo appletov. Ker uporabniki sami dodajajo applete na strani, lahko pride no nezaželenih zlorab. Priporoča se uporaba t.i. podpisanih javanskih programčkov (signed applets), ki jih uporabniki pred uporabo odobrijo ali ne, odvisno od podpisnika. Vendar pa to dostikrat upočasni proces izdelave določene strani, saj se morajo ustvarjalci programčkov ubadati še s podpisovanjem le-teh. Prav zaradi tega je pri tem okolju omogočeno tudi dodajanje nepodpisanih programčkov. Ustvarjalec tega wiki okolja in te naloge prisega na poštenost in neškodoželjnost uporabnikov.

Tretji, najmanj nevaren, način zlorabe strani je brisanje oz. spreminjanje vsebine strani. Wiki okolje je narejeno tako, da shrani vsako spremembo določene strani v drugo datoteko ter mapo na računalniku, na katerem je nameščena. To pomeni, da v primeru zlorabe administrator preprosto nadomesti zlorabljeno datoteko. Seveda je težje slediti manjšim 'popravkom', ko nekdo zlonamerno spremeni samo eno besedo v besedilu in s tem zmanjša pravilnost podatkov.

Preden razvijemo okolje v celoti, je priporočljivo namerno zlorabiti okolje iz nekega oddaljenega računalnika, da vidimo, v kakšni meri in kako je okolje ranljivo. To pomeni, da prosimo prijatelje - računalnikarje, naj poskušajo narediti čimveč škode na našem okolju. Seveda je skozi celotno fazo razvoja priporočljivo imeti vsaj pol ducata varnostnih kopij.

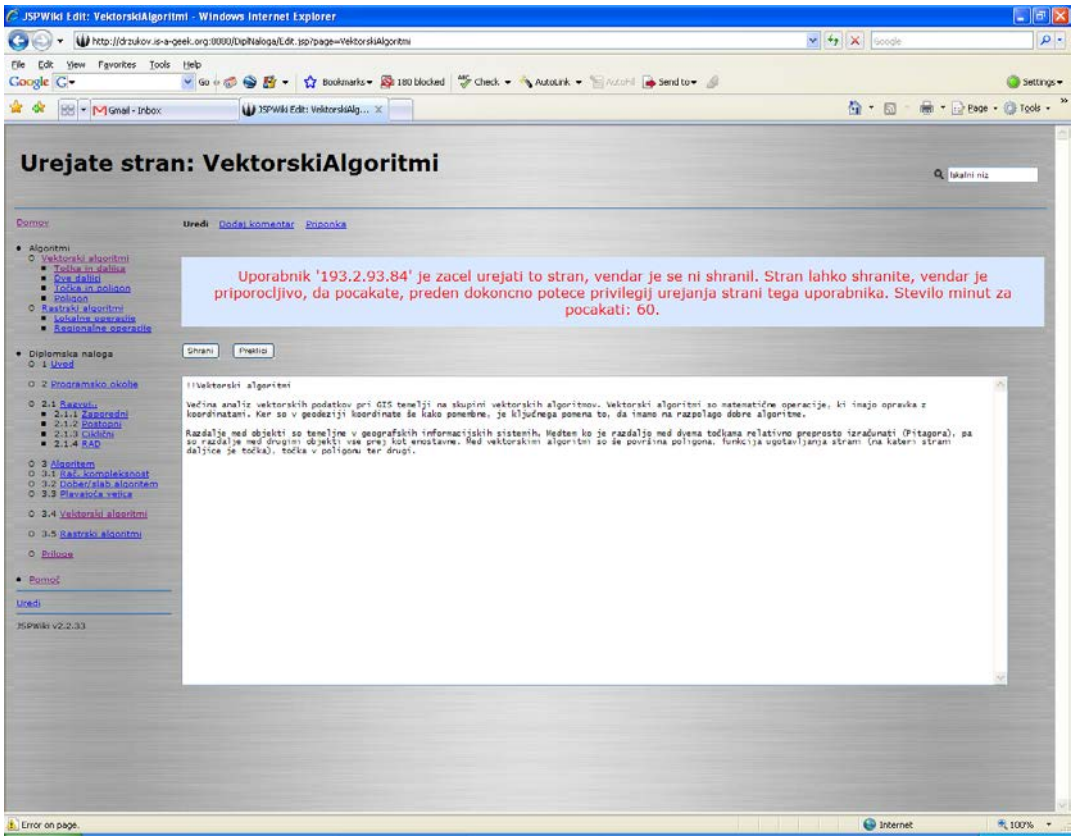
2.2.5 Testiranje

Testiranje okolja se je izvajalo ves čas razvoja, pri tem pa mislimo predvsem na možne hrošče znotraj okolja. Želeli smo preveriti vse situacije, v katerih se uporabnik lahko znajde. Se pravi, da smo sprožili vsako povezavo, ki jo okolje omogoča ter poskušali zlorabiti okolje na

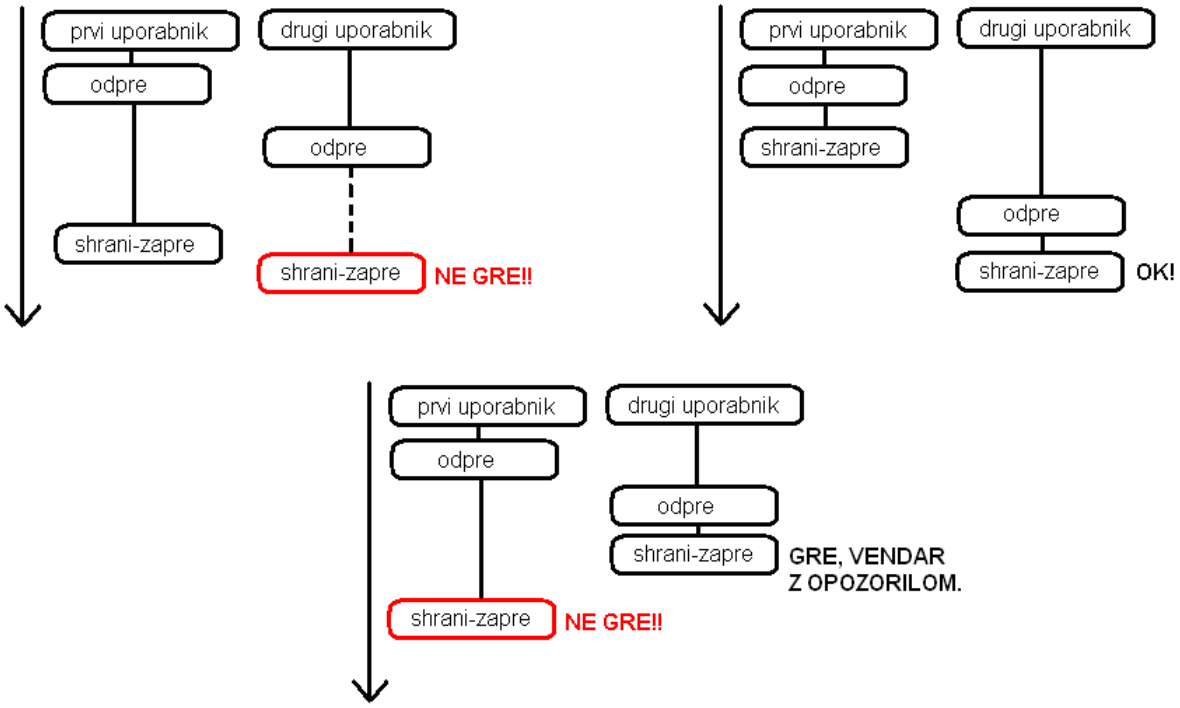
vse načine, ki smo si jih zamislili. Seveda smo skušali tudi preveriti, kaj se zgodi, ko okolje hkrati uporablja več uporabnikov ter jih več ureja isto stran.

Za razliko od podobnih okolij se je naše okolje nahajalo lokalno na administratorjevem računalniku, dostopno samo takrat, ko je bil ta računalnik povezan z internetom, na njem pa je bil v teku strežniški program. Dostopnost do tega računalnika je bila mogoča le s predhodno objavo naslova internetnega protokola (IP address) uporabniku. Prav zaradi tega je bilo testiranje skromnejše od priporočenega, v katerem se izdela nekaj preizkusnih različic ter povabi k sodelovanju veliko število uporabnikov.

Slika 9 prikazuje, kaj se zgodi, ko dva uporabnika hkrati urejata isto stran. Dokler je v fazi urejanje, se uporabnikoma zdi, da ni nič narobe, ko pa hoče drugi uporabnik shraniti izvod, ga ne more, ker ga je pred tem shranil že prvi uporabnik. Do napake ne bi prišlo, če bi drugi uporabnik odprl stran kasneje, kot jo je prvi uporabnik shranil.



Slika 9: Primer sporočila, ko dva uporabnika hkrati urejata isto stran



Slika 10: Diagram poteka urejanja strani

2.2.6 Vzdrževanje

Vzdrževanje okolja je običajno prepuščeno tistemu, ki to okolje vzpostavi. Seveda pa tudi tu ločimo več vrst vzdrževanja.

Vzdrževanje sistema, se pravi strežnika oz. računalnika, ki poganja celotno okolje, je naloga tistega, pri katerem je ta strežnik nameščen. To pomeni brezhibno delovanje strojne opreme, dograjevanje oz. dodajanje trdih diskov, če primanjkuje prostora ali pa preprosto zamenjavo celotnega strežnike, v primeru, da je ta že dotrajan in ne dosega nekih osnovnih standardov hitrosti.

Vzdrževanje strežnika in programske opreme je prepuščeno za to usposobljenim administratorjem ali drugim tehničnim delavcem ter obsega predvsem posodabljanje ali zamenjavo programske opreme, do katere lahko privede njena zastarelost in s tem neuporabnost.

Vzdrževanje podatkov je naloga tako uporabnikov kot vzdrževalcev. Če gre za sistem, ki se poganja v enem jeziku, npr. v slovenščini, lahko za podatke skrbijo za to zaposlene osebe. Wikipedia kot največje wiki okolje ima le šest stalno zaposlenih ljudi, ki skrbijo tako za podatke kot za delovanje okolja. Seveda pa ti zaposleni ne morejo skrbeti za podatke, ki so v drugih jezikih, ki jih ne govorijo. Tu pridejo v poštev uporabniki, ki popravijo podatke ali obvestijo administratorja, da je nekdo zlorabil določen podatek.

3 ALGORITEM

Algoritem je podroben opis računskega procesa, s katerim lahko izvedemo določeno operacijo. Koncept algoritma se je razvil kot sredstvo pri reševanju matematičnih problemov, kot na primer iskanje skupnega delitelja ali množenje števil.

Beseda algoritem ima zanimiv izvor. Al-Khwārizmī, perzijski astronom in matematik, je leta 825 napisal razpravo z naslovom *O računanju s hindujskimi števili,* ki je bila v latinščino prevedena v 12. stoletju z naslovom *Algoritmi de numero Indorum,* katerega pomen pa je najverjetneje bil *Al-Khwārizmī o računanju s hindujskimi števili*. Ime se je skozi leta v latinščini obdržalo ter sedaj pomeni računsko metodo.

Kako opišemo posamezni postopek, je odvisno od tega, kdo bo ta algoritem izvajal: človek ali računalnik. Pri računalnikih govorimo o računalniških programih, medtem ko so pri ljudeh algoritmi zapisani v naravnem jeziku. Primer ne tako preprostega algoritma za ljudi je recept za pripravo mozzarelinega mošnjička s česnovo omako.

Posebna vrsta algoritmov so geometrični algoritmi, ki operirajo nad predmeti prostorskega značaja. Vzemimo za primer problem, v katerem moramo določiti, ali je točka znotraj zanke ali ne. Medtem ko človeški možgani z lahkoto ugotovijo (seveda pri enostavnih zankah), ali je točka znotraj ali ne, pa tega za računalniški program ne bi mogli trditi.

Obstaja lahko več vrst algoritmov za isti problem. Denimo, da imamo mrežo slovenskih cest in bi radi izračunali najkrajšo razdaljo od Prešernovega trga v Ljubljani do Glavnega trga v Mariboru. Najbolj enostaven (vendar ne nujno najboljši!) algoritem je izračun vseh možnih kombinacij med tema dvema točkama. Da je takšen algoritem počasen in potraten (procesorske moči), najbrž ni potrebno posebej omenjati. Vendar obstajajo boljši algoritmi za izračun te naloge. Eden takšnih je algoritem z izločevanjem. Zdrava pamet nam pove, da se iz Ljubljane do Maribora ne bomo peljali skozi Koper ali Kranj. Torej lahko ti dve regiji (kot še mnogo drugih) izključimo iz postopka.

Računalniku bi lahko to povedali takole:
1) Vzemi regijo Ljubljana z okolico;
2) Vzemi regijo Štajerska;
3) Poglej, če je med njima v ravni črti še kakšna druga regija; če je, dodaj;
4) Izračunaj.

3.1 Računska kompleksnost

Kako preveriti, ali je naš algoritem dovolj hiter in kako mu dodeliti oznako, ki bi podajala njegovo učinkovitost, nam pove računska kompleksnost.

Računska kompleksnost se lahko nanaša na čas, ki je potreben za izvedbo algoritma ali na računalniški pomnilnik, potreben za izvedbo. Ker je za naše namene potrebno relativno malo računalniškega pomnilnika, nas bolj zanima računska kompleksnost za čas izvajanja algoritma.

Definicija računske kompleksnosti je sledeča (Harrie, 2005). Če je funkcija, ki opisuje procesorski čas, enaka g(n), pri čemer je n število vhodnih podatkov, je računska kompleksnost posameznega algoritma enaka O(f(n)), če obstaja takšna konstanta C ter celo število n_o, da velja:

$$C*f(n) >= g(n) \text{ za vse } n>n_o. \qquad (3.1)$$

Primer: Imamo množico n točk v prostoru, opremljenih s kartezičnimi koordinatami x in y. Zanima nas, katera od točk ima najmanjšo x koordinato. To storimo tako, da za vsako točko preverimo, ali morda ustreza iskanemu pogoju, kar pomeni, da moramo preveriti n točk. Če nas tudi zanima, katera ima najmanjšo y koordinato, lahko to storimo na dva načina. Prvi način bi bil ta, da bi ločeno od x koordinate pogledali za vsako točko tudi y koordinato (slika 11). V tem primeru bi imel ta algoritem Ordo(n^2). Boljši in hitrejši način bi bil, da skupaj z x

koordinato pogledamo tudi za y koordinato (slika12). V tem primeru bi imel algoritem Ordo(n).

```
clc
format short

for i=1:10
    M(i,1) = rand(1);
    M(i,2) = rand(1);
end
M

minx = M(1,1);
miny = M(1,2);

for i=1:10
    if M(i,1) < minx
        minx = M(i,1);
    end
end

for i=1:10
    if M(i,2) < miny
        miny = M(i,2);
    end
end

minx
miny
```

```
M =
    0.1573    0.4075
    0.4078    0.0527
    0.9418    0.1500
    0.3844    0.3111
    0.1685    0.8966
    0.3227    0.7340
    0.4109    0.3998
    0.5055    0.1693
    0.5247    0.6412
    0.0162    0.8369
minx =
    0.0162
miny =
    0.0527
```

Slika 11: Primer slabše programske kode, napisane v jeziku matlab

```
clc
clear
format short

for i=1:10
    M(i,1) = rand(1);
    M(i,2) = rand(1);
end
M

minx = M(1,1);
miny = M(1,2);

for i=1:10
    if M(i,1) < minx
        minx = M(i,1);
    end

    if M(i,2) < miny
        miny = M(i,2);
    end

end

minx
miny
```

```
M =
    0.7140    0.5152
    0.6059    0.9667
    0.8221    0.3178
    0.5877    0.1302
    0.2544    0.8030
    0.6678    0.0136
    0.5616    0.4546
    0.9049    0.2822
    0.0650    0.4766
    0.9837    0.9223
minx =
    0.0650
miny =
    0.0136
```

Slika 12: Primer boljše programske kode, napisane v jeziku matlab

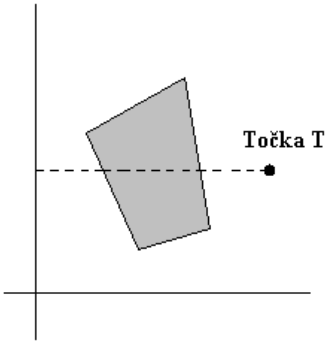
3.2 Dober in slab algoritem

Razvijalci se poslužujejo različnih metod, kako privarčevati pri času in procesorski moči, zato so algoritmi zgrajeni površno ter ne upoštevajo vseh možnih situacij, ki bi se pri izvajanju algoritma lahko zgodile. Dober algoritem je tak, ki predvidi vse možne situacije, ki lahko nastanejo med izvajanjem. Tak algoritem ne more zatajiti, lahko pa je, ob velikem številu podatkov, le zelo počasen. Boljši algoritmi so tisti, ki so hkrati dobri, vendar za zmanjšanje časa pri izvajanju uporabljajo hevristike.

Poglejmo primer algoritma, ki nam pove, ali je točka v poligonu ali ne. Pri tem velja omeniti, da je treba predhodno definirati, kaj naj algoritem naredi in česa ne. Če nas zanima, ali je točka v poligonu in je to tisto, za kar razvijamo algoritem, potem izločimo tudi točke, ki ležijo na meji poligona, ker jih pojmujemo za zunanje. Pri našem primeru bomo privzeli zgornjo definicijo, torej nas zanima le, če je točka v poligonu.

Poznamo več vrst algoritmov za to nalogo. Najpreprostejši je s štetjem, kolikokrat daljica iz točke do ene koordinatne osi seka poligon.

Osnovni princip je sledeč: narišemo daljico od točke do ene od koordinatnih osi. Ker mora veljati, da leži celoten poligon v istem kvadrantu, postavimo koordinatno os izven poligona oz. tako, da se dotika skrajnega roba poligona. Daljica naj bo vzporedna drugi koordinatni osi (slika 13).

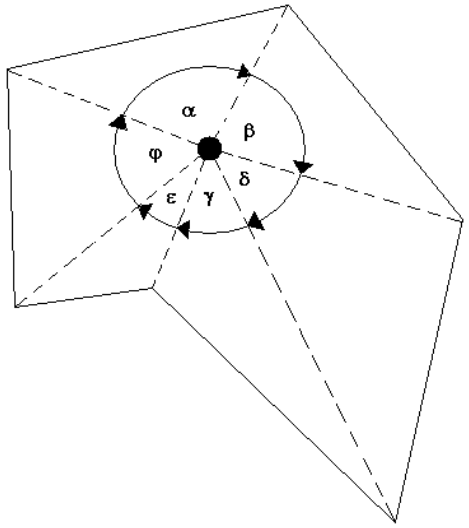


Slika 13: Preprost način izvedbe algoritma točka v poligonu

Če ta daljica seka poligon sodo število krat, točka ne leži v poligonu, v kolikor pa daljica seka poligon liho število krat, takrat točka leži v poligonu.

Seveda obstajajo še druge situacije. Daljica lahko potuje skozi eno izmed ogljišč ali pa je del stranice poligona. O podrobni obravnavi tega algoritma bomo povedali več v naslednjih poglavjih.

Zgornji algoritem se šteje za preprostejšega, obstajajo pa seveda še težji kakor tudi bolj zahtevni ter časovno potratni algoritmi. Eden takšnih je algoritem, ki na podlagi vsote kotov preveri, ali je točka v poligonu ali ne (slika 14).



Slika 14: Princip računanja algoritma točka v poligonu s seštevanjem kotov

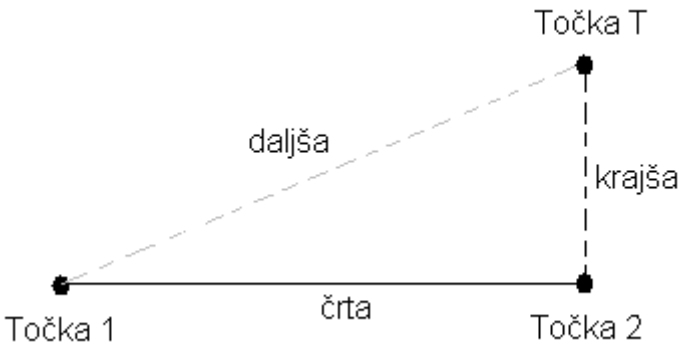
Iz točke, ki jo preverjamo, potegnemo daljice do posameznih ogljišč poligona. Vsota kotov, ki jo tvori točka z zaporednimi pari ogljišč, mora biti enaka večkratniku 360°. Če ni, je točka zunaj poligona.

Ker zgoraj omenjeni algoritem vsebuje računanje s kvadratnim korenom, arcus kosinusom, deljenjem, skalarnim ter vektorskim produktom, velja splošno prepričanje, da je ta algoritem najslabši možen algoritem za to vrsto naloge. Vendar je – po zgornji definiciji – to še vedno dober algoritem.

3.3 Algoritmi in (od)plavajoča vejica

V matematiki lahko brez težav uporabljamo realna števila, pri čemer nam ni potrebno posebej paziti, kako bomo zaokroževali npr. kvadratne korene, ker nam tega ni potrebno početi.

Zaradi uporabe matematičnih pravil pri programiranju algoritmov pa lahko to prinese neljube zaplete. Poglejmo spodnji primer:



Slika 15: Primer računanja razdalje točke od daljice

Da bi ugotovili razdaljo *Točke T* od *črte*, moramo najprej ugotoviti, ali je ta točka zunaj ali znotraj območja črte. Vidimo lahko, da je točka zunaj območja črte, natančneje na meji območja, torej je razdalja od nje to črte enaka njeni razdalji do najbližje točke; v tem primeru do *Točke 2*. Razdalja je torej enaka razdalji *krajša*.

Algoritem (natančneje je opisan v naslednjih poglavjih) pa od nas zahteva, da preverimo, kje se točka nahaja. Če velja: $\text{daljša}^2 < \text{krajša}^2 + \text{črta}^2$, potem se točka nahaja znotraj območja, v nasprotnem primeru pa zunaj območja.

Poglejmo na konkretnem primeru. Recimo, da so koordinate točk naslednje: Točka 1 (0,0); Točka 2 (5,0); Točka T (5,1). Izračunamo lahko posamezne razdalje. *črta* = 5, *krajša* = 1, *daljša* = koren 26. Če preverimo zgornji algoritem, kje naj bi se točka nahajala, dobimo:

$$\sqrt{26}^2 < 1^2 + 5^2 \qquad (3.2)$$

kar ne drži, saj sta obe strani neenačaja enaki, torej točka leži zunaj območja črte.

Če pa računamo z računalnikom, dobimo pri uporabi natančnosti s šestnajstimi decimalkami za enačbo koren 26 rezultat 5,0990195135927848. Seveda pa kvadrat dobljenega rezultata ni 26, temveč vedno malenkost manj; v našem primeru 25,9999999999999997.

Če sedaj preverimo zgornji algoritem o položaju točke, dobimo:

$$25{,}9999999999999997 < 1^2 + 5^2 \qquad (3.3)$$

Enačba (3.3) seveda drži, torej se točka nahaja znotraj območja črte, to pa pripelje do napačnega rezultata.

Ne glede na število decimalk, ki jih uporabljamo, realnih števil pri računalniškem programiranju ne bomo nikoli mogli zapisati dovolj dobro. To izhaja iz tega, ker lahko v računalniškem pomnilniku shranimo samo določeno količino podatkov. Pri zapisu števila z dvojno natančnostjo moramo za zapis uporabiti 8 bajtov pomnilnika, pri štirikratni natančnosti pa že 16 bajtov. Zato smo pri zapisovanju števil omejeni z velikostjo pomnilnika. Za vsakodnevno uporabo ne potrebujemo več kot dvojne natančnosti.

Da se izognemo nepotrebnim zapletom, uporabimo dogovor, da če je neka približna vrednost dovolj dobra, potem jo štejemo za pravo.

Pri programiranju lahko za zgornji primer uporabimo sledeč dogovor:

$$\text{Int}\,(\,\text{daljša}^2 * 10000\,) < \text{Int}\,((\,\text{krajša}^2 + \text{črta}^2\,) * 10000\,) \qquad (3.4)$$

kar se bere kot: vsako stran neenačaja pomnožimo z neko vrednostjo, odvisno od tega, kako točen želimo imeti algoritem ter to vrednost zaokrožimo na celo število.

Za zgornji primer tako velja:

$$2600000 < 2600000, \tag{3.5}$$

kar seveda ne drži, torej se nahaja točka zunaj območja črte.

3.4 Vektorski algoritmi

Večina analiz vektorskih podatkov pri GIS temelji na skupini vektorskih algoritmov. Vektorski algoritmi so matematične operacije, ki imajo opravka s koordinatami. Ker so v geodeziji koordinate še kako pomembne, je ključnega pomena to, da imamo na razpolago dobre algoritme.

Razdalje med objekti so temeljne v geografskih informacijskih sistemih. Medtem ko je razdaljo med dvema točkama relativno preprosto izračunati (Pitagora), pa so razdalje med drugimi objekti vse prej kot enostavne. Med vektorskimi algoritmi so še površina poligona, funkcija ugotavljanja strani (na kateri strani daljice je točka), točka v poligonu ter drugi.

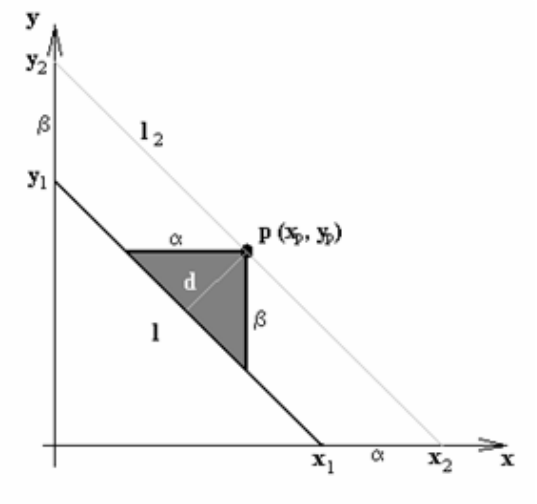
3.4.1 Razdalja med točko in premico

Za začetek poglejmo, kako izračunamo razdaljo med točko in premico. Povedati je potrebno, da v vektorskih algoritmih ne uporabljamo premic, saj ne moremo predstavljati količin z neskončnimi vrednostmi. Prav zato moramo biti pri uporabi spodnje enačbe še posebej pozorni.

Dve premici l_1 in l_2 (slika 16) lahko predstavimo kot:

$$l_1 = \{(x, y) a \cdot x + b \cdot y + c_1 = 0\} \tag{3.6}$$

$$l_2 = \{(x, y) a \cdot x + b \cdot y + c_2 = 0\} \tag{3.7}$$



Slika 16: Razdalja od točke do premice

Ker sta premici vzporedni, sta parametra a in b za obe premici enaka, razlikujeta se le parametra c_1 in c_2.

Površino osenčene ploskve lahko zapišemo na dva načina:

$$P = \frac{1}{2} \cdot \alpha \cdot \beta \tag{3.8}$$

$$P = \frac{1}{2} \cdot \sqrt{\alpha^2 + \beta^2} \cdot d \tag{3.9}$$

Iz teh dveh enačb lahko izračunamo d:

$$d = \frac{\alpha \cdot \beta}{\sqrt{\alpha^2 + \beta^2}} \tag{3.10}$$

Če podrobno pogledamo sliko 16, lahko razberemo velikosti α in β:

$$\alpha = |x_1 - x_2| = \left|\frac{(c_1 - c_2)}{a}\right| \tag{3.11}$$

$$\beta = |y_1 - y_2| = \left|\frac{(c_1 - c_2)}{b}\right| \tag{3.12}$$

pri čemer smo uporabili x = 0 ter y = 0, za primer koordinatnih osi.

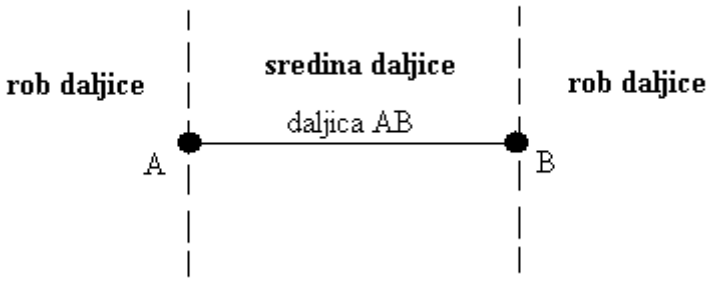
Končno lahko izračunamo velikost dolžine med točko in premico:

$$d = \frac{\left(\left|\frac{c_1 - c_2}{a}\right| \cdot \left|\frac{c_1 - c_2}{b}\right|\right)}{\sqrt{\left|\frac{c_1 - c_2}{a}\right|^2 + \left|\frac{c_1 - c_2}{b}\right|^2}} = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}} = \frac{\left|(a \cdot x_p + b \cdot y_p + c_1)\right|}{\sqrt{a^2 + b^2}} \tag{3.13}$$

Zgornjo enačbo bomo kasneje uprabili pri izračunu razdalje od točke do daljice.

3.4.2 Razdalja med točko in daljico

Ko računamo razdaljo med točko in daljico, lahko nastopita dva primera. Točka je lahko na *sredini* oziroma na *robu* daljice (slika 17).



Slika 17: Razdalja od točke do daljice

Če se točka nahaja na robu daljice, za izračun razdalje zadošča Pitagorjev izrek, v kolikor pa je točka na sredini, uporabimo enačbo za izračun razdalje od točke do premice (enačba 3.13). Da preverimo, ali je točka v sredini ali ne, uporabimo naslednji izraz:

(krajša dolžina)^2 < (dolžina daljice)^2 + (krajša dolžina)^2 => točka leži na sredini daljice
(krajša dolžina)^2 ≥ (dolžina daljice)^2 + (krajša dolžina)^2 => točka leži na robu daljice

3.4.3 Razdalja med dvema daljicama

Razdalja med dvema daljicama, ki se ne sekata, je bodisi enaka razdalji med dvema točkama bodisi med točko ene daljice ter drugo daljico. Dovolj je, da izračunamo razdalje med točkami ter daljicami.

3.4.4 Razdalja med dvema poligonoma

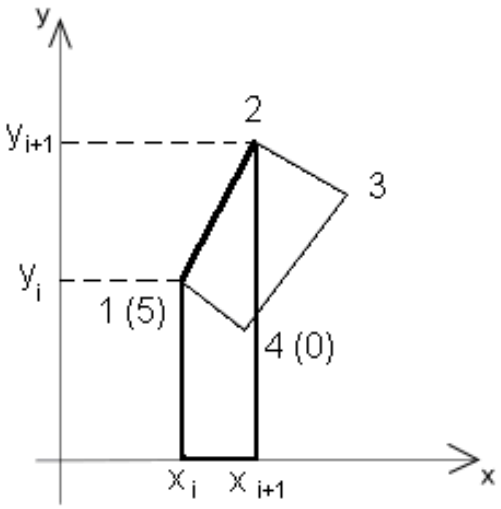
Razdaljo med dvema poligonoma, ki se ne sekata, dobimo tako, da izračunamo posamezne razdalje med vsemi stranicami obeh poligonov.

3.4.5 Površina poligona

Površine poligonov v GIS se računajo izključno iz koordinat. Spodaj je predstavljen algoritem, ki izračuna površino poligona, katerega koordinate so podane zaporedno v smeri urinega kazalca. Algoritem upošteva samo preproste poligone, t.j. poligone brez lukenj.

Celoten poligon razdelimo na trapezoide, ki jih dobimo tako, da iz posameznih točk potegnemo daljice do x-osi (slika 18). Te daljice potem predstavljajo vzporedne stranice trapezoidov. Površina posameznega trapezoida je tako:

$$P(i) = \frac{(x_{i+1} - x_i) \cdot (y_{i+1} + y_i)}{2} \qquad (3.14)$$



Slika 18: Površina poligona

Če sedaj uporabimo enačbo (3.14) za izračun posameznih površin trapezoidov, dobimo za zgornjo polovico poligona pozitivne vrednosti, za spodnjo polovico poligona pa negativne vrednosti površin. Zaključimo lahko, da je površina poligona enaka vsoti vseh površin trapezoidov:

$$P = \sum_{i=1}^{n} \frac{(x_{i+1} - x_i) \cdot (y_{i+1} + y_i)}{2} \tag{3.15}$$

Pri tem velja poudariti, da morajo biti točke oštevilčene tako, kot na sliki 18 in mora veljati, da je prva točka hkrati enaka točki z indeksom za 1 večjim od zadnje točke ter da je zadnja točka hkrati enaka točki z indeksom za 1 manjšim od prve točke.

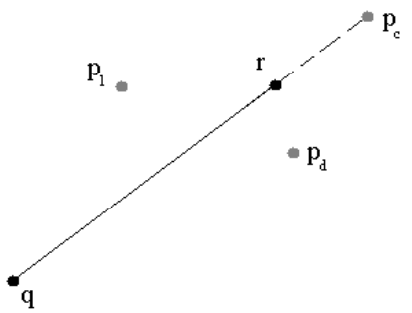
Enačbo(3.15) lahko poenostavimo, in sicer:

$$
\begin{aligned}
P &= \sum_{i=1}^{n} \frac{(x_{i+1} - x_i)\cdot(y_{i+1} + y_i)}{2} = \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} y_i (x_{i+1} - x_i) + \sum_{i=1}^{n} y_{i+1}(x_{i+1} - x_i) \right) = \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} y_i (x_{i+1} - x_i) + \sum_{i=1}^{n} y_i (x_i - x_{i-1}) \right) = \\
&= \frac{1}{2} \sum_{i=1}^{n} y_i (x_{i+1} - x_{i-1})
\end{aligned}
\tag{3.16}
$$

V primeru uporabe geodetskega koordinatnega sistema moramo samo zamenjati oznaki x oz. y v zgornji enačbi.

3.4.6 Funkcija ugotavljanja strani

Enačbo 3.16 lahko uporabimo tudi za ugotavljanje, na kateri strani daljice leži posamezna točka. V algoritem vnesemo obe točki daljice ter točko, za katero želimo izvedeti, na kateri strani leži. V kolikor bodo točke kolinearne, bo površina poligona enaka nič, v primeru, ko pa bodo točke nekolinearne, bo površina poligona bodisi negativna bodisi pozitivna.



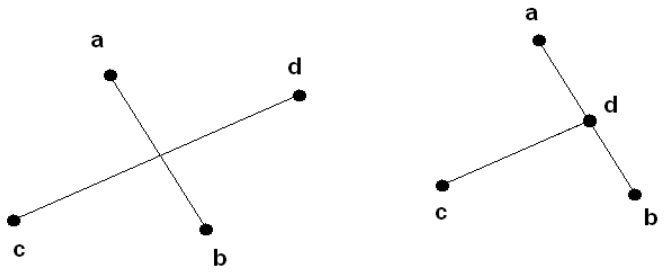
Slika 19: Točka na različnih straneh daljice

Pri programiranju je pogosto uporabno, da napišemo funkcijo, ki vrne naslednje vrednosti:

```
                =  -1 , če leži točka na levi strani daljice
stran(p,q,r)    =   0 , če je točka kolinearna z daljico
                =   1 , če leži točka na desni strani daljice
```

3.4.7 Presek daljic

Pri preseku daljic mislimo predvsem na to, ali se dve daljici sekata ali ne. Pri naši definiciji se daljici ne sekata takrat, ko leži točka ene daljice na drugi daljici (slika 20 desno).



Slika 20: Presek daljic (levo) ter situacija, kjer se daljici samo srečata in ne sekata (desno)

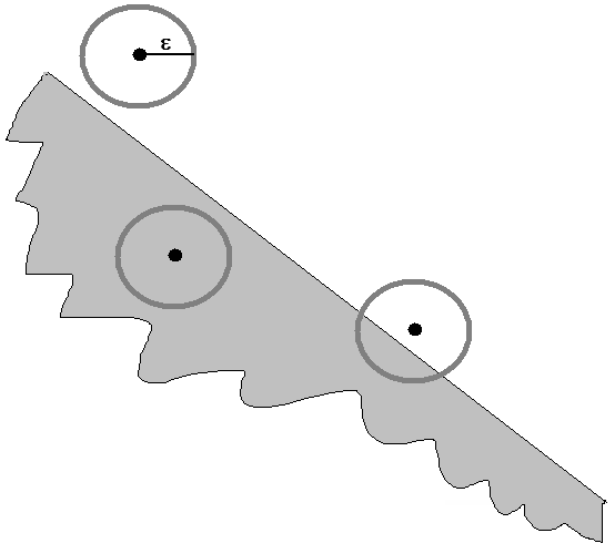
Presek daljic izračunamo s pomočjo funkcije ugotavljanja strani, in sicer:

$$\left.\begin{array}{l} stran(a,b,c) \neq stran(a,b,d) \\ stran(c,d,a) \neq stran(c,d,b) \end{array}\right\} \quad \text{daljici a-b in c-d se sekata} \qquad (3.17)$$

3.4.8 Točka v poligonu

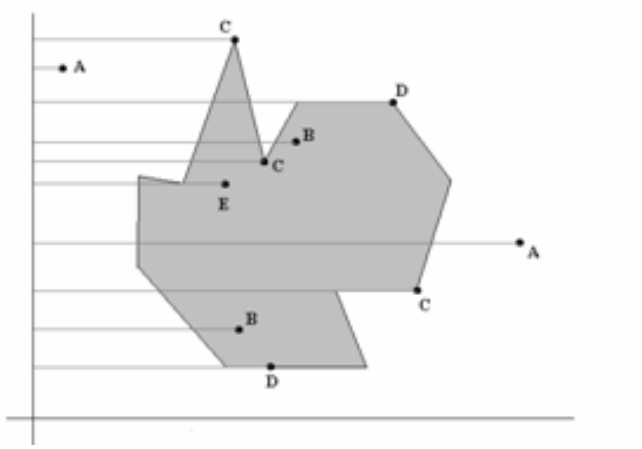
Kot smo omenili že v prejšnjih poglavjih, obstaja več vrst algoritmov za primer točke v poligonu. Tukaj se bomo osredotočili zgolj na že prej opisanega, torej s pomočjo štetja, kolikokrat določena črta seka poligon.

Če bi ubrali matematični pristop, bi morali definirati neko poljubno majhno okolico ε, s pomočjo katere bi preverili, ali leži točka v poligonu, na robu ali izven njega (slika 21).



Slika 21: Matematični način reševanja problema z uporabo okolice ε

Pri programiranju se lotimo zadeve nekoliko drugače (Shimrat, 1962).

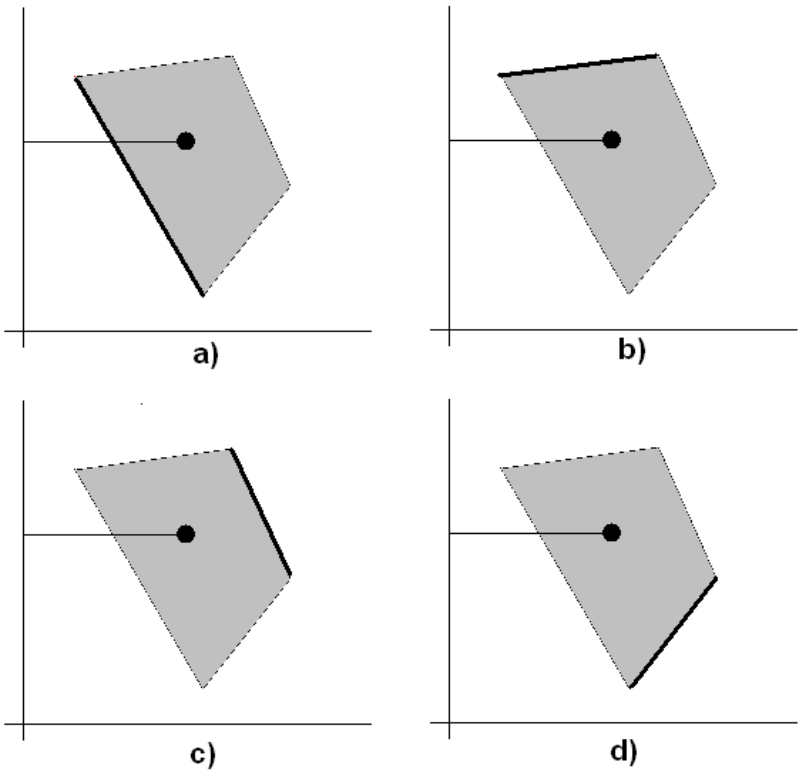


Slika 22: Programerski način reševanja istega problema

Najpreprosteje je, da potegnemo črto od točke, za katero želimo preveriti, ali je v poligonu ali ne, do ene od koordinatnih osi. Pri tem moramo paziti, da leži celoten poligon v istem kvadrantu. V nasprotnem primeru potegnemo črto od točke do osi, ki se nahaja povsem izven poligona.

Nastopijo lahko različne situacije. Črta lahko seka poligon enkrat, večkrat, lahko potuje skozi točko ali linijo. Boljši algoritmi se razlikujejo od slabših po tem, da upoštevajo vse možne situacije.

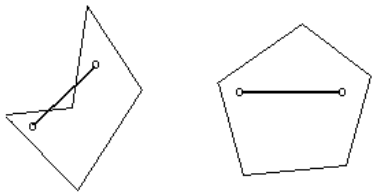
Preprost algoritem bi bil sledeč: za vsak rob poligona preverimo, ali se morda seka z referenčno črto. Na spodnji sliki lepo vidimo potek takšnega algoritma. Kot kaže slika, črta seka rob poligona samo enkrat, torej se točka nahaja v poligonu.



Slika 23: Potek algoritma po korakih

3.4.9 Konveksna lupina

Algoritem se uporablja za ugotavljanje razsežnosti množice točk. Konveksna lupina je najmanjši poligon, ki vsebuje vse točke dane množice. Po definiciji je za konveksni poligon značilno, da lahko med dvema poljubnima točkama v tem poligonu potegnemo daljico, ki je v celoti vsebovana v tem poligonu.



Slika 24: Primer nekonveksne (levo) ter konveksne množice (desno)

Najpreprostejši algoritem bi bil sledeč:

```
za vsak i
        za vsak j
                dodaj segment med i in j konveksni lupini
                za vsak k (ki ni i ali j)
                        če je k desno od segmenta
                                briši segment med i in j
```

Ta algoritem ima računsko kompleksnost reda O(n3), kjer je n število točk v množici.

Primer boljšega algoritma je naslednji:

```
najdi točko z najmanjšo y-koordinato, nastavi jo kot začetno točko i
        potegni vodoravno črto, ki se konča v točki i
                ponavljaj
                {
                        za vsako točko, ki ni i
                                izračunaj kot do i v obratni smeri urinega kazalca
                                        točko z najmanjšim kotom označi za k
                                                dodaj segment med i in k konveksni lupini
                                                        označi točko k kot i
                }
                dokler točka i ni enaka začetni točki
```

3.5 Rastrski algoritmi

Na kratko omenimo še rastrske algoritme. Obstajajo tri vrste rastrskih podatkov v GIS:

- digitalne karte,
- digitalni modeli reliefa in digitalni modeli višin ter
- digitalni ortofoto načrti, torej podobe.

Za vsakega od teh podatkov obstajajo različne metode, kako iz teh podatkov dobiti določene informacije. Za digitalne ortofoto načrte obstajajo razna filtriranja, kar pomeni, da z matrico prečesamo celotno območje, kot rezultat pa dobimo novo karto. Pri digitalnih modelih reliefa oz. višin pride v upoštev senčenje, analiza padca vode (ang. flow analysis), pri digitalnih kartah pa razne analize stroškov, multikriterijsko odločanje ter generalizacija.

3.5.1 Lokalne ter regionalne operacije

Lokalne operacije so tiste vrste rastrskih algoritmov, pri katerih je poljuben slikovni element (i,j) na novi karti odvisen samo od slikovnega elementa (i,j) na izvirni karti, pri čemer sta i in j številka vrstice oz. stolpca.

Te algoritme uporabljamo za manipuliranje s slikami. Slika 25 prikazuje množenje vrednosti vsakega slikovnega elementa z vrednostjo 1.2, kar v praksi rahlo posvetli izvirno podobo.

100	100	110	140
110	110	100	150
130	100	190	200
140	140	100	100

x 1.2

120	120	132	168
132	132	120	180
156	120	228	240
168	168	120	120

Slika 25: Lokalna operacija, prikazana s slikovnimi elementi

Primer regionalne operacije pa je filtriranje. Filtri so matrike, ki jih poženemo čez izvirni raster, delujejo pa na tako velikem območju, kot je velikost samega filtra. Izhodni raster je praviloma manjše velikosti kot izvirni, predvsem zaradi omejenega delovanja filtrov na robovih.

Slika 26 prikazuje filtriranje izvirne podobe s filtrom velikosti 2x2. Filter, kakršen je prikazan spodaj, nam vrne povprečno vrednost štirih slikovnih elementov v izvirni sliki ter jo zapiše v novo matriko, ki predstavlja novo podobo.

100	100	110	140
110	110	100	150
130	100	190	200
140	140	100	100

⊗

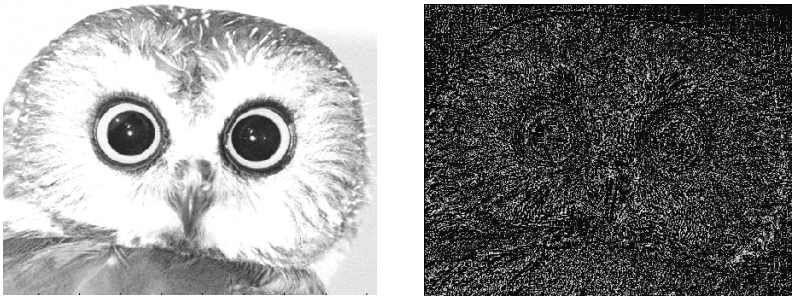
0.25	0.25
0.25	0.25

102	110	127
112	122	137
130	137	160

Slika 26: Regionalna operacija, prikazana s slikovnimi elementi in filtrom

Primeri rastrskih operacij:

Slika 27: Izvrina podoba pred operacijo



Slika 28: Podoba po lokalni operaciji (levo) ter regionalni operaciji (desno)

4 UPORABNOST OKOLJA

Kot smo že omenili, lahko okolje uporabimo za prikaz različnih besedil, slik ter drugih dodatkov. Z uporabo le-tega lahko objavljamo članke ter druge raziskovalne projekte. Napredno lahko urejamo z uporabo ukazov CSS, ki jih umestimo med dvojne znake za odstotek (%%). Z uporabo priponk lahko uporabnikom ponudimo, da si kakšne datoteke tudi snamejo. Velikost in vrsta datoteke ne omejujeta objave; ta je odvisna le od prostora, ki je na voljo na strežniku. Vendar pa se s tem uporabnost okolja še ne konča.

Okolje lahko uporabimo tudi za prikaz pomembnih povezav, ki jih lahko uporabniki dodajajo ali brišejo, v kolikor ne služijo namenu. Prav tako je na voljo upraba javanskih programčkov, ki jih preprosto dodamo kot priponke ter prikličemo z enostavnim ukazom v urejevalniku besedila.

Vzpostavimo lahko tudi forum, vendar pa je v takšni obliki precej ranljiv, saj lahko spreminjamo prispevke drugih udeležencev. Z dodatno nadgradnjo okolja lahko zahtevamo od uporabnikov, da se v okolje prijavijo z uporabniškim imenom in geslom. Tako lahko spremljamo, kdo je dodal oz. spremenil določeno stran ter mu ob morebitni zlorabi prepovemo uporabljati okolje.

Uporabniki, ki so prijavljeni v okolje, lahko objavljajo svoje spletne dnevnike (ang. blog), ki postajajo vse bolj priljubljeni. Za zahtevnejše uporabnike lahko vzpostavimo tudi obveščanje preko RSS tehnologije, tako da prejmejo obvestilo takoj, ko se vsebina določene strani spremeni.

Okolje je primerno za objavljanje študijskega gradiva, kjer lahko združimo opise nalog, datoteke, komentarje ter javanske programčke. Prav tako je lahko primerno za objavljanje tečajev različnih panog. Okolje podpira tudi nadgradnjo s SQL bazo.

Kombinacij, s katerimi lahko upravljamo okolje, je veliko. Odvisno je le od tega, kaj želimo ponuditi uporabnikom.

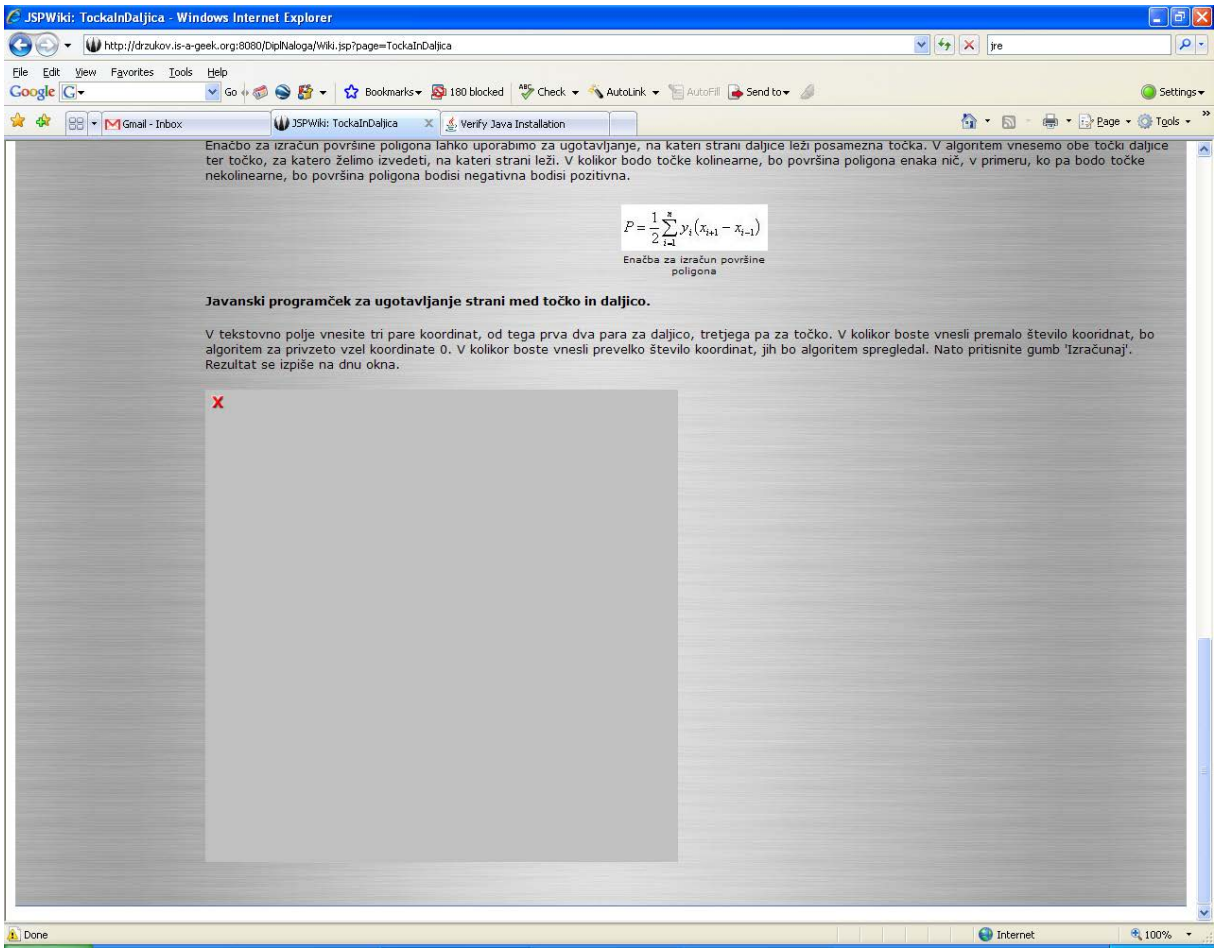
5 ZAKLJUČEK

Brez zadržkov lahko rečemo, da bi programsko okolje potrebovalo dodatne izboljšave, saj je za razvoj resnih aplikacij potrebnih več ljudi. Kot pomanjkljivost lahko omenimo tudi to, da je bilo okolje preizkušeno na samo dveh različnih brskalnikih. Kot pri veliko aplikacijah bi morala tudi ta nositi podatke o minimalni zahtevani strojni in programski opremi, saj se pri izvajanju algoritmov lahko hitro zgodi, da računalnik preprosto zmrzne.

Ker se pri razvoju brskalnikov niso pojavili določeni standardi, se še danes, po skoraj petnajstih letih razvoja brskalnikov, vidijo razlike pri uporabi. Okolje je bilo testirano na dveh trenutno najbolj priljubljenih brskalnikih, Internet Explorerju različice 7.0 nadgradnje Beta 2 podjetja Microsoft, ter Mozilli Firefox različice 2.0.0.6 podjetja Mozilla Foundation. Kot se je izkazalo, je orodje JSPWiki najbolj prilagodljivo brskalniku podjetja Microsoft. Vse to se je pokazalo v majhnih podrobnostih, kot so dostopnost do iskalnega polja, ki je bila v Firefoxu mogoča le z uporabo tipkovnice ali pa z vnosom podob v urejevalniku, ki jih je Firefox ob uporabi istega vnosa poravnal na levo stran, Internet Explorer pa na sredino. Vendar pa se pri normalni uporabi okolja takšnih malenkosti ne opazi.

Za normalno uporabo ter boljšo preglednost se priporoča uporaba 17-palčnega zaslona, saj se pri manjših velikostih zaslonov določena besedila prekrivajo. To lahko odpravimo tako, da zmanjšamo velikost prikazanega besedila, kar nam omogoča vsak brskalnik.

Prav tako je za ogled javanskih programčkov potrebno javino pogonsko okolje (ang. Java Runtime Environment) različice 6 ali novejše, saj se v nasprotnem primeru prikaže samo prazno okno (slika 29). V kolikor pa javanske programčke razvijemo s starejšo različico jave, potem za njihov prikaz zadostuje starejša različica javinega pogonskega okolja.



Slika 29: Prikaz nedelovanja javanskih programčkov

Okolje je dostopno brezplačno pod licenco GPL, za katero ima vse avtorske pravice Fakulteta za gradbeništvo in geodezijo Univeze v Ljubljani. Skupaj s izvirno kodo mora biti objavljena tudi licenca, ki se ponavadi nahaja v datoteki lgpl.txt.

Okolja, ki ponujajo možnost uporabnikovega sodelovanja, so brez dvoma v vzponu, zato imajo aplikacije, kot je ta, svetlo prihodnost. Na to kaže tudi dejstvo, da jih lahko uporabljamo tako za poljudne kot tudi za znanstvene namene. Ker praktično ni omejitev pri številu objavljenih strani, odvisne so le od prostora, ki je na voljo, lahko takšna okolja uporabljamo za veliko število različnih področij, združenih v eno samo okolje. Eno takšnih področij so prav gotovo informacijski sistemi.

Poleg vektorskih algoritmov, ki so predstavljeni v tej nalogi, lahko javanske programčke uporabimo tudi za obdelovanje rastrskih, mrežnih ter topoloških algoritmov. Težava se zna pojaviti le tam, kjer želimo določene podatke shraniti na naš lokalni disk, saj je zaradi varnosti to večkrat onemogočeno.

Poleg geografskih informacijskih sistemov lahko javanske programčke znotraj okolja uporabimo še za druga področja v okviru geodezije. Tukaj mislimo predvsem na obdelavo podob in računanje raznih koeficientov pri fotogrametriji, računanje geodetskih nalog pri inženirski geodeziji, izravnalne račune pri višji geodeziji ter drugo.

Prav tako je okolje lahko uporabno med samim študijskim procesom, kjer poteka komunikacija med profesorji in študenti, predvsem ko gre za podajanje učnega gradiva ter komentiranje vsebine spletne strani.

Če pogledamo širše, se lahko okolje uprablja za objave novic, kjer pa vsakdo prispeva k posodobljenosti in točnosti informacij. Te novice so lahko opremljene s slikovnim ali drugim večpredstavnostnim gradivom, komentarji ter raznimi viri.

Okolje je prijazno tudi do uporabnikov, ki želijo brezplačno ponuditi svoje delo v pogled ali prenos še drugim uporabnikom spleta. Gre predvsem za ljudi, ki želijo svojo programsko opremo, podatke ali svoj pogled na določeno zadevo narediti dostopno čim večji množici ljudi.

Glavna prednost takšnega okolja pred drugimi je ta, da se urejanje lahko dogaja že med tem, ko mi brskamo po določeni strani. S tem nam je prihranjeno pošiljanje popravljenih datotek na strežnik, kar posledično pripelje do manjše porabe časa. Poleg tega pa je s tem večja verjetnost, da bo naključni uporabnik okolja dejansko sodeloval pri sooblikovanju te strani.

Kako bo okolje dostopno uporabnikom, je odvisno le od avtorja okolja, vendar pa mora ta vedno upoštevati dejstvo, da se bo okolje nenehno dopolnjevalo ter da bo, ob preveliki zaščiti ali nedostopnosti, tudi izgubilo prvotni pomen.

VIRI

UPORABLJENI VIRI

- Burrough A. Peter, McDonnel A. Rachael. 1998. Principles of Geographical Information Systems. New York, Oxford University Press: 333 str.

- Harrie, L. 2005. Lecture notes in GIS Algorithms. Lund, Lund University. GIS Centre and Department of Physical Geodesy and Ecosystem Analysis.

- Martin James. 1991. Rapid Application Development. Toronto, Macmillan Coll Div.

- Shimrat, M. 1962. Algorithm 112, Position of Point Relative to Polygon. Communications of the ACM. Calgary, University of Alberta: str. 434.

- Šumrada, R. 2005. Strukture podatkov in prostorske analize. Ljubljana, Univerza v Ljubljani, Fakulteta za gradbeništvo in geodezijo: 284 str.

- Šumrada, R. 2005, Tehnologija GIS. Ljubljana, Univerza v Ljubljani, Fakulteta za gradbeništvo in geodezijo: 330 str..

- Worboys F. Michael, Matt Duckham. 2004. GIS: A Computing perspective. Taylor & Francis: 426 str.

OSTALI VIRI

- Brittain Jason, Darwin Ian F. Tomcat: The Definitive Guide. O'Reilly Books.

- Mesojedec Uroš, Fabjan Borut. 2004. Java2: temelji programiranja. Ljubljana, Založba Pasadena: 596 str.

- Pahor David, Drobnič Matija. 2002. Leksikon računalništva in informatike. Ljubljana, Založba Pasadena: 786 str.

- Schildt Herbert. 2007. Swing: A Beginner's Guide. New York, McGraw-Hill: 590 str.

PRILOGE

PRILOGA A: VEKTORSKI ALGORITMI – IZVIRNE KODE

PRILOGA B: RASTRSKI ALGORITMI - IZVIRNE KODE

PRILOGA C: WIKI PRIROČNIK

PRILOGA D: WIKI.JSP - IZVIRNA KODA

PRILOGA E: JAVANSKI PROGRAMČEK PRI IZVAJANJU

PRILOGA F: DODAJANJE KOMENTARJA

PRILOGA G: DODAJANJE PRIPONK

PRILOGA H: UREJANJE GLAVNEGA KAZALA

PRILOGA I: ISKANJE NIZA

PRILOGA A: VEKTORSKI ALGORITMI – IZVIRNE KODE

Razdalja med točko in daljico

Matlab

```
% lintoc.h
%
% Funkcija za izračun razdalje od točke do linije (daljice).
% Funkcija vrne vrednost (razdaljo).
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | lintoc((x1,y1,x2,y2,xt,yt);                          |
% |                                                      |
% | x1,y1,x2,y2  --koordinate daljice                    |
% | xt,yt        --koordinati točke                      |
% | ;  -- ne prikaže vrnjene vrednosti                   |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function razdalja = lintoc(x1,y1,x2,y2,xt,yt)
format long

crta  = sqrt((x1-x2)^2+(y1-y2)^2);
crta1 = sqrt((x1-xt)^2+(y1-yt)^2);
crta2 = sqrt((x2-xt)^2+(y2-yt)^2);

if (crta1 > crta2)
    daljsa = crta1;
    krajsa = crta2;
else
    daljsa = crta2;
    krajsa = crta1;
end

if(int16((daljsa^2)*10000) < int16((crta^2+krajsa^2)*10000))
    %disp('crta je v sredini');
    razdalja = krajsa*daljsa / sqrt(krajsa^2+daljsa^2);
else
    %disp('crta je na robu');
    razdalja = krajsa;
end

disp(sprintf('razdalja: %f',razdalja));

end
```

Java

```
/*  double tockaDaljica (double[] daljica, double[] tocka)
 *
 *  Funkcija sprejme niz koordinat tock daljice ter niz
 *  koordinat ene tocke. Ce sta v nizu vec kot dva para koordinat,
 *  potem funkcija vzame samo prva dva.
 *
 *  Funkcija vrne vrednost razdalje.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static double tockaDaljica (double[] daljica, double[] tocka) {

        double crta = Math.sqrt((daljica[0] - daljica[2])*(daljica[0] -
                      daljica[2]) + (daljica[1] - daljica[3])*(daljica[1] - daljica[3]));
        double crta1 = Math.sqrt((daljica[0] - tocka[0])*(daljica[0] - tocka[0]) +
                      (daljica[1] - tocka[1])*(daljica[1] - tocka[1]));
        double crta2 = Math.sqrt((daljica[2] - tocka[0])*(daljica[2] - tocka[0]) +
```

```
                        (daljica[3] - tocka[1])*(daljica[3] - tocka[1]));

        double krajsa, daljsa;

        if (crta1 > crta2) {
                daljsa = crta1;
                krajsa = crta2;
        }
        else {
                daljsa = crta2;
                krajsa = crta1;
        }

        if ((daljsa*daljsa)*10000 < (crta*crta+krajsa*krajsa)*10000) {
                return krajsa*daljsa / (Math.sqrt(krajsa*krajsa+daljsa*daljsa));
        }
        else {
                return krajsa;
        }
}
```

Razdalja med dvema daljicama

 # Matlab

```
% linlin.h
%
% Funkcija za izračun razdalje med dvema daljicama
% Funkcija vrne vrednost (razdaljo).
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | linlin(A,B);                                         |
% |                                                      |
% | A = (x1,y1,x2,y2)  --koordinate prve daljice         |
% | B = (x3,y3,x4,y4)  --koordinate druge daljice        |
% | ;  -- ne prikaže vrnjene vrednosti                   |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function razdalja = linlin(A,B)

clc,clf

a=[A(1),A(2)]; b=[A(3),A(4)];
c=[B(1),B(2)]; d=[B(3),B(4)];

X1=[a(1),b(1)]; Y1=[a(2),b(2)];
X2=[c(1),d(1)]; Y2=[c(2),d(2)];


line(X1,Y1,'LineWidth',2,'LineStyle',':'), hold on, axis equal,
line(X2,Y2,'LineWidth',2)

if ( and(stran(a(1),a(2),b(1),b(2),c(1),c(2))~=...
         stran(a(1),a(2),b(1),b(2),d(1),d(2)), ...
         stran(c(1),c(2),d(1),d(2),a(1),a(2))~=...
         stran(c(1),c(2),d(1),d(2),b(1),b(2)))  )
         razdalja = 0;
         disp('Liniji se sekata.');

else
    disp('Liniji se ne sekata,');

    raz1 = sqrt(((a(1)-c(1))^2) + ((a(2)-c(2))^2));
    raz2 = sqrt(((a(1)-d(1))^2) + ((a(2)-d(2))^2));

    raz3 = sqrt(((b(1)-c(1))^2) + ((b(2)-c(2))^2));
    raz4 = sqrt(((b(1)-d(1))^2) + ((b(2)-d(2))^2));
```

```
    raz = [raz1,raz2,raz3,raz4];
    razdalja = min(raz);
    disp(sprintf('najmanjsa razdalja med njima je: %.5f\n',razdalja));

end
```

Java

```
/*  double odmikDaljic(double[] daljica, double[] daljica2)
 *
 *  Funkcija sprejme dva niza koordinat tock dveh daljic.
 *  Ce sta v nizu vec kot dva para koordinat, potem funkcija
 *  vzame samo prva dva.
 *
 *  Funkcija vrne vrednost 0, če se daljici sekata. Če se daljici
 *  ne sekata, vrne funkcija vrednost razdalje.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static double odmikDaljic (double[] daljica, double[] daljica2) {

        if (presekDaljic(daljica, daljica2)==1)
                return 0;
        else {
                double raz1 = Math.pow(Math.pow( daljica[0]-daljica2[0],2 ) +
                                Math.pow( daljica[1]-daljica2[1],2 ),0.5);

                double raz2 = Math.pow(Math.pow( daljica[2]-daljica2[0],2 ) +
                                Math.pow( daljica[3]-daljica2[1],2 ),0.5);

                double raz3 = Math.pow(Math.pow( daljica[2]-daljica2[2],2 ) +
                                Math.pow( daljica[3]-daljica2[3],2 ),0.5);

                double raz4 = Math.pow(Math.pow( daljica[0]-daljica2[2],2 ) +
                                Math.pow( daljica[1]-daljica2[3],2 ),0.5);

                double raz[] = {raz1,raz2,raz3,raz4};

         double min = Double.MAX_VALUE;
         for (int i = 0; i < raz.length; i++) {
             if (raz[i] < min)
                min = raz[i];
         }

                return min;
        }
}
```

Površina poligona

Matlab

```
% povrsina.h
%
% Funkcija, ki izračuna površino poligona.
% Funkcija vrne vrednost (površino).
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | povrsinaPoligona(A);                                 |
% |                                                      |
% | A = (x1,y1;   --koordinate točk poligona             |
% |      x2,y2;     v smeri urinega kazalca              |
% |        :                                             |
% |      xn, yn)                                         |
% |                                                      |
```

```
% | ;  -- ne prikaže vrnjene vrednosti                  |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function area = povrsinaPoligona(A)

B = [A;A(1,1),A(1,2)];
[n,m] = size(A);

for i=1:n
    scatter(A(i,1),A(i,2)), hold all,
    Graphx=[B(i+1,1) B(i,1)];
    Graphy=[B(i+1,2) B(i,2)];

    plot(Graphx,Graphy,'--rs','LineWidth',2,'MarkerEdgeColor','k',...
                'MarkerFaceColor','g','MarkerSize',5)
    axis('equal');
    axis('off');
end

C = [A(n,1),A(n,2);B];

area = 0;

for i=2:n+1
    area = area + C(i,2)*(C(i+1,1)-C(i-1,1));
end

area = area/2;
```

Java

```
/*  double povrsinaPoligona(double[] poligon)
 *
 *  Funkcija sprejme niz koordinat poligona. Koordinate morajo
 *  biti podane kot si sledijo v smeri urinega kazalca.
 *
 *  Funkcija vrne vrednost (povrsino poligona).
 *
 *  Jure Kop, geo-uni, 2007
 */

public static double povrsinaPoligona (double[] poligon) {

        double povrsina = 0;
        int vel = poligon.length;
        double[] tPoligon = new double[vel+4];
        for(int i=0; i<vel; i++) {
                tPoligon[0]     = poligon[vel-2];
                tPoligon[1]     = poligon[vel-1];
                tPoligon[i+2]   = poligon[i];
                tPoligon[vel+2] = poligon[0];
                tPoligon[vel+3] = poligon[1];
        }

        for(int i=1;i<(vel+1)/2; i++)
                povrsina = povrsina + tPoligon[2*i+1]*(tPoligon[2*i+2]-tPoligon[2*i-2]);

        return povrsina/2;
}
```

Funkcija ugotavljanja strani

Matlab

```
% stran.h
%
% Funkcija, ki preveri, na kateri strani linije leži točka.
% Funkcija vrne 0 v primeru kolinearnosti, 1 v primeru, ko je točka na
% desni strani linije ter -1, ko je točka na levi stani linije,
% gledano s prve proti drugi točki.
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | stran((x1,y1,x2,y2,x3,y3);                           |
% |                                                      |
% | x1,y1,x2,y2  --koordinate daljice                    |
% | x3,y3        --koordinati toèke                      |
% | ;  -- ne prikaže vrnjene vrednosti                   |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function test = stran(x1,y1,x2,y2,x3,y3)

vsota = 0.5 * (y1*(x2-x3) + y2 * (x3-x1) + y3 * (x1-x2));
    if (vsota > 0)
        test = 1;
        else if (vsota<0)
        test = -1;
        else
        test = 0;
    end
end
```

Java

```
/*  int stran(double[] daljica, double[] tocka)
 *
 *  Funkcija sprejme niz koordinat tock daljice ter niz
 *  koordinat ene tocke. Ce sta v nizu vec kot dva para koordinat,
 *  potem funkcija vzame samo prva dva.
 *
 *  Funkcija vrne vrednost 1, ce lezi tocka na desni strani daljice,
 *  vrednost -1, ce lezi na levi strani daljice ter vrednost 0,
 *  ce so tocke kolinearne.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static int stran (double[] daljica, double[] tocka) {
       double vsota = 0.5 * (daljica[1]*(daljica[2]-tocka[0]) + daljica[3] * (tocka[0]-
daljica[0]) +
                     tocka[1] * (daljica[0]-daljica[2]));

       if (vsota > 0)
              return 1;
       else if (vsota < 0)
              return -1;
       else
              return 0;
}
```

Presek daljic

Matlab

```
% presekDaljic.h
%
% Funkcija, ki preveri, ali se podani dve daljici sekata.
% Funkcija vrne vrednost 1, če se daljici sekata ter
% vrednost 0, če se ne sekata.
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | presekDaljic(A,B);                                   |
% |                                                      |
% | A = (x1,y1,x2,y2)  --koordinate prve daljice         |
% | B = (x3,y3,x4,y4)  --koordinate druge daljice        |
% | ;  -- ne prikaže vrnjene vrednosti                   |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007


function test = presekDaljic(A,B)
clf

a = [A(1),A(3)]; b = [A(2),A(4)];
c = [B(1),B(3)]; d = [B(2),B(4)];

line(a,b,'LineWidth',2,'LineStyle',':'), hold on, axis equal,
line(c,d,'LineWidth',2)

if (    (stran(A(1),A(2),A(3),A(4),B(1),B(2)) ~= ...
         stran(A(1),A(2),A(3),A(4),B(3),B(4))) & ...
        (stran(B(1),B(2),B(3),B(4),A(1),A(2)) ~= ...
         stran(B(1),B(2),B(3),B(4),A(3),A(4)))   );

    disp('daljici se sekata');
    test = 1;
else
    disp('daljici se NE sekata');
    test = 0;
end
end
```

Java

```
/*  int presekDaljic (double[] daljica, double[] daljica2)
 *
 *  Funkcija sprejme dva niza koordinat tock dveh daljic.
 *  Ce sta v nizu vec kot dva para koordinat, potem funkcija
 *  vzame samo prva dva.
 *
 *  Funkcija vrne vrednost 1, ce se daljici sekata, ter
 *  vrednost 0, če se daljici ne sekata.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static int presekDaljic (double[] daljica, double[] daljica2) {

        double[] toc1 = {daljica2[0], daljica2[1]};
        double[] toc2 = {daljica2[2], daljica2[3]};
        double[] toc3 = {daljica[0],  daljica[1]};
        double[] toc4 = {daljica[2],  daljica[3]};

        if ( (stran(daljica, toc1) != stran(daljica,toc2)) &&
                (stran(daljica2,toc3) != stran(daljica2,toc4)) )
                return 1;
        else
                return 0;
}
```

Točka v poligonu

Matlab

```
% tvp.h
%
% Funkcija, ki preveri, ali je točka v poligonu.
% Funkcija vrne vrednost 1, če točka leži v poligonu,
% ter vrednost 0, če leži zunaj oz. na meji poligona.
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | tvp(A,x,y);                                          |
% |                                                      |
% | A = (x1,y1;   --koordinate točk poligona             |
% |      x2,y2;                                          |
% |        :                                             |
% |      xn, yn)                                         |
% | x,y           --koordinati točke                     |
% | ;  -- ne prikaže vrnjene vrednosti                   |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function test = tvp(A,x,y)
clf

n = size(A,1);
B = [A;A(1,1),A(1,2)];

for i=1:n
    scatter(A(i,1),A(i,2)), hold all,

    Graphx=[B(i+1,1) B(i,1)];
    Graphy=[B(i+1,2) B(i,2)];

    plot(Graphx,Graphy,'--rs','LineWidth',2,'MarkerEdgeColor','k',...
                'MarkerFaceColor','g','MarkerSize',10)
    axis('equal');
    %axis('off');

end

scatter(x,y, 'MarkerEdgeColor','k','MarkerFaceColor','r'), hold all

count = 0;

for i=1:n
    a = [0,y];
    b = [x,y];
    c = [B(i+1,1),B(i+1,2)];
    d = [B(i,1),B(i,2)];

    if(and(stran(a(1),a(2),b(1),b(2),c(1),c(2))~=...
           stran(a(1),a(2),b(1),b(2),d(1),d(2)), ...
           stran(c(1),c(2),d(1),d(2),a(1),a(2))~=...
           stran(c(1),c(2),d(1),d(2),b(1),b(2))));
        count = count + 1;
    end
end

if (mod(count, 2) == 0)
    disp ('Tocka je zunaj poligona!')
    test = 0;
else
    disp('Tocka je znotraj poligona')
    test = 1;
end
```

Java

```
/*  int poligonTocka(double[] poligon, double[] tocka)
 *
 *  Funkcija sprejme niz koordinat poligona. Koordinate morajo
 *  biti podane kot si sledijo v smeri urinega kazalca.
 *  Funkcija sprejme tudi niz koordinat tocke.
 *
 *  Funkcija vrne vrednost 1, ce tocka lezi v poligonu, ter
 *  vrednost 0, ce lezi zunaj oz. na meji poligona.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static int poligonTocka (double[] poligon, double[] tocka) {

        int stevec = 0;
        int vel = poligon.length;
        double[] tPoligon = new double[vel+2];
        for (int i=0; i<vel; i++) {   tPoligon[i] = poligon[i]; }

        tPoligon[vel] = poligon[0];
        tPoligon[vel+1] = poligon[1];

        double min = Double.MAX_VALUE;
       for (int i=0; i < vel/2; i++) {
        if (poligon[2*i+1] < min)
                min = poligon[i];
    }

    for(int i=1; i<vel/2; i++) {

                double a[] = {min-1,tocka[1]};
                double b[] = {tocka[0], tocka[1]};
                double c[] = {tPoligon[2*i],tPoligon[2*i+1]};
                double d[] = {tPoligon[2*i-2], tPoligon[2*i-1]};
                double daljica1[] = {a[0],a[1],b[0],b[1]};
                double daljica2[] = {c[0],c[1],d[0],d[1]};
                double toc1[] = {a[0],a[1]};
                double toc2[] = {b[0],b[1]};
                double toc3[] = {c[0],c[1]};
                double toc4[] = {d[0],d[1]};

                if ( (stran(daljica1, toc3) != stran(daljica1,toc4)) &&
                        (stran(daljica2, toc1) != stran(daljica2,toc2)) )
                       stevec++;
        }

        if (stevec % 2 == 0)
                return 0;
        else
                return 1;
}
```

Konveksna lupina

Matlab

```
% konvlup.h
%
% Funkcija, ki določi točke konveksne lupine.
% Funkcija ne vrača vrednosti.
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | konvlup(A)                                           |
% |                                                      |
% | A = (x1,y1;   --koordinate točk                      |
% |      x2,y2;                                          |
% |        :                                             |
% |      xn, yn)                                         |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function konvlup(B)

clc, clf
n=size(B,1);
A=B;

P=[];

lupina = [];
s = 1;

for i=1:n
    for j=1:n
        P(s,1) = i;
        P(s,2) = j;
        P(s,3) = 0;
        s = s+1;
    end
end

v = size(P,1);
PP = [];
vv = 1;
for i = 1:v
    PP = [PP; A(P(vv,1),1)  A(P(vv,1),2)  A(P(vv,2),1)  A(P(vv,2),2) ];
    vv = vv+1;
end

vv = 1;
CC = [];
KK = [];
testt = 1;

for i = 1:v
   nn = 1;
   testt=1;
   stiri = 0;
   for j = 1:n
       c =  stran(PP(vv,1), PP(vv,2), PP(vv,3), PP(vv,4), A(nn,1), A(nn,2));
       CC = [CC;c];

       if (c < 0)
           testt=0;
       end
       if (c==0)
           stiri = stiri+1;
       end
        nn = nn+1;
       %disp(sprintf('%d, %d, %d',c,testt, stiri));
   end
```

```
   if and((testt ~= 0),(stiri ~= n))
      % disp('DA');
       KK = [KK;PP(vv,1), PP(vv,2), PP(vv,3), PP(vv,4) ] ;
   end

  % disp('---');
   vv = vv + 1;
end

%za linije
KK;
k = size(KK,1);
kkk= 1;
for i = 1:k
    graphx = [KK(kkk,1)  KK(kkk,3)];
    graphy = [KK(kkk,2)  KK(kkk,4)];
    kkk = kkk+1;
    %axis('equal');
    plot(graphx,graphy,'--rs','LineWidth',2,'MarkerEdgeColor','k',...
              'MarkerFaceColor','g','MarkerSize',10), hold on, axis equal

end
X = B(:,1);
Y = B(:,2);
scatter(X,Y,'o','filled');

KoordTock = [KK(:,1),KK(:,2)]
end
```

Java

```
/*  void konveksnaLupina(double[] nizKoordinat )
 *
 *  Funkcija sprejme niz koordinat. Nato iz teh koordinat določi
 *  koordinate točk konveksne lupine. Podane točke med seboj
 *  ne smejo biti kolinearne.
 *
 *  Funkcija ne vrača vrednosti.
 *
 *  Jure Kop, geo-uni, 2007
 */
public static void konveksnaLupina(double[] nizKoordinat) {

        int count;
        int n = nizKoordinat.length;
        double[] daljica = new double[4];
        double[] tocka = new double[2];

        for (int i=0; i<(n/2); i++) {
                for (int j=0; j<(n/2); j++) {
                        count = 0;
                        for (int k=0; k<(n/2); k++) {
                                if((k!=i)&&(k!=j)) {

                                        daljica[0] = nizKoordinat[2*i];
                                        daljica[1] = nizKoordinat[2*i+1];
                                        daljica[2] = nizKoordinat[2*j];
                                        daljica[3] = nizKoordinat[2*j+1];
                                        tocka[0]   = nizKoordinat[2*k];
                                        tocka[1]   = nizKoordinat[2*k+1];

                                        int str = stran(daljica, tocka);

                                        if (str!=-1) {
                                                count = 0;
                                                break;
                                        }
                                        else if(str==-1)
```

```
                                        count++;

                                    if (count==(n/2)-2) {

                                        System.out.printf("Daljica: " + daljica[0] + " ");
                                        System.out.printf(" " + daljica[1] + "  -> ");
                                        System.out.printf(" " + daljica[2] + " ");
                                        System.out.printf(" " + daljica[3] + "\n");
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
```

PRILOGA B: RASTRSKI ALGORITMI - IZVIRNE KODE

Lokalne operacije

Matlab

```
% rasLokalni.h
%
% Funkcija, ki vsak slikovni element izvirne podobe pomnoži
% z danim faktorjem.
% Funkcija ne vrača vrednosti.
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | rasLokalni('podoba.format', faktor)                  |
% |                                                      |
% | podoba       --ime podobe                            |
% | format       --format podobe                         |
% | faktor       --željeni faktor                        |
%  ------------------------------------------------------
%                               Jure Kop, geo-uni, 2007

function rasLokalni(ime, faktor)
clc,clf

a = imread(ime);
size(a)
image(a), hold

vrst  = size(a,1); stolp = size(a,2); glob  = size(a,3)

b = double(a(:,:,:));
d = double(a);

for i=1:vrst
    for j=1:stolp
        b(i,j,1) = d(i,j,1) * faktor;
        b(i,j,2) = d(i,j,2) * faktor;
        b(i,j,3) = d(i,j,3) * faktor;
    end
end

ima = uint8(b);
figure(2), image(ima);

end
```

Java

```
/*  double[][] rasterLokalni(int[][] matrika, double faktor)
 *
 *  Funkcija opravi lokalno rastrsko operacijo,
 *  ter vrne popravljeno podobo v obliki nove matrike.
 *
 *  Vhodna matrika mora biti kvadratna.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static double[][] rasterLokalni(int[][] matrika, double faktor) {

        int vel = matrika.length;
        double nMatrika[][] = new double[vel][vel];
        for(int i=0; i<vel;i++)
                for(int j=0; j<vel; j++)
                        nMatrika[i][j] = matrika[i][j] * faktor;

        return nMatrika;
}
```

Regionalne operacije

Matlab

```
% rasRegionalni.h
%
% Funkcija, ki skozi celotno podobo požene dani filter
% dimenzije 3x3.
% Funkcija ne vraèa vrednosti.
%  ------------------------------------------------------
% | Navodila za uporabo:                                 |
% |                                                      |
% | rasRegionalni('podoba.format', filter)               |
% |                                                      |
% | podoba       --ime podobe                            |
% | format       --format podobe                         |
% | filter       --željeni filter dimenzije 3x3          |
%  ------------------------------------------------------
%                                Jure Kop, geo-uni, 2007

function rasRegionalni(ime, filter)
clc,clf

a = imread('owl.jpg');
size(a);
image(a), hold;

vrst  = size(a,1);
stolp = size(a,2);
glob  = size(1,3);

d = double(a);

for i = 1:vrst-2
    for j = 1:stolp-2
        for k = 1:glob
        tempM1 = [d(i,j,k), d(i,j+1,k), d(i,j+2,k);
                  d(i+1,j,k), d(i+1,j+1,k), d(i+1,j+2,k);
                  d(i+1,j,k), d(i+1,j+1,k), d(i+1,j+1,k)];

        novaM(i,j,k) = filter(1,1)*tempM1(1,1)+...
                       filter(1,2)*tempM1(1,2)+...
                       filter(1,3)*tempM1(1,3)+...
                       filter(2,1)*tempM1(2,1)+...
                       filter(2,2)*tempM1(2,2)+...
                       filter(2,3)*tempM1(2,3)+...
                       filter(3,1)*tempM1(3,1)+...
                       filter(3,2)*tempM1(3,2)+...
                       filter(3,3)*tempM1(3,3);

            end
     end
end

ima = uint8(novaM);
figure(2), image(ima);
```

Java

```
/*  double[][] rasterRegionalni(int[][] matrika, int[][] filter)
 *
 *  Funkcija opravi regionalno rastrsko operacijo,
 *  ter vrne popravljeno podobo v obliki nove matrike.
 *
 *  Vhodna matrika mora biti kvadratna, filter pa dimenzije 3x3.
 *
 *  Jure Kop, geo-uni, 2007
 */

public static double[][] rasterRegionalni(int[][] matrika, int[][] filter) {

        int vel = matrika.length;
        double nMatrika[][] = new double[vel-2][vel-2];

        for(int i=0; i<vel-2; i++) {
                for(int j=0; j<vel-2; j++) {
                                nMatrika[i][j] =  matrika[i][j]     * filter[0][0] +
                                                  matrika[i][j+1]   * filter[0][1] +
                                                  matrika[i][j+2]   * filter[0][2] +
                                                  matrika[i+1][j]   * filter[1][0] +
                                                  matrika[i+1][j+1] * filter[1][1] +
                                                  matrika[i+1][j+2] * filter[1][2] +
                                                  matrika[i+2][j]   * filter[2][0] +
                                                  matrika[i+2][j+1] * filter[2][1] +
                                                  matrika[i+2][j+2] * filter[2][2];
                }
        }
        return nMatrika;
}
```

PRILOGA C: WIKI PRIROČNIK

Besedila

Pri navadnem pisanju besedila ne potrebujemo posebnega predznanja, saj besedilo preprosto napišemo, kot bi pisali v katerem koli urejevalniku besedil. Edino, kar se pri urejevanju besedila razlikuje od drugih urejevalnikov, je prelom vrstice. Nova vrstca v urejevalniku še ne pomeni končen prelom vrstice. To storimo z dvojnim prelomom vrstice ali s posebnim znakom.

Slike

Če je slika dostopna kjer koli na spletu, jo lahko preprosto dodamo tako, da v oglate oklepaje vstavimo naslov te slike.

Primer: `[www.tam_kjer_je_slika.org/slika.png]`

Nadomestno besedilo lahko nastavimo takrat, ko uporabnik noče ali ne more videti slike.

Primer: `[tu je slika| www.tam_kjer_je_slika.org/slika.png]`

Sliko lahko dodamo k besedilo tudi takrat, ko je bila kot priponka dodana h kakšni drugi strani.

Primer: `[Domov/slika.png]` doda sliko z imenom slika.png na stran, ki jo trenutno urejamo. Slika.png mora biti pripeta k strani Domov.

Lahko pa uporabimo tudi naslednji ukaz.

[{Image src='ime_podobe.končnica' caption = 'Napis, ki se pojavi pod sliko.' align = 'center'}], pri čemer so podprti vsi standardni formati, kot so JPG, BMP, GIF ter PNG.

Javanski programčki

Javanske programčke, ki smo jih prej pripeli določeni strani, lahko prikličemo z ukazom:

[{Applet code='Razred' archive='Datoteka.jar']

pri čemer morajo biti vse datoteke shranjene v arhivski datoteki s končnico jar. Kot atribut jim lahko dodamo širino ter višino.

Naslavljanje

Poznamo tri velikosti naslavljanj, majhno, srednje ter veliko, ki jih določimo tako, da pred besedilo vstavimo en, dva oz. tri klicaje. Vsako naslavljanje ustvari tudi t.i. imensko sidro, na katerega se lahko sklicujemo kjer koli iz besedila.

Označevanje besedil

Z uporabo zvezdice (*) pred besedilom dodamu le-temu označbo, piko. Za globlje nivoje uporabimo več zvezdic. *Primer:*

```
* Ena
* Dva
* Tri
** Tri.Ena
```

Pri čemer dobimo:

- Ena
- Dva
- Tri
  - Tri.Ena

Oštevilčena besedila

Z uporabo lojtre (#) besedilo oštevilčimo. *Primer*: zapis

```
# Ena
# Dva
# Tri
## Tri.Ena
```

ustvari:

1. Ena
2. Dva
3. Tri
    1. Tri.Ena

Če želimo besedilo napisati v več vrsticah, moramo pri naslednji vrstici dodati najprej presledek. Tako se besedilo v novi vrstici samodejno doda oštevilčenemu besedilu.

Definicije

Definicije zapišemo z uporabo podpičja ter dvopičja.

;__Java__:"Baje nek otok oz. programski jezik."

Oblikovanje besedila

Za krepko pisavo uporabimo dva podvezaja (_) na vsaki strani besedila. Za poševno pisavo na vsako stran besedilo vstavimo dva apostrofa (').

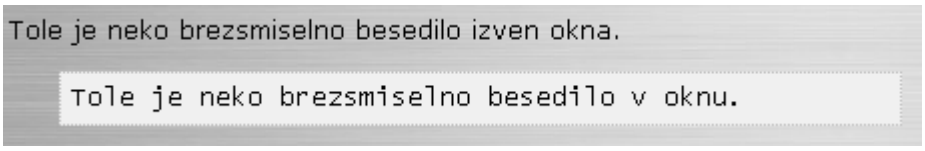
__krepko__ **krepko**

''poševno'' *poševno*

Predoblikovano besedilo

Če želimo besedilo izpisati v posebnem oknu, kot na primer programsko kodo, potem ga vstavimo med tri zavite oklepaje. Primer:

```
Tole je neko brezsmiselno besedilo izven okna.
{{{
Tole je neko brezsmiselno besedilo v oknu.
}}}
```



Povezave

Na drugo stran znotraj wiki okolja se povežemo tako, da med oglate oklepaje vstavimo ime strani, na katero se želimo povezati, npr. [Domov]. Če želimo tej povezavi dodati besedilo,

lahko to storimo takole: [Klik|Domov]. Pri tem se povežemo na stran *Domov*, besedilo, ki se izpiše, je *Klik*.
Na tuje strani se povežemo tako, da v oglate oklepaje napišemo celoten naslov te strani, [http://java.sun.com]. Podprti protokoli so http:, ftp:, mailto:, https: ter news:.

Opombe

Opombe so sestavljene iz dveh delov. Prvi del, ime opombe v oglatem oklepaju [1], ustvari povezavo. Drugi del, lojtro (#) z imenom opombe v oglatem oklepaju [#1], pa dodamo dejanskemu besedilu opombe, ponavadi na dnu strani.

Tabele

Tabele ustvarimo z uporabo enojnih oz. dvojnih pokončnih črt (|). Dvojne uporabimo takrat, ko želimo določeno vrstico poudariti, npr. prvo vrstico. Tabelo preprosto zaključimo tako, da ustvarimo prazno vrstico, ki pa je brez pokončne črte.

Primer:

```
|| Oseba || Atribut 1 || Atribut 2
| ''Jure'' | __Nekaj__ | Haha
| [DomacaStran] | NekajDrugega | Hihi
```

Dobimo:

Oseba	Atribut 1	Atribut 2
Jure	**Nekaj**	Haha
DomacaStran	NekajDrugega	Hihi

Komentarji

Z uporabo stavka *%%commentbox <besedilo> %%* lahko ustvarimo dodaten okvir s komentarji, ki prispevajo k boljši preglednosti strani. Za sam izgled okvirja skrbi datoteka jspwiki.css. Spodaj je prikazan primer uporabe komentarja.

Konflikti

Ob hkratnem urejevanju iste strani bo JSPWiki tistemu, ki bo stran začel prvi urejati, dovolil, da jo tudi shrani, drugemu uporabniku pa bo pokazal konfliktno stran. Konfliktno stran dobimo tudi, če se z gumbom *nazaj* (back) v našem brskalniku želimo vrniti na stran za urejanje, saj brskalnik še kar misli, da ureja prejšno stran (slika 9).

Brisanje strani

Stran lahko izbrišemo tako, da izbrišemo njeno vsebino ter povezavo, ki kaže na to stran. Ta stran bo sicer še vedno obstajala na matičnem računalniku, vendar pa bo očem javnosti nevidna. Prav tako je možno, da za bolj občutljive strani administrator poskrbi, da so zaščitene proti brisanju.

Dodajanje novih strani

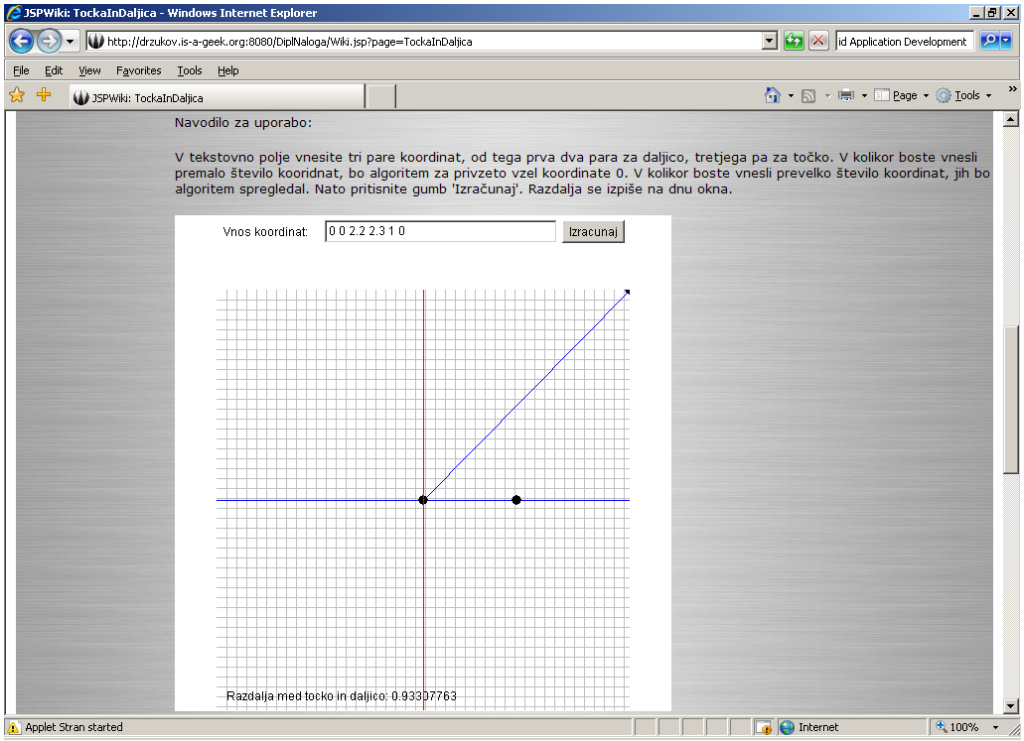
Novo stran dodamo preprosto tako, da ustvarimo povezavo na to stran (ime strani v oglatih oklepajih) ter nanjo kliknemo. Ko bomo prvič kliknili na to povezavo, nas bo program samodejno preusmeril na stran za urejanje.

PRILOGA D: WIKI.JSP - IZVIRNA KODA

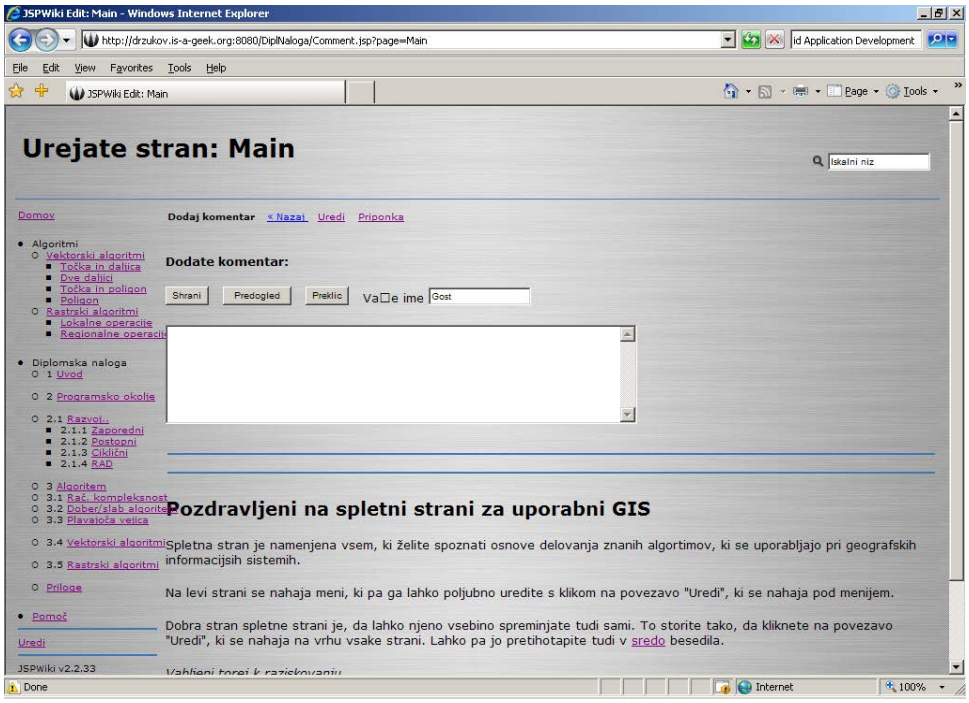
```
<%@ page import="org.apache.log4j.*" %>
<%@ page import="com.ecyrd.jspwiki.*" %>
<%@ page import="com.ecyrd.jspwiki.tags.WikiTagBase" %>
<%@ page import="com.ecyrd.jspwiki.auth.permissions.ViewPermission" %>
<%@ page import="com.ecyrd.jspwiki.auth.*" %>
<%@ page errorPage="/Error.jsp" %>
<%@ taglib uri="/WEB-INF/jspwiki.tld" prefix="wiki" %>
<%!
    public void jspInit()
    {
        wiki = WikiEngine.getInstance( getServletConfig() );
    }
    Category log = Category.getInstance("JSPWiki");
    WikiEngine wiki;

%><%

    WikiContext wikiContext = wiki.createContext( request, WikiContext.VIEW );
    String pagereq = wikiContext.getPage().getName();

    NDC.push( wiki.getApplicationName()+":"+pagereq );

    log.info("Request for page '"+pagereq+"' from "+request.getRemoteAddr()+" by
"+request.getRemoteUser() );

    String redirect = wiki.getRedirectURL( wikiContext );

    if( redirect != null )
    {
        response.sendRedirect( redirect );
        return;
    }

    AuthorizationManager mgr = wiki.getAuthorizationManager();
    UserProfile currentUser  = wiki.getUserManager().getUserProfile( request );

    if( !mgr.checkPermission( wikiContext.getPage(),
                              currentUser,
                              new ViewPermission() ) )
    {
        if( mgr.strictLogins() )
        {
            log.info("User "+currentUser.getName()+" has no access - redirecting to login
page.");
            String msg = "Unknown user or password.<br>Please try again.";
            session.setAttribute( "msg", msg );
            String pageurl = wiki.encodeName( pagereq );
            response.sendRedirect( wiki.getBaseURL()+"Login.jsp?page="+pageurl );
            return;
        }
        else
        {
            //
            //  Do a bit of sanity check here.  FIXME: Should this be somewhere else?
            //
            if( pagereq.equals("LoginError") )
            {
                throw new WikiSecurityException("Looped config detected - you must not prevent
view access to page LoginError AND have strictLogins set to true!");
            }

            log.info("User "+currentUser.getName()+" has no access - displaying message.");
            response.sendRedirect( wiki.getViewURL("LoginError") );
        }
    }

    pageContext.setAttribute( WikiTagBase.ATTR_CONTEXT,
                              wikiContext,
                              PageContext.REQUEST_SCOPE );
```

```
    //
    //  Alright, then start responding.
    //

    response.setContentType("text/html; charset="+wiki.getContentEncoding() );

    String contentPage = wiki.getTemplateManager().findJSP( pageContext,
                                                            wikiContext.getTemplate(),
                                                            "ViewTemplate.jsp" );

%><wiki:Include page="<%=contentPage%>" /><%

    NDC.pop();
    NDC.remove();
%>
```

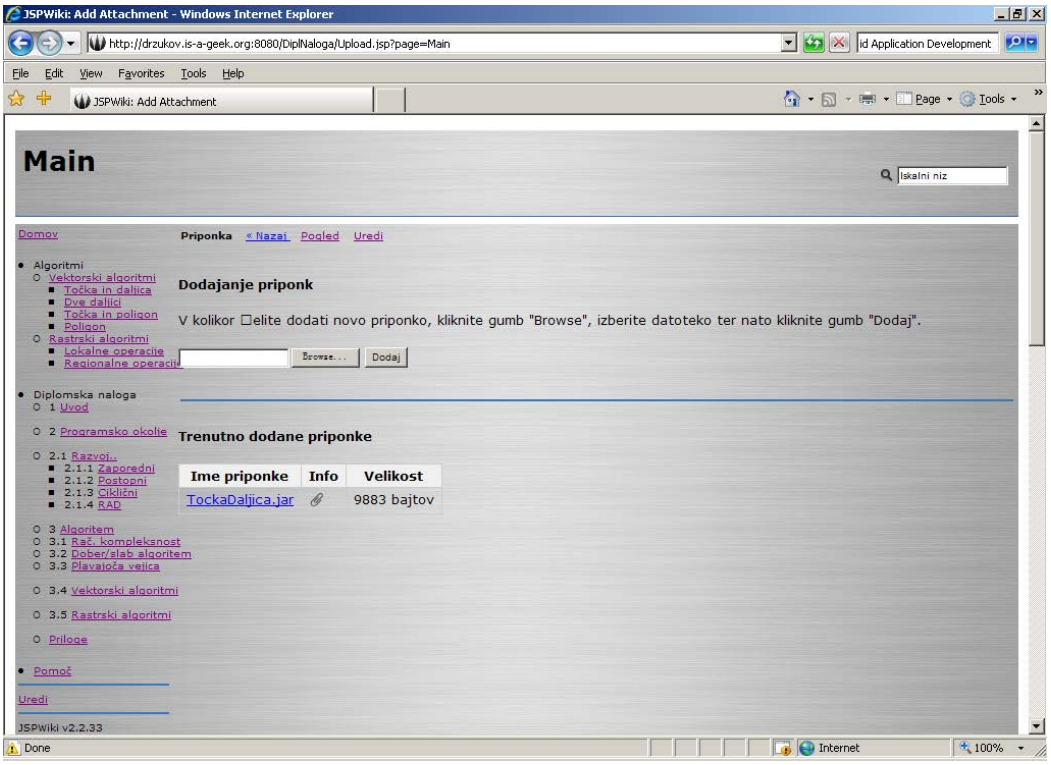
PRILOGA E: JAVANSKI PROGRAMČEK PRI IZVAJANJU



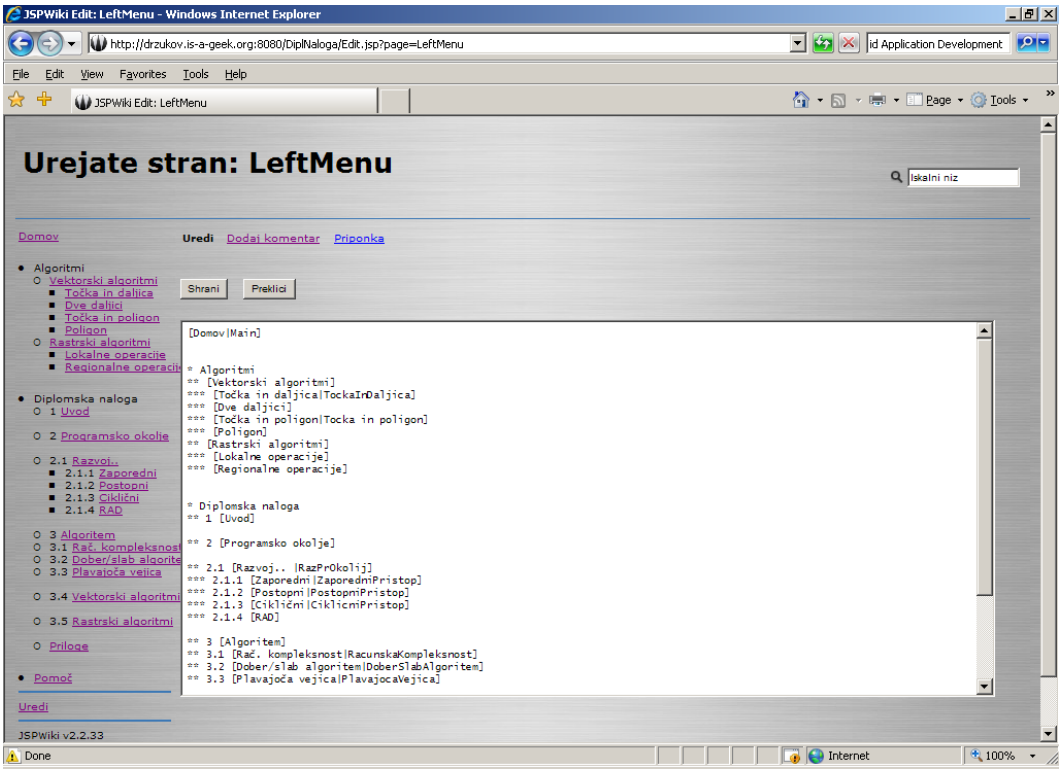
PRILOGA F: DODAJANJE KOMENTARJA



PRILOGA G: DODAJANJE PRIPONK



PRILOGA H: UREJANJE GLAVNEGA KAZALA



PRILOGA I: ISKANJE NIZA