UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Miha Zorc

Razvoj programa za pregledovanje arheoloških podatkov na risbah v programu AutoCAD

Diplomska naloga

Ljubljana, 2007

Zahvala

Zahvaljujem se vsem, ki so kakor koli pripomogli k implementaciji moje diplomske naloge. Iskreno se zahvaljujem za potrpljenje pri popravljanju napak mentorju mag. Matiji Lokarju. Zahvala našemu šefu Tikotu za vse tehnične pripomočke, Rokotu, ki mi je podal prve napotke pri delu z AutoCAD-om ter Evici za strpno prevajanje tujih besedil ter moralni podpori. Zahvaljujem se tudi staršem za vso pomoč v času študija.

KAZALO

Program dela

V diplomski nalogi opišite, kako je potekal razvoj programa za pregledovanje podatkov, ki so del risb narejenih s programom AutoCAD. Na kratko opišite tudi tehnologijo za razvoj aplikacije, torej sam program AutoCAD in programski jezik Visual Basic for Applications.

Mentor:
mag. Matija Lokar

Povzetek

V arheologiji se pri dokumentiranju zaradi množice podatkov za pridobivanje in obdelavo leteh uporablja računalnik. Kot najbolj učinkovit program za obdelavo se je izkazal program AutoCAD. Risba v AutoCAD-u je sestavljena iz objektov, ki se jih preko orodne vrstice nanaša na risalno površino. Objekt je lahko črta, lomljenka, točka ... AutoCAD ima vgrajena dva programska jezika, ki sta namenjena izključno programiranju v AutoCAD-u. Eden med njima je Visual Basic for Application (VBA).
Za lažje pregledovanje podatkov na risbah v AutoCADu, sem razvil program MiniExplorer. Napisan je v programskem jeziku VBA za AutoCAD in namenjen pregledovanju in urejanju arheološke dokumentacije. Program je sestavljen iz več logičnih enot, ki so nastale s sodelovanjem arheologov.

Math. Subj. Class. (2000): 68N15, 68P10, 68U15
Computing Review Class. System (1998): D.1.5, D.1.1, D.1.7, D.3.3, E.0, E.1, E.5, J.6, J.5

Ključne besede: arheologija, dokumentacija, AutoCAD, Visual Basic, objekti, gradniki, MiniExplorer, grafični vmesnik, programska koda

Keywords: archaeology, documentation, AutoCAD, Visual Basic, objects, controls, MiniExplorer, graphic interface, program code

1. UVOD

Že vrsto let se ukvarjam z obdelavo podatkov pri arheoloških izkopavanjih. Gre predvsem za natančno dokumentiranje lokacij nahajališč posameznih najdb. Zato obdelava podatkov poteka s programom AutoCAD, ki je eden boljših programov za natančno tehnično risanje. Sprva je delo s programom AutoCAD predstavljalo precejšen zalogaj spoznavanja novih možnosti. Z leti pa je delo postalo rutina. Podatkov, ki jih je bilo potrebno voditi na risbah, je bilo vedno več, zato je tudi obdelava postala kompleksnejša. Porajalo se je vprašanje, ali lahko vnos teh podatkov in njihovo kasnejše pregledovanje olajšamo. Odgovor se je skrival v vgrajenem programskem jeziku.

Pri študiju smo precej časa posvetili programskim jezikom in delu z njimi. Zato sem se odločil, da preizkusim svoje znanje programiranja. Nastal je projekt Mini Explorer. Sprva je bil načrt, da bi bil to program, ki bi le olajšal pregledovanje že obdelanih podatkov. Sčasoma je program postal veliko več. Na začetku je program vseboval samo nekaj preprostih funkcij za pohitritev uporabe nekaterih orodij v AutoCADu. Skozi razvoj programa so se porajale vedno nove možnosti, ki so nudile enostavnejše in hitrejše rešitve. Program je dobival vedno več funkcij. Razvit je bil v programskem jeziku Visual Basic for Application, ki je del AutoCAD-a. Sicer v času študija tega jezika nismo spoznali (ukvarjali smo se predvsem s programskim jezikom Java), a mi zaradi dobrega poznavanja Jave, sam prehod na nov jezik ni delal večjih težav.

Diplomska naloga je v grobem razdeljena na dva dela. V prvem delu je kratka predstavitev načina pridobivanja podatkov in opis njihove obdelave. Prav tako so predstavljene osnovne značilnosti in način dela z AutoCAD-om in Visual Basic-om. V drugem delu je predstavljen projekt Mini Explorer. V nalogi bom predstavil samo nekatere ključne programske prijeme, ki sem jih uporabil pri razvoju programa Mini Explorer, saj bi bil opis razvoja celotnega programa preobsežen.

2. ZAJEM IN OBDELAVA PODATKOV

V tem razdelku bom skušal na kratko razložiti, kako poteka zajem podatkov na arheoloških izkopavanjih, kako se jih obdela ter kako lahko delo pospešimo. Spoznali bomo, kako poteka zajem podatkov na arheološkem terenu ter kako te podatke vnesemo v računalnik. Nato bomo spoznali program AutoCAD 2006, v katerem se ti podatki obdelajo. V naslednjem razdelku pa bomo spoznali še Visual Basic for Applications, tako sam jezik, kot tudi razvojno okolje, ki je vgrajeno v program AutoCAD.

2.1 Princip dela na arheoloških izkopavanjih

2.1.1 Mreža kvadrantov

Za potrebe dokumentiranja se na arheološkem najdišču vzpostavi relativni koordinatni sistem z nekim fiksnim izhodiščem glede na absolutni koordinatni sistem (Gauss – Krueger).[1] Nato se zaradi lažje orientacije na terenu, po celem arheološkem najdišču vzpostavi mreža kvadrantov velikosti 5 x 5 metrov. Po abcisni osi jih označimo s številkami, po ordinati pa s črkami.

[1] Absolutni koordinatni sistem po Gauss – Krueger-ju je koordinatni sistem, ki površino Zemlje razdeli na določene cone z pripadajočimi koordinatami.

2.1.2 Dokumentiranje

Pri arheoloških izkopavanjih se ukvarjamo z dokumentiranjem arheoloških podatkov. Pri pridobivanju in obdelavi podatkov arheologi uporabljajo tako imenovano stratigrafsko enoto (v nadaljevanju SE). To je enota, ki omogoča arheologom določevanje položaja in odnosov med SE v nekem stratigrafskem depozitu. Stratigrafski depozit je zaporedje dogodkov, ki so se zvrstili na nekem prostoru. SE je lahko jama, polnilo jame, nasip, jarek, del zidu, skelet, grob, posamezna zemeljena plast, skupek razbitih posod itd. Na podlagi stratigrafije nekega prostora se lahko datirajo posamezne najdbe in gradbene ostaline, ki pripadajo istemu časovnemu obdobju.

Zajem podatkov poteka s pomočjo geodetske naprave, ki se imenuje totalna postaja. To je naprava, ki nam zelo natančno izmeri koordinate poljubnih točk v prostoru. Le te so merjene glede na izhodiščno točko, kjer je postavljena totalna postaja. Izhodiščne točke totalne postaje, tako imenovana stojišča, so znane točke, ki jih dobimo na Geodetski upravi RS. Totalna postaja shranjuje izmerjene koordinate v integriran spomin tako, da se jih lahko kasneje prenese v računalnik.

Vse stratigrafske enote se dokumentirajo opisno, s fotografijami in prostorsko, glede na smernice internega standarda TSS400+. Prostorsko dokumentiranje vsebuje klasično arheološko risbo, uporabo totalne postaje in georeferencirane digitalne fotografije - foto skice (dalje FS), ki se jih dodatno računalniško obdela. Dokumentiranje s totalno postajo zajema izmere obrisov, izohips, višin in profilov arheoloških struktur, izkopnega polja in sond. Tako pridobljene podatke se vnese v računalnik in dodatno interpretira (poveže linije, opremi z oznakami in opombami).

Kompleksnejše SE oziroma strukture se vertikalno posname z digitalnim fotoaparatom. Fotografirano strukturo se pred tem opremi z markerji (fototočkami). Tem markerjem se s totalno postajo izmeri koordinate. Te se nato tlorisno vnese v računalnik, prav tako tudi digitalno fotografijo. Fotografije se nato georeferencira. To pomeni, da z ustreznimi programskimi orodji fototočke na fotografiji poveže z izmerjenimi koordinatami in tako fotografijo umesti v prostor. Strukture na fotografijah (FS) se v naslednji fazi vektorizira in opremili z interpretacijami arheologov na terenu.

2.1.3 Prenos podatkov s totalne postaje

Za prenos podatkov v računalnik se uporablja preprost program Wincomms, ki je na voljo na spletni strani proizvajalca totalne postaje. Tako se v računalnik prenese tekstovna datoteka, v kateri so zabeležene koordinate pobranih podatkov. S programom Prolink, ki ga tudi dobimo na omenjeni spletni strani, podatke izvozimo kot datoteko v formatu AutoCAD. Vsa nadaljna obdelava podatkov poteka v programu AutoCAD 2006, ki ga bomo spoznali v naslednjem razdelku.

2.2 Osnove programa AutoCAD 2006

AutoCAD je najbolj razširjen program za tehnično risanje. V podjetju Autodesk, ki izdeluje program AutoCAD, pravijo, da končnica CAD pomeni *računalniško podprto načrtovanje (computer-aided design)*.

Prva različica AutoCAD-a, ki je delovala še v operacijskem sistemu DOS, je prišla na tržišče leta 1982. AutoCAD je bil prvi program za računalniško podprto risanje, narejen za uporabo na namiznih računalnikih. Takrat je večina drugih programov za tehnično risanje delovala le na visoko zmogljivih delovnih postajah,zato se je hitro uveljavil.

AutoCAD je zelo obsežen program, zato bomo v tem razdelku spoznali samo nekatera orodja, ki se pri obdelavi arheoloških podatkov najpogosteje uporabljajo.

AutoCAD kot osnovne gradnike uporablja objekte. Objekti so točke, črte, pravokotniki ...

Te objekte nanašamo na risalno površino. Ko jih nanesemo na risalno površino, jih lahko premikamo, izbiramo, jim spreminjamo velikost, jih rotiramo itd. Objektom lahko natančno določamo lastnosti kot so koordinate, velikost in podobno.

2.2.1 Prednosti programa AutoCAD

Uspeh AutoCAD-a je v dobršni meri tudi posledica njegove odprte zasnove. Delo s programom s pomočjo podprogramov v različnih programskih jezikih, ki so v AutoCAD-u podprti, lahko avtomatiziramo. Tako lahko uporabljamo programska jezika AutoLISP in Visual Basic for Applications. To sta sicer splošno namenska programska jezika, a prilagojena posebej za programiranje AutoCAD-a z vidika končnega uporabnika.
Zaradi tega je AutoCAD zelo prilagodljiv risarski program. Program omogoča trirazsežno modeliranje in vizualizacijo površine teles, dostop do zunanjih zbirk podatkov, inteligentno kotiranje, uvažanje in izvažanje datotek v različnih oblikah zapisa, podporo internetu in še veliko več. Zaradi vseh omenjenih značilnosti, dobro zasnovane pomoči ter dobre podpore programskim jezikom je bil AutoCAD dobra izbira tudi za delo arheologov.

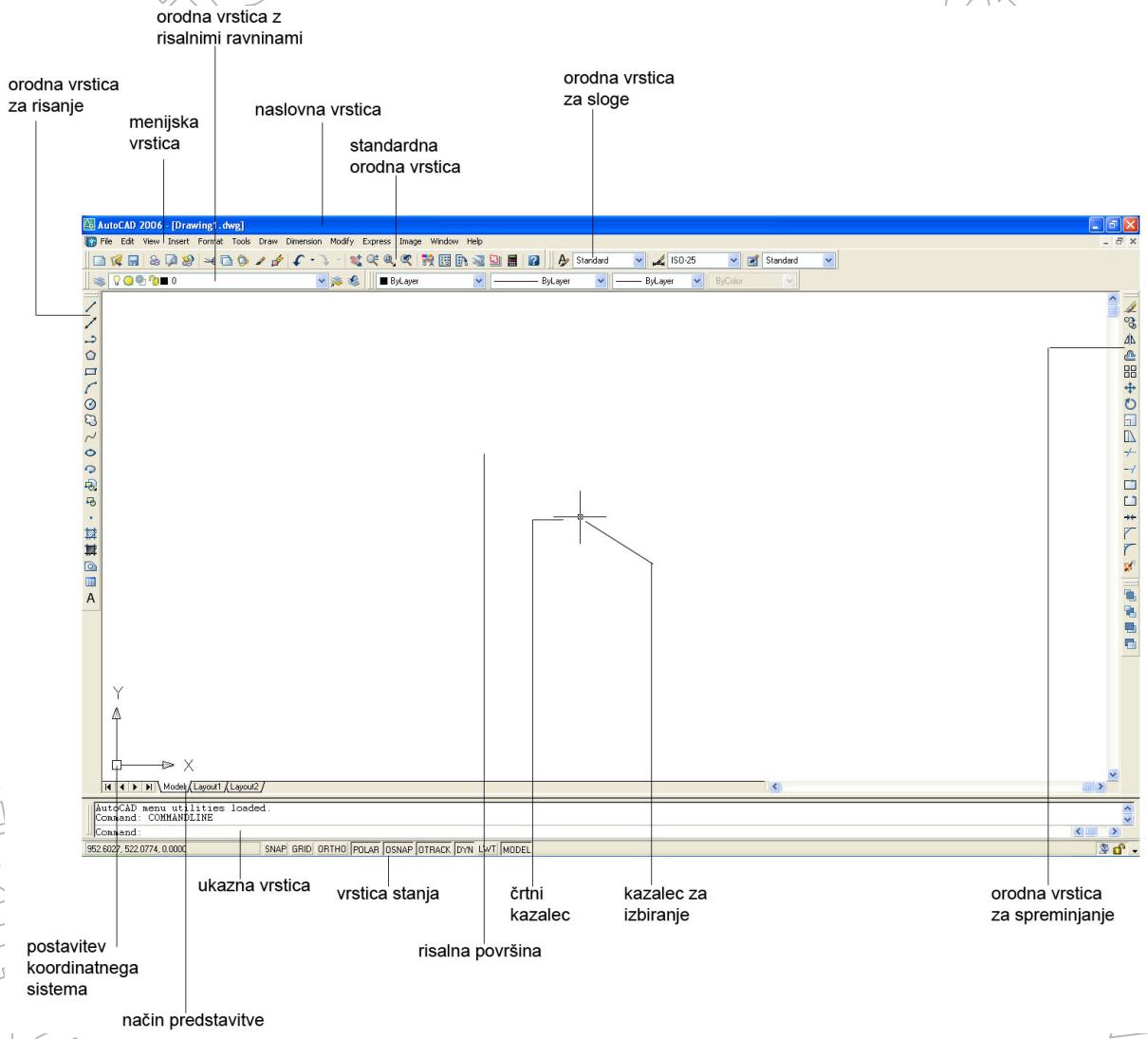
2.2.2 Predstavitev delovnega okolja

Slika 1 prikazuje okno, ki se pojavi ob zagonu AutoCAD-a. Seveda je lahko okno nekoliko drugačno, saj lahko sami določamo, katera orodja hočemo videti na zaslonu in katera ne.
Zaslon AutoCAD-a sestavljajo štiri pomembna območja:

- Risalna površina
- Meniji in orodne vrstice
- Ukazna vrstica
- Vrstica stanja

2.2.2.1 Risalna površina

V AutoCAD-u rišemo na glavno prazno površino na sredini zaslona. Tej površini pravimo *grafično okno* ali *risalna površina* (Slika 1). Risalno površino si lahko predstavljamo kot list skicirnega papirja. Poglavitna razlika med papirjem in računalniško risalno površino je v tem, da je računalniška risalna površina lahko poljubno velika.



Slika 1: Delovno okolje AutoCAD 2006

Ob spodnjem robu risalne površine se nahaja jeziček *Model* (glej način predstavitve na Sliki 1). Rišemo na risalni površini na tem jezičku. Na ostalih dveh jezičkih, ki se imenujeta *Layout*, postavimo risbo za izris. Na njih oblikujemo risbo, ki jo želimo natisniti s tiskalnikom, zrisati z risalnikom ali izvoziti v datoteko v formatu pdf ali jpg.

2.2.2.2 Ikona uporabniškega koordinatnega sistema(UCS)

V levem spodnjem kotu risalne površine lahko opazimo simbol z dvema puščicama (Slika 1). Temu simbolu pravimo ikona uporabniškega koordinatnega sistema (UCS – *User Coordinat System*). Puščici kažeta v pozitivni smeri osi X in Y. Ikona nam pomaga pri orientaciji na risalni površini.

2.2.2.3 Črtni kazalec

Na Sliki 1 lahko na sredi risalne površine opazimo dve prekrižani črti s kvadratkom v sredini. Kvadratku pravimo *kazalec za izbiranje*. Z njim izbiramo objekte na risalni ravnini. Če torej kliknemo, izberemo objekt, na katerem je omenjeni kazalec za izbiranje. Črtama pravimo *črtni kazalec*. Pomaga nam pri lažjem sledenju kazalca za izbiranje po risalni površini. Kazalec za izbiranje in črtni kazalec premikamo z miško. Ob spodnjem delu zaslona, na levi strani vrstice stanja (Slika 1), lahko vidimo koordinati X in Y, ki se spreminjata hkrati s premikanjem miške. Kažeta položaj kazalca za izbiranje na risalni površini. Več o vrstici stanja bomo izvedeli kasneje.

2.2.2.4 Meniji in orodne vrstice

Ob zgornjem robu zaslona je naslovna vrstica, tik pod njo pa menijska vrstica. Pod njima sta dve orodni vrstici. Poleg tega sta na zaslonu še dve orodni vrstici, in sicer *orodja za risanje* (Draw) in *orodja za spreminjanje* (Modify). Ponavadi sta zasidrani na levi in desni strani zaslona (Slika 1). Ukaze v AvtoCAD-u, s katerimi rišemo, urejamo itd., izvajamo prek menijev in orodnih vrstic.

Menije in orodne vrstice je mogoče prilagajati lastnim potrebam, zato je okno lahko videti drugačno kot tisto na Sliki 1. AutoCAD ponuja še veliko več orodnih vrstic. Prikažemo jih, kadar jih potrebujemo. Na voljo so, na primer, orodne vrstice *Dimension* (kotiranje), *View* (pogled), itd.. Nekatera orodja, ki smo jih pri našem delu uporabljali najpogosteje, bomo spoznali kasneje.

2.2.2.5 Ukazna vrstica

V spodnjem delu zaslona je posebno okence, v katerem so navadno prikazane tri vrstice besedila. Vidimo lahko besedo *Command:* (Slika 1). To je *ukazna vrstica*. Vse ukaze, ki jih drugače izvajamo preko menujskega sistema in orodnih vrstic, lahko izvedemo tudi tako, da jih natipkamo v ukazno vrstico. Tudi kadar ukaz izvedemo z izbiro iz menija ali z gumbom iz orodne vrstice, je včasih koristno pogledati v ukazno vrstico, da vidimo, kako se AutoCAD odzove. V ukazno vrstico namreč AutoCAD izpisuje določena obvestila, ki se nanašajo na zadnji izvedeni ukaz.

2.2.2.6 Vrstica stanja

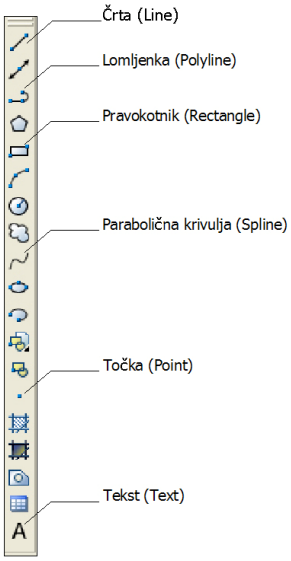
Čisto na dnu zaslona je *vrstica stanja* (Slika 1). Na levi strani sta koordinati X,Y, ki označujeta položaj kazalca za izbiranje. V vrstici stanja so tudi številni gumbi, ki pa jih pri našem delu nismo uporabljali, zato jih tudi ne bom opisoval.

2.2.3 Uporaba nekaterih orodij

V tem razdelku bom predstavil nekatera orodja, ki jih najpogosteje uporabljamo pri delu z AutoCAD-om v arheologiji. Najprej si bomo pogledali orodja iz orodne vrstice za risanje. Nato si bomo bolj podrobno ogledali orodno vrstico z risalnimi ravninami (layer), saj se večina mojega nadaljnega dela nanaša ravno nanje. Na koncu bomo spoznali še nekatera orodja iz orodne vrstice za spreminjanje.

2.2.3.1 Orodja iz orodne vrstice za risanje

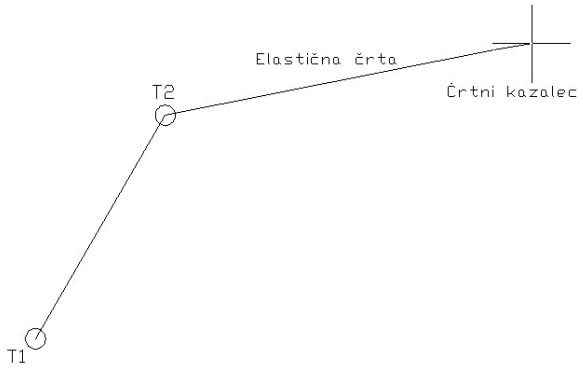
Kot pove že samo ime orodne vrstice, s temi orodji rišemo po risalni površini. Ogledali si bomo risanje preprostih črt ter krivulj, dodajanje besedila, ustvarjanje točk in nekaterih drugih geometrijskih konstrukcij. Vsako orodje predstavlja v AutoCAD-u svoj objekt. Vsak objekt ima svoje lastnosti. Več o objektih, njihovih lastnostih, funkcijah nad njimi in podobno si bomo ogledali v razdelku o VBA.



Slika 2: Orodna vrstica za risanje

Črta (Line)

Z ukazom Line lahko narišemo preprosto črto od ene točke do druge. Ko izberemo prvo točko in premaknemo črtni kazalec do lokacije druge točke, vidimo črto, ki nam kaže, kje bo narisana črta potem, ko določimo drugo točko (glej Sliko 3). Objekti črte imajo dva konca (prvo in zadnjo točko). Če nadaljujemo z izbiranjem točk, AutoCAD nariše daljico med vsako izbrano točko in predhodnjo točko. Vsak narisan segment črte (vsaka daljica) je samostojen objekt, ki ga lahko po potrebi premikamo ali izbrišemo. Za zaključek tega ukaza pritisnemo na Enter. Črte lahko narišemo tudi z vnosom koordinat obeh končnih točk v ukazno vrstico.



Slika 3: Risanje črt

Pravokotnik (Rectangle)

Pravokotnik narišemo tako, da v orodni vrstici izberemo gumb Rectangle (Slika 2). Ukaz zahteva določitev dveh nasprotnih oglišč pravokotnika (glej Sliko 4). Na risalni površini s klikom levega gumba določimo prvo oglišče pravokotnika. Ob premikanju miške se samodejno ustvarja pravokotnik. Ob drugem kliku na levi gumb smo določili nasproti ležeče oglišče in s tem dobili končni pravokotnik. Oglišči lahko podamo tudi prek ukazne vrstice.

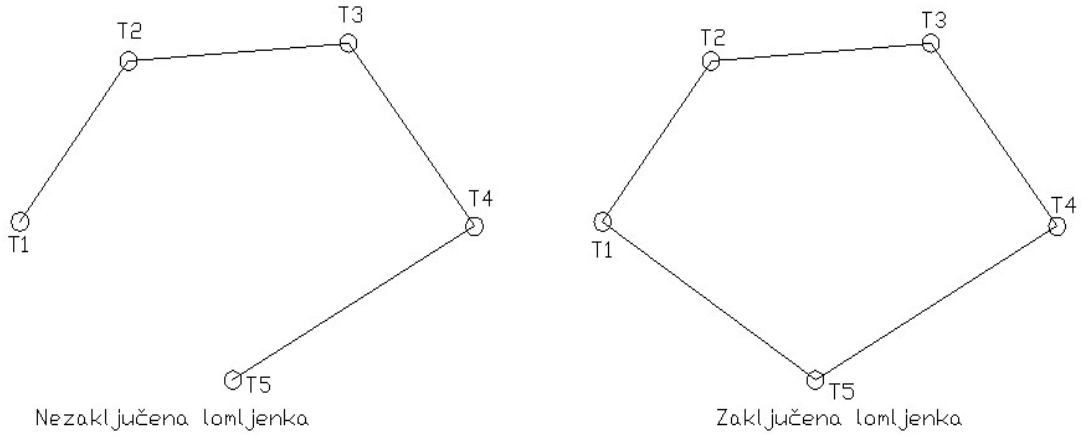
Slika 4: Risanje pravokotnika

Točka (Point)

Točko narišemo tako, da v orodni vrstici izberemo gumb Point (Slika 2). Nato se z miško premaknemo na ustrezno mesto na risalni površini. Ob kliku levega gumba smo tam ustvarili točko. Če poznamo natančne koordinate točke, lahko te podamo tudi preko ukazne vrstice.

Lomljenke (Polyline)

Včasih je zelo priročno, če lahko celoten niz daljic s skupnimi krajišči, obravnavamo kot en objekt. Takrat narišemo lomljenko, ki je samostojen objekt. Lomljenko narišemo tako, da v orodni vrstici za risanje izberemo gumb Polyline (Slika 2). Nato s klikom miškinega levega gumba na risalni površini podajamo točke, med katerimi se izrisujejo segmenti lomljenke. V nasprotju s podobnim risanjem zaporedja daljic, tu segmenti niso samostojni objekti. Za zaključek ukaza pritisnemo tipko *Enter*. S tem smo narisali nezaključeno lomljenko. Lahko pa združimo konec in začetek lomljenke tako, da namesto pritiska na tipko Enter v ukazno vrstico vtipkamo *Close* in lomljenko zaključimo. Tako lomljenko imenujemo zaključena lomljenka.

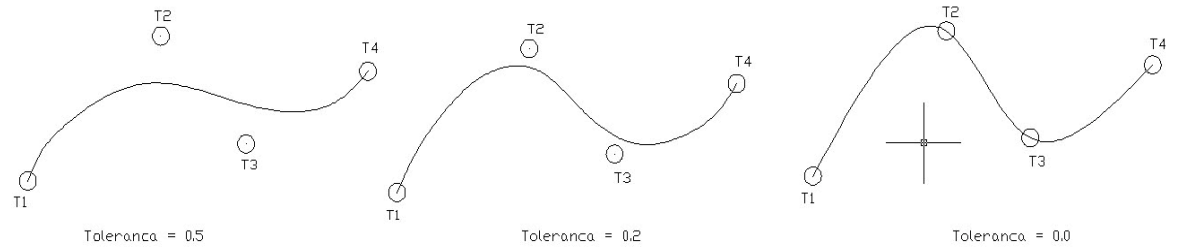


Slika 5: Nezaključena in zaključena lomljenka

Parabolični zlepki (Spline)

Ukaz Spline ustvari tip zlepka, ki se imenuje neenakomeren racionalen B-zlepek. Zlepek je gladka krivulja, ki je umeščena vzdolž podanih kontrolnih točk. Parabolični zlepek narišemo tako, da v orodni vrstici za risanje izberemo gumb Spline. Prav tako kot lomljenko ga

izrisujemo segmentno. S klikom levega gumba na risalni površini določamo točke, med katerimi se zlepki izrisujejo. Tudi parabolični zlepki so, ravno tako kot lomljenke, lahko zaključeni ali nezaključeni. Opcija *Fit Tolerance* uporabljamo za kontroliranje, kako se parabolični zlepek prilagaja kontrolnim točkam (glej Sliko 6). Nizka vrednost tolerance ustvari zlepek zelo blizu kontrolnih točk. Toleranca z vrednostjo 0 (nič) ustvari zlepek, ki poteka skozi kontrolne točke.



Slika 6: Sprememba zlepka glede na toleranco

Tekst (Mtext, Text)

Za dodajanje teksta na risalno površino imamo v AutoCAD-u dve možnosti. Dodajamo lahko:

- Enovrstični tekst (Single Line Text)
- Večvrstični tekst (Multiline Text)

Če za dodajanje teksta izberemo gumb iz orodne vrstice za risanje (Slika 2), bomo ustvarili objekt večvrstičnega teksta. Do obeh možnosti pa lahko dostopamo prek menijske vrstice (Slika 1) *Draw* → *Text* .

Pri programiranju moramo paziti na to, ali imamo opraviti z eno ali večvrstičnim tekstom, saj sta objekta enovrstičnega in večvrstičnega teksta različna. Imata drugačne lastnosti in metode in se tudi obnašata drugače. Več o tem si bomo ogledali kasneje.

2.2.3.2 Orodna vrstica z risalnimi ravninami

Risalne ravnine so zelo uporabne, kadar želimo večim različnim objektom na risbi določiti iste lastnosti. Z njimi si olajšamo razlikovanje med različnimi elementi risbe. Z risalnimi ravninami lahko risbo uredimo na več načinov. Med drugim lahko:

- risalnim ravninam dodelimo različne privzete lastnosti objektov, kot so barve, vrste črt in debeline črt. To pomeni, da vsi objekti na neki risalni ravnini privzamejo tisto barvo, vrsto črte ... kot smo jo določili tej risalni ravnini. Seveda lahko posameznemu objektu na neki risalni ravnini kasneje podamo drugačne lastnosti: npr. spremenimo barvo ali vrsto črte ... Če kasneje spremenimo določeno privzeto lastnost objektov neke ravnine, se spremembe poznajo na vseh objektih na tej risalni ravnini, razen na objektih, katerim smo eksplicitno določili drugačne lastnosti.
- določimo, ali bodo objekti na ravnini vidni ali ne. Tako lahko npr. na eni ravnini narišemo samo vse najdbe kosti. Če v določenem prikazu ne potrebujemo prikaza teh nahajališč, vse objekte na ravnini enostavno »skrijemo« s tem, da naredimo to ravnino nevidno. Če pa bi se želeli posvetiti samo najdbam kosti, lahko naredimo nevidne vse ostale ravnine. S tem se lahko posvetimo samo tistim objektom, ki jih moramo narisati ali urediti.
- določimo, kateri objekti bodo natisnjeni s tiskalnikom.

- risalno ravnino zaklenemo. Tako predmetov na njej ni mogoče spreminjati. Na ta način lahko preprečimo nehotene spremembe tistih delov risbe, ki so že končani. Tako lahko po zarisu izkopa arheološkega terena zaklenemo ravnino s črto, ki prikazuje izkopno polje, da je ob samem pregledovanju terena ne bi slučajno po pomoti prenesli na kakšno drugo mesto.

Kaj so risalne ravnine

Risalne ravnine so način upravljanja, urejanja in kontroliranja vizualne razporeditve risbe. Dejansko gre za to, da si celotno risbo lahko predstavljamo, kot da je sestavljena iz več prozornih plasti. Taka plast je risalna ravnina. Posamezni objekt je vedno narisan na neki plasti. Te plasti potem lahko po potrebi vključujemo ali izključujemo iz risbe in na ta način organiziramo določene lastnosti risbe. Uporablja jih tudi veliko drugih programov za risanje (npr Adobe PhotoShop,...).

Poglejmo si preprost arheološki primer. Recimo, da kopljemo neko jamo in naletimo na neko posodo. Ko skopljemo nekaj centimetrov nižje, najdemo še eno posodo. V računalniku to predstavimo tako, da prvo posodo damo na eno risalno ravnino, drugo posodo na drugo. Tako si ob vklopu prve in izklopu druge risalne ravnine lahko pogledamo prvo posodo, ob izklopu prve in vklopu druge risalne ravnine pa drugo posodo. Če želimo pogledati odnose med obema, vklopimo obe risalni ravnini.

Poleg privzetih lastnosti objektov kot so barve, vrste črt in debeline črt, mora imeti vsaka risalna ravnina tudi ime. Vse risbe imajo risalno ravnino z imenom *0* (nič). Tej risalni ravnini ni možno spreminjati imena ter jo zamrzniti. Vse ostale lastnosti ji lahko poljubno spremenimo. Novo risalno ravnino izdelamo tako, da ji določimo ime, barvo, vrsto črt in debelino črt. Ime je obvezno, ostalo se nastavi samodejno.

Ravnine imajo štiri lastnosti. Stanje teh lastnosti določajo, ali je risalna ravnina vidna, ali se obnavlja, ali jo lahko urejamo in ali bo izrisana:

- **On/Off:** Vklopljene risalne ravnine so vidne na risalni površini. Izklopljene ravnine niso vidne, še vedno pa lahko delamo tudi z njimi (spreminjamo objekte na njih, dodajamo nove objekte ...). Vse ravnine so privzeto vklopljene (imajo stanje On).
- **Thawed/Frozen:** Če ravnino zamrznemo, z njo ni mogoče početi ničesar. Prav tako zamrznjena ravnina ni vidna. Privzeto so vse ravnine odmrznjene. Pri praktičnem delu je torej razlika med izklopljeno in zamrznjeno ravnino v tem, da izklopljenim risalnim ravninam lahko dodajamo objekte, ki jih seveda ne vidimo, zamrznjenim pa ne.
- **Unlocked/Locked:** Nezaklenjene risalne ravnine so vidne in jih je mogoče urejati. To je privzeta nastavitev. Tudi zaklenjene risalne ravnine so vidne in jim je možno dodajati objekte, vendar jih ni mogoče urejati. Če je risalna ravnina nezaklenjena in izklopljena, jo je mogoče urejati, vendar ni vidna.
- **Plottable/Not Plottable:** Določimo, ali naj se objekti na risalni ravnini natisnejo s tiskalnikom ali ne.

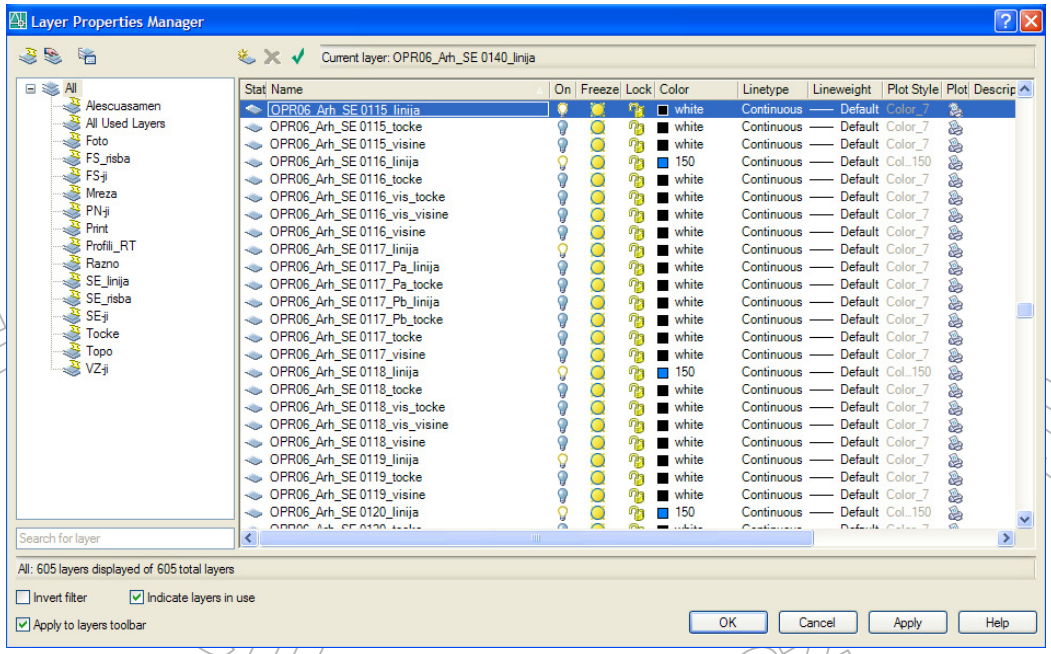
Prve tri lastnosti torej lahko kombiniramo v naslednje kombinacije:

On	Thawed	Unlocked	Objekti so vidni in jih lahko poljubno spreminjamo in dodajamo nove
On	Thawed	Locked	Objekti so vidni in jih lahko dodajamo, vendar jih ne moremo spreminjati
On	Frozen	Unlocked	Objekti niso vidni in jih ni mogoče spreminjati ali dodajati
On	Frozen	Locked	Objekti niso vidni in jih ni mogoče spreminjati ali dodajati
Off	Thawed	Unlocked	Objekti niso vidni, a jih lahko poljubno spreminjamo in dodajamo nove
Off	Thawed	Locked	Objekti niso vidni, a jih lahko dodajamo, vendar ne moremo spreminjati
Off	Frozen	Unlocked	Objekti niso vidni in jih ni mogoče spreminjati ali dodajati
Off	Frozen	Locked	Objekti niso vidni in jih ni mogoče spreminjati ali dodajati

Tabela 1: Kombinacija lastnosti risalnih ravnin

Izdelava novih risalnih ravnin

Novo risalno ravnino izdelamo tako, da v orodni vrstici risalnih ravnin odpremo upravitelja lastnosti risalnih ravnin (Layer Properties Manager). Odpre se okno, kot ga vidimo na Sliki 7. V tem pogovornem oknu so prikazane vse trenutne risalne ravnine in njihove lastnosti. Nove risalne ravnine lahko ustvarimo tukaj. To storimo tako, da v levem oknu kliknemo desni miškin gumb ter izberemo *New Layer*. Ustvari se risalna ravnina s privzetim imenom. Aktiven ostane stolpec, ki se imenuje *Name* (glej Sliko 7). Zapišemo še poljubno ime ter potrdimo z levim gumbom miške. Če imena ne spremenimo, ostane privzeto. Pri izbiri imena je smiselno, da uporabimo tako ime, ki nam bo povedalo, kakšni objekti se na tej ravnini nahajajo.



Slika 7: Layer Properties Manager

Seveda to ni edini način izdelave novih risalnih ravnin. V nadaljevanju bom pokazal, kako lahko ustvarimo nove risalne ravnine tudi s pomočjo vgrajenega programskega jezika Visual Basic for Application.
Vsaka risalna ravnina predstavlja eno vrstico v Layer Properties Manager-ju (glej Sliko 7). Vsaka vrstica je razdeljana na več stolpcev. Vsak stolpec predstavlja eno lastnost. Prvi stolpec z imenom *Status*, nam kaže, ali je risalna ravnina aktivna. Če je, se ob njej nahaja zelena kljukica. Vedno je aktivna le ena ravnina. Risalno ravnino naredimo aktivno tako, da jo izberemo, na njej pritisnemo desni gumb miške ter izberemo *Set current*. Na to risalno ravnino bomo poslej dodajali objekte. V drugem stolpcu je ime risalne ravnine. Tretji stolpec z imenom *On* nam omogoča, da določimo ali je risalna ravnina vidna ali ne. To storimo tako, da z levim gumbom miške kliknemo na ikono, ki ponazarja žarnico. Če žarnica »sveti«, je risalna ravnina vidna, drugače pa nevidna. Podobno lahko spreminjamo tudi ostale lastnosti.

2.2.3.3 Orodja iz orodne vrstice za spreminjanje

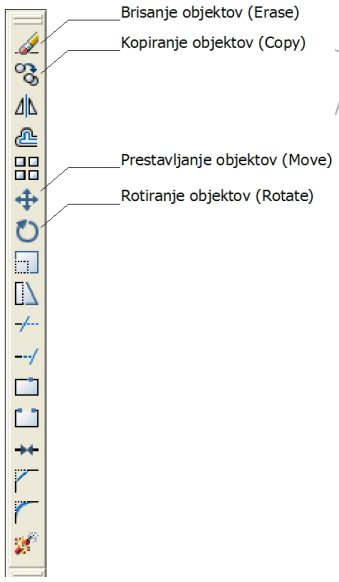
Spreminjanju risbe pravimo *urejanje*. Večina ukazov za urejanje je zbranih v orodni vrstici za spreminjanje (Modify).
Objekt, ki ga želimo urediti, moramo najprej izbrati. To storimo tako, da kazalec za izbiranje z miško prestavimo na željeni objekt ter kliknemo levi gumb miške. Izberemo lahko tudi več objektov hkrati. To storimo tako, da z levim gumbom kliknemo na vse objekte, ki jih želimo urediti. V primeru, da smo izbrali napačen objekt, s tipko Esc prekinemo izbiranje, kar nam omogoča vnovično izbiranje novih objektov. Ukazi za urejanje veljajo za vse trenutno izbrane objekte.

Brisanje objektov

Pri risanju se skorajda ne moremo izogniti brisanju določenih objektov. Ukaz za brisanje, *Erase* (Slika 8), je zelo preprost. Nima namreč nobene izbire. Objekt izbrišemo tako, da ga izberemo in v orodni vrstici za spreminjanje kliknemo gumb za brisanje. Naslednja možnost je, da objekt izberemo, ter pritisnemo tipko Delete. Če je izbranih več objektov, s klikom na ukaz Erase, oziroma pritiskom na tipko Delete, pobrišemo vse.

Prestavljanje objektov

Predmete na risbi prestavljamo z ukazom *Move* (Slika 8). Predmet prestavimo tako, da ga izberemo na risalni površini (kliknemo nanj) in v orodni vrstici za spreminjanje kliknemo gumb za prestavljanje. Potem objekt z miško poljubno prenašamo po risalni površini. Ko ga namestimo na željeno mesto, pritisnemo levi gumb miške. Če je bilo izbranih več objektov, prestavimo vse. Pri tem se ohranijo razmerja (razdalja) med izbranimi objekti.
Če želimo prestaviti še kak objekt, kliknemo nanj in ponovno izberemo gumb za prestavljanje.



Slika 8: Orodna vrstica za spreminjanje

Kopiranje objektov
Kopiranje objektov je zelo podobno prestavljanju. Edina razlika je v tem, da pri kopiranju AutoCAD ne odstrani objekta s prvotnega mesta.

Rotiranje objektov
V AutoCAD-u lahko objekt ali več objektov preprosto vrtimo okrog izbrane izhodiščne točke. Objekt zarotiramo tako, da ga najprej izberemo na risalni površini. V orodni vrstici izberemo gumb Rotate. Nato na risalni površini izberemo točko, ki označuje središče rotacije. Ob premiku miške objekt rotira okoli te točke. Ko smo ga pravilno namestili, njegovo lego potrdimo z levim miškinim gumbom. Če poznamo natančen kot rotacije v stopinjah, ga lahko vnesemo preko ukazne vrstice. Pri tem velja, da izhodiščni poltrak kota kaže horizontalno v desno in da stopinje naraščajo v smeri, ki je nasprotna smeri urinega kazalca. Če je izbranih več objektov, zarotiramo vse objekte glede na skupno središče rotacije.

2.2.4 Povzetek

Kot sem že omenil, je AutoCAD zelo obsežen program. Na kratko smo si pogledali njegovo delovno okolje. Delovno okno smo razdelili na štiri ključne enote:

- Risalna površina
- Meniji in orodne vrstice
- Ukazna vrstica
- Vrstica stanja

Vsako enoto smo si v grobem ogledali. Nadaljevali smo z opisom nekaterih orodij iz orodnih vrstic ter menijev. Opisa so bila deležna samo orodja, ki sem jih uporabljal pri razvoju programa, katerega opis je jedro diplome. Nekoliko podrobneje smo razložili princip dela z risalnimi ravninami.

3. VISUAL BASIC FOR APPLICATIONS

Visual Basic je programski jezik, ki omogoča hitro in preprosto izdelavo programov. Odkod tako ime? Prvi del - Visual – pomeni, da grafični vmesnik izdelujemo s pomočjo množice vnaprej določenih elementov, ki jim pravimo gradniki. To so npr. okenca za vnos besedila, ukazni gumbi, seznami, potrditvena polja, izbirni gumbi in še vrsta drugih. Te gradnike nanašamo na osnovno in ostala okna programa (ali kot jim pogosto tudi rečemo – na obrazce) in jih tam razvrstimo in določimo izgled (velikost, barvo, …). Drugi del imena je povezan s programskim jezikom BASIC, ki je med programerji zelo priljubljen in omogoča izdelavo preprostih, pa tudi zelo zahtevnih programov. A sam jezik je precej drugačen kot BASIC, ki ga poznamo iz začetkov razvoja hišnih in osebnih računalnikov, saj je sedaj objektno usmerjen.

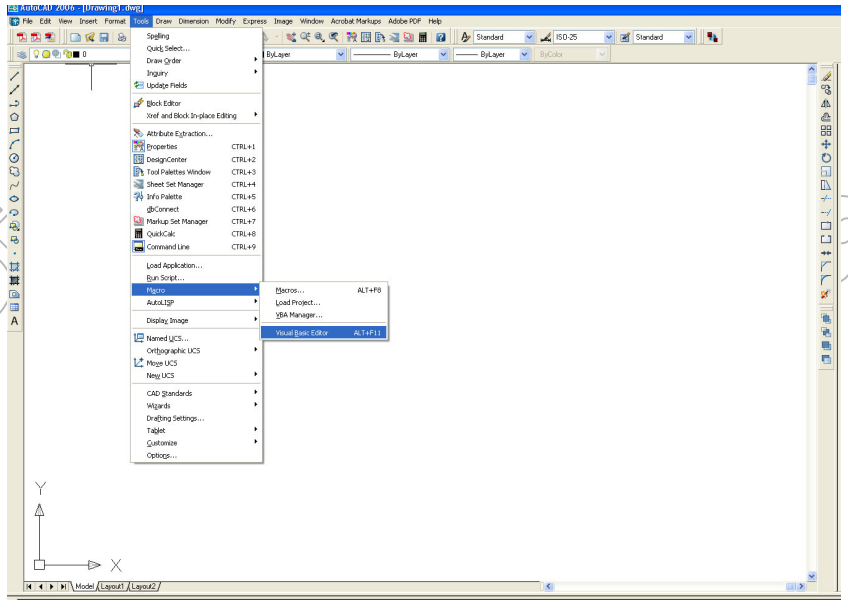
Danes poznamo več različic Visual Basica, ki jih lahko uporabljamo tudi pri programiranju spletnih strani (npr. VB Script) ali pa v okviru nekaterih priljubljenih programskih paketov, kot sta npr. Microsoft Office in AutoCAD (Visual Basic for Applications - v nadaljevanju VBA). Osnovna značilnost VBA, po kateri se razlikuje od »velikega brata« Visual Basica, je v tem, da je tesno naslonjen na programski paket v katerem deluje. To je predvsem razvidno iz objektnega modela, ki ga podpira. Tako VBA v AutoCAD-u pozna objekte, ki predstavljajo plasti (Layer), točke (AcPoint), črte (AcLine) ... VBA znotraj programa Excel pa pozna objekte, ki predstavljajo stolpec, celico, grafikon … torej tiste objekte, ki so sestavni del preglednic. Pri sami razlagi VBA se bom naslonil na programski jezik java, ki smo ga spoznali na fakulteti. Tako nekaterih pojmov z VBA ne bom razlagal, ampak samo omenil.

3.1 Razvojno okolje

Za začetek si najprej oglejmo razvojno okolje programskega jezika Visual Basic for Applications.

3.1.1 Zagon

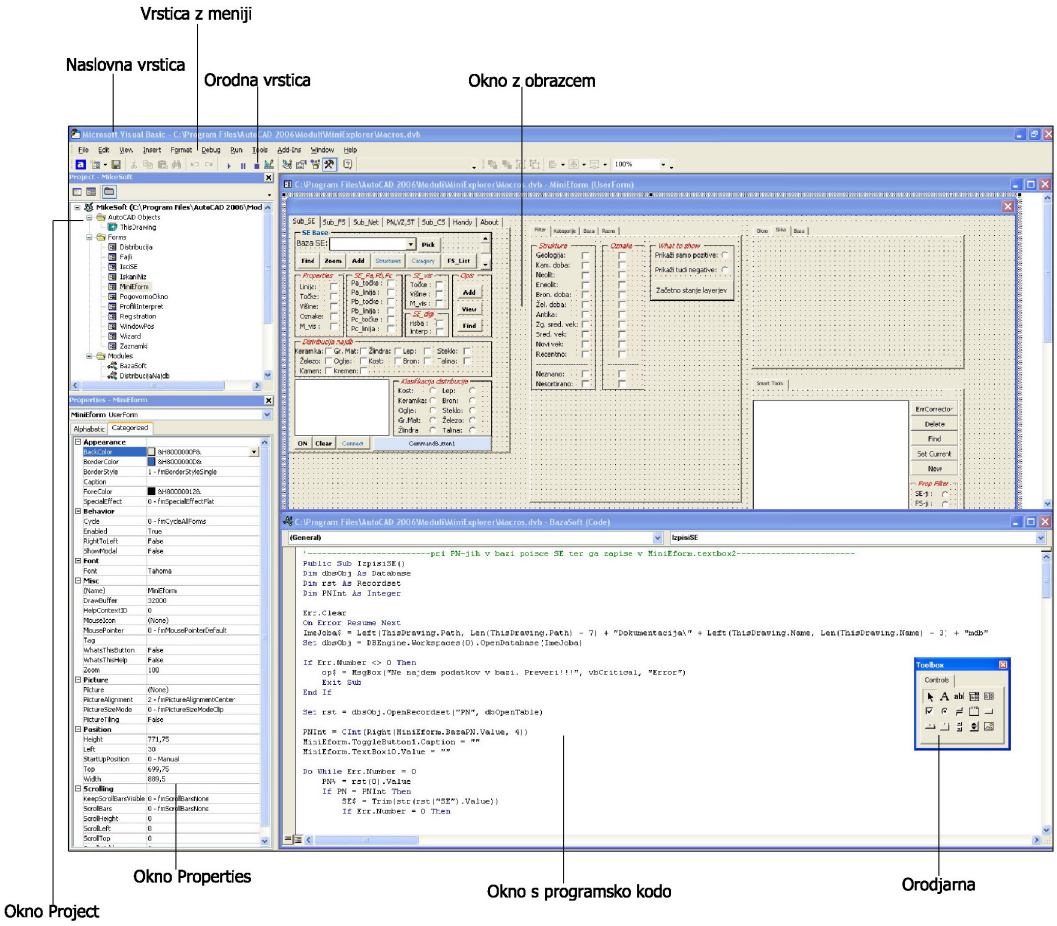
V AutoCAD – u dostopamo do okolja, kjer uporabljamo VBA prek menija *Tools* → *Macro* → *Visual Basic Editor*. Okolje se imenuje VBA IDE (kar je kratica za *Integrated Development Environment* ali integrirano razvojno okolje).



Slika 8: Zagon programskega okolja VBA

3.1.2 Programsko okno

Po zagonu okolja zagledamo programsko okno. Opišimo ga.



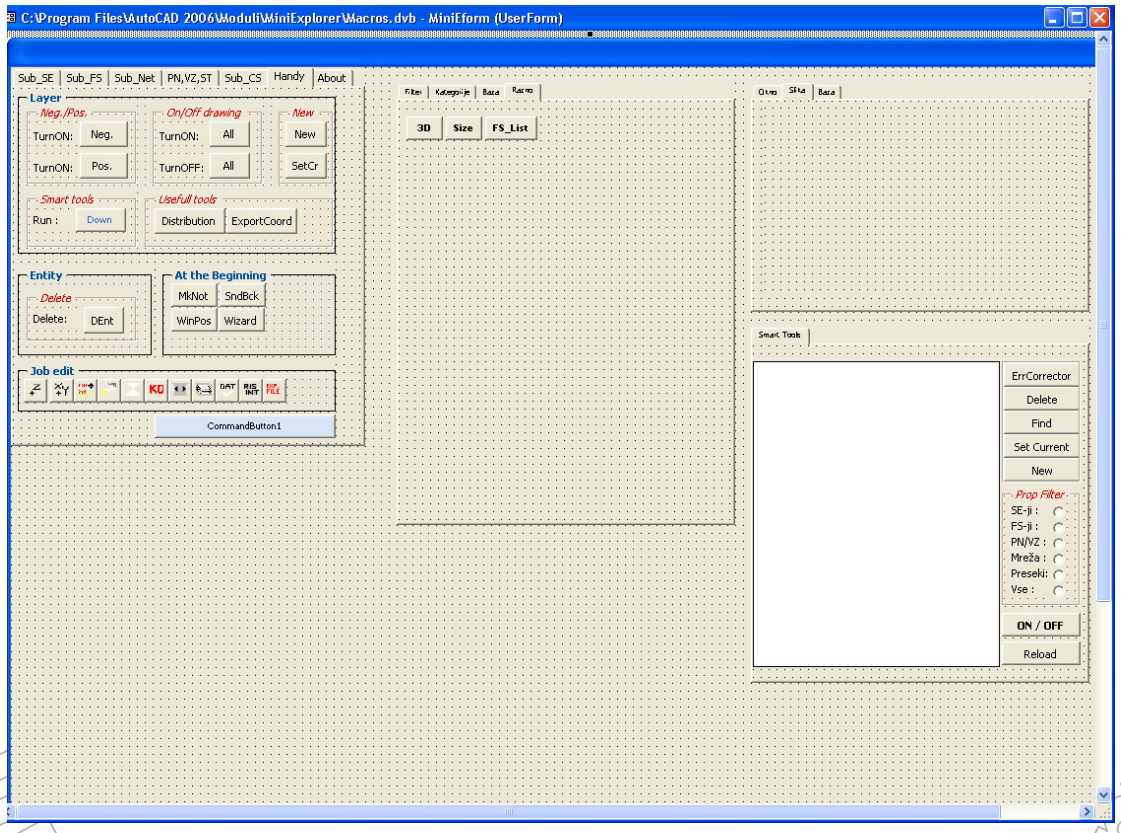
Slika 9: Razvojno okolje VBA

Naslovna vrstica
Na vrhu programskega okna je naslovna vrstica z imenom programa (Microsoft Visual Basic) in trenutno odprtim projektom.

Vrstica z meniji
Naslovni vrstici sledi vrstica z meniji, v katerih najdemo vse ukaze, ki so nam na voljo pri delu.

Orodna vrstica
Orodna vrstica ponuja hiter dostop do najuporabnejših ukazov, ki jih lahko poženemo kar s klikom na ikono. Ko prvič poženemo program, se na zaslonu prikaže orodna vrstica Standard, druge orodne vrstice pa lahko po želji prikažemo z ukazom *View* → *Toolbars*.

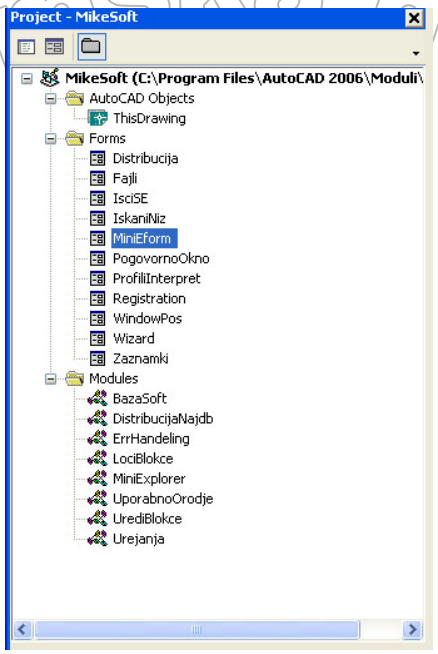
Obrazec



Slika 10: Obrazec MiniExplorer

Obrazec je okno, ki predstavlja vmesnik med programom in uporabnikom. Uporabnik torej s programom komunicira prek obrazca. Vanj zapisuje besedila, klika na gumbe, izbira elemente s seznama in podobno. Po drugi strani pa program v obrazec izpisuje rezultate in se tako odziva na podatke in izbire, za katere se je odločil uporabnik. Na obrazec vstavimo razne kontrole in slikovne elemente. Na Sliki 10 imamo primer obrazca programa MiniExplorer, na katerem imamo kombiniran seznam, ukazne gumbe, izbirne gumbe, oznake, okvirje...

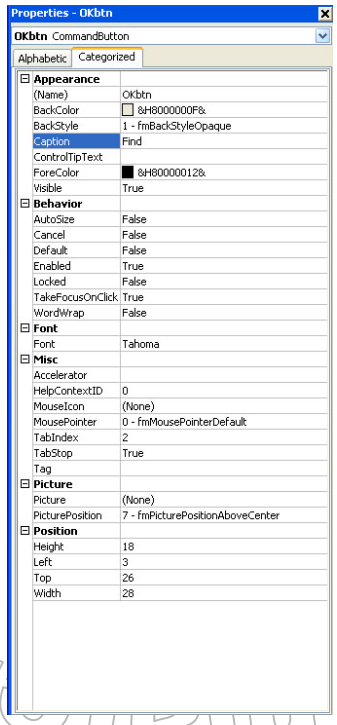
Okno Project



Slika 11: Okno Project

Projekt je zbirka različnih vrst datotek, ki skupaj tvorijo celoten program. V projekt sodijo obrazci, moduli s programsko kodo in razne kontrolne datoteke. Vse te elemente prikažemo v oknu Project. Več o projektih si bomo ogledali v razdelku Projekti.

Okno Properties

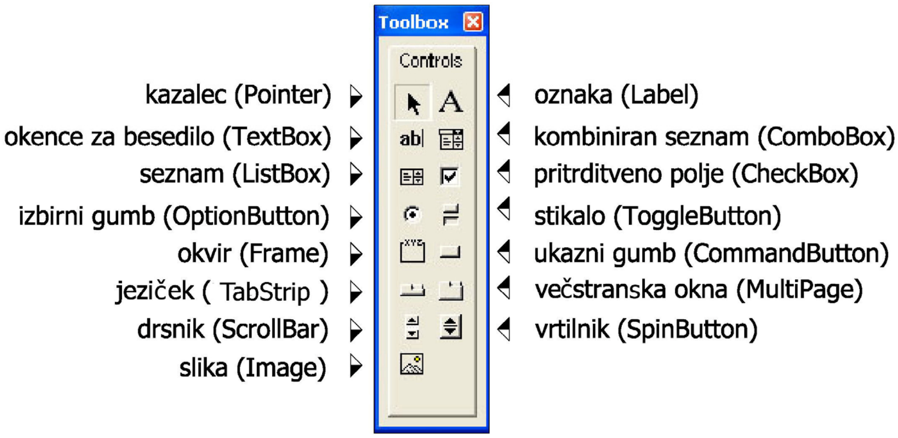


Slika 12: Okno Properties

V oknu Properties prikazujemo lastnosti obrazca in gradnikov, ki so na njem. Na vrhu okna je okence trenutno aktivnega gradnika (npr. OKbtn), v seznamu, ki ga odpremo s klikom na puščico, pa so naštete še vse preostale kontrole, ki so prisotne v obrazcu. Na zavihkih so bodisi po kategorijah, bodisi po abecedi urejene vse lastnosti, ki pripadajo v okencu navedenemu gradniku. S klikom na ustrezno lastnost jo lahko spremenimo.

Orodjarna (Toolbox)

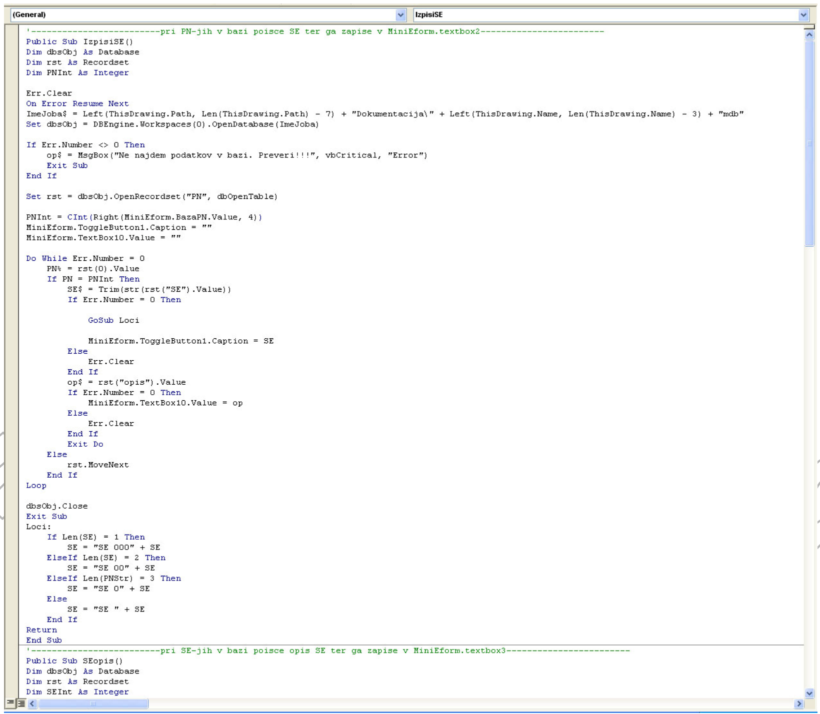
Orodjarna vsebuje raznovrstne gradnike, ki jih med pripravo programa vstavljamo v obrazec ter kazalec (Pointer) s pomočjo katerega izbiramo gradnike, ki so trenutno na obrazcu.



Slika 13: Orodjarna

Če pogledamo Sliko 10, je obrazec opremljen s standardnimi gradniki iz orodjarne. Več o gradnikih si bomo ogledali v razdelku Gradnja pogovornega okna MiniExplorer.

Okno s programsko kodo



Slika 14: Okno s programsko kodo

V okno s kodo zapisujemo programsko kodo oz. ukaze jezika VBA, ki krmilijo delovanje programa. Kadar okno s kodo ni prikazano na zaslonu, ga prikažemo z ukazom *View → Code*. Vedeti moramo, da ima vsak obrazec svoje okno s pripadajočo kodo, projekt pa lahko vsebuje kodo, ki je zapisana v raznih modulih.

3.2 Projekti

Pri delu v VBA imamo vseskozi opravka s projekti. Projekt je skupina datotek, ki jih potrebujemo pri izdelavi programa. V datotekah so shranjeni podatki o obrazcih, moduli s programsko kodo, slike, ki se uporabljajo v programu in podobno.
Vsak projekt lahko vsebuje naslednje vrste datotek:

- Projektno datoteko. Ta ima podaljšek `.dvb`
- Po eno tekstovno datoteko za vsak obrazec. Te datoteke imajo podaljšek .frm
- Po eno binarno datoteko za vsak obrazec. Te datoteke imajo podaljšek .frx in vsebujejo podatke o lastnostih kontrol na obrazcih.
- Po eno datoteko s podaljškom .bas za vsak standardni modul (*standard module*).
- Po eno datoteko s podaljškom .cls za vsak razredni modul (*class module*).

3.2.1 Sestavni deli projekta

Kot smo že omenili, lahko projekt vsebuje obrazce (*form module*), standardne module (*standard module*) in razredne module (*class module*).

Obrazci
Obrazec (Form) je v VBA vgrajen razred. V programu uporabljeni primerki obrazcev (torej objekti tipa Form) so shranjeni v posebnih modulih (*form module*) – tekstovnih datotekah s podaljškom .frm. Tu najdemo podroben opis obrazca in gradnikov na njem. Poleg tega so tu prisotne še najave spremenljivk, konstant in zunanjih procedur, ki jih uporabljamo v obrazcih.

Standardni moduli
Standardni modul (s*tandard module*) lahko vsebuje najave javnih (*public*) spremenljivk, konstant, tipov, funkcij in procedur.

Razredni moduli
Poleg v VBA vgrajenih razredov (kot je npr. Form), si lahko tudi sami napišemo svoje razrede. Razredni moduli (*class module*) so podobni obrazcem, le da nimajo vidnega uporabniškega vmesnika. Koristni so, kadar želimo izdelati svoje razrede, kamor vključimo tudi vso programsko kodo, s katero določimo lastnosti in postopke nad objekti tega razreda.

3.3 Spremenljivke, konstante in tipi podatkov

Pri izračunih moramo pogosto shranjevati začasne vrednosti. V ta namen uporabljamo spremenljivke. Vsaka spremenljivka ima svoje ime in tip. Tip določa vrsto podatkov, ki jih lahko shranimo v spremenljivko. Če se tip podatka ne sklada s podatkom, ki ga spremenljivki priredimo, pride do napake. Tudi konstante lahko hranijo vrednosti, vendar se te vrednosti, za razliko od spremenljivk, med izvajanjem programa ne spreminjajo.

3.3.1 Spremenljivke

Kot smo že omenili, uporabljamo spremenljivke za hrambo podatkov. Vsaka spremenljivka ima svoje ime in tip. Če spremenljivki ne določimo tipa, se uporabi privzeti tip Variant. Ta je

med vsemi najbolj prilagodljiv, saj lahko vanj shranimo vsakovrstne podatke. Ima pa tudi svoje pomankjivosti. Zaseda namreč precej več prostora v pomnilniku kot drugi tipi, ki so posebej prirejeni vrstam podatkov.

Najava spremenljivk

Ob najavi spremenljivk povemo programu, da želimo to spremenljivko uporabljati v kodi. Običajno spremenljivke najavimo s stavkom *Dim*. Tega sestavlja rezervirana beseda Dim, ki ji sledi ime spremenljivke in tip podatkov, ki jih bo hranila. Spremenljivka je lahko tudi objekt.

```
Dim Ime_Spremenljivke [As Tip_podatkov]
Dim Ime_Spremenljivke [As Tip_objekta]
```

Doseg in trajanje spremenljivk

Doseg spremenljivk določa, v katerih delih programske kode bodo spremenljivke vidne. Poznamo lokalne in globalne spremenljivke.

Spremenljivke, ki jih najavimo znotraj procedure, so lokalne spremenljivke. Obstajajo le toliko časa, dokler se procedura izvaja. Tako spremenljivko je mogoče uporabljati le znotraj procedure, zato je druge procedure ne poznajo. Zaradi tega ni težav, če enaka imena spremenljivk uporabljamo v več procedurah.

Lokalne spremenljivke je moč uporabljati le v proceduri, v kateri jo najavimo. Vrednosti spremenljivk, ki jih najavimo s stavkom Dim, se izgubijo, ko se izvajanje procedure konča. To je moč preprečiti s stavkom Static, saj tedaj spremenljivka zadrži svojo vrednost tudi, ko se procedura izvede do konca.

```
Dim Ploscina As Single
Static Uporabnik As String
```

Spremenljivkam, ki jih najavimo na začetku obrazca ali modula, pravimo globalne spremenljivke. Glede na to, ali jih uporabljamo v enem modulu ali pa v celem programu, so privatne ali javne. Privatne so tiste, ki so uporabne samo v vseh tistih procedurah, ki so zapisane na tem obrazcu ali modulu. Najavimo jih z določilom Private.

```
Private Ploscina As Single
```

Kadar potrebujemo spremenljivko, ki bi bila na voljo v celotnem programu, torej v vseh obrazcih in modulih, si pomagamo z določilom Public, npr.:

```
Public Ploscina As Single
```

Globalne spremenljivke vedno najavimo na začetku obrazca ali modula. Ne moremo jih najaviti v proceduri.

3.3.2 Konstante

Kadar se v programski kodi večkrat pojavijo enaka števila, je koristno, da jih najavimo kot konstante. Poznamo dve vrsti konstant. Prve so vgrajene v VBA, druge pa podajamo sami. Oglejmo si najavo konstant:

```
[Public][Private] Const Ime_konstante [As Tip_podatkov] = Izraz
```

Primer v VBA vgrajenih konstant je konstanta `vbGray`, ki je celo število in pomeni oznako za sivo barvo.

3.3.3 Tipi podatkov

Številčni tipi
VBA pozna več tipov, ki so namenjeni hrambi števil. Števila so lahko cela ali realna. Operacije s celimi števili delujejo hitreje kot z realnimi.
Pri celih številih imamo na voljo tipe Byte, Integer in Long. Najpogosteje uporabljen tip je Byte. Obsega sicer le števila od 0 do 255, vendar so podatki pogosto v tem obsegu. Koristen je tudi za hrambo dvojiških podatkov. V pomnilniku zasede 1 zlog.

```
Dim Stevilo As Byte
```

Tip Integer zajema cela števila med -32.768 in +32.767, v pomnilniku zasede 2 zloga.

```
Dim Stevilo As Integer
```

V spremenljivkah tipa *Long* lahko hranimo števila med -2.147.483.648 in +2.147.483.647 in v pomnilniku zasedejo 4 zloge

```
Dim Stevilo As Long
```

Pri realnih številih poznamo tipa Single in Double. VBA pozna tudi tip Currency, vendar ga ne bom opisoval, ker ga v MiniExplorerju ne uporabljam.

Tip *Single* zajema števila od $-3.4 * 10^{38}$ do $+3.4 * 10^{38}$ in zasede 4 zloge. Pri uporabi tipa Single je natančnost približno 6 decimalnih mest.

```
Dim Stevilo As Single
```

Zmogljivejši je tip Double z obsegom od $-1.8 * 10^{308}$ do $+1.8 * 10^{308}$ in okvirno natančnostjo 14 decimalnih mest. Ta zasede 8 zlogov.

```
Dim Stevilo As Double
```

Nizi znakov
Kadar imamo opraviti s spremenljivko, ki vedno hrani znake, je koristno, da jo najavimo kot niz znakov, oziroma kot tip *String*. Npr.:

```
Dim Ime As String
```

Niz je poljubno zaporedje znakov med dvojnima narekovajema ("Niz"). Število znakov, ki jih zmore hraniti spremenljivka tipa *String*, je praktično neomejeno, saj v njih lahko hranimo do 2.000.000.000 znakov. Na srečo VBA sam poskrbi, da spremenljivke tipa String zasedajo le toliko prostora, kot je potrebno za trenutno število znakov, saj bi drugače kaj hitro zmanjkalo pomnilnika. Nizi v VBA so torej spremenljive dolžine.
Pri delu z nizi imamo na voljo številne funkcije, ki jih bomo spoznali kasneje.

Logični tipi
Za logični tip *Boolean* se običajno odločimo pri delu s spremenljivkami, za katere vemo, da bodo zavzele le dve vrednosti (True ali False). Privzeta vrednost spremenljivke Boolean je False.

```
Dim Logicna As Boolean
```

Tip Variant
Spremenljivka tipa *Variant* lahko hrani podatke poljubne vrste, izjema so le nizi nespremenljive dolžine. Kot smo omenili, v VBA ni potrebno najaviti spremenljivk, oziroma tudi ni potrebno, da pri najavi določimo tip spremenljivke. Vse spremenljivke, ki jih ne najavimo ali, ki jim ob najavi ne določimo tipa, so tipa Variant.
Pri delu s tipom Variant velja nekaj previdnosti. Pri izvajanju aritmetičnih operacij ali funkcij nad takimi spremenljivkami mora v spremenljivki biti res število, v nasprotnem pride do napake.

Objekti
Objektne spremenljivke se lahko sklicujejo na poljubne objekte v našem ali kakem drugem programu. Če najavimo spremenljivko tipa *Object*, ji lahko pozneje s stavkom *Set* (glej razdelek Prireditveni stavek) priredimo poljuben objekt, ki ga program pozna:

```
Dim Predmet As Object
Set Predmet = New Form1
```

Bolje je, da namesto splošnega tipa Object uporabimo ustrezne konkretne razrede, npr. Form:

```
Dim Predmet As Form
Set Predmet = New Form1
```

Kakšni objekti, oziroma razredi so v VBA v AutoCAD-u na voljo, si bomo ogledali v razdelku AutoCAD-ovi objekti in postopki nad njimi.

Polja
Polja so posebna podatkovna struktura, ki nam omogoča, da množico podatkov istega tipa shranimo pod istim imenom spremenljivke, nato pa jih ločimo po indeksih. Pogosto to poenostavi kodo, saj lahko v zankah s pomočjo indeksov učinkovito dostopamo do posameznih elementov polja.
Vsako polje ima zgornjo in spodnjo mejo. To sta indeksa prvega in zadnjega elementa. Prvi element polja ima indeks enak 0. Če bi radi za indeks prvega elementa uporabili kakšno drugo število in ne 0, moramo to eksplicitno zahtevati. Poglejmo si to na primerih:

```
Dim Imena (30) As Integer
Dim Imena (1 To 30) As Integer
```

V prvem primeru smo najavili polje z 31 števili. Indeks prvega elementa je 0, zadnjega pa 30. V drugem primeru smo najavili polje s 30 elementi, kjer je prvi indeks 1 in zadnji 30. V nasprotju s programskim jezikom java je torej parameter pri deklaraciji zadnji indeks in ne velikost polja.Polja so lahko statična ali dinamična. Statična polja vedno vsebujejo nespremenljivo število elementov, medtem ko se število elementov v dinamičnem polju lahko spreminja med delovanjem programa.

Statična polja
Statična polja lahko tako kot ostale spremenljivke, glede na željen doseg najavimo na več načinov. Če želimo, da bi bilo polje dostopno v celem programu, ga najavimo v modulu z določilom *Public*. Kadar pa naj bo polje dostopno le znotraj modula ali obrazca, ga v tem modulu ali obrazcu najavimo tako, da uporabimo določilo *Private*. Lokalno polje najavimo s stavkom `Dim`, ki ga zapišemo v proceduro. Ko statičnem polju nastavimo velikost, je kasneje ne moremo spremeniti.

```
Dim Imena (30) As String
Dim Stevila (15) As Integer
```

Dinamična polja
Pogosto med programiranjem ne vemo točno, koliko elementov bo v polju. VBA nam zato ponuja rešitev v obliki dinamičnega polja. Tu je moč število elementov spreminjati tudi med izvajanjem programa. Dinamična polja zasedejo manj prostora v pomnilniku, saj jih po potrebi lahko zmanjšamo.
Najavimo jih ravno tako kot statična, le da tukaj ne zapišemo števila elementov.

```
Dim Imena () As String
Dim Stevila () As Integer
```

Število elementov določimo pozneje s stavkom *ReDim*.

```
Redim Imena (30)
Redim Stevila(15)
```

Vsakič, ko s stavkom *ReDim* spremenimo število elementov polja, se vrednosti v polju izgubijo.
Obstaja pa način, da se podatki v polju ohranijo. V tem primeru moramo za besedo ReDim zapisati še besedo *Preserve*. Zavedati pa se je treba, da je operacija lahko precej časovno potratna. Če želimo na primer povečati velikost polja za 5, ne da bi izgubili vrednosti v polje že vnesenih elementov, zapišemo:

```
ReDim Preserve Imena(UBound(Imena)  + 5)
```

Funkciji LBound in UBound
Funkcija `LBound` vrne najmanjši, `UBound` pa največji indeks elementov v polju. Oblika zapisa je:

```
LBound(Polje)
UBound(Polje)
```

3.4 Prireditveni stavek
Ob najavi se spremenljivki priredi inicializacijska vrednost, ki je odvisna od tipa spremenljivke. Številčne spremenljivke dobijo vrednost 0, nizi postanejo prazni, spremenljivke tipa Variant dobijo vrednost Empty, spremenljivke tipa Boolean pa vrednost False.
Najpreprostejši način spreminjanje vrednosti spremenljivk je z uporabo prireditvenega stavka. Spremenljivki priredimo novo vrednost tako, da zapišemo njeno ime, enačaj in izraz. Tip vrednosti izraza se mora ujemati s tipom spremenljivke. Poglejmo nekaj primerov:

```
Dim Stevilo As Integer, Dim Niz As String, Dim Log As Boolean
Stevilo = 30 + 8 * 8
Niz = "test"
Log = True
```

Omeniti velja še posebnost pri prirejanju objektov. Pri objektih pri prireditvenem stavku pred objektno spremenljivko vedno uporabimo besedico *Set*. Z besedico Set spremenljivko inicializiramo in s tem pripravimo, da ji lahko priredimo neko vrednost. Poglejmo si na primeru, kako trenutno risalno ravnino priredimo novemu objektu tipa risalna ravnina.

```
Dim NovObjekt As AcadLayer
Set NovObjekt = ThisDrawing.ActiveLayer
```

Več o AutoCAD-ovih objektih si lahko preberemo v naslednjem poglavju.

3.5 AutoCAD-ovi objekti in postopki nad njimi

Seznam vseh AutoCAD-ovih objektov dobimo tako, da odpremo iskalnik objektov. V vrstici z meniji izberemo *View* → *Object Browser*. Tukaj lahko vidimo tudi vse postopke in lastnosti, ki jih ima nek objekt. Objekti so urejeni v knjižnicah. Vsi predmeti, s katerimi delamo v AutoCAD-u, so v njegovi knjižnici. Seveda lahko dodamo tudi druge knjižnice (npr: Microsoft Excel), vendar tega ne bom opisoval, ker v nadaljevanju uporabljam samo objekte iz AutoCAD-ove knjižnice.
V levem podoknu *Iskalnika objektov*, poimenovanem *Classes*, vidimo seznam objektov. V VBA poznamo tako posamezne objekte kot zbirke objektov (*collection*). Zbirka objektov je množica istovrstnih objektov. Za primer vzemimo objekt risalne ravnine. Ta se imenuje `AcadLayer`, zbirka risalnih ravnin pa `AcadLayers`. Namen zbirk je delo s skupinami objektov. Običajno vse elemente zbirke pregledamo s stavkom For Each.
Poglejmo kako bi najavili obe vrsti spremenljivk v VBA:

```
Dim RisRavnina As AcadLayer
Dim RisRavnine As AcadLayers
```

Vsakemu od objektov lahko:

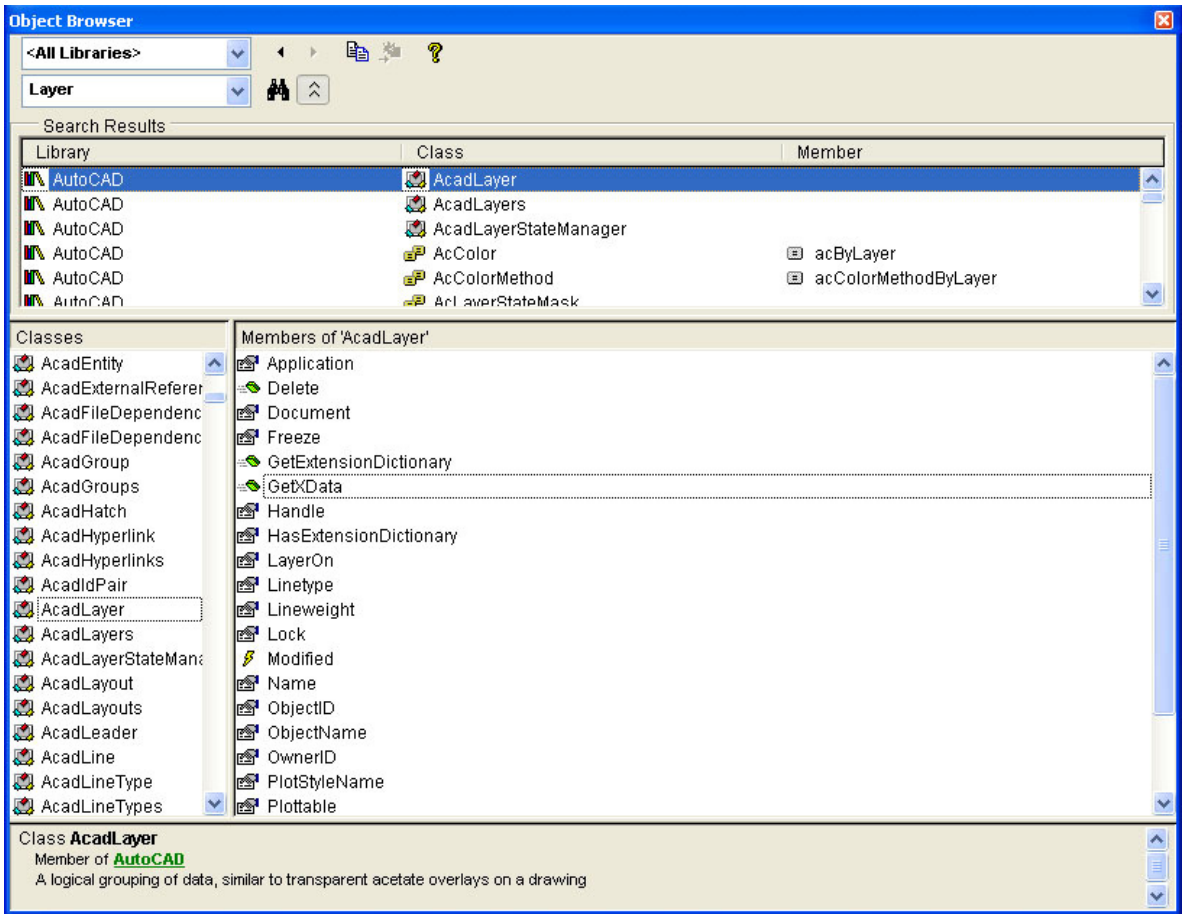
- spreminjamo lastnosti (barva, napis, položaj ...) ali pa
- nad njim izvajamo postopke (objekt izbrišeno, zarotiramo ...).

Lastnosti in postopki, ki jih lahko izvajamo na nekem objektu, so našteti v desnem podoknu Iskalnika objektov z imenom *Members* (glej Sliko 15) .
Objekti vsebujejo tudi druge objekte, med sabo pa so ločeni s piko. Nato za piko dodamo želeni postopek ali lastnost.
V VBA je AutoCAD kot program objekt. Tudi trenutna risba je objekt. Če želimo določiti objekt na risbi, moramo določiti, za kateri program gre, risbo in šele nazadnje določimo objekt na risbi. Poglejmo si primer, kjer želimo na risalno površino dodati objekt Črta(Line):

```
Application.ActiveDocument.ModelSpace.AddLine(ZacetnaTocka,KoncnaTocka)
```

Drugo ime za objekt `Application.ActiveDocument` je `ThisDrawing` (trenutna risba), zato lahko zgoraj napisano kodo napišemo tudi takole:

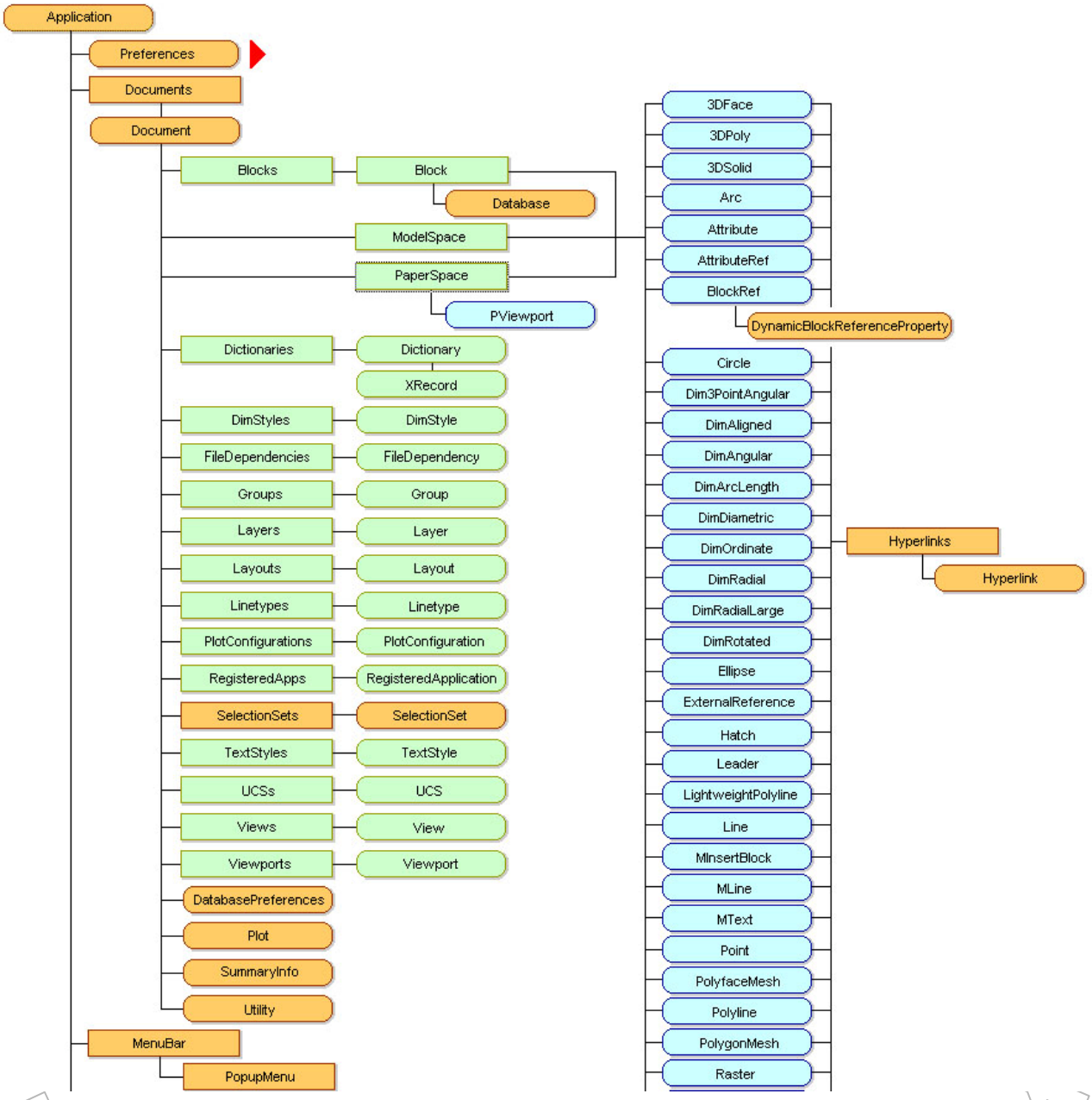
```
ThisDrawing.ModelSpace.AddLine(ZacetnaTocka,KoncnaTocka)
```



Slika 15: Iskalnik objektov (Object Browser)

Če želimo z nekim objektom delati, moramo vedeti, na katero mesto v razporeditveni lestvici sodi. Zgradbo razporeditvene lestvice najhitreje prikažemo tako, da v AutoCAD-u izberemo *Help → Developer Help*. Tam poiščemo AutoCAD-ov objektni model (Slika 16).
Poglejmo si preprost primer, kjer želimo v spremenljivko tipa String zapisati ime trenutno aktivne risalne ravnine. V pomoč nam je lastnost `Name`, ki je ena od lastnosti risalnih ravnin. Pove nam ime risalne ravnine.

```
Dim RisRavnina As AcadLayer
Dim ImeRisRavnine As String

Set RisRavnina = ThisDrawing.ActiveLayer
ImeRisRavnine = RisRavnina.Name
```



Slika 16: AutoCAD-ov objektni model (Object Model)

3.6 Kontrolne strukture

Med kontrolne strukture prištevamo pogojne stavke ter zanke.

3.6.1 Pogojni stavki

Večkrat se moramo na nekem mestu v programu odločiti, kaj naj program naredi. V tem primeru postavimo pogoj. Če je pogoj izpolnjen, se izvede prvi del kode, v nasprotnem primeru pa se ta del preskoči in se izvede drugi del kode. VBA pozna tri vrste pogojnih stavkov: stavek *If...Then*, stavek *If...Then...Else* in stavek *Select Case*.

Stavek If...Then

Stavek nastopa v dveh oblikah: kot enovrstični stavek in kot stavek, ki ga zaključimo z *End If*
Enovrstični stavek *If...Then* uporabimo takrat, kadar bi radi izvedli le en stavek in to takrat, ko

je pogoj izpolnjen. Oblika zapisa je:

```
If Pogoj Then Stavek
```

Ukaz se izvede, če je pogoj izpolnjen (ima vrednost *True*). Pogoj je poljuben izraz, ki ima logično vrednost. Primer:

```
If Stevilo < 10 Then Log = True
```

V stavku If lahko, kadar je pogoj izpolnjen, izvedemo tudi več stavkov. Ti stavki sestavljajo blok ukazov. Za zadnjim ukazom v bloku zapišemo besedi *End If*, ki označujeta konec bloka. Oblika zapisa je:

```
If Pogoj Then
     Stavki
End If
```

Pri obeh oblikah velja da, če pogoj ni izpolnjen, se izvajanje nadaljuje na stavku, ki stavku *If* sledi. V primeru enovrstičnega pogojnega stavka je to naslednja vrstica, pri drugi obliki pa stavek, ki sledi *End If*.

Stavek If...Then...Else
Kadar želimo, da se nekaj izvede, če je pogoj izpolnjen, nekaj drugega pa, če pogoj ni izpolnjen, uporabimo obliko:

```
If Pogoj Then
     Stavki1
Else
     Stavki2
End If
```

Če je pogoj izpolnjen, se bodo izvedli *Stavki1*, če pa ne, pa *Stavki2*. Potem se izvajanje nadaljuje za *End If*.

Stavek Select Case
VBA pozna še en pogojni stavek – *Select Case*. Ta je podoben stavku *If*, omogoča pa vejitev glede na vrednost nekega izraza. S stavkom *Select Case* najprej določimo izraz, ki ga bomo preverjali, nato pa s stavki *Case* podamo možne vrednosti tega izraza in ukaze, ki se izvedejo, kadar izraz zavzame te vrednosti.

```
Select Case Izraz
     Case Vrednost1
          Stavki
     Case Vrednost2
          Stavki
          …
     Case Vrednostn
          Stavki
     Case Else
          Stavki
End Select
```

Če izraz ne zavzame nobene vrednosti, navedenih v *Case* delu se izvedejo stavki v *Case Else* delu, če tega ni pa se izvajanje takoj nadaljuje za *End Select*. Primer takega stavka je npr.:

```
Dim Material As String
Dim StKov, StKam, StKer, StKost, StBron, StStek, StNez As Integer
 ...
Select Case Material
       Case "Kovina"
          StKov = StKov + 1
       Case "Kamen"
          StKam = StKam + 1
       Case "Keramika"
          StKer = StKer + 1
       Case "Kost"
          StKost = StKost + 1
       Case "Bron"
          StBron = StBron + 1
       Case "Steklo"
          StStek = StStek + 1
       Case Else
          StNez = StNez + 1
End Select
```

Tovrstno kodo MiniExplorer uporablja pri štetju posameznih kosov najdb, ki se nahajajo v bazi. V baznem stolpcu se nahajajo vrste materiala posameznih najdb (keramika, kamen, steklo...). Ko posamezno vrstico v bazi preberemo, se vrsta materiala zapiše v spremenljivko Material. Stavek Select Case nato preveri, kateri niz je zapisan v spremenljivki Material, ter ga primerja z nizom pod posameznim Case stavkom. Če torej spremenljivka Material vsebuje niz Kamen, se spremenljivka StKam poveča za ena. Če se material ne sklada z nobeno vrednostjo posameznih Case stavkov, se poveča število neznanih najdb (StNez).

3.6.2 Zanke

Zanke uporabljamo pri izvajanju opravil, ki jih je treba večkrat ponoviti.

Stavek For...Next

Stavek oziroma zanko *For...Next* uporabljamo takrat, kadar moramo neko dejanje večkrat ponoviti. Zanka se začne s stavkom *For* in konča s stavkom *Next*. Za rezervirano besedo *For* zapišemo spremenljivko, ki je števec zanke. Ta torej določa, kolikokrat se zanka ponovi. Števec se povečuje ali zmanjšuje po 1. Vendar se lahko odločimo tudi za kako drugo vrednost, ki jo napišemo za besedo *Step*. Ko se zanka izvede prvič, se uporabi začetna vrednost števca. Pri naslednjem izvajanju zanke se števec poveča ali zmanjša za korak, ki smo ga podali s *Step*. Izvajanje zanke se ponavlja, dokler je pogoj izpolnjen, tj. dokler je števec manjši ali enak končni vrednosti (oziroma večji ali enak, če je korak negativen).

```
For Števec = Začetna_vrednost To Končna_vrednost [Step Velikost_koraka]
     Stavki
Next Števec
```

Običajno se zanka *For...Next* ponovi za vse vrednosti števca. Vendar lahko izvajanje zanke prekinemo že prej. To naredimo tako, da na mestu, kjer želimo izvajanje zanke prekiniti, zapišemo stavek *Exit For*. Ta stavek običajno zapišemo v kombinaciji s stavkom *If*, tako da podamo pogoj, ki mora biti izpolnjen, da se izvajanje zanke prekine.

Stavek For Each...Next

Stavek *For Each...Next* je zelo podoben stavku *For...Next*. Razlika je v tem, da izvrši ukaze na vseh objektih v zbirki (*collection*) ali na vseh elementih polja (*array*). Zelo koristen je takrat, kadar ne poznamo njihovega števila.

```
For Each Element In Zbirka
     Stavki
Next Element
```

Poglejmo si za zgled primer, kjer želimo prešteti vse risalne ravnine na neki risbi. Deklariramo spremenljivko tipa risalna ravnina ter se sprehodimo skozi celo zbirko risalnih ravnin. Število risalnih ravnin na začetku je 0, na vsakem koraku pa število povečamo za 1. Končni rezultat je število vseh risalnih ravnin na neki risbi.

```
Dim RisRavnina As AcadLayer
Dim StRisRavnin As Integer

StRisRavnin = 0
For Each RisRavnina In ThisDrawing.Layers
    StRisRavnin = StRisRavnin + 1
Next RisRavnina
```

Stavek Do...While

V zanki *Do...While* se stavki ponavljajo, vse dokler je pogoj izpolnjen. Ko pogoj ni več izpolnjen, se izvajanje prekine, program pa se nadaljuje s prvim stavkom, ki sledi stavku *Loop,* s katerim sicer končamo zanko. Pogoj lahko postavimo na začetek ali konec zanke. V drugem primeru se zanka izvede vsaj enkrat, ne glede na pogoj. V prvem primeru mora biti pogoj že na začetku izpolnjen, če želimo, da se zanka izvede vsaj enkrat. Zanka ima, glede na mesto, kjer postavimo pogoj, dve obliki:

```
Do While Pogoj
     Stavki
Loop
```

ali pa

```
Do
     Stavki
Loop While Pogoj
```

Tudi zanko *Do...While* lahko po potrebi prekinemo, še preden se izvede do konca. Tedaj uporabimo ukaz *Exit Do*.

3.7 Procedure in funkcije

Programsko kodo običajno razdelimo na manjše logične enote, ki jim pravimo procedure. Uporaba procedur je koristna, kadar nameravamo ista opravila izvajati na več mestih v programu. Iskanje napak v programu je bistveno lažje, če je koda razdeljena na procedure. Procedure, ki smo jih izdelali v enem programu, lahko brez večjih omejitev uporabljamo tudi v drugih programih. V VBA se srečujemo s tremi vrstami procedur. Najprej so tu navadne procedure (`Sub procedure`), ki opravijo neko delo, a ne vrnejo nobene vrednosti. Med posebne procedure sodijo funkcije (`Function procedure`), ki vedno vrnejo neko

vrednost. Na koncu omenimo še procedure, s katerimi določamo lastnosti (`Property procedure`) in lahko z njimi prirejamo vrednosti ter se sklicujemo na objekte.

3.7.1 Navadne procedure

Navadna procedura je sestavljena iz kode, ki se izvede, ko jo pokličemo ali, ko se sproži nek dogodek. Vstavimo jo v obrazec, standardni (*standard module*) ali razredni (*class module*) modul. Oblika zapisa je:

```
[Private | Public][Static] Sub Ime_procedure(Argumenti)
     Stavki
End Sub
```

Argumente v proceduri najavimo tako kot pri najavi spremenljivk. V VBA poznamo dve vrsti navadnih procedur: splošne procedure (*general procedure*) in dogodkovne procedure (*event procedure*). V splošni proceduri povemo programu, kako naj izvede neko opravilo. Splošno proceduro moramo v programu eksplicitno poklicati, ko jo želimo uporabiti. Na drugi strani pa dogodkovna procedura ni dejavna, dokler se ne sproži dogodek, ki jo požene npr. klik na miškin gumb, premik miške ...

Splošne procedure

Splošne procedure so koristne, kadar imamo kodo, za katero želimo, da se izvede ob več različnih dogodkih. Namesto da bi enako kodo zapisovali na vsa ta mesta, si pripravimo proceduro, ki jo nato po potrebi kličemo.
Kot zgled napišimo preprosto splošno proceduro, ki naredi vidne samo modre risalne ravnine. Vse ostale risalne ravnine izklopimo.

```
Public Sub PrikaziModreRisRavnine()
'spremenljivka tipa risalna ravnina
Dim RisRavnina As AcadLayer

'za vse risalne ravnine v zbirki
For Each RisRavnina In ThisDrawing.Layers
    'če je privzeta barva risalne ravnine modra
    If RisRavnina.TrueColor.ColorIndex = acBlue Then
        'risalno ravnino vklopimo
        RisRavnina.LayerOn = True
    'če privzeta barva risalne ravnine ni modra
    Else
        'risalno ravnino izklopimo
        RisRavnina.LayerOn = False
    End If
Next RisRavnina
End Sub
```

Še več zgledov pa je v razdelku Projekt MiniExplorer, kjer opisujemo program MiniExplorer.

Dogodkovne procedure

Ko nek objekt, kot je obrazec ali kak drug gradnik, prepozna dogodek, kot je klik, premik miške in podobno, se samodejno sproži dogodkovna procedura, ki smo jo v kodi povezali s tem dogodkom. Ker ime te procedure vzpostavi povezavo med objektom in kodo, bi lahko rekli, da so dogodkovne procedure pripete obrazcem in gradnikom.
Kako zapisujemo dogodkovne procedure? Kadar gre za proceduro, povezano z nekim gradnikom, zapišemo ime gradnika, ki mu sledi znak _, na koncu pa še ime dogodka. Za zgled

zapišimo začetno vrstico procedure, ki se izvede ob kliku na ukazni gumb z imenom DodajNovoRavnino:

```
Private Sub DodajNovoRavnino_Click ()
```

Več o dododkovnih procedurah si bomo ogledali v razdelku Obrazci in gradniki.

Klicanje procedur
Kako proceduro kličemo, je odvisno od vrste in mesta procedure ter načina njene uporabe v programu. Ker procedura ne vrne nobene vrednosti, njenega imena ne moremo uporabljati v izrazih. Zato je klic procedure vedno samostojen stavek. Proceduro *Sub* lahko kličemo na dva načina. Pri prvem zapišemo stavek *Call*, nato ime procedure ter na koncu med oklepaje navedemo še argumente.

```
Call Ime_procedure (Argumenti)
```

Pri drugem načinu zapišemo samo ime procedure, ki ji sledijo argumenti, v tem primeru brez oklepajev.

```
Ime_procedure Argumenti
```

Procedure, ki so zapisane v drugih modulih, je moč priklicati na kateremkoli mestu v programu. Včasih moramo poleg imena procedure podati še modul, v katerem je zapisana. To moramo storiti takrat, ko imamo več procedur z istim imenom napisanih v različnih modulih.

3.7.2 Funkcije

Funkcija je posebna vrsta procedure, za katero je značilno, da vrne neko vrednost. VBA pozna celo vrsto vgrajenih funkcij. Lahko sestavljamo tudi svoje, kar počnemo s stavkom Function.

```
[Private | Public][Static] Function Ime_funkcije (Argumenti) [As Tip]
     Stavki
End Function
```

Funkcije imajo, podobno kot spremenljivke, določen tip podatkov, ki ga vrnejo. Tega zapišeno za besedo *As*. Če to izpustimo, funcija vrača rezultat tipa Variant. Za razliko od programskega jezika java tukaj ne uporabljamo stavka *return*. Da določimo, kakšno vrednost vrača funkcija, uporabimo spremenljivko z enakim imenom, kot jo ima funkcija. Vrednost te spremenljivke ob zaključku funkcije bo vrednost, ki jo vrne funkcija. Za zgled napišimo funkcijo, ki nam spremenljivko tipa Integer spremeni v niz.

```
Public Function Pretvornik(Stevilo As Integer) As String
     Pretvornik = Str(Stevilo)
End Function
```

Funkcijo nato lahko kličemo kot funkcije, ki so vgrajena v VBA. Primer:

```
Dim Stevilo As Integer
Dim Niz As String

Stevilo = 17
Niz = Pretvornik(Stevilo) → "17"
```

3.8 Nizi in števila

Nizi in števila so podatki, s katerimi se pri pisanju programov največ srečujemo. Niz je tista vrsta podatka, ki ga v programu MiniExplorer največ uporabljamo. Zato si bomo ogledali nekaj najpomembnejših vgrajenih funkcij, ki so namenjeni delu z nizi. Za primerjalne nize sem določil imena risalnih ravnin, ker ravno s temi nizi v MiniExplorerju največ operiram.

3.8.1 Funkcije nad nizi

Asc

Funkcija Asc vrne kodo ASCII prvega znaka v nizu. Koda ASCII je seveda celo število. Oblika zapisa je:

```
Asc(Niz)
```

Nekaj primerov:

```
Asc("A") → 65
Asc("a")  → 97
Asc("SE 0118") → 83
```

Chr

Funkcija Chr ima nasprotni učinek kot funkcija Asc. Vrne namreč znak, ki ima to kodo.

```
Chr(ASCII_koda)
```

InStr

Funkcija InStr vrne mesto, kjer se nek niz prvič pojavi znotraj danega niza.

```
InStr ([Začetek,]Niz1,Niz2[, Primerjava])
```

Prvi argument (celo število) ni obvezen, določa pa mesto, kjer se začne iskanje. Če ga izpustimo, se iskanje začne pri prvem znaku. Drugi in tretji argument sta obvezna. Argument Niz1 označuje niz, v katerem iščemo drug niz, ki ga podamo z argumentom Niz2. Zadnji argument, ki je spet število, ni obvezen, določa pa način primerjave znakov. Pri klicu za zadnji argument običajno uporabimo eno od vgrajenih konstant. Možnosti primerjav lahko vidimo v spodnji tabeli. Če zadnjega argumenta ne upoštevamo, se privzame konstanta -1 (Option Compare).

Vrednost	Opis
-1 - vbUseCompareOption	Primerjava v skladu z nastavitvijo Option Compare.
0 - vbBinaryCompare	Binarna primerjava.
1 - vbTextCompare	Tekstovna primerjava.
2 - vbDatabaseCompare	Primerjava na osnovi podatkov iz podatkovne zbirke v formatu programa Microsoft Access.

Tabela 2: Konstante za primerjavo

V nasprotju s polji (in javo), je indeks prvega znaka 1 in ne 0. Poglejmo si primer primerjave dveh nizov:

```
Dim Niz1 As String
```

```
Dim Niz2 As String
Dim LogSpr As Boolean


Niz1 = "OPR06_Arh_SE 0118_linija"
Niz2 = "SE 0118"
LogSpr = False

If InStr(Niz1, Niz2) > 0 then
     LogSpr = True
End If
```

V primeru preverimo, če je Niz2 vsebovan v nizu Niz1. Vsebovan je takrat, ko nam funkcija InStr vrne število večje od nič. Poglejmo si še nekaj primerov, kjer uporabljamo različne načine primerjave zankov.

```
Dim Niz1 As String
Dim Niz2 As String

Niz1 = "OPR06_Arh_SE 0118_linija"
Niz2 = "se 0118"

'če za primerjalni algoritem uporabimo 0 (binarna primerjava)postane
'funkcija neobčutljiva na male in velike črke, zato mora funkcija v nizu
'Niz1 najti enako besedo, kot je besedica Niz2; ker v nizu Niz1 ne najde
'besede iz niza Niz2, funkcija vrne vrednost 0
InStr(Niz1,Niz2,0) → 0

'če za primerjalni algoritem uporabimo tekstovno primerjavo, postane
'funkcija neobčutljiva na male in velike črke, tako da nam v našem
'primeru vrne število 11
InStr(Niz1,Niz2,1) → 11
```

Če se odločimo za način primerjave Option Compare, določimo način primerjave znotraj modula. S tem lahko dosežemo, da ima vsak modul svoj način primerjave. Njegova sintaksa je naslednja:

```
Option Compare{Binary | Text | Database}
```

Ta stavek mora stati pred vsemi procedurami v modulu. Če stavka ne uporabimo, je privzeta primerjava binarna.
Poglejmo si to na primeru. Ustvarimo nov modul z imenom `TestniModul`. Na njem zapišemo proceduro `TestnaProcedura`, kjer s funkcijo InStr preverjamo, če se Niz2 nahaja v nizu Niz1:

```
'pred proceduro zapišemo primerjavo; v primeru smo uporabili tekstovno
'primerjavo
Option Compare Text
'v modul zapišemo proceduro TestnaProcedura
Public Sub TestnaProcedura()
Dim Niz1 As String, Niz2 As String
Dim Odg As Varinat
Niz1 = "SE 0118"
Niz1 = "se 0118"

'ker je primerjava tekstovna, niz Niz2 najdemo v nizu Niz1
```

```
Odg = InStr(Niz1, Niz2) → 1
End Sub
```

Primerjavo s konstanto 2 (vbDatabaseCompare) uporabimo, ko delamo s podatki, ki izvirajo iz baze Microsoft Access.

Len
Funkcija Len vrne dolžino niza. Oblika zapisa je:

```
Len(Niz)
```

Nekaj primerov:

```
Len("SE 0118") → 7
Len("") → 0
Len("OPR06_Arh") → 9
```

Mid
Funkcija Mid vrne del niza.

```
Mid(Niz, Začetek [, Dolžina])
```

Drugi argument predstavlja indeks prvega znaka v nizu, ki ga vrne funkcija. Če je indeks slučajno večji od dolžine niza, je rezultat prazen niz. Tretji argument ni obvezen, določa pa dolžino vrnjenega niza. Če ga ne zapišemo, dobimo vse znake od začetne lege do konca niza.

Nekaj primerov:

```
Mid("OPR06_Arh_SE 0118_linija", 11, 7) → "SE 0118"
Mid("OPR06_Arh_SE 0118_linija", 1, 5) → "OPR06"
Mid("OPR06_Arh_SE 0118_linija", 19, 6) → "linija"
```

Split
Funkcija Split razdeli niz na več podnizov, ki postanejo elementi polja (array).

```
Split(Niz [, Ločilo] [, Število_podnizov] [, Primerjava])
```

Prvi argument je niz, ki ga želimo razdeliti, drugi pa določa ločilo med posameznimi podnizi oz. besedami. Če ga ne zapišemo, se upošteva privzeto ločilo – presledek. Ločilo torej pove od kje do kje je en element. S tretjim argumentom določimo število podnizov, ki naj jih funkcija vrne. Če ne zapišemo ničesar, vrne funkcija vse podnize. Četrti argument določa način primerjave ob iskanju ločila v nizu, ko niz delimo. Kakšni načini primerjave so na voljo, smo opisali pri funkciji InStr (glej Tabelo 2). Poglejmo si na preprostem primeru, kako dani niz razdelimo na več podnizov in jih zapišemo v polje:

```
Dim Besedilo As String
Dim PoljeBesed() As String

Besedilo = " OPR06_Arh_SE 0118_linija "
'funkcija Split ob znaku _ razdeli besedilo na podnize, ter jih zapiše v
'polje; npr. v PoljeBesed(2) je niz "SE 0118", v PoljeBesed(3) pa niz
'"linija"
PoljeBesed = Split(Besedilo,"_")
```

Str
Funkcija Str pretvori število v niz. Oblika zapisa je:

```
Str(Število)
```

Poglejmo si preprost primer, kjer število tipa Double spremenimo v niz :

```
Dim Stevilo As Double
Dim SteviloVniz As String

Stevilo = 3.14
SteviloVniz = Str(Stevilo) → "3.14"
```

StrComp
Funkcija StrComp vrne rezultat primerjave dveh nizov. Oblika zapisa je:

```
StrComp(Niz1, Niz2 [, Primerjava])
```

Prva dva argumenta sta niza, ki ju želimo primerjati. Tretji argument ni obvezen, določa pa način primerjave.
Če je prvi niz manjši od drugega, potem vrne funkcija vrednost -1. Kadar je prvi niz večji, je rezultat 1, če pa sta niza enaka, funkcija vrne vrednost 0. Kakšni so načini primerjave, ki jih imamo na voljo, smo opisali pri funkciji InStr (glej Tabelo 2).

LTrim, RTrim, Trim
Funkcija LTrim vrne niz, iz katerega odstrani vse presledke, ki so pred prvim od presledka različnim znakom v nizu. Podobno funkcija RTrim vrne niz, kjer argumentu odstrani vse presledke za zadnjim znakom, ki ni presledek. Najbolj splošna je funkcija Trim, ki odstrani vse presledke pred prvim in za zadnjim znakom v nizu.

```
LTrim("   SE 0118  ") → "   SE 0118"
RTrim("   SE 0118  ") → "SE 0118  "
LTrim("   SE 0118  ") → "SE 0118"
```

Left in Right
Funkcija Left vrne prvih nekaj znakov, funkcija Right pa zadnjih nekaj znakov v danem nizu.

```
Left("OPR06_Arh_SE 0118_linija", 5) → "OPR06"
Right("OPR06_Arh_SE 0118_linija", 6) → "linija"
```

Drugi argument torej določa, koliko znakov z leve oz. desne strani niza vrne funkcija.

3.8.2 Matematične funkcije
Navedimo nekaj matematičnih funkcij, ki so v programu MiniExplorer najpogosteje uporabljene.

Abs
Funkcija Abs vrne absolutno vrednost števila.

```
Abs(Število)
```

Cos
Funkcija Cos vrne kosinus števila, ki je kot, izražen v radianih.

```
Cos(Število)
```

Exp
Funkcija Exp izračuna eksponent števila pri osnovi *e*.

```
Exp(Število)
```

Fix
Funkcija Fix vrne celi del realnega števila. Če je število negativno, vrne prvo celo število, ki je večje ali enako danemu številu.

```
Fix(Število)
```

Nekaj primerov:

```
Fix( 1.89 )  → 1
Fix( -2.33 ) → -3
```

Round
Funkcija Round zaokroži število na izbrano število decimalk.

```
Round (Število [, Decimalke])
```

Drugi argument ni obvezen, določa pa število decimalk, ki jih dobimo pri zaokrožitvi števila. Če ga ne zapišemo, funkcija zaokroži število na najbližje celo število.

Sqr
Funkcija Sqr vrne kvadrat števila.

```
Sqr(Število)
```

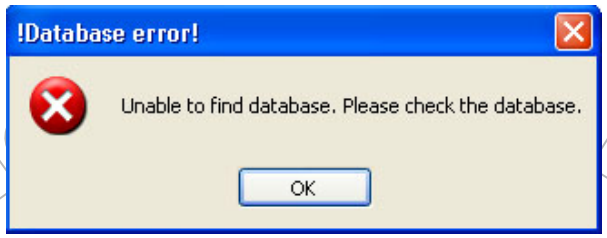
3.9 Sporočilna okna

V okolju Windows se pogosto srečujemo z raznimi okenci, v katerih se prikazujejo sporočila. Včasih moramo v taka okenca kaj vpisati ali pa se odločiti za kako možnost, tako da kliknemo na ukazni gumb. VBA pozna dve osnovni vrsti sporočilnih oken. To sta *MsgBox* in *InputBox*. V prvem nam program izpisuje razna sporočila. Drugega pa običajno uporabimo za to, da preko njega podamo dodatne podatke, ki jih program potrebuje.

3.9.1 Funkcija MsgBox

Funkcija *MsgBox* je namenjena prikazu okna z raznimi opozorili in sporočili. V njem je torej sporočilo ter vsaj en ukazni gumb. Gumbov je lahko tudi več, poleg tega pa so lahko v oknu tudi razne ikone.

```
MsgBox(Sporočilo [, Gumbi][, Naslov][, Pomoč, Kontekst])
```



Slika 17: Sporočilno okno MsgBox z gumbom OK

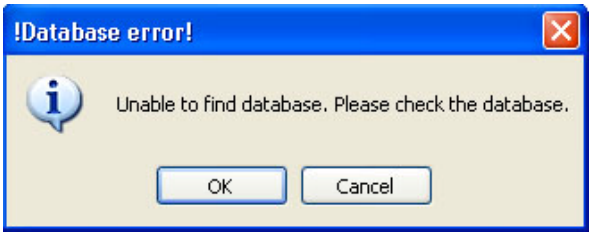
Od vseh argumentov je obvezen le prvi – sporočilo. Če želimo sporočilo zapisati v več vrsticah, ustvarimo prelom tako, da v niz vtaknemo znak za prehod v novo vrsto. To najlažje naredimo z vgrajeno konstanto *vbCrLf*. Tako se niz

```
"Unable to find database." + vbCrLf + "Please check the database."
```

izpiše kot

```
Unable to find database.
Please check the database.
```

Kot drugi argument običajno uporabimo vsoto konstant, ki določa kombinacijo gumbov, ikone in privzetega gumba. Tako je *vbYesNo* konstanta, ki pove, da se pojavita gumba *Yes* in *No*, *vbOKCancel* konstanta za kombinacijo gumbov *OK* in *Cancel*. Če argument spustimo, dobimo (kot na Sliki 17) gumb *OK* ter ikono kritične napake. Z izrazom *vbExclamation* + *vbOKCancel* dobimo sliko, kot kaže zgled:



Slika 18: Sporočilno okno MsgBox z gumboma OK in Cancel

Tretji argument je besedilo, ki se izpiše v naslovni vrstici sporočilnega okna (na Sliki 18 napis !Database Error!). Četrti argument je ime datoteke s pomočjo. Kadar ga zapišemo, moramo obvezno podati tudi peti argument, ki nam pove, kje v omenjeni datoteki dobimo pomoč o tem sporočilu.
Funkcija *MsgBox* nam kot rezultat vrne celo število od 1 do 7, kar pove na kateri gumb je uporabnik kliknil. Vrednosti, ki jih prejmemo s klikom na različne gumbe, si lahko ogledamo v Tabeli 3.

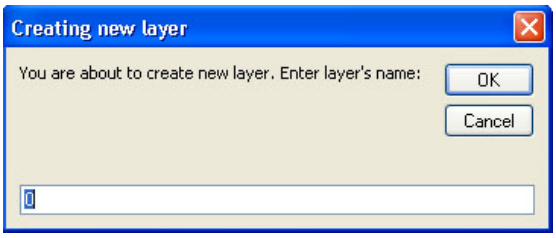
vrnjena konstanta	Vrednost	Opis
vbOk	1	Uporabnik je kliknil gumb OK
vbCancel	2	Uporabnik je kliknil gumb Cancel
vbAbort	3	The User Clicked Abort
vbRetry	4	The User Clicked Retry
vbIgnore	5	The User Clicked Ignore
vbYes	6	The User Clicked Yes
vbNo	7	The User Clicked No

Tabela 3: Konstante gumbov

3.9.2 Funkcija InputBox

Funkcija *InputBox* prikaže okno s sporočilom, v katerem je tudi vnosna vrstica, kamor uporabnik zapiše poljubne znake. Funkcija vrača niz znakov, ki jih je uporabnik vpisal.

```
InputBox(Sporočilo[, Naslov][, Privzeto][, X][, Y][, Pomoč, Kontekst])
```



Slika 19: Sporočilno okno InputBox

Tudi tu je obvezen le prvi argument. Drugi argument je besedilo, ki se izpiše v naslovni vrstici okna. Tretji argument je privzeti niz znakov, ki se izpiše v okencu, še preden karkoli vpišemo. Nato sta dva argumenta, ki določata lego okenca. Če ju ne podamo, se okno odpre na sredini ekrana. Zadnja dva argumenta sta kot pri funkciji *MsgBox* namenjena pomoči.
Če kliknemo gumb OK, funkcija *InputBox* vrne vsebino okenca. Kadar pa kliknemo gumb Cancel, funkcija vrne prazen niz.
Če bi želeli s pomočjo te funkcije prebrati npr. kot, za katerega želimo zavrteti nek lik in ga shraniti v spremenjlivko *KotZasuka* tipa *Integer*, bi napisali:

```
Dim NizKotaZasuka As String
Dim KotZasuka As Iteger

'vpišemo stopinje v sporočilno okno
NizKotaZasuka = InputBox("Enter angle value:", "Angle")
'vrnjeni niz NizKotaZasuka pretvorimo v celo število KotZasuka s pomočjo
'funkcije CInt
KotZasuka = CInt(NizKotaZasuka)
```

Seveda moramo vpisati v sporočilno okno celo število, drugače pri pretvorbi niza v število pride do napake.

3.10 Obrazci in gradniki

Vsak program, napisan v VBA, lahko razdelimo na dva dela: programsko kodo in programski vmesnik. Programska koda se izvaja v ozadju, tako da jo uporabnik nikoli ne vidi, vidni del je

le obrazec. Ta je pravzaprav osnova programskega vmesnika, ki je namenjen komuniciranju med uporabnikom in programom. Obrazec je objekt, ki vsebuje gradnike, kot so okenca za besedilo, izbirni gumbi, pritrditvena polja, ukazni gumbi, seznami, drsniki in še mnoge druge. Seveda so tudi gradniki objekti s svojimi lastnostmi, postopki in dogodki. Skupno ime za obrazec z gradniki je pogovorno okno.
Podrobneje si bomo zgled izdelave takega pogovornega okna ogledali v razdelku Projekt MiniExplorer, ko bomo opisali način izgradnje pogovornega okna programa MiniExplorer.

3.10.1 Izdelava pogovornih oken

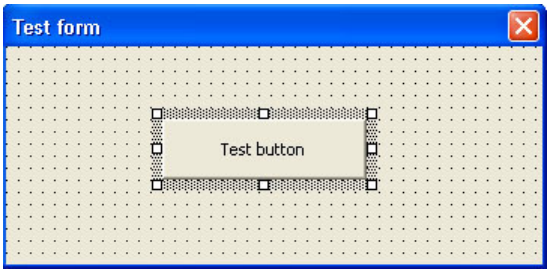
V VBA se pogovorna okna imenujejo *uporabniški obrazci (UserForm)*. Naredimo jih tako, da izberemo *Insert* → *UserForm*. VBA naredi prazno okno, ki bo šele postalo pogovorno okno. Hkrati se pokaže orodjarna (glej poglavje Razvojno okolje). Orodjarna vsebuje orodja za izdelavo pogovornih oken.

3.10.2 Spreminjanje lastnosti pogovornega okna

Ko dobimo nov obrazec, ga najprej preimenujemo. V oknu *Properties* poiščemo lastnost *Name* in jo spremenimo iz privzetega imena *UserForm1* v smisleno ime, ki nam bo povedalo za kakšen obrazec gre (glej poglavje Razvojno okolje).
Ena od lastnosti, ki jo bomo običajno spreminjali, je tudi *Caption* (naslov). Pogovorno okno mora imeti naslov, ki pove njegov namen. Ko v okno z lastnostmi tipkamo naslov, se sočasno izpisuje v programskem oknu.

3.10.3 Dodajanje gradnikov na obrazec

Najpogosteje uporabljen gradnik je ukazni gumb. Gradnik te vrste, torej gumb, uporabimo takrat, kadar potrebujemo objekt, na katerega bi radi kliknili, da bi izvedli neko dejanje. Na obrazec ga dodamo tako, da kliknemo ustrezno ikono v orodjarni. Z miško ga premaknemo na obrazec. Na Sliki 20 lahko vidimo obrazec z novim ukaznim gumbom.



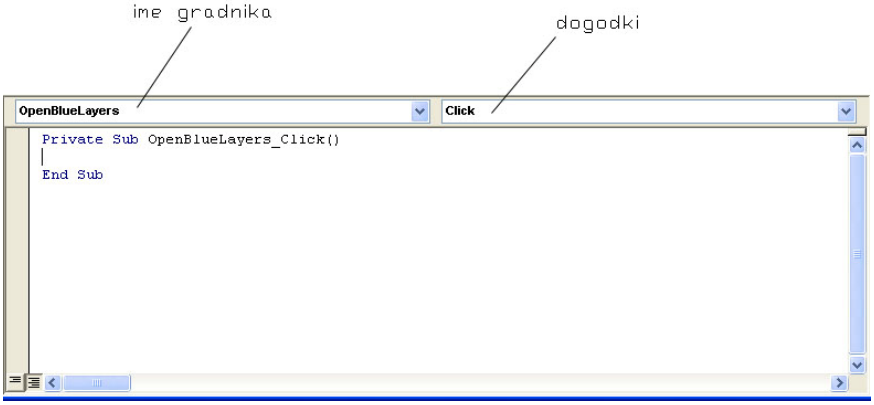
Slika 20: Obrazec z ukaznim gumbom

Okvir in ročke kažejo na to, da je gumb trenutno izbrani predmet. Gumb lahko poljubno prestavljamo po obrazcu. Z levim gumbom miške ga primemo in prestavimo na željeno mesto. V obrazec vedno dodamo toliko gumbov, kolikor jih potrebujemo. Če že vnaprej vemo, koliko gumbov bomo potrebovali, običajno dodamo vse hkrati.
Tako kot ima obrazec svoje lastnosti, jih ima tudi vsak gradnik. Navadno spremenimo vsaj napis na gumbu (na Sliki 20 - Test button) in ime gumba.
Podobno velja tudi za druge gradnike. Več o tem pa si bomo ogledali pri opisu izgradnje uporabniškega vmesnika programa MiniExporer.

3.10.4 Pisanje kode v VBA za ukazni gumb

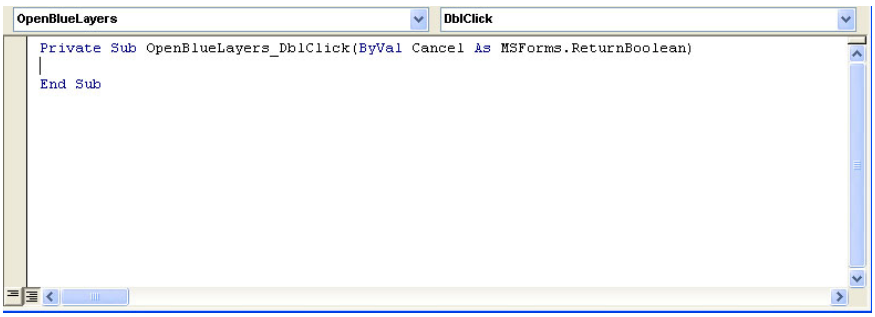
Ko pišemo kodo za gradnike, se moramo zavedati, da imamo opravka z dogodkovnim programiranjem. Ko na primer kliknemo gumb, se sproži dogodek klik na gumb. Tak dogodek lahko prestrežemo in povemo, kaj naj se ob tem dogodku zgodi. V ta namen moramo

napisati ustrezno dogodkovno proceduro. To storimo tako, da dvokliknemo gradnik. Odpre se okno s programsko kodo. Koda v oknu s pogramsko kodo se zapiše samodejno (glej Sliko 21). Dvoklik nas avtomatično postavi znotraj kode privzete dogodkovne procedure za kliknjeni objekt. Če bi radi napisali proceduro za kak drug objekt in/ali kak drug dogodek, moramo izbrati nov par. V oknu s programsko kodo sta dva padajoča seznama. V prvem se nahajajo imena vseh gradnikov, ki so na obrazcu, kjer smo gradnik dvokliknili. V drugem pa vsi dogodki, ki se lahko na izbranem gradniku zgodijo.
Prazne dogodkovne procedure (torej take kamor nismo napisali prav nič kode), okolje VBA avtomatsko zbriše.



Slika 21: Izbor dogodka za gradnik

Na Sliki 21 imamo primer, kjer se koda za izbran gradnik sproži, ko na njega kliknemo. Če bi hoteli, da se koda sproži ob dvokliku, bi izbrali sledeče:



Slika 22: Dogodek dvoklik

3.10.5 Prikaz pogovornih oken in zagon programa

Če želimo naš program pognati, moramo prikazati pogovorno okno in ga uporabiti. Pogovorno okno prikažemo tako, da naredimo standardno proceduro in napišemo kodo, kjer zapišemo *ime_obrazca.Show*.

```
Public Sub RunForm()
  TestForm.Show
End Sub
```

To proceduro z imenom `RunForm` samo še poženemo. To storimo z ukazom *Run → Run* Sub. S tem smo sprožili prikaz pogovornega okna in nadaljnje odvijanje programa bo odvisno od dogodkov, ki se bodo dogajali.Pogovorna okna VBA so modalna, kar pomeni, da morajo

biti zaprta, preden AutoCAD naredi kar koli drugega. Ko je koda v VBA izvedena, pogovorno okno zapremo tako, da na koncu programa napišemo *Unload Me*.

Oglejmo si celotni postopek s katerim bi napisali in uporabili program, ki vsebuje en gumb s klikom katerega naredimo vidne samo risalne ravnine modre barve. Ustrezno proceduro, ki doseže, da so vidne samo risalne ravnine modre barve, smo že zapisali v razdelku Navadne procedure. Dogodek, ki bo gumb sprožil, pa bomo nastavili na dvoklik.

- Naredimo nov obrazec. To storimo tako da v vrstici z meniji kliknemo *Insert* → *UserForm*. V oknu Properties mu spremenimo ime (name) ter naslov (caption). Poimenujmo ga `TestForm`. Enako nastavimo tudi napis (naslov).
- V orodjarni izberemo gradnik gumb ter ga z miško prestavimo na obrazec `TestForm`. Preimenujemo ga v `OpenBlueLayers`.
- Dvokliknemo na ustvarjeni gumb. Odpre se okno s programsko kodo. Med možnimi dogodki za ta gradnik izberemo dvoklik. V tej dogodkovni proceduri pokličemo proceduro, ki smo jo zapisali v razdelku Navadne procedure. Ob koncu obvezno zapišemo `Unload me`, da se obrazec ob koncu zapre. Zapišemo torej sledečo kodo:

```
Private Sub OpenBlueLayers_DblClick(ByVal Cancel As_
                                    _MSForms.ReturnBoolean)
  Call PrikaziModreRisRavnine
  Unload Me
End Sub
```

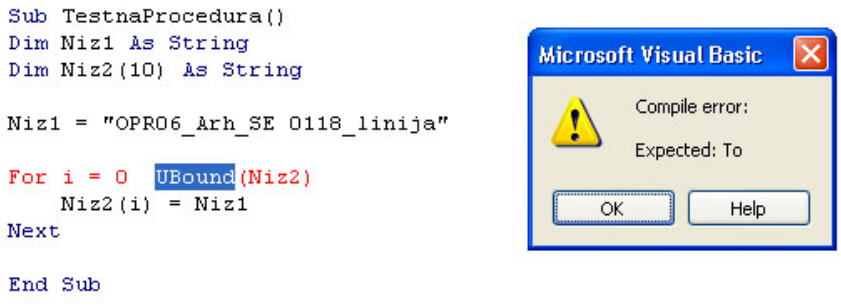
- Sedaj moramo samo še pognati pogovorno okno. To storimo tako, da ustvarimo novo navadno proceduro ter v njo zapišemo sledečo kodo:

```
Public Sub RunForm()
  TestForm.Show
End Sub
```

- Sedaj proceduro z imenom `RunForm` samo še poženemo. To storimo z ukazom *Run* → *Run Sub*.
- Če bomo kliknili na gumb, bomo na risbi videli le tiste objekte v AutoCADu, ki so na ravninah s privzeto modro barvo. Okno z gumbom se bo avtomatsko zaprlo.

3.11 Odpravljanje in lovljenje napak

Žal se v programe skoraj vedno prikradejo take ali drugačne napake. Najpreprosteje odkrijemo napake pri tipkanju, ko na primer napačno zapišemo ime spremenljivke, gradnika, konstant in podobno. Med trdovratnejše napake pa sodijo logične napake, ko program sicer deluje, a ne daje pričakovanih rezultatov.
Najbolje je, da se enostavnim tipkarskim napakam ognemo že med pisanjem programa. VBA pozna orodje, ki sproti preverja skladnjo izrazov, medtem ko jih zapisujemo. Takoj, ko v neki vrstici naredimo napako, nas to orodje ob prehodu v novo vrstico opozori. Med tovrstne napake sodijo manjkajoči deli izrazov kot so oklepaji, narekovaji in podobno. Poglejmo si na primeru:



Slika 23: Sporočilo ob napaki

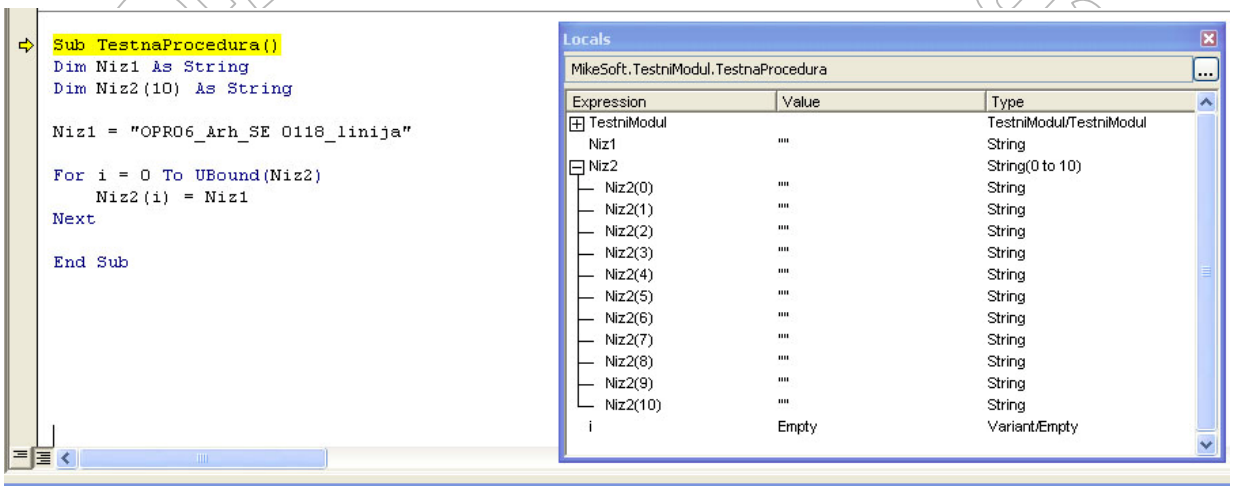
VBA samodejno poskrbi zato, da so imena ključnih besed in lastnosti zapisane z veliko začetnico in prikazana v modri barvi. To zagotavlja, da med tipkanjem naredimo čim manj napak.
Kadarkoli med delovanjem programa pride do napake, se izvajanje programa takoj prekine. Včasih pa bi to radi naredili sami, tudi če ni prišlo do napake. To lahko naredimo med samim izvajanjem programa s pritiskom na kombinacijo tipk *Ctrl + Break*, ali pa z uporabo ukaza *Run → Break*. Naslednja možnost je, da v kodi nastavimo mesto oziroma vrstico, kjer naj se izvajanje programa prekine. Tej točki pravimo prekinitvena točka (Breakpoint). Seveda moramo to narediti v oknu Code, kjer vidimo programsko kodo. Z miško kliknemo v sivo področje, levo od kode ali pa označimo celo vrstico in izvedemo ukaz *Debug → Toggle Breakpoint* (Slika 24).

Slika 24: Prekinitvena točka

Izvajanje programa ob prekinitvi lahko nadaljujemo na več načinov. Lahko nadaljujemo z normalnim izvajanjem, kar naredimo z ukazom *Run → Run Macro*, lahko pa izvedemo samo naslednjo vrstico (*Debug → Step Into*) ali proceduro (*Debug → Step Over*).
Pogosto pride do napake, ker spremenljivka nima pričakovane vrednosti. Tak primer je na primer deljenje s številom 0. Zato je zelo koristno, če lahko med izvajanjem kode sproti spremljamo vrednost spremenljivk. V VBA lahko to počnemo z okni Immediate, Locals ali Watches.
Poglejmo si cel postopek kako s pomočjo okna Locals, spremljamo delovanje programa. Kot primer uporabimo proceduro iz primera.

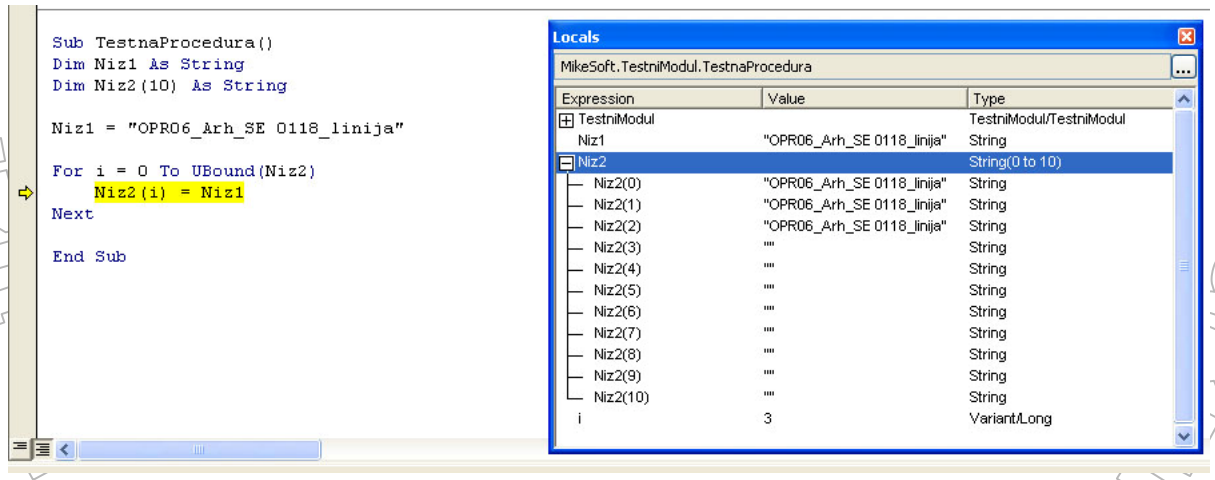
- Kazalec postavimo v prvo vrstico v programu. V meniju *Debug* izberemo *Step Into*. Vrstica, ki se trenutno izvaja, se obarva rumeno, v oknu Locals se nastavijo spremenljivke, ki jih bomo v programu uporabljali (glej Sliko 25). V stolpcu Expression nam v prvi vrstici kaže v katerem modulu imamo zapisano proceduro, ki jo

preverjamo. V našem primeru imamo proceduro TestnaProcedura zapisano na modulu TestniModul. V naslednjih vrsticah prvega stolpca imamo navedene vse lokalne spremenljivke, ki jih v proceduri uporabljamo. Ker je Niz2 polje, seveda vsebuje več nizov. V drugem stolpcu imamo zapisane vrednosti spremenljivk. Na začetku so vsi nizi prazni (Slika 25). V tretjem stolcu imamo zapisane tipe spremenljivk, ki jih v proceduri uporabljamo. Ker spremenljivko *i* nisem posebej definiral, je tipa Variant.



Slika 25: Vrednosti spremenljivk v oknu Locals ob izvedbi prve vrstice

- Ob vsakem novem pritisku *Debug → Step Into* se izvede naslednja vrstica programa. Po desetih korakih izvedbe si lahko rezultate ogledamo na Sliki 26.



Slika 26: Okno Locals po desetih korakih izvedbe

- Če ne pride do napak, imamo ob koncu v oknu Locals zapisane vse vrednosti posameznih spremenljivk. Tako lahko vrednosti primerjamo z našimi pričakovanji. Če med izvajanjem naletimo na napako, program ustavimo z ukazom *Run → Reset*. Napako popravimo ter zopet po korakih preverimo pravilnost delovanja kode.

Seveda pa kljub vsem tem orodjem in postopkom ni nujno, da bomo na ta način odpravili vse napake. Na srečo lahko včasih na določenih mestih v programu vnaprej predvidimo, da tam utegne priti do napake ob izvajanju programa. Tako lahko datoteka, ki jo odpiramo, ne obstaja, med izvajanjem programa lahko pride do deljenja z 0 in podobno. Nekatere od teh

napak lahko pričakujemo ali jih celo moramo pričakovati, drugih seveda ne. VBA, podobno kot programski jezik java, pozna mehanizme, s katerimi lahko prestrezamo napake, ki se zgodijo med izvajanjem programa.
Poglejmo zelo preprosto ogrodje za prestrezanje napak:

```
Sub Prestrezi()
    'napoved spremenljivk
    '...

    On Error GoTo Napaka
    'ostanek procedure
    '...

Napaka:
MsgBox "Napaka" & Err.Number & " " & Err.Description
    Resume Next
End Sub
```

Če med izvajanjem dela podprograma (označenega z ostanek procedure) pride do napake, ki bi drugače povzročila, da bi program nehal delovati (bi se sesul), se to ne zgodi. Dobimo številko napake, ter njen opis. Tako imamo priložnost, da začnemo obravnavati določene napake na ustrezen način.

3.12 Povzetek

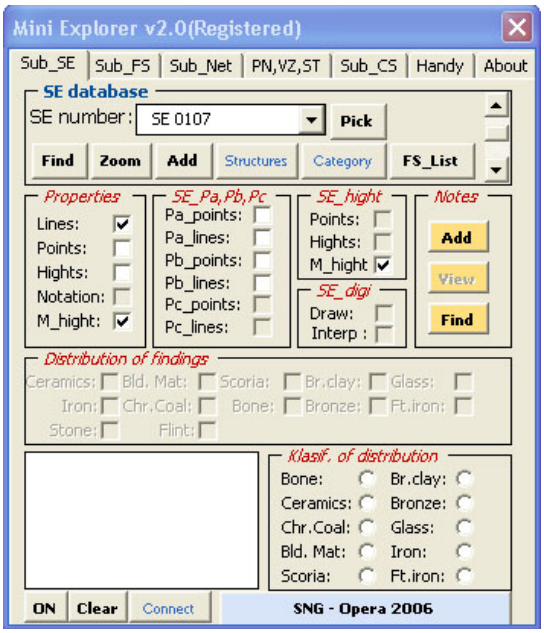
VBA nam v AutoCAD-u ponuja precejšnje možnosti dela z objekti, ki smo jih narisali na risbi. Programiranje je dokaj enostavno in učinkovito. Tako lahko določene postopke, ki bi nam vzeli precej časa, avtomatiziramo.
VBA ni edino orodje za programiranje v AutoCAD-u. Poleg Visual LISP-a obstaja še programsko orodje ObjectARX. ObjectARX temelji na programskem jeziku C++ in omogoča popolno predmetno povezavo z AutoCAD-om. Programi, napisani v tem okolju, so precej hitrejši od tistih, napisanih v VBA.
V naslednjem poglavju si bomo ogledali praktični del diplomske naloge. Nekatere postopke, ki nam v AutoCAD-u vzamejo veliko časa, bomo poskušali s pomočjo ustreznih programov v VBA avtomatizirati in poenostaviti.

4. PROJEKT MINIEXPLORER

MiniExplorer je program znotraj AutoCADa, ki omogoča enostaven vpogled v arheološko dokumentacijo. Na začetku je bil ta program le skupek nekaj enostavnih modulov, ki so arheologom olajševali vsakdanje delo z AutoCADom. Kasneje je postal bolj kompleksen pregledovalnik, ki omogoča urejanje ter pregledovanje arheološke dokumentacije.
Program je razdeljen na več logičnih enot. Sama razporeditev posameznih zavihkov v programu je nastala v sodelovanju z arheologi (Slika 27). Zaradi obsežnosti programa si bomo podrobno ogledali samo nekatere funkcije programa.
Tekst v pogovornem oknu in vsa sporočila programa so v angleškem jeziku. Za to sem se odločil iz več razlogov:

- Pogovorna okna v angeščini omogočajo lažje mednarodno sodelovanje.
- Ker program AutoCAD ni poslovenjen, so ostala sporočila in ukazi v anglenščini. Zato je večini uporabnikov uporaba najpogostejših angleških ukazov in sporočil v pogovornih oknih domača.
- Veliko ukazov in sporočil v pogovornih oknih ima v angleščini krajšo obliko in bolj nazoren pomen kot v slovenščini.

Ker je v času razvoja potekalo večje mednarodno sodelovanje, je prav prvi razlog bistveno prispeval k odločitvi o vmesniku v angleškem jeziku.
Spremenljivke v kodi in komentarji so v slovenščini zaradi lažje kontrole nad programom. Tako lažje najdemo napake, ki se prikradejo v kodo.



Slika 27: Pogovorno okno MiniExplorer

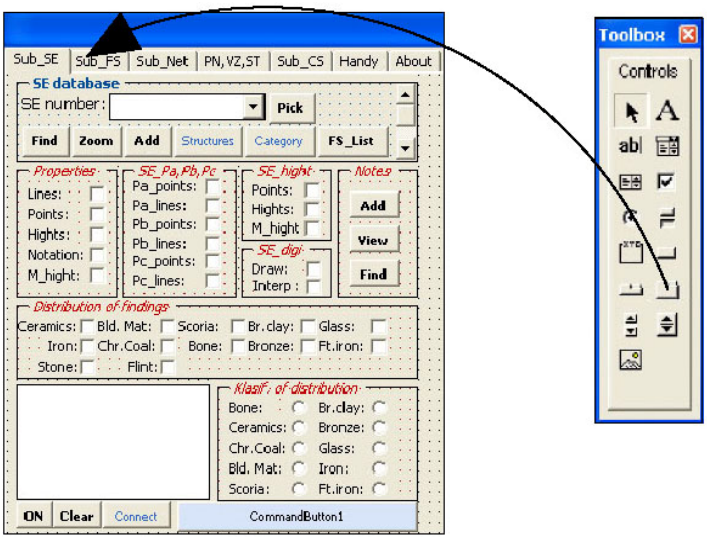
Pri vsakem modulu, ki ga bom podrobneje predstavil, bom napisal tudi, kje v programu se nahaja. Predstavitev posameznega modula bo potekala v treh fazah:

- Predstavitev problema, ki ga bomo reševali
- Enostavni opisni algoritem
- Programska koda, napisana v VBA, opremljena s podrobnimi komentarji

Bolj zapletene postopke bom posebej razložil. Ker so nekatere vrstice v programski kodi daljše, kot jih sprejme urejevalnik, sem to vrstico končal z znakom _, kar pomeni, da se ista vrstica kode nadaljuje v naslednji vrstici.
V naslednjem razdelku si najprej oglejmo potek gradnje in uporabo gradnikov, ki so potrebni za izgradnjo uporabniškega vmesnika programa MiniExplorer.

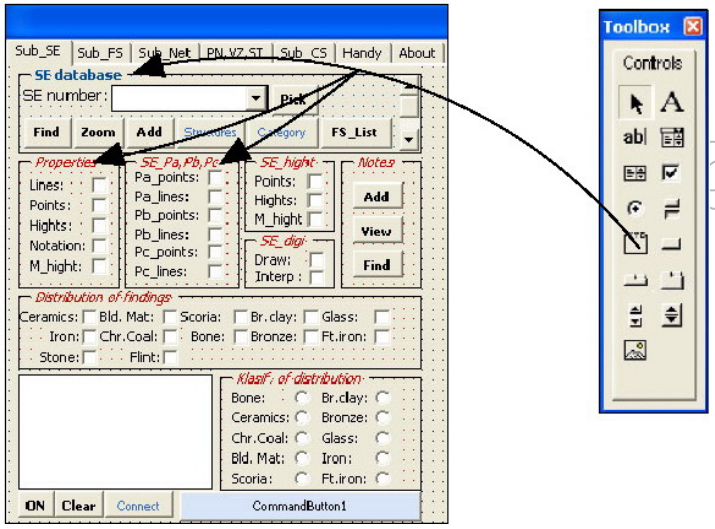
4.1 Gradnja pogovornega okna MiniExplorer

Pogovorno okno je verjetno najpomembnejši del nekega programa. Z njim uporabnik komunicira s programsko kodo. Za gradnjo pogovornega okna MiniExplorer sem porabil veliko časa, saj je okno sestavljeno iz cele palete gradnikov.
Vse se začne s preprostim obrazcem, na katerega nanašamo nove gradnike. Prvi gradnik, ki sem ga dodal na obrazec, je bilo *večstransko okno* (MultiPage). Le-tega z miško prenesemo na obrazec ter ga poljubno raztegnemo. V primeru MiniExplorerja sem večstransko okno raztegnil čez cel obrazec tako, da obrazca praktično ne vidimo. Večstranska okna nam omogočajo, da pogovorno okno razdelimo na več logičnih enot (v MiniExplorerju na Sub_SE, Sub_FS, Sub_Net ...) oziroma zavihkov. Ko ustvarimo gradnik tipa večstransko okn*o*, imamo ustvarjena le dva zavihka. Nove zavihke dodajamo tako, da na gradniku kliknemo desni gumb ter izberemo ukaz *New page*.



Slika 28: Dodajanje zavihka na večstransko okno (MultiPage)

Na posameznem zavihku večstranskega okna spet pripravimo logične enote v katere združimo gradnike. V ta namen sem uporabil gradnik *okvir* (Frame). Okvir je zelo uporaben prav v ta namen – za ločevanje med sabo povezanih enot. Tako na Sliki 28 na zavihku Sub_SE vidimo okvire SE database, Properties, Notes ... Poleg vizualnega združevanja gradnikov je okvir pomemben tudi pri izbirnih (ali kot jim tudi včasih rečemo - radijskih) gumbih. Namreč izbirni gumbi znotraj posameznega okvira tvorijo celoto. To pomeni, da je med njimi lahko sočasno izbran le eden.



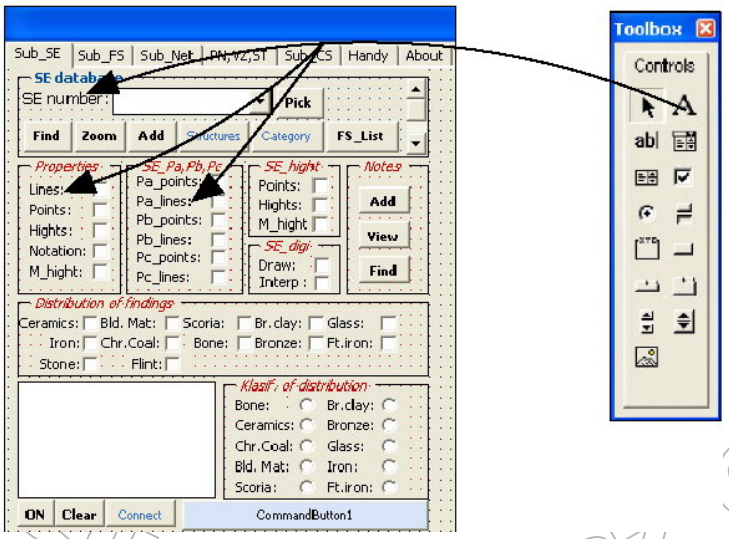
Slika 29: Okvir (Frame)

Ob vsakem gradniku se običajno nahaja kratek opis, kaj se ob kliku na določen gradnik, zgodi. Tudi ta opis je seveda gradnik. Imenuje se *oznaka* (Label). Oznaka je zelo preprost gradnik. Ko jo vstavimo v obrazec, ji dodelimo ime (`Name`) in napis (`Caption`), lahko pa določimo še druge lastnosti, kot so vrsta, velikost in barva pisave (`Font`), poravnava besedila in podobno. Poglejmo si primer, kako med delovanjem programa spremenimo napis oznake z imenom `PrimerOznake`:

```
PrimerOznake.Caption = "Spremenjen napis"
```

S tem stavkom smo spremenili napis oznake na »Spremenjen napis«. Poglejmo si še primer, kako bi naredili oznako odebeljeno (Bold):

```
PrimerOznake.Font.Bold = True
```

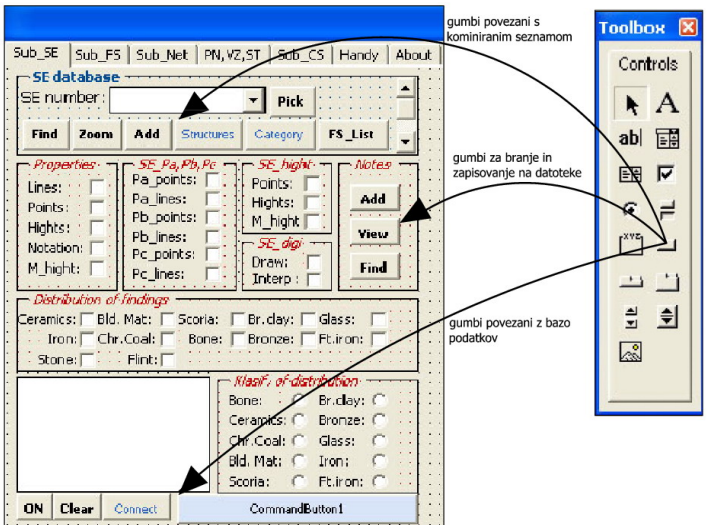


Slika 30: Oznaka (Label)

Naslednji zelo pomemben gradnik je *ukazni gumb* (CommandButton), ki je praktično nepogrešljiv v vsakem programu. Ob kliku nanj se izvrši neko dejanje. V MiniExplorerju je

gumbov zelo veliko (npr. Find, Zoom, Add ...). Oglejmo si nekaj uporabnih lastnosti ukaznih gumbov:

- Lastnost `Cancel`: Lastnost `Cancel` določa, ali ima ukazni gumb funkcijo Cancel, s katero prekličemo opravljene spremembe (vrednost True), ali ne (vrednost False). Kadar ima lastnost vrednost True, je klik na tak gumb enakovreden pritisku na tipko Esc. Če torej kliknemo na ta gumb, ali pa pritisnemo na tipko Esc, se sproži dogodek Click tega ukaznega gumba. V ustrezni dogodkovni proceduri (imeGumba_Click) poskrbimo, da se razveljavijo morebitne spremembe, ki smo jih pred tem naredili. Na obrazcu je lahko samo en gumb, katerega lastnost Cancel ima vrednost True. Smiselno je, da je napis na njem tak, da pove, da bomo s tem nekaj preklicali.
- Lastnost `Picture`: Lastnost `Picture` določa sliko, ki se pojavi na ukaznem gumbu, ko je ta dostopen in še ni pritisnjen. Sliko lahko izberemo med izdelavo programa ali pa jo med izvajanjem programa podamo s funkcijo `LoadPicture`. Slika se prikaže na sredini gumba. Če pa ima gumb napis (lastnost `Caption`), se pojavi nad napisom. Prevelike slike se samodejno obrežejo. Pri gradnji MiniExplorerja sem se uporabi slik ogibal, saj z njihovo uporabo upočasnimo delovanje programa.



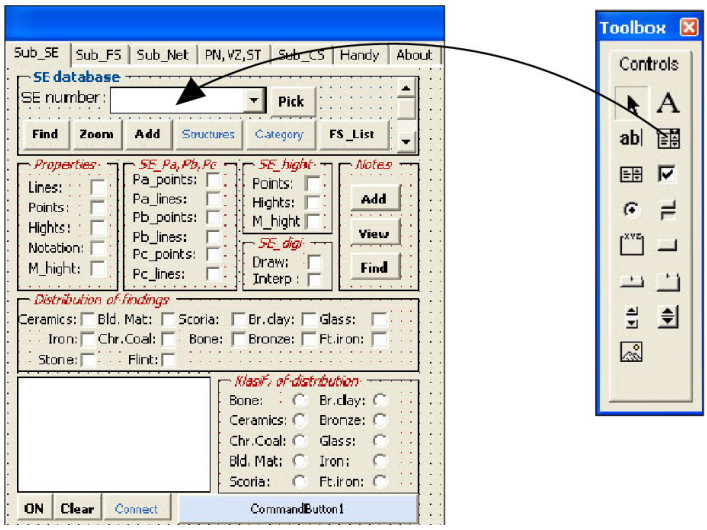
Slika 31: Ukazni gumb (CommandButton)

Naslednji gradnik uporabimo, kadar želimo uporabnikom omogočiti izbiranje med prednastavljenimi vrednostmi, kadar pa med njimi ni ustrezne, vnos svojega podatka. Ta gradnik se imenuje *kombinirani seznam* (ComboBox). Vanj je moč shraniti poljubne elemente. Elemente v kombiniran seznam dodajamo s pomočjo funkcije `AddItem`. Poglejmo si , kako napolnimo gradnik kombinirani seznam z imenom `KmbSez`, ki se nahaja na obrazcu z imenom `TestniObrazec`:

```
Dim PoljeSE() As String
Dim NizSE As String

'zapišemo niz z SE-ji
NizSE = "SE 0315, SE 0185, SE 0567, SE 0689, SE 0319"

's funkcijo Split zapišemo SE-je v polje, kot delilni element uporabim
'vejico
PoljeSE = Split(NizSE, ",")
```

```
'za vse elemente v polju PoljeSE
For i = 0 To UBound(PoljeSE)
    'vstavljamo nize v kom. seznam
    TestniObrazec.KmbSez.AddItem(PoljeSE(i))
Next
```

VBA v kombiniranem seznamu uporablja indekse od 0 dalje. Prvi element ima torej indeks 0, drugi indeks 1 ...



Slika 32: Kombinirani seznam (ComboBox)

Za brisanje elementov s kombiniranega seznama poskrbi funkcija `RemoveItem`. Kot parameter sprejme indeks elementa, ki ga mora odstraniti.
Pogosto potrebujemo informacijo o tem, koliko elementov je v seznamu. To nam pove lastnost `ListCount`. Če ima kombiniran seznam pet elementov, bo vrednost lastnosti `ListCount` 5, indeksi pa bodo seveda 0, 1, 2, 3, 4. Poglejmo si primer procedure, ki ji za argument podamo indeks elementa, ki ga želimo izbrisati iz kombiniranega seznama KmbSez:

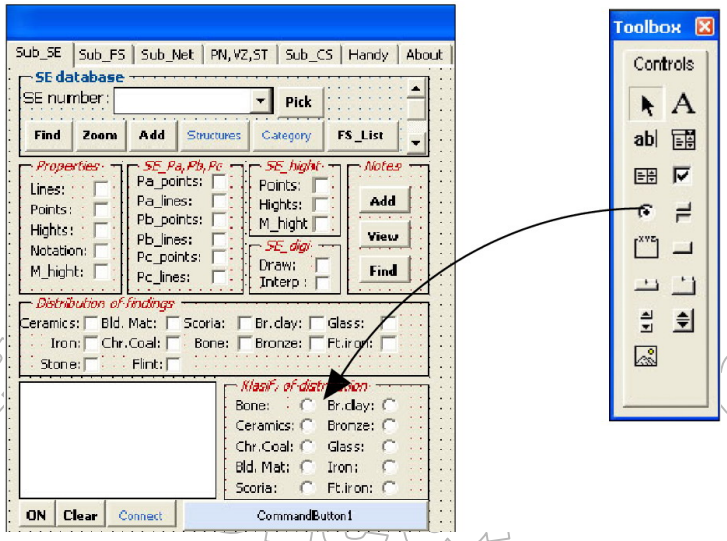
```
Sub BrisiElt(Indeks As Long)
    'če je v kombiniranem seznamu več elementov kot je velikost indeksa,
    'element iz seznama lahko izbrišem
    if TestniObrazec.KmbSez.ListCount > Indeks Then
        'brisanje elementa iz kom. seznama
        TestniObrazec.KmbSez.RemoveItem(Indeks)
    End If
End Sub
```

S funkcijo `Clear` naenkrat pobrišemo celoten kombinirani seznam.
Lastnost `ListIndeks` določa indeks elementa, ki je prikazan v okencu kombiniranega seznama, z lastnostjo `Value` pa izvemo, kaj je trenutno vsebina vnosnega okenca. Poglejmo si to na zgledu. Uporabimo kar kombinirani seznam, kot smo ga ustvarili pri zgledu dodajanja elementov v kombiniran seznam. Želimo, da bi se ob startu pogovornega okna v kombiniranem seznamu pojavil niz SE 0185. Zato `ListIndeks` nastavimo na vrednost 1. Z lastnostjo `Value` preverimo, ali je trenutni označeni element v kombiniranem seznamu res SE 0185. To storimo s pomočjo sporočilnega okna MsgBox, ki nam izpiše trenutno označen element v komb. seznamu.

```
Dim PoljeSE() As String
Dim NizSE As String
Dim Odg As String

NizSE = "SE 0315, SE 0185, SE 0567, SE 0689, SE 0319"
PoljeSE = Split(NizSE, ",")

For i = 0 To UBound(PoljeSE)
    TestniObrazec.KmbSez.AddItem (PoljeSE(i))
Next

'nastavimo komb. seznam na indeks z nizom SE 0185
TestniObrazec.KmbSez.ListIndex = 1

'izpišemo trenutno nastavljeni element v komb. seznamu, ki je besedica
'SE 0185
Odg = MsgBox("Lastnost Value nam vrne: " & TestniObrazec.KmbSez.Value)
```

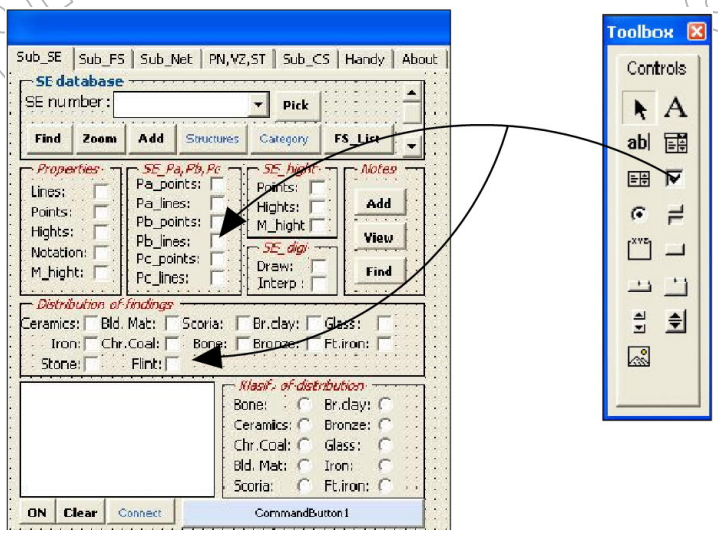
Pri MiniExplorerju je kombiniran seznam zapolnjen z nizi, ki so oblike: SE 0001, SE 0103 ... in predstavljajo oznake risalnih ravnin. Te nize dobimo s postopkom pregledovanja risalnih ravnin in preverjanjem njihovega imena. Postopek je precej dolg, zato ga ne bom opisoval.
MiniExplorer vsebuje še kar nekaj gradnikov. Eden od njih je *izbirni gumb* (OptionButton). Včasih mu rečemo tudi radijski gumb. Namenjen je prikazu izbir, med katerimi lahko uporabnik izbere. Izbirni gumbi vedno nastopajo v skupinah. Uporabnik se med njimi lahko odloči le za eno izbiro, saj se le-te med seboj izključujejo. Kot smo že omenili, izbirne gumbe združimo v skupine tako, da jih vstavimo v okvir (Frame). Izbirne gumbe sem uporabil na več mestih v programu. Uporabnik se lahko npr. odloči, da bi rad na risbi videl vse kosti ali vse bronaste najdbe ali vse keramične izdelke ... Seveda se lahko odloči zgolj za eno kategorijo najdb, saj se med sabo izključujejo.
Omeniti velja lastnost `Value`, ki določa stanje izbirnega gumba. Ta je lahko potrjen (vrednost True) ali nepotrjen (vrednost False). Kadar vstavimo na obrazec skupino izbirnih gumbov, običajno med njimi izberemo enega, ki bo potrjen, ko se obrazec odpre. To naredimo prav z lastnostjo `Value`.

```
IGSteklo.Value = True
```



Slika 33: Izbirni gumb (OptionButton)

Pritrditveno polje (CheckBox) je gradnik v obliki kvadratka, ki je lahko prazen ali pa je v njem kljukica, kadar je potrjen. Tudi pritrditvena polja pogosto združujemo v skupine. Vendar se posamezna prireditvena polja med sabo ne izključujejo kot pri izbirnih gumbih. Tudi tu je osnovna lastnost `Value`, ki pove, če je polje izbrano ali ne.

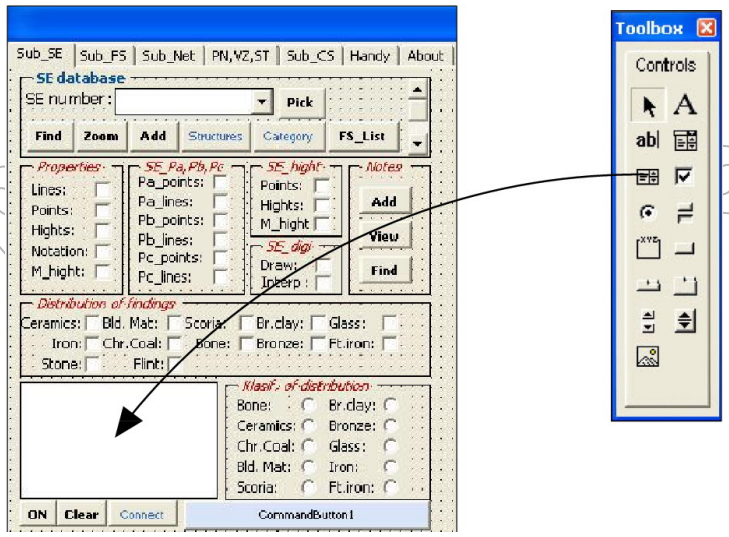


Slika 34: Pritrditveno polje (CheckBox)

Omembe vredna sta še dva gradnika. Prvi je *seznam* (ListBox). Namenjen je prikazu elementov, ki ji uporabnik lahko le označuje. V tem je tudi poglavitna razlika med seznamom ter kombiniranem seznamu. V kombiniranem seznamu lahko uporabnik izbere element ali pa ga sam vpiše v okence. Pri navadnem seznamu lahko elemente samo izbira. Kadar je elementov več, kolikor jih na za seznam predvidenem prostoru lahko prikažemo, se seznam samodejno opremi z drsniki, ki omogočajo pregledovanje vseh elementov. Podobno kot pri kombiniranem seznamu tudi tu lahko dodajamo in brišemo elemente s postopki `AddItem`, `RemoveItem` in `Clear`. Za razliko od kombiniranega seznama je možno v navadnem seznamu izbrati več kot en element hkrati. V MiniExplorerju imamo na vsakem zavihku en

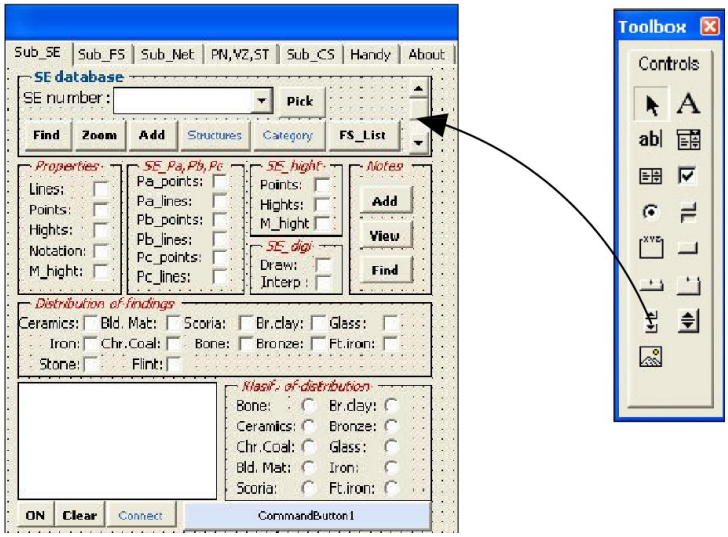
tak seznam. Uporabljamo ga za prikaz rezultatov.
Opišimo to na arheološkem primeru. Če bi hoteli, da se nam izpišejo vsi SE-ji, ki vsebujejo fragmente keramike, bi pod oknom *Klasif. of distribution* pritisnili radijski gumb *Ceramics*. Ob kliku bi se izvedla procedura, ki iz seznama SE-jev, ki so v kombiniranem seznamu, v navaden seznam zapiše vse dobljene rezultate (Slika 35). V tem seznamu lahko sedaj določene SE označimo in manipuliramo samo z izbranimi SE-ji. Podobno velja tudi za druga izbiranja.

Slika 35: Izpis vseh SE-jev, ki vsebujejo keramiko



Slika 36: Seznam (ListBox)

Drugi gradnik se imenuje *drsnik* (ScrollBar). Ta omogoča pregledovanje daljših seznamov. Omeniti velja nekaj lastnosti tega gradnika. Tako ima lastnosti `Max` in `Min`. Prva nam določa največjo, druga pa najmanjšo vrednost drsnika. Lastnost `Value` nam vrne trenutno vrednost na drsniku. V MiniExplorerju sem ga povezal z gradnikom kombiniran seznam, kar nam omogoča lažje, zlasti pa hitrejše, pregledovanje elementov v kombiniranem seznamu.



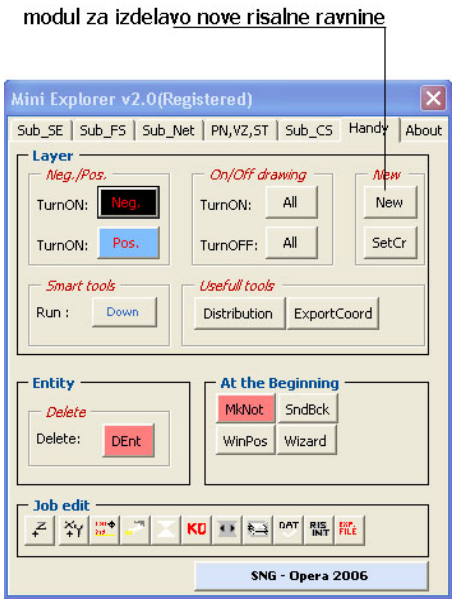
Slika 37: Drsnik (ScrollBar)

Vsi zavihki večstranskega okna imajo zelo podobno zasnovo kot prvi zavihek. Tega sem opisal zato, ker je najobsežnejši in sestavljen iz največ različnih gradnikov.
Uporabniški vmesnik tako imamo. Sedaj mu moramo "vdahniti" še vsebino. V naslednjih razdelkih bom opisal nekaj postopkov, ki jih lahko izvedemo s pomočjo programa MiniExplorer.

4.2 Modul za izdelavo novih risalnih ravnin

Pri delu z AutoCAD–om ves čas delamo z risalnimi ravninami. Novo risalno ravnino v AutoCAD–u lahko ustvarimo v upravitelju lastnosti risalnih ravnin (glej Stran 16). V programu MiniExplorer smo želeli imeti tudi modul, ki bi nam omogočal, da risalno ravnino

ustvarimo programsko. Zakaj pa bi sploh pisali lasten program, ki bi nam omogočal programsko izdelavo nove risalne ravnine? Odgovor je preprost - ker je programsko dodajanje nove ravnine veliko hitrejše. Ko imamo v upravljalniku risalnih ravnin nekaj tisoč risalnih ravnin, že samo odpiranje upravljalnika traja celo večnost. Najpomembneje pa je, da lahko program priredimo tako, kot to zahteva uporabnik.
Modul se v MiniExplorerju nahaja pod zavihkom *Handy* → *New* (Slika 36).



Slika 38: Modul za izdelavo nove risalne ravnine

Algoritem

Modul za izdelavo novih risalnih ravnin je dokaj enostaven. Ime nove risalne ravnine vnese uporabnik prek sporočilnega okna InputBox. Ker navadno ustvarimo več risalnih ravnin hkrati, ki pa se med sabo razlikujejo le po zadnji besedi (Slika 41), določimo kot privzeti niz znakov v sporočilnem oknu InputBox kar ime trenutno aktivne risalne ravnine. Tako nam ni potrebno vsakič pisati celega imena risalne ravnine. Naslednji korak je zelo pomemben. Lahko se namreč zgodi, da bo uporabnik vnesel tako ime, da risalna ravnina s tem imenom že obstaja. Zato preverimo, če je temu tako. Lastnost `Layers` objekta `ThisDrawing` vsebuje seznam vseh obstoječih risalnih ravnin. Z zanko `For Each` se sprehodimo čez vse risalne ravnine, ter s pomočjo vgrajene funkcije primerjamo zahtevano ime z imenom pregledovane risalne ravnine. Če risalna ravnina z zahtevanim imenom že obstaja, ponudimo uporabniku dve možnosti:

- Program naj se vrne na začetek, ter ponovno zahteva ime risalne ravnine.
- Program se enostavno zaključi.

Če risalna ravnina s tem imenom še ne obstaja, ustvarimo novo risalno ravnino z željenim imenom. Ker bomo nove objekte po vsej verjetnosti dodajali ravno na to novo risalno ravnino, ob koncu programa ponudimo še možnost, da ustvarjeno ravnino proglasimo za trenutno. Modul se imenuje `UstvariRavnino`.

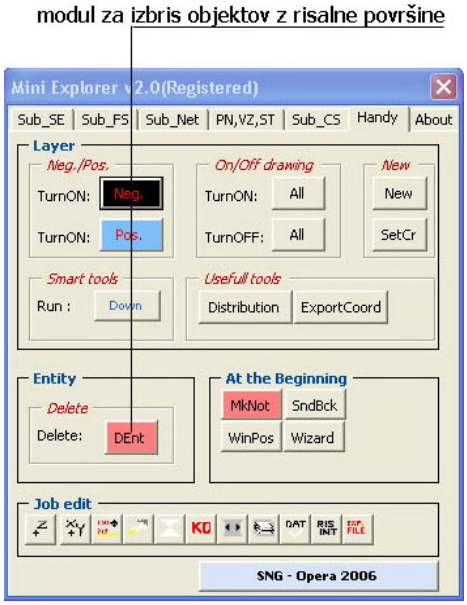
Programska koda

```
Public Sub UstvariRavnino()
'objekt s katerim se bomo sprehajali po risalnih ravninah

Dim RisRav As AcadLayer
'objekt, ki bo postal nova risalna ravnina
Dim NovaRavnina As AcadLayer
'spremenljivka, ki jo bomo potrebovali pri sporočilnih oknih tipa MsgBox
Dim Odg As Variant
Dim ZacasnoIme As String
Dim Ime As String

Zacetek:
'ime trenutno aktivne risalne ravnine
ZacasnoIme = ThisDrawing.ActiveLayer.Name

'vnos imena nove ravnine
Ime = InputBox("You are about to create new layer. Enter layer's name:",_
               _"Creating new layer", ZacasnoIme)

'če pustimo okence prazno, ponudimo možnost ponovnega vnosa imena risalne
'ravnine
If Len(Ime) = 0 Then
    Odg = MsgBox("Please, enter layer's name." & vbCrLf & "Repeat _
                 _selection?", vbYesNo, "Error")

    'če pritisnemo gumb No, zaključimo modul in ne naredimo ničesar
    If Odg = vbNo Then
        Exit Sub
     'če pritisnemo gumb Yes, se vrnemo na začetek, ter ponovno zahtevamo
     'ime ravnine
    Else
        GoTo Zacetek
    End If
End If

'preverimo, ali že obstaja risalna ravnina s podanim imenom
For Each RisRav In ThisDrawing.Layers 'pregledamo vse obstoječe ravnine
    'ali sta imeni enaki
    If StrComp(RisRav.Name, Ime) = 0 Then
        'zopet ponudimo dve možnosti: vračanje ali zaključek
        Odg = MsgBox("This layer already exists." & vbCrLf & " Repeat_
                     _selection?", vbYesNo, "Error")
        'če pritisnemo gumb No, zaključimo modul
        If Odg = vbNo Then
            Exit Sub
        'če pritisnemo gumb Yes, ponovno zahtevamo ime
        Else
            GoTo Zacetek
        End If
    End If
Next RisRav

'ustvarimo risalno ravnino z željenim imenom
Set NovaRavnina = ThisDrawing.Layers.Add(Ime)
'ponudimo možnost nastavitve ustvarjene risalne ravnine kot trenutne
Odg = MsgBox("Set this layer as current?", vbYesNo, "Current layer")

'nova risalna ravnina naj ne bo trenutna
```

```
If Odg = vbNo Then
    Exit Sub
'nova ravnina naj bo trenutna
Else
    ThisDrawing.ActiveLayer = NovaRavnina
End If

End Sub
```

4.3 Modul za izbris objektov z risalnih ravnin

Na posamezni risalni površini v AutoCAD-u imamo lahko ogromno objektov (točke, črte, parabolične krivulje ...). Recimo, da bi radi izbrisali vse objekte z neke risalne ravnine. V AutoCAD-u to storimo tako, da označimo vse objekte, nato pa pritisnemo tipko Delete. Ta proces je zelo dolgotrajen. Zlasti se zadeva zaplete, če so objekti z različnih risalnih ravnin med sabo prepleteni. Poglejmo si to na primeru. Recimo, da imamo na nekem prostoru več SE-jev. Vsak SE je v računalniku predstavljen s tremi risalnimi ravninami (Slika 41). Da lahko izbišemo vse točke z neke risalne ravnine, moramo sprva izklopiti vse ostale risalne ravnine, da ne bi po pomoti izbrisali kakšnega objekta iz druge risalne ravnine. Nato moramo vse objekte označiti ter izbrisati. Risalno ravnino, kateri smo odstranili objekte, moramo še ročno odstraniti z upravljalnika risalnih ravnin. Problem lahko dokaj enostavno rešimo z VBA. V MiniExplorerju lahko ta modul najdemo pod zavihkom *Handy* → *DEnt*, ki izbriše vse objekte z ene ali več risalnih ravnin, ne da bi pri tem morali vse te objekte posebej označiti (dovolj je en sam na ravnino), kar nam prihrani ogromno časa.
Zaradi lažjega razumevanja izdelave samega programa sem sam opis programa razdelil na več delov. Najprej je predstavljena razlaga posameznega dela, temu sledi koda.



Slika 39: Modul za izbris objektov z risalne površine

Algoritem

Najprej moramo programu podati objekte, ki jih želimo izbrisati. Tej nalogi je kos AutoCAD-ov objekt, ki se imenuje `AcadSelectionSet`. Lahko si ga predstavljamo kot polje, v katerega lahko shranimo različne AutoCAD-ove objekte (točke, črte...). Program jih nato obravnava kot eno enoto. Vgrajeno ima funkcijo `SelectOnScreen`, s katero lahko poljubno dodajamo objekte v objekt. Ko se izvaja ta funkcija, z miško kliknemo na željene

objekte na risalni površini. Objekti, ki jih kliknemo, se dodajo v objekt tipa `AcadSelectionSet`. Ko izberemo vse željene objekte, pritisnemo tipko Enter. Da se izognemo napakam, upoštevamo možnost, da uporabnik ne izbere nobenega objekta.

```
Dim Klik As AcadSelectionSet
Dim Odg As Variant

'nastavimo objekt z imenom Izbira
Set Klik = ThisDrawing.SelectionSets.Add("Izbira")

'uporabnik izbere objekte, ki jih želi izbrisati; ponudimo mu možnost za
'izhod iz programa z gumbom Cancel, brez brisanja objektov ter gumb Ok za
'izbor objektov
Odg = MsgBox("Select objects. All objects on the same layer as_
             _selected ones will be erased. Do you want to_
             _continue?", vbOKCancel + vbExclamation, "Select_
             _objects")

'uporabnik ne želi nadaljevati - izhod
If Odg = vbCancel Then Exit Sub

'uporabnik bo dodal objektu Klik objekte s klikanjem
Klik.SelectOnScreen

'če uporabnik ni izbral nobenega objekta - izhod z opozorilom
If Klik.Count = 0 Then
    Odg = MsgBox("No objects selected!", vbCritical, "Error")
    Exit Sub
End If
```

Sedaj imamo v spremenljivki `Klik` zbrane objekte, ki določajo ravnine, na katerih želimo izbrisati vse objekte. Zato moramo določiti ravnino, na kateri se posamezni objekt nahaja. To nam omogoča funkcija `Layer`, ki nam vrne niz z imenom risalne ravnine.
Imena ravnin bomo shranjevali v dinamično polje. V to polje torej z zanko shranimo vsa imena risalnih ravnin vseh objektov v objektu `Klik`. Tu pa naletimo na problem. Ob izbiri objektov smo namreč lahko izbrali več objektov z iste risalne ravnine. Poskrbeti moramo, da se v polju imena risalnih ravnin ne ponavljajo. Vsako mora nastopiti natanko enkrat. Zato pregledamo polje in v novo polje shranimo le različna imena ravnin.

```
Dim PoljeIzb() As String
Dim TrenutniNiz As String
Dim KoncnoPolje As String
Dim Objekt As AcadEntity
Dim SteviloRazlicnih As Integer
Dim i As Integer, j As Integer

i = -1
'v polje zapišemo vsa imena risalnih ravnin iz zbirke Klik
For Each Objekt In Klik
    i = i + 1
    'dinamično polje sproti povečujemo
    ReDim Preserve PoljeIzb(0 To i)
    'v polje zapišemo imena risalnih ravnin
    PoljeIzb(i) = Objekt.Layer
Next Objekt
'poskrbimo, da se vsako ime risalne ravnine pojavi zgolj enkrat
SteviloRazlicnih = 0
```

```
'sprehod skozi vsa imena, ki se nahajajo v polju PoljeIzb
For i = 0 To UBound(PoljeIzb)
    TrenutniNiz = PoljeIzb(i)
    's trenutnim elementom primerjamo vse ostale elemente
    For j = i + 1 To UBound(PoljeIzb)

        'v trenutku, ko se ime pojavi večkrat, ga v polju nadomestimo s
        'praznim nizom
        If PoljeIzb(j) = TrenutniNiz Then PoljeIzb(j) = ""
    Next

'če niz ni prazen, ime dodamo v končno polje
If PoljeIzb(i) <> "" Then
    SteviloRazlicnih = SteviloRazlicnih + 1
    ReDim Preserve KoncnoPolje(0 To SteviloRazlicnih)
    KoncnoPolje(SteviloRazlicnih) = PoljeIzb(i)
End If
Next
```

Nadaljevanje je dokaj enostavno. Sedaj nam preostane samo še, da izbrišemo objekte, ki se nahajajo na risalnih ravninah, ki so shranjene v polju. Ta problem rešimo z dvojno zanko. Zunanja zanka se sprehaja skozi vsa imena risalnih ravnin, shranjenih v polju, notranja zanka pa za vsak objekt na risalni površini pogleda ime risalne ravnine na kateri leži. Če se ime sklada z imenom v polju, objekt izbrišemo.

```
'za vsa imena ravnin v končnem polju
For i = 0 To UBound(KoncnoPolje)
    'pregledamo vse objekte na risalni površini
    For Each Objekt In ThisDrawing.ModelSpace
        'če se imeni ujemata, objekt izbrišemo
        If Objekt.Layer = KoncnoPolje(i) Then
            Objekt.Delete
        End If
    Next Objekt
Next
```

Kljub izbrisu vseh določenih objektov, se risalne ravnine, na katerih so se ti objekti nahajali, ohranijo. Program dopolnimo tako, da uporabnika vprašamo, kaj naj s temi ravninami stori. Ponudimo mu dve možnosti:

- Risalne ravnine naj program odstrani.
- Program naj risalne ravnine ne izbriše.

Če se uporabnik odloči za izbris, lahko naletimo na probleme. Risalne ravnine, ki je nastavljena kot trenutna, ni mogoče izbrisati. Podobno se zgodi, če je risalna ravnina zamznjena ali zaklenjena. Da nam ni potrebno za vsako ravnino preverjati njenega stanja, bomo brisanje vedno poskusili izvesti. A napake bomo s stavkom `On Error` ulovili in uporabnika obvestili, da določena ravnina ni bila izbrisana.
Izbris risalnih ravnin rešimo na zelo podoben način, kot smo opravili z izbrisom objektov. Uporabimo dvojno zanko, kjer zunanja teče skozi polje z imeni, notranja pa skozi vse risalne ravnine. Če se imeni ujemata, to risalno ravnino odstranimo. Če naletimo na problem, opisan prej, uporabnika obvestimo o tem, katere risalne ravnine program ne more odstraniti.

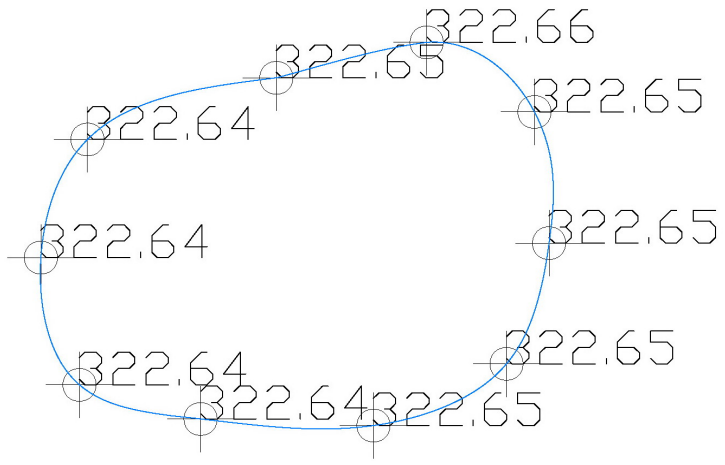
```
Dim RisRavnina As AcadLayer
Dim ShraniIme As String
```

```
'ob primeru napake se premaknemo v razdelek napak
On Error GoSub Napaka

'za vse elemente v končnem polju
For i = 0 To UBound(KoncnoPolje)
   'za vse ravnine na risbi
   For Each RisRavnina In ThisDrawing.Layers
      'imena se skladajo - izbris risalne ravnine
      If RisRavnina.Name = KoncnoPolje(i) Then
         'zapomnimo si ime risalne ravnine, da ga ob primeru napake lahko
         'posredujemo uporabiku
         ShraniIme = KoncnoPolje(i)
         'izbris risalne ravnine; tu lahko pride do napake
         RisRavnina.Delete
      End If
   Next RisRavnina
Next

'obravnava napake
Napaka:
'v primeru napake uporabniku javimo, katere ravnine ne moremo izbrisati
Odg = MsgBox("Layer " & risRavnina & " can't be deleted! Check if this_
            _layer is set to current, frozen or locked.")

'vrnemo se nazaj v zanko, kjer se brisanje nadaljuje
Return
```

4.4 Modul Zoom

Vsaka računalniško obdelana stratigrafska enota je sestavljena iz treh vrst AutoCAD-ovih objektov:

- Točk (AcadPoints)
- Enovrstičnih tekstov (AcadText)
- Paraboličnega zlepka (AcadSpline)



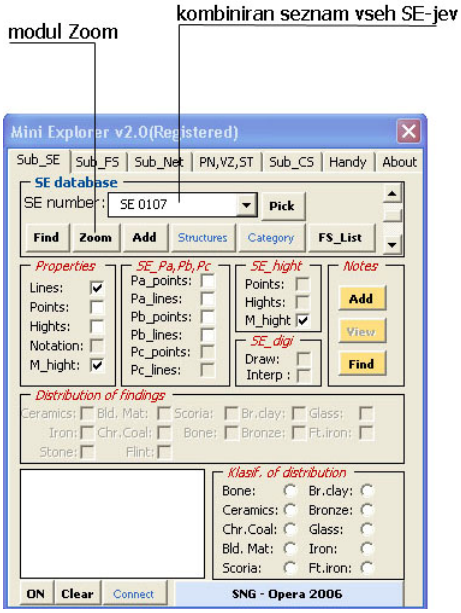
Slika 40: Predstavitev SE-ja v AutoCAD-u

Parabolični zlepek dobimo s smiselno povezavo točk, ki jih prejmemo s totalne postaje. Tekst

ob točkah predstavlja nadmorsko višino vsake točke. Vsaka vrsta objekta ima predpisano risalno ravnino Vsaka SE torej zavzema tri svoje risalne ravnine. Vsaka od teh ravnin v svojem imenu vsebuje številko SE enote v obliki *SE xxxx*. (Slika 41)

Slika 41: Risalne ravnine za vsak SE

Arheološki teren lahko vsebuje na tisoče takih SE-jev. Orientacija in iskanje določenega SE-ja tako postaneta skoraj nemogoča. Potrebno je poseči po modulu, ki nam na zaslonu hitro približa željeno enoto. To pomeni, da mora biti algoritem napisan tako, da uporablja čim manj operacij, ki lahko modul upočasnijo. Modul se v programu MiniExplorer nahaja pod zavihkom *Sub_SE → Zoom* (Slika 42).



Slika 42: Modul Zoom

Podatek o tem, kateri SE naj se približa, dobimo iz gradnika kombinirani seznam, ki smo ga ob startu programa MiniExplorer zapolnili z vsemi SE-ji, ki se nahajajo na risbi. Tega dela postopka ne bom opisoval.

Algoritem

AutoCAD ima že vgrajen ukaz *zoom*. Ker nam VBA omogoča direktno pošiljanje ukazov v ukazno vrstico, nam je to v pomoč, da spoznamo, kakšna mora biti točno oblika ukaza. Če v ukazni vrstici zapišemo ukaz *zoom*, nam vrne sledeče:

```
Command: zoom
Specify corner of window, enter a scale factor (nX or nXP), or
[All/Center/Dynamic/Extents/Previous/Scale/Window/Object] <real time>:
```

Ob samem razvoju modula smo prišli do zaključka, da optimalno izbiro omogoča podizbira *Window*. Ta od nas zahteva dve točki, ki pomenita oglišči pravokotnega dela risbe, ki jo bomo videli na zaslonu, da jo lahko približa. Naša glavna naloga je torej, da poiščemo dve točki, ki določata pravokotnik, ki vsebuje ustrezno SE. Izjema so SE-ji, ki vsebujejo zgolj eno točko. V

tem primeru uporabimo podizbiro *Center*, kjer je potrebno podati zgolj eno točko. Sam postopek razdelimo na dva osnovna koraka. V prvem sestavimo polje koordinat vseh točk, ki pripadajo tej SE. V drugem pa bomo med temi koordinatami poiskali tisti dve točki, ki določata pravokotnik, ki vsebuje vse ostale točke.
Definiramo dve polji. V prvo shranjujemo koordinato x vsake točke, ki pripada ustrezni SE, v drugo pa koordinato y. Koordinato z lahko zanemarimo, ker pri tem modulu ni potrebna. Prvi korak izvedemo tako, da se z zanko sprehodimo čez vse objekte na risbi. Za vsak objekt pogledamo ime risalne ravnine na kateri se nahaja. Ker ime risalne ravnine vsebuje tudi oznako SE, to preverimo. Če ime risalne ravnine torej vsebuje niz, izbran v kombiniranem seznamu SE in je ta objekt tudi točka, smo naleteli na pravi objekt. Njegovi koordinati uvrstimo v pripravljeni polji.

```
Dim Tocka As AcadPoint
Dim KoordinataX() As Double, KoordinataY() As Double
Dim Objekt As AcadEntity
Dim KoordinateTocke As Variant
Dim i As Integer

i = -1
'za vse objekte na risalni površini
For EaSch Objekt In ThisDrawing.ModelSpace
    'če je objekt točka, ter ime risalne ravnine vsebuje ime izbrane SE
    If TypeOf Objekt Is AcadPoint And InStr(Objekt.Layer, BazaSE.Value)_
    _Then
        i = i + 1 'našli smo ustrezno točko
        ReDim Preserve KoordinataX(0 To i)
        ReDim Preserve KoordinataY(0 To i)
        Set Tocka = Objekt
        'funkcija Coordinates nam v spremenljivko tipa Variant zapiše
        'koordinato točke v obliki polja s tremi elementi
        KoordinateTocke = Tocka.Coordinates
        'pod prvim indeskom v polju se nahaja koordinata x
        KoordinataX(i) = KoordinateTocke(0)
        'pod drugim indeskom se nahaja koordinata y
        KoordinataY(i) = KoordinateTocke(1)
    End If
Next Objekt
```

Sedaj potrebujemo pravokotnik, ki bo vseboval vse te točke. Ustrezni diagonalni točki dobimo tako, da v polju s koordinatami x poiščemo največjo in najmanjšo vrednost. Ravno tako določimo y. Potrebno pa je še malo previdnosti. Lahko se nam namreč pripeti, da nek SE vsebuje samo eno točko. Pred vstopom v zanko še preverimo, da polje gotovo vsebuje vsaj dva vnosa.

```
Dim x1 As Double, x2 As Double, y1 As Double, y2 As Double

'nastavimo začetne vrednosti - koordinate prve točke v polju
x1 = KoordinataX(0)
x2 = x1

y1 = KoordinataY(0)
y2 = y1

'zagotovo sta v polju vsaj dve točki
If UBound(KoordinataX) > 0 Then
For j = 1 To i
```

```
        'če je trenutni x manjši od x-a v spremenljivki x1, je najmanjši x do
        'sedaj, trenutni x
        If KoordinataX(j) < x1 Then
            x1 = KoordinataX(j)
        'če je trenutni x večji od x-a v spremenljivki x2, je največji x do
        'sedaj, trenutni x
        ElseIf KoordinataX(j) > x2 Then
            x2 = KoordinataX(j)
        End If

        If KoordinataY(j) < y1 Then
            y1 = KoordinataY(j)
        ElseIf KoordinataY(j) > y2 Then
            y2 = KoordinataY(j)
        End If
    Next
    End If
```

Sedaj imamo vse podatke, ki jih potrebujemo. AutoCAD-u moramo naročiti, naj v risalno okno prikaže tisti del risbe, ki ga določa izračunani pravokotnik. Zato bomo potrebovali vgrajeni ukaz *zoom*. V VBA-ju lahko v AutoCAD–ovo ukazno vrstico pošljemo (in s tem izvedemo) ukaz s pomočjo funkcije `SendCommand`. Funkcija kot argument prejme niz, ki ga posreduje ukazni vrstici. AutoCAD glede na niz izvrši predpisano dejanje. Mi mu bomo podali ukaz *zoom* s pripadajočimi koordinatami. Problem razdelimo na dva dela:

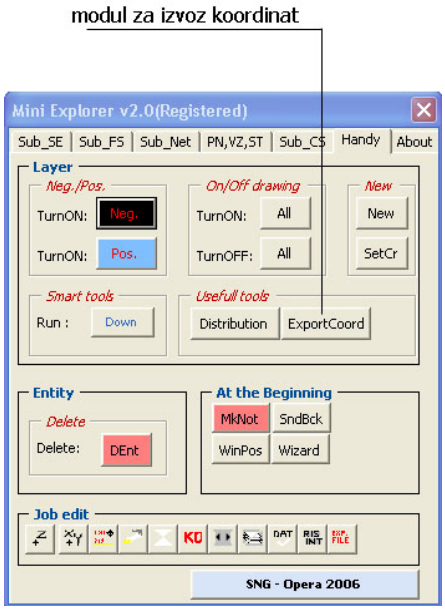
- SE vsebuje zgolj eno točko.
- SE vsebuje vsaj dve točki.

V prvem primeru v ukazno vrstico AutoCAD–a pošljemo ukaz `zoom c`, kar pomeni *Zoom Center*. Ta ukaz ima tri parametre. Prva dva določata točko, tretji pa je realno število, ki pove stopnjo, koliko naj AutoCAD točko približa. Manjše kot je število, bolj nam točko približa. Po daljšem testiranju smo se kot najprimernejši faktor, odločili za število `0.5`. Rekli smo, da moramo ukaz oblikovati kot niz. Zato moramo koordinati točke, ki sta tipa `Double` spremeniti v niz.
V drugem primeru uporabimo ukaz `zoom w`, kar pomeni *Zoom Window*. Podamo mu dve točki. Ravno tako kot v prvem primeru moramo tudi tu koordinate tipa `Double` spremeniti v niz.

```
Dim SkupenNiz As String

'samo ena točka
If UBound(KoordinataX) = 0 Then
    'sestavimo niz, ki ga prejme funkcija SendCommand
    SkupenNiz = "'zoom c " + Trim(str(x1)) + "," + Trim(str(y1)) + " " +_
                _"0.5" + " "

'več kot ena točka
Else
    skupenNiz = "'zoom w " & str(x1) & "," & Trim(str(y1)) & str(x2) &_
                "," & Trim(str(y2)) + " "
End If

'pošljemo niz v ukazno vrstico
ThisDrawing.SendCommand(skupenNiz)
```

4.5 Modul za izvoz koordinat

Pri arheoloških izkopavanjih večkrat uporabljamo tudi druge grafične programe, zlasti take, ki omogočajo predstavitve v treh dimenzijah (na primer 3Dmax in podobni). Večina teh programov potrebuje kot podatek tekstovno datoteko, na kateri imamo zapisane koordinate točk, ki jih potem prikažejo v treh dimenzijah. Modul `IzvoziTocke` nam je pri tem v veliko pomoč. Vsem v AutoCAD-u označenim točkam prebere koordinate, ter jih zapiše v tekstovno datoteko. Koordinate vsake točke zapiše v svojo vrstico, med seboj pa jih loči z vejico. Modul se v MiniExplorerju nahaja pod zavihkom *Handy* → *ExportCoord* (Slika 43).



Slika 43: Modul za izvoz koordinat

Algoritem

Najprej programu podamo točke, s katerimi želimo operirati. Pri tem si pomagamo z objektom `AcadSelectionSet` in vgrajeno funkcijo `SelectOnScreen`. Objekt smo že spoznali pri opisu modula za izbris objektov na risalni ravnini. Omogoča nam, da s pomočjo miške na risalni ravnini označimo objekte (točke, linije ...), ki jih želimo obravnavati kot eno enoto. Seveda ne smemo prezreti, če uporabnik ne označi nobenega objekta. V primeru, da tega ne upoštevamo, lahko pride do napake. Ko izberemo točke, se z zanko sprehodimo skozi vse objekte, ki smo jih prej izbrali. Pri vsakem sprotnem objektu preverimo ali je objekt točka. Če objekt ni točka, preverimo naslednji objekt. Točkam preberemo koordinate s funkcijo `Coordinates`. Koordinate shranimo v spremenljivko tipa `Variant`. Koordinate, shranjene v spremenljivki, so tipa `Double`. Ker nadalje opreriramo z nizi, jih s funkcijo `Str` iz tipa `Double` spremenimo v tip `String`. Ob sami pretvorbi lahko koordinate tudi poljubno zaokrožimo. V našem primeru smo se odločil, da je zadostna natančnost izvoženih koordinat na dve decimalki natančno. Pri zaokroževanju pa lahko naletimo na problem. Zaokrožujemo z vgrajeno funkcijo `Round`, ki kot podatek sprejme spremenljivko tipa `Double` ter željeno natančnost. Poglejmo si na primeru:

```
x = 5.342
y = Round(x, 2) → 5.34 'pravilna zaokrožitev

x = 5.399
```

```
y = Round(x, 2) → 5.4 'željeno y = 5.40

x = 5.999
y = Round(x, 2) → 6 'željeno y = 6.00
```

V prvem primeru funkcija vrne pravilen rezultat na dve decimalki natančno. Pri ostalih dveh pa ni tako. Po potrebi torej dobljenemu nizu dodamo ničle. V ta namen smo napisali pomožno funkcijo `ZaokroziKoordinato`. Funkcija sprejme za argument niz (število pretvorjeno v niz), ki mu po potrebi doda ničle. Vrne nam pravilno obliko niza. To storimo za vse tri koordinate (x, y, z). Nato sledi zapis koordinat v skupen niz. Koordinate v skupnem nizu ločimo z vejico. Ko zapišemo vse tri koordinate v vrstico, se s pomočjo vgrajene konstante `vbCrLf` pomaknemo v novo vrstico. To storimo za vse označene točke. Niz je tako pripravljen za izpis na tekstovno datoteko. Ker previdnosti ni nikoli odveč, preverimo velikost niza. Če je velikost niza nič, tedaj med vsemi objekti, ki smo jih označili na risalni ravnini, ni bilo nobene točke. Uporabnika na to opozorimo ter program zaključimo. Pri izpisu niza na datoteko si pomagamo z objektom `FileSystemObject`. Ta nam omogoča dostop do datotek na računalniku. Pri delu z zunanjimi datotekami moramo biti zelo previdni pri podajanju poti do datotek. Če datoteka, kamor pot kaže, ne obstaja, pride do napake. V našem primeru smo se odločili, da vse izvoze datotek shranjujemo v imenik *c:\Text files*. Zaradi možnosti, da imenik ne obstaja, preverimo njegov obstoj. To storimo s funkcijo `FolderExists`, ki ji podamo pot, ki jo preveri. Če imenik ne obstaja, ga ustvarimo s funkcijo `CreateFolder`. Naslednja zelo pomembna odločitev je ime datoteke, v katero koordinate izvažamo. Če bi bilo ime vedno isto, bi prejšnji izvoz vedno prepisali z novim. To moramo preprečiti. Rešitev se nam ponuja s sprotnim sestavljanjem vedno novega imena. Vsaka datoteka je sestavljena iz imena *export* ter tekočega števila (*export0, export1, export2*...). To storimo tako, da s pomočjo zanke sestavljamo ime. Besedici *export* zaporedoma dodajamo indeks ter preverjamo obstoj datoteke s tem imenom. Če datoteka s tem imenom ne obstaja, smo našli pravo ime za izvozno datoteko. Nato izpišemo skupni niz s koordinatami na datoteko. Ker je možno, da v direktoriju *Text files* obstaja na tisoče datotek, uporabnika s pomočjo sporočilnega okna obvestimo o natančni poti do ustvarjene datoteke (npr: *c:\Text files\export30.txt*).
Modul se imenuje `IzvoziTocke`.

Programska koda

```
Public Sub IzvoziTocke()
Dim Klik As AcadSelectionSet
Dim Tocka As AcadPoint
Dim SkupniNiz As String
Dim Datoteka As Variant
Dim PotDoDatoteke As String
Dim Objekt As AcadEntity
Dim Koordinate As Variant
Dim KoordinataX As String, KoordinataY As String, KoordinataZ As String
Dim Odg As String

'ustvarimo novo zbirko tipa SelectionSet z izbranim imenom Izvoz
Set Klik = ThisDrawing.SelectionSets.Add("Izvoz")
'uporabimo funkcijo za izbor objektov
Klik.SelectOnScreen

'če nismo označili nobenega objekta - izhod z opozorilom
If Klik.Count = 0 Then
```

```
    Odg = MsgBox("No objects selected!", vbCritical, "Error")
    Exit Sub
End If
'za vse objekte v zbirki Klik
For Each Objekt In Klik
    'če je trenutni objekt točka
    If TypeOf Objekt Is AcadPoint Then
        Set Tocka = Objekt


        'izpišemo koordinate v spremenljivko tipa Variant
        Koordinate = Tocka.Coordinates

        'koordinate zaokrožimo na dve decimalki ter pretvorimo v niz
        KoordinataX = Trim(str(Round(Koordinate(0), 2)))
        'pretvorjen niz podamo funkciji ZaokroziKoordinato za argument,
        'vrne pa nam pravilno zaokrožen niz
        KoordinataX = ZaokroziKoordinato(KoordinataX)
        KoordinataY = Trim(str(Round(Koordinate(1), 2)))
        KoordinataY = ZaokroziKoordinato(KoordinataY)
        KoordinataZ = Trim(str(Round(Koordinate(2), 2)))
        KoordinataZ = ZaokroziKoordinato(KoordinataZ)

        'sproti sestavljamo celoten niz in vsakič skočimo v novo vrstico
        SkupniNiz = SkupniNiz + KoordinataX + "," + KoordinataY + "," +_
                    _KoordinataZ + vbCrLf
    End If
Next Objekt

'med vsemi izbranimi objekti, nismo izbrali nobene točke - izhod z
'opozorilom
If Len(SkupniNiz) = 0 Then
    MsgBox ("No points were selected.")
    Exit Sub
End If

'ustvarimo VBA objekt za obdelavo datotek
Set Datoteka = CreateObject("Scripting.FileSystemObject")
'nastavimo pravilen direktorij za izvoz
PotDoDatoteke = "c:\Text files"

'preverimo obstoj direktorija
If Datoteka.FolderExists(PotDoDatoteke) = False Then
    'če direktorij ne obstaja ga ustvarimo
    Datoteka.CreateFolder(PotDoDatoteke)
End If

'deklariramo dodatne spremenljivke, ki jih v nadaljevanju potrebujemo
Dim i As Integer, j As Integer
i = 0
j = 1
'neskončna zanka - zanka se konča, ko najdemo pravo ime za datoteko
Do While j > 0
    'sestavljamo ime datoteke
    PotDoDatoteke = "c:\Text files\export" & i & ".txt"
    'če datoteka ne obstaja, smo našli pravo ime
    If Datoteka.FileExists(PotDoDatoteke) = False Then
        'ustvarimo novo datoteko
        Datoteka.CreateTextFile(PotDoDatoteke)
        'odpremo datoteko za pisanje
```

```
            Open PotDoDatoteke For Output As #1
            'zapišemo niz na datoteko
            Print #1, SkupniNiz
            'zapremo datoteko
            Close #1
            'skok iz neskončne zanke, ker smo našli pravo ime
            Exit Do
        End If
        i = i + 1
    Loop

    'uporabniku javimo na katero datoteko smo zapisali koordinate
    Odg = MsgBox("Coordinates were exported to file " & potDoDatoteke & ".",_
                 _"Reminder")
    End Sub
    'pomožna funkcija za zaokroževanje koordinat
    Public Function ZaokroziKoordinato(Koordinata As String) As String
    'če niz ni pravilne oblike
    If Not Right(Koordinata, 3) Like ".##" Then
        'če je število oblike npr.: 3.4 mu dodamo eno ničlo, tako da postane
        'oblike 3.40
        If Right(Koordinata, 3) Like "#.#" Then
            ZaokroziKoordinato = Koordinata & "0"
            'število je pravilno zaokroženo - funkcijo končamo
            Exit Function
        'če je število oblike npr.: 3 mu dodamo dve ničli, tako da postane
        'oblike 3.00
        ElseIf Right(Koordinata, 3) Like "###" Then
            ZaokroziKoordinato = Koordinata & ".00"
            'število je pravilno zaokroženo - funkcijo končamo
            Exit Function
        End If
    End If
    'število je bilo že ob vstopu v funkcijo pravilno zaokroženo, tako da
    'vrnem niz nespremenjen
    ZaokroziKoordinato = Koordinata
    End Function
```

5. ZAKLJUČEK

AutoCAD se je v arheologiji izkazal kot idealen program za arheološko dokumentiranje. Obdelava podatkov poteka hitro in učinkovito. Zlasti kasnejše pregledovanje obdelanih podatkov je zelo enostavno. Z uporabo programa MiniExplorer postane pregledovanje za povprečnega uporabnika bistveno bolj praktično.

Program MiniExplorer nam omogoča pregledovanje in obdelavo podatkov skoraj brez predhodnega poznavanja programa AutoCAD. Ker je napisan v programskem jeziku Visual Basic for Applicaton, ga lahko zelo hitro dopolnimo ali mu dodamo nove funkcije. Program MiniExplorer se ponaša tudi z možnostjo povezave z zunanjimi podatkovnimi bazami. Arheologi večino podatkov beležijo v podatkovne baze, ki pa so za pregledovanje dokaj okorne. S možnostjo povezave teh podatkov z vizualno podobo v programu AutoCAD se vrednost podatkovnim bazam poveča.

Visual Basic mi je pomenil zelo dobro izhodiščno osnovo. Programi napisani v njem so dokaj hitri in zanesljivi. Seveda obstajajo tudi hitrejši programski jeziki. AutoCAD na primer nudi možnost povezave s programskim jezikom C++. Programi napisani v C++ so bistveno hitrejši kot programi napisani v VBA, zato bo verjetno nadaljni razvoj programa MiniExplorerja potekal v tem programskem jeziku. Osnove, ki sem si jih pridobil pri programiranju v VBA, pa mi bodo pri tem prav gotovo v pomoč.

6. LITERATURA

1. Ellen Finkelstein, AutoCAD 2005 in AutoCAD LT 2005, Biblija, 2004
2. Aleš Šular, Spoznajmo Visual Basic, 2001
3. Matjaž Prtenjak, Visual Basic za Aplikacije, 1998
4. Scott Warner, Naučite se sami Visual Basic, 1998
5. Bill Bulchard, David Pitzer, Znotraj AutoCAD-a 2000, 1999