KONZORCIJ ŠOLSKIH CENTROV

REPUBLIKA SLOVENIJA
MINISTRSTVO ZA ŠOLSTVO IN ŠPORT

Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

Načrtovanje programskih aplikacij

NPA

1.del

Srečo Uranič

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KONZORCIJ ŠOLSKIH CENTROV
REPUBLIKA SLOVENIJA
MINISTRSTVO ZA ŠOLSTVO IN ŠPORT
Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

SPLOŠNE INFORMACIJE O GRADIVU

Izobraževalni program

Tehnik računalništva

Ime modula

Načrtovanje programskih aplikacij – NPA 4

Naslov učnih tem ali kompetenc, ki jih obravnava učno gradivo

Vizuelno programiranje, dedovanje, knjižnice, večokenske aplikacije, delo z bazami podatkov, testiranje in dokumentiranje, nameščanje aplikacij.

Avtor: Srečo Uranič
Recenzent: Matija Lokar
Lektor: v lekturi
Datum: Februar 2011
CIP:

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

POVZETEK/PREDGOVOR

Gradivo ***Načrtovanje programskih aplikacij*** je namenjeno dijakom 4. letnika izobraževalnega programa SSI – Tehnik računalništva in dijakom 5 letnika PTI - Tehnik računalništva. Pokriva vsebinski del modula Načrtovanje in razvoj programskih aplikacij, kot je zapisan v katalogu znanja za ta program v poklicnem tehniškem izobraževanju. Kot izbrani programski jezik sem izbral programski jezik C# v razvojnem okolju Microsoft Visual C# 2010 Express. Gradivo ne vsebuje poglavij, ki so zajeta v predhodnem izobraževanju. Dostopna so v mojih mapah http://uranic.tsckr.si/ na šolskem strežniku TŠCKR (to so poglavja osnove za izdelavo konzolnih aplikacij, metode, varovalni bloki, razhroščevanje, tabele, zbirke, strukture, naštevanje, razredi in objekti, rekurzije, metoda main in datoteke). V gradivu bomo večkrat uporabljali kratico *OOP* – Objektno Orireniriranо Programiranje.

V besedilu je veliko primerov programov in zgledov, od najenostavnejših do bolj kompleksnih. Bralce, ki bi o posamezni tematiki radi izvedeli več, vabim, da si ogledajo tudi gradivo, ki je navedeno v literaturi.

Gradivo je nastalo na osnovi številnih zapiskov avtorja, črpal pa sem tudi iz že objavljenih gradiv, pri katerih sem bil "vpleten". V gradivu je koda, napisana v programskem jeziku, nekoliko osenčena, saj sem jo na ta način želel tudi vizualno ločiti od teksta gradiva.

Literatura poleg teoretičnih osnov vsebuje številne primere in vaje. Vse vaje, pa tudi številne dodatne vaje in primeri so v stisnjeni obliki na voljo na strežniku TŠC Kranj http://uranic.tsckr.si/VISUAL%20C%23/, zaradi prevelikega obsega pa jih v literaturo nisem vključil še več.

Programiranja se ne da naučiti s prepisovanjem tujih programov, zato pričakujem, da bodo dijaki poleg skrbnega študija zgledov in rešitev pisali programe tudi sami. Dijakom tudi svetujem, da si zastavijo in rešijo svoje naloge, ter posegajo po številnih, na spletu dosegljivih primerih in zbirkah nalog.

Gradivo ***Načrtovanje programskih aplikacij*** vsebuje teme: izdelava okenskih aplikacij, lastnosti in dogodki okenskih gradnikov, dogodki tipkovnice, miške in ure, kontrola in validacija uporabnikovih vnosov, sporočilna okna, dedovanje, virtualne in prekrivne metode, polimorfizem, knjižnice, pogovorna okna, delo s tiskalnikom, nadrejeni in podrejeni obrazci, dostop in ažuriranje podatkov v bazah podatkov, transakcije, testiranje in dokumentiranje aplikacije, izdelava namestitvenega programa.

Ključne besede: gradniki, lastnosti, metode, dogodki, validacija (preverjanje pravilnosti), sporočilna okna, pogovorna okna, razred, dedovanje, polimorfizem, virtual, override, using,

knjižnica, tiskalnik, baza, transakcija, MDI, DataSet, SQL, testiranje, dokumentiranje, namestitev.

Legenda: Zaradi boljše preglednosti je gradivo opremljeno z motivirajočo slikovno podporo, katere pomen je naslednji:

-  informacije o gradivu;
-  povzetek oz. predgovor;
-  kazala;
-  učni cilji;
-  napoved učne situacije;
-  začetek novega poglavja;
-  posebni poudarki;
-  nova vaja, oziroma nov primer;
-  rešitev učne situacije;
-  literatura in viri.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO SLIK:

KAZALO TABEL

UČNI CILJI

Učenje programiranja je privajanje na algoritmični način razmišljanja. Poznavanje osnov programiranja in znanje algoritmičnega razmišljanja je tudi nujna sestavina sodobne funkcionalne pismenosti, saj ga danes potrebujemo praktično na vsakem koraku. Uporabljamo oz. potrebujemo ga:

- pri vsakršnem delu z računalnikom;
- pri branju navodil, postopkov (pogosto so v obliki "kvazi" programov, diagramov poteka);
- za umno naročanje ali izbiranje programske opreme;
- za pisanje makro ukazov v uporabniških orodjih;
- da znamo pravilno predstaviti (opisati, zastaviti, ...) problem, ki ga potem programira nekdo drug;
- ko sledimo postopku za pridobitev denarja z bankomata;
- ko se odločamo za podaljšanje registracije osebnega avtomobila;
- za potrebe administracije – delo z več uporabniki;
- za ustvarjanje dinamičnih spletnih strani;
- za nameščanje, posodabljanje in vzdrževanje aplikacije;
- ob zbiranju, analizi in za dokumentiranje zahtev naročnika, komuniciranje in pomoč naročnikom;
- za popravljanje "tujih" programov

CILJI

- Spoznavanje oz. nadgradnja osnov programiranja s pomočjo programskega jezika C#.
- Poznavanje in uporaba razvojnega okolja Visual Studio za izdelavo programov.
- Načrtovanje in izdelava preprostih in kompleksnejših programov.
- Privajanje na algoritmični način razmišljanja.
- Poznavanje razlike med enostavnimi in kompleksnimi programskimi strukturami.
- Ugotavljanje in odpravljanje napak v procesu izdelave programov.
- Uporaba znanih rešitev na novih primerih.
- Spoznavanje osnov sodobnega objektno orientiranega programiranja.
- Manipuliranje s podatki v bazi podatkov.
- Testiranje programske aplikacije in beleženje rezultatov testiranja.
- Ob nameščanju, posodabljanju in vzdrževanju aplikacije.
- Ko zbiramo, analiziramo in dokumentiramo zahteve naročnika, komuniciramo z njim in

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

mu pomagamo.

- Izdelava dokumentacije in priprava navodil za uporabo aplikacije.
- Uporaba več modulov v programu in izdelava lastne knjižnice razredov in metod.
- Delo z dinamičnimi podatkovnimi strukturami.

TRI V VRSTO

Igra ***Tri v vrsto*** je igra za dva igralca, ki polagata ploščice dveh barv na mrežo 3 x 3 polj (ali pa rišeta križce in krožce na karo papirju). Zmaga tisti, ki mu prvemu uspe postaviti v vrsto ali diagonalo tri svoje ploščice.

Radi bi napisali program, ki bo igralcema omogočal igranje te enostavne igrice. V ta namen bomo šli preko vrste gradnikov. Slednje bomo tudi ilustrirali z različnimi zgledi. Spoznali bomo osnove razvojnega okolja, pojasnili kako gradimo aplikacijo s pomočjo obrazca in grafičnih gradnikov, ki jih polagamo na obrazce, ter kako pišemo odzivne dogodke. Naučili se bomo uporabljati enega od najkompleksnejših gradnikov *DataGridView* in dinamično ustvarjati okenske gradnike in njihove dogodke. Obenem bomo ponovili, kako pišemo lastne metode in kakšen je pomen varovalnih blokov.

MICROSOFT Visual C# 2010 - OSNOVNI POJMI: Rešitve (Solutions) in projekti (projects)

Microsoft Visual Studio je razvojno okolje (IDE – *Integrated Development Enviromnment*), ki omogoča razvoj konzolnih, namiznih in spletnih aplikacij na platformah Microsoft Windows, *Windows Mobile*, *Windows CE*, *.NET Framework*, *.NET Compact Framework* in *Microsoft Silverlight.* V okviru razvojnega okolja lahko uporabljamo različne programske jezike: Visual C#, Visual C++ , J# ali pa Visual Basic.

Glede na potrebe programerjev Microsoft ponuja kar nekaj verzij razvojnega okolja: *Professional* (namenjen posameznikom za osnovne razvojne naloge), *Premium* (namenjen posameznikom in projektnim skupinam za razvoj aplikacij) in *Ultimate* (obširna množica orodij ki pomagajo razvijalcem in skupinam izdelati visoko kvalitetne projekte od oblikovanja do instalacije). Poleg tega Microsoft ponuja še kar nekaj drugih sorodnih orodij za spodbujanje kreativnosti in izdelavo zahtevnih in sodobnih rešitev.

Najzanimivejši, ker so dostopni brezplačno, pa so za široke množice prav gotovo produkti *Visual Studio Express*. Odločimo se lahko za različice, ki podpirajo katerega koli od prej naštetih

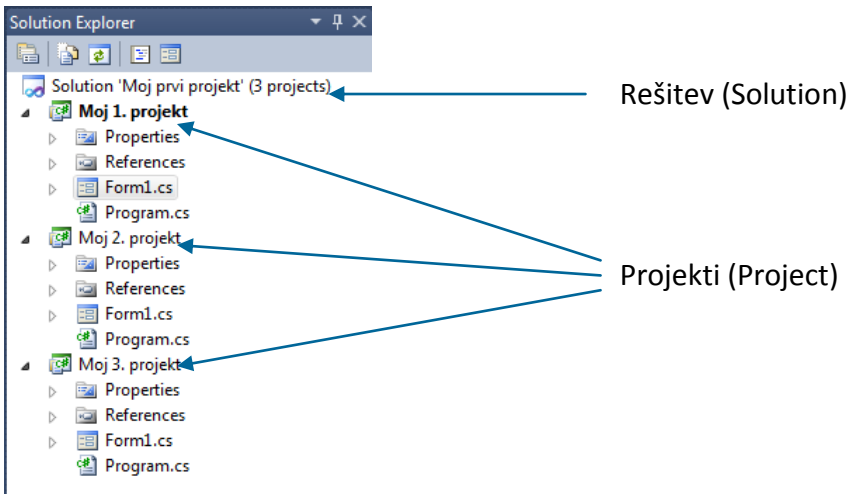
jezikov. V tej literaturi bomo uporabljali najnovejšo različico *Visual C# 2010 Express* (kratica ***VCE***).

Največja organizacijska enota v okolju *MS Visual Studio* in tako tudi v *Visual C# 2010 Express* je **rešitev** - **Solution**. Posamezna rešitev obsega *celotno* delo, ki se tiče nekega problema. Vsebuje enega ali več projektov **(Project)**, a običajno je projekt en sam. Projekt vsebuje izvorne datoteke in še vse ostale datoteke (npr. slike) , ki jih potrebuje prevajalnik za izdelavo izvršilne datoteke (.exe) oz. knjižnjice (.dll). Rešitev, ki vsebuje več projektov, je primerna za velike projekte, ki jih razvija več razvijalcev hkrati. Le-tem je tako omogočeno neodvisno delo na projektih, ki skupaj tvorijo ustrezno rešitev.

Projekt vsebuje izvorne datoteke aplikacije, ki jih prevajalnik (*compiler*) prevede v objektne datoteke vrste *.obj*, povezovalnik (*linker*) pa le-te poveže v *izvedbeno* datoteko aplikacije vrste *.exe* ali *dinamično* knjižnico vrste *.dll*. Oba postopka skupaj se imenujeta gradnja projekta (*building*).

Vsaka izvedba neke aplikacije zahteva ločen projekt. Tudi če želimo npr. za isto aplikacijo zgraditi izvedbeno datoteko vrste *.exe* in izvedbeno datoteko vrste dinamične knjižnice *.dll*, moramo uporabiti za vsako od njiju ločen projekt.

Okno *Solution Explorer* prikazuje rešitev, projekte in seznam njihovih datotek:



Slika 1: Okno z rešitvijo in projekti v njej.

Zapomnimo si:

- Posamezen odprt primerek *Visual C# 2010 Express* v operacijskem sistemu Windows obsega le eno rešitev.
- Rešitev obsega enega ali več projektov ter morebitne odvisnosti med njimi.
- Posamezen projekt lahko pripada eni ali pa več rešitvam.

- Posamezni projekti v neki rešitvi so lahko tudi v različnih programskih jezikih (takih, ki jih podpira razvojno orodje).

Razvojno orodje običajno ustvari (razen če spremenimo nastavitve) za vsak projekt ločeno mapo na disku. V tej mapi sta tudi podmapi *"obj"* ter *"bin"***,** v katere *VCE* shranjuje datoteke, ki sestavljajo projekt.

VCE uporablja dva tipa datotek (*.sln* in ***.suo***) za shranjevanje nastavitev, ki ustrezajo neki rešitvi. Ti dve datoteki, ki ju s skupnim imenom imenujemo *rešitev - solution files*, izdela *Solution Explorer*. Opremi ju z vsemi informacijami in podatki, ki so potrebni za prikaz grafičnega vmesnika in upravljanje z datotekami našega projekta. Razvijalcu projekta se tako ob odpiranju razvojnega okolja ni potrebno vsakič ukvarjati z okoljskimi nastavitvami.

Rešitev (*Solution*) je torej sestavljena iz dveh datotek:

- *.sln (Visual Studio Solution)*, ki vsebuje popoln opis o vsebini rešitve. Organizira projekt in njegove komponente v skupno rešitev in zagotovi ustrezno okolje s potrebnimi referencami in njihovim položajem na mediju - disku; datoteko lahko uporablja več razvijalcev aplikacije.

- ***.****suo (Solution User Options)*, ki vsebuje podatke o pozicijah oken. Datoteka je specifična za posameznega razvijalca, hrani pa vse potrebne nastavitve, zato da jih pri ponovnem odpiranju projekta ni potrebno nastavljati ponovno.

Datoteka *.sln* je nepogrešljiva, datoteka *.suo* pa je pogrešljiva: rešitev lahko zgradimo tudi brez nje.

Okenske aplikacije (Windows Forms) v Visual C# 2010 Express Edition

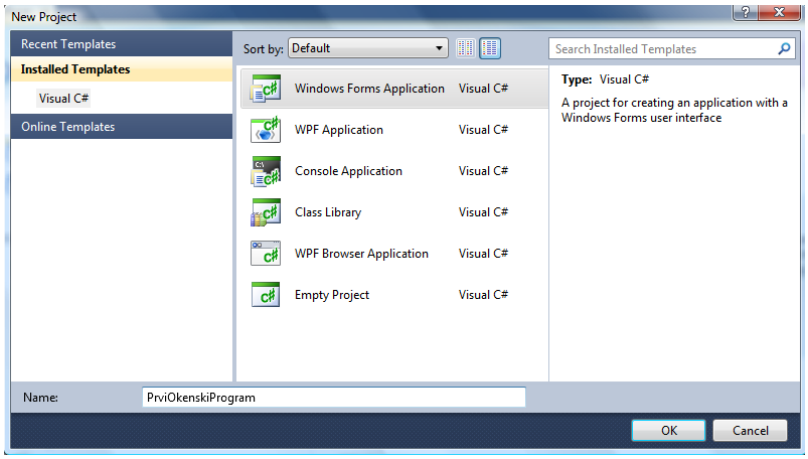
Razvojno orodje **Visual C#** vsebuje tudi vsa potrebna orodja za razvoj okenskih aplikacij. S pomočjo **Visual Designer**-ja lahko ustvarimo uporabniški vmesnik, ki temelji na t.i. obrazcih (*Forms*). Enostavni projekti lahko vsebujejo en sam obrazec, kompleksnejši pa celo množico različnih obrazcev, ki jih uporabnik po potrebi odpira in zapira. Pri gradnji aplikacije na obrazce postavljamo gradnike, **Visual C#** pa sam generira programsko kodo, ki ustreza uporabniškemu vmesniku, ki ga oblikujemo. Največja razlika med gradnjo konzolnih in vizuelnih aplikacij pa je v tem, da smo pri konzolnih aplikacijah v glavnem delali z eno samo datoteko, v katero smo pisali kodo, pri vizualnih aplikacijah pa je datotek več, vsaka od njih pa ima točno določen pomen. Pri gradnji projektov se bomo opirali na predznanje, ki smo ga pridobili v preteklosti: zanke, tabele,

metode, strukture, razredi in objekti, datoteke in še kaj. Vse o teh podatkovnih tipih in številne vaje lahko najdete npr. v datotekah

- http://uranic.tsckr.si/C%23/C%23.pdf,
- http://uranic.tsckr.si/C%23/Prakticno%20programiranje.pdf,
- http://uranic.tsckr.si/VISUAL%20C%23/VISUAL%20C%23.pdf,
- http://uranic.tsckr.si/C%23/.

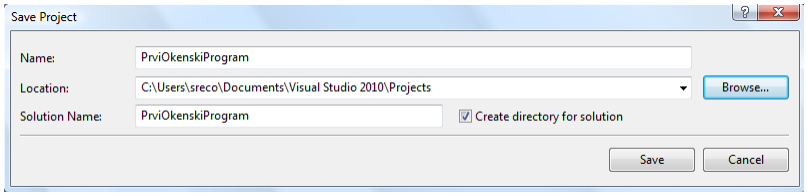
Ustvarjanje osnovnih okenskih aplikacij v okolju Visual C#

Za izdelavo prve okenske aplikacije odpremo meni *File → New Project*. Odpre se novo okno **New Project** v katerem izberemo opcijo **Windows Forms Application**. Na dnu okna, v polje *Name*, napišimo še ime naše prve okenske aplikacije, npr. *PrviOkenskiProgram* in kliknemo gumb *OK*. Na ekranu se prikaže celotno razvojno okolje za gradnjo novega projekta. Še preden naredimo karkoli, novo začeti projekt takoj shranimo. Izberemo *File→Save All* (ali pa kliknemo ikono *Save All* v orodjarni).



Slika 2: Ustvarjanje novega projekta.

Odpre se pogovorno okno *Save Project* za shranjevanje projekta. Ime projekta (*Name*) je že zapisano, izbrati ali potrditi moramo le še lokacijo (*Location*), kjer bo naš projekt shranjen. Privzeta nastavitev ja kar mapa Project znotraj mape *Visual Studio 2010*, seveda pa se lahko s klikom na gumb *Browse* prestavimo v drugo mapo ali pa kjerkoli ustvarimo novo mapo, v kateri bo naš projekt.



Slika 3: Vnos imena projekta in mape, kjer bo le-ta shranjen.

Čeprav smo v prejšnjem odstavku zapisali, da v polje *Name* vnesemo ime naše prve okenske aplikacije, to ne drži povsem. Dejansko gre za ime imenskega prostora (*namespace*) znotraj katerega bomo ustvarili nov projekt. Dejansko ime projekta, ki ga delamo znotraj določenega imenskega prostora, bomo vnesli šele ob kliku na *SaveAll* v razvojnem okolju. In kakšen je pomen pojma imenski prostor: zaenkrat bo dovolj da

vemo, da kadar različni razvijalci delajo na istem projektu, uporabljajo vsak svoj imenski prostor za razvoj. Znotraj le-tega, jim potem ni potrebno skrbeti za probleme, ki lahko nastanejo zaradi podvajanja imen (npr. imen razredov). Mi bomo v naših projektih običajno imenski prostor poimenovali enako kot projekt.

Razvojno orodje nam ponuja tudi možnost, da je ime rešitve (*Solution Name*) drugačno od imena programa, a v naših projektih imena rešitve zaenkrat ne bomo spreminjali. Kljukica v polju *Create Directory for solution* omogoča, da bo v mapi, ki smo jo izbrali ali ustvarili za naš projekt (*Location*), ustvarjena nova mapa z imenom našega projekta, v tej mapi pa bodo vse datoteke, ki sestavljajo naš projekt.

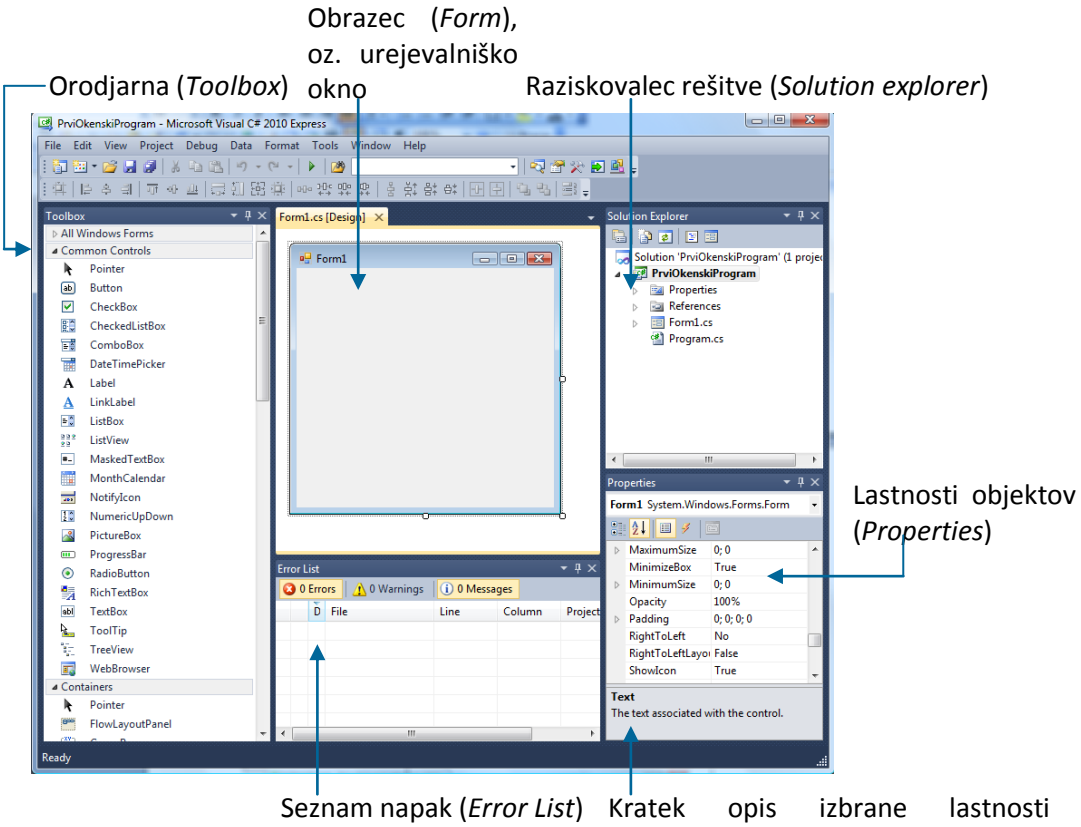
Projekt dokončno shranimo s klikom na gumb *Save*. Pri vsakem prevajanju se bo novo stanje zapisalo v izbrano mapo. Obenem se bo vselej ustvaril tudi izvršilni program (datoteke s končnico *.exe - executable file*), ki nastane kot končni rezultat našega projekta. Ta datoteka se nahaja v mapi *imeProjekta\bin\Debug* znotraj mape, v kateri je naš projekt. Če želimo našo že izdelano aplikacijo prenesti na drug računalnik, je potrebno le skopirati izvršilno datoteko in le-to prenesti na drug računalnik (na katerem pa mora biti nameščen *Microsoftov Framework* – to je programska oprema, ki jo namestimo na računalnik na katerem teče operacijski sistem *Windows*, vsebuje pa številne knjižnjice z rešitvami za uporabnike in razvijalce aplikacij).

Posvetimo se sedaj najprej opisu razvojnega okolja. Po vnosu imena projekta in shranjevanju imamo na vrhu razvojnega okolja t.i. *glavno okno*, ki vsebuje vrstico z meniji, pod njo pa orodjarno s hitrimi gumbi za realizacijo posebno pomembnih opravil. Pod glavnim oknom je odprt nov prazen obrazec (*Windows Form*) v oblikovalskem pogledu (*Design View*), okoli njega pa nekaj odprtih oken, ki nam pomagajo pri gradnji projekta.

Število oken, ki so prikazana v razvojnem okolju je odvisno od nastavitev. Zgornja slika prikazuje najpogosteje odprta okna, pri čemer pa lahko katerokoli od oken tudi minimiziramo oz. zopet prikažemo. Če je npr okno *Toolbox* skrito, kliknemo na *Toolbox* na levem robu ekrana in ko se prikaže, ga lahko s klikom na priponko fiksiramo na tem delu ekrana. Če pa je npr. okno *Toolbox* povsem zaprto, ga ponovno odpremo z izbiro *View → Other Windows→Toolbox*. Enako pravilo velja tudi za okni *Solution Explorer* in *Properties,* pa seveda za vsa ostala okna (razen glavnega, ki je odprto ves čas).

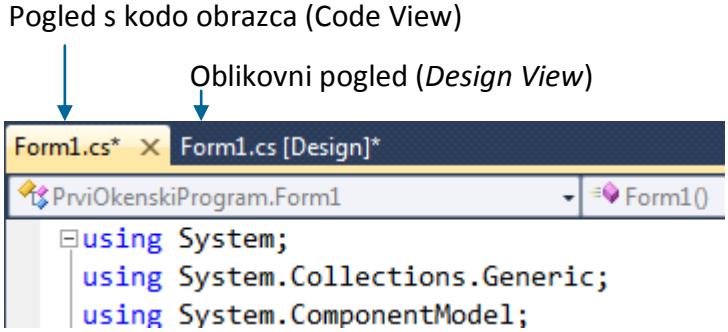
Okna, ki sestavljajo razvojno okolje imajo naslednji pomen:

- *Toolbox*: sestavljena je iz več palet z gradniki. Projekt v Visual C# namreč gradimo tako, da na obrazce postavljamo gradnike. Na paleti *All Windows Forms* so zbrani prav vsi gradniki po abecednem redu, na ostalih paletah pa le gradniki, ki po neki logiki spadajo skupaj: na paleti *Common Controls* so tako zbrani najpogosteje uporabljeni gradniki, na paleti *Menus & Toolbars* gradniki za delo z meniji, orodjarno, ipd. Posamezno skupino gradnikov odpremo s klikom miške na znak **+** pred imenom skupine, zapremo pa s klikom na znak **–** pred imenom skupine.



Slika 4: Razvojno okolje.

- Obrazec oz. urejevalniško okno: privzeto je v tem oknu slika obrazca, na katerega bomo polagali gradnike. S tipko *F7* (ali pa klikom na *View→Code*) lahko prikažemo datoteko, v katero bomo pisali kodo (spremenljivke, objekte, metode, odzivne dogodke), ki pripadajo temu obrazcu. V oknu *Code/Design View* se pokaže nov zavihek. Če sedaj kliknemo zavihek (*Tab*), ki ima dodano besedico *[design]* se vrnemo nazaj na oblikovni pogled obrazca.



Slika 5: Preklapljanje med oblikovnim pogledom (*Design*) in pogledom s kodo obrazca.

S tipkama *Shift+F7* (oziroma *View→Designer*) preklopimo nazaj na vizuelni del obrazca.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

- Solution Explorer: raziskovalec rešitve je okno, v katerem so prikazane vse mape in datoteke znotraj našega projekta. Pomen posameznih datotek in map bomo spoznavali kasneje, ko bomo ustvarjali projekte.

- Properties: okenske (vizuelne) aplikacije gradimo tako, da na obrazec postavljamo gradnike, ki jih jemljemo iz okna *Toolbox*. V oknu *Properties* so prikazane lastnosti in dogodki tistega gradnika na obrazcu, ki je trenutno izbran (tudi obrazec je gradnik). Med njimi je tudi ime *(name)*, ki določa ime posameznega gradnika. Z () je označeno, da ne gre za lastnost z imenom name. S pomočjo tega imena se v projektu na posamezne gradnike tudi sklicujemo. Ime mora biti enolično na nivoju celotnega projekta. Pri tem nam je razvojno okolje v veliko pomoč, saj nam za vsak nov gradnik predlaga privzeto ime. *Visual C#* poimenuje imena gradnikov tako, da jim zaporedoma dodeli imena z dodano številko na koncu (*label1*, *label2*, *label3*,… ali pa *button1*, *button2*, …). Imena gradnikov lahko poljubno spremenimo (to je pri obsežnejših projektih celo priporočljivo oz. skoraj nujno, saj sicer izgubimo pregled nad gradniki), a pri tem je potrebno paziti, da se imena ne podvajajo, sicer bomo dobili obvestilo o napaki. Kadar poimenujemo gradnike s svojim imenom, je priporočljivo, da uporabljamo zveneča imena: npr *gumbVnos*, *gumbZapri*, *gumbAzuriraj* … Na začetku v naših projektih privzetih imen gradnikov ne bomo spreminjali, a le v primerih, ko gradnikov istega tipa ne bo veliko.

- *Error List:* okno s seznamom napak. Ko namreč nek projekt prevedemo, so v njem lahko napake, ki so prikazane v tem oknu. Klik na posamezno vrstico v tem oknu nas postavi v urejevalnik kode na mesto, kjer se ta napaka nahaja.

Razvojno okolje nam ponuja tudi druga okna, ki nam pomagajo pri gradnji projekta, ki pa jih zaenkrat ne bomo potrebovali.

Če se v oknu *Toolbox* z miško zapeljemo čez poljuben gradnik in za trenutek počakamo, se v okvirčku izpiše osnovni namen tega gradnika.

Več o posameznih oknih bomo spoznali v nadaljevanju, ob ustvarjanju projektov.

V naslednji tabeli so zbrane osnovne bližnjice za določena opravila v razvojnem okolju, ki jih lahko naredimo s pomočjo tipkovnice.

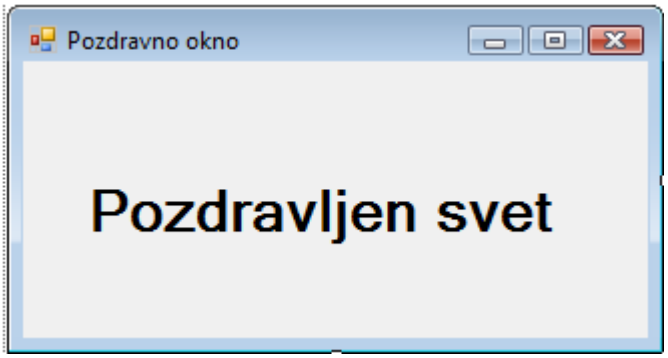
Bližnjica	Razlaga
F7	Preklop med pogledom *Design View* in *Code View*.
Shift+F7	Preklop med pogledom *Code View* in *Design View.*
F9	Nastavitev prekinitvene točke.
F12	Skok na definicijo spremenljivke, objekta ali metode.

Shift+F12	Iskanje vseh referenc metode ali spremenljivke.
Ctrl+M, Ctrl+M	Razširi / Skrij del kode (npr. vsebino metode) v urejevalniku.
Ctrl+K, Ctrl+C	Zakomentiraj označeni del kode v urejevalniku.
Ctrl+K, Ctrl+U	Odkomentiraj označeni del kode v urejevalniku.
Shift+Alt+Enter	Preklop med celostranskim pogledom na urejevalniško okno in normalnim pogledom.
Ctrl+I	Zaporedno iskanje določene besede (spremenljivke, metode, ..); vse enake besede v oknu se označijo.

Tabela 1: Seznam bližnjic v razvojnem okolju.

Nadaljujmo sedaj z gradnjo našega prvega okenskega programa, poimenovanega *PrviOkenskiProgram*. Najprej kliknimo na obrazec, se postavimo v okno *Propeties*, izberemo lastnost *Text* in vpišemo besedilo "Pozdravno okno". Vnos potrdimo z <*Enter*> oz. ponovnim klikom na obrazec. Privzeto ime obrazca (lastnost *Name*) je *Form1* in ga ne bomo spreminjali. Na obrazec sedaj postavimo naš prvi gradnik. V oknu *Toolbox* izberimo (kliknimo) gradnik *Label* – oznaka (nahaja se na paleti *Common Controls*), nato pa kliknimo na površino obrazca na mesto, kamor želimo postaviti ta gradnik. Dobil je privzeto ime *label1*. Vsak gradnik, ki ga postavimo na obrazec ima na začetku privzeto ime in lastnosti, ki pa jih lahko spremenimo v oknu *Properties*. Spomnimo se, da ga najdemo v pregledovalniku lastnosti pod (name). Gradnik je namenjen prikazu poljubne informacije na obrazcu. Pogosto ga tudi postavimo ob kak drug gradnik in vanj zapišemo besedilo, s katerim opišemo namen tega gradnika.

Prepričajmo se, da je gradnik *label1* na obrazcu izbran in se preselimo v okno *Properties* (če le-to ni odprto, ga odpremo z *View→Other Windows→Properties Window*). Med lastnostmi najprej poiščimo lastnost *Text* in jo spremenimo v "*Pozdravljen svet*". Poiščimo še lastnost *Font*, kliknimo na tripičje in v pogovornem oknu izberemo določeno vrsto pisave, stil in velikost. V urejevalniškem oknu nato izberemo obrazec (na njegovih robovih se prikažejo kvadratki) in ga ustrezno pomanjšamo (razširimo), da dobimo približno takole sliko.

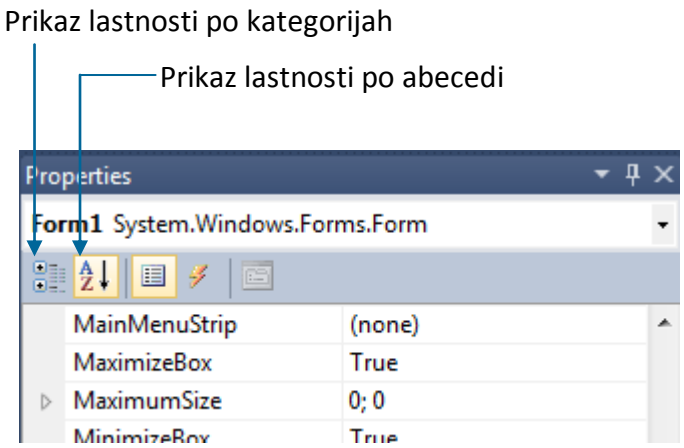


Slika 6: Prvi okenski program

Gradnikom na obrazcu lahko poljubno spreminjamo velikost. Izberemo gradnik, ki ga želimo povečati/pomanjšati (tako da nanj kliknemo…) , nato pa ga z miško raztegnemo/skrčimo na želeno velikost. Če je na obrazcu več gradnikov, je možna njihova poravnava tako, da nek gradnik približamo drugemu in prikažeta se vodoravna in navpična črta, ki omogočata natančno poravnavo.

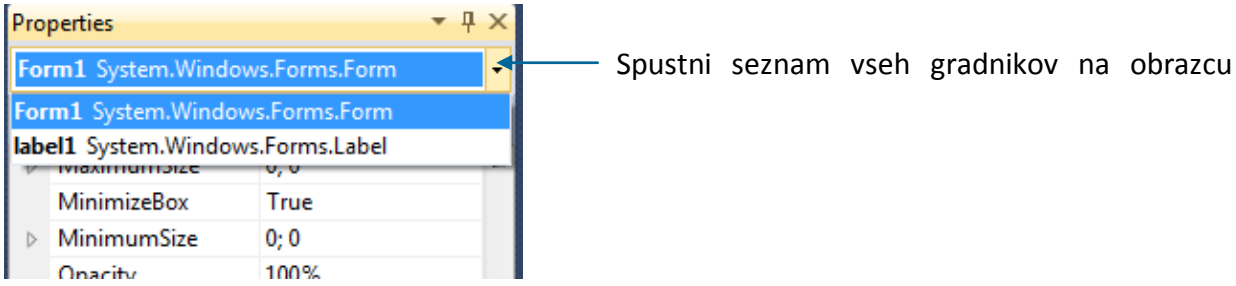
Ker smo v oknu *Design View* nazadnje kliknili na obrazec (tudi obrazec je gradnik), so v oknu *Properties* sedaj prikazane lastnosti obrazca. Vsak od gradnikov ima cel kup lastnosti, posamezne izmed njih pa bomo spoznavali v nadaljevanju.

Lastnosti gradnikov v oknu *Properties* so lahko nanizane na dva načina: po abecednem redu ali pa po kategorijah. S klikom izberemo način, ki nam trenutno bolj ustreza.



Slika 7: Urejanje lastnosti gradnikov.

V oknu *Properties* lahko vidimo tudi seznam vseh gradnikov, ki smo jih postavili na obrazec. Poskrbimo za to, da bo izbran gradnik obrazec (*Form1*), nato v oknu *Properties* odpremo spustni seznam vsebovanih gradnikov.

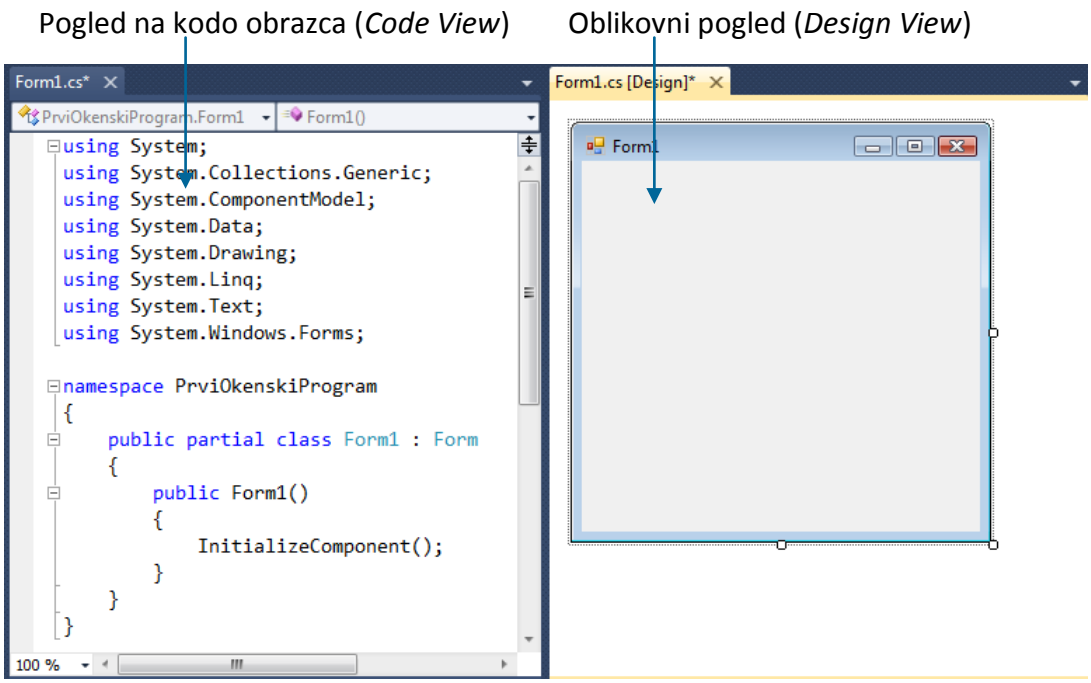


Slika 8: Seznam vseh gradnikov na obrazcu.

Odpre se okno v katerem so nanizani vsi gradniki, ki so trenutno na obrazcu. Izberemo lahko kateregakoli in v oknu *Properties* se prikažejo njegove lastnosti v oknu *Design View* pa je ta gradnik izbran.

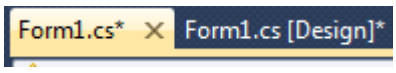
Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Omenjeno je že bilo, da je vsak obrazec v *Visual C#* predstavljen na dva načina: v oblikovnem pogledu (*Form1.cs [design])* in v pogledu za urejanje kode (*Form1.cs*). Oblikovni pogled (*Design View*) in pogled za urejanje kode (*Code View*) sta med seboj neodvisna, tako da lahko na ekranu hkrati prikažemo oba pogleda, kar je še posebej ugodno kadar imamo na razpolago velik monitor.



Slika 9: V urejevalniškem oknu imamo lahko hkrati odprta oba pogleda na obrazec: pogled *Code View* in pogled *Design View*.

Kadar je na vrhu okna (zavihka oz. okna urejevalnika), v katerem je naša koda, znak zvezdica (*), to pomeni, da je bila na tem obrazcu narejena vsaj ena sprememba, potem, ko smo ga nazadnje shranili. Ročno shranjevanje ni potrebno, saj se shranjevanje izvede avtomatsko takoj, ko poženemo ukaz za prevajanje. Seveda pa je ročno shranjevanje priporočljivo takrat, kadar smo zapisali veliko vrstic kode, pa prevajanja še ne želimo pognati.



Slika 10: Zvezdica na vrhu okna pomeni, da zadnje stanje projekta še ni shranjeno.

Oba pogleda v urejevalniškem oknu lahko na ekranu dobimo takole: preklopimo najprej na pogled za urejanje, nato pa z miško primimo zavihek *Form1.cs [design]* in ga povlecimo nad pogled za urejanje kode. Spustimo tipko na miški in prikaže se *Pop Up* meni s tremi opcijami. Izberimo npr. opcijo *New Vertical Tab Group*. Okno za urejanje kode se sedaj razdeli na dva dela: v enem imamo predstavljen pogled za urejanje kode obrazca, v drugem pa oblikovni pogled obrazca. Seveda lahko novo okno kadarkoli zopet zapremo in se tako vrnemo v prejšnji pogled.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Če smo na obrazec pomotoma postavili gradnik, ki ga ne potrebujemo, ga lahko odstranimo na dva načina: označimo ga s klikom miške in pritisnemo tipko *Delete* na tipkovnici, ali pa kliknemo desno tipko miške in v prikazanem *Pop Up* meniju izberemo opcijo *Delete*.

Preglejmo sedaj še vsebino okna **Code View**:

Na začetku so stavki, ki predstavljajo vključevanje že obstoječih imenskih prostorov v naš projekt.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

Sledi napoved imenskega prostora, v katerem ustvarjamo naš projekt (njegovo ime je enako imenu projekta) in nato deklaracija razreda *Form1*, ki je izpeljan (deduje – o dedovanju bomo govorili kasneje) iz splošnega razreda *Form* (osnoven, prazen obrazec). Pojasnimo še pomen rezervirane besedice *partial* v glavi razreda *Form1*: ta besedica označuje, da je na tem mestu napisan le del razreda *Form1* (*partial* - delno), ostali del (tisti del, ki ga pri polaganju gradnikov na obrazec za nas gradi razvojno okolje) pa je napisan v drugi datoteki (vsebino te datoteke lahko prikažemo v urejevalniškem oknu tako, da v *Solution Explorerju* kliknemo na vrstico *Form1.Designer.cs*). V konstruktorju razreda Form1 je klicana metoda *InitialComponent*, ki pa se dejansko nahaja v datoteki *Form1.Designer.cs*. To datoteko za nas gradi razvojno okolje. S tem so razvijalci razvojnega okolja poskrbeli za to, da ni celotna koda na enem mestu, kar bi imelo za posledico nepreglednost in težave pri ažuriranju projektov. Bistvena prednost pa je v tem, da je na ta način ločena koda, ki jo pišemo sami in koda, ki jo za nas gradi razvojno okolje. Namesto klica metode *InitialComponent* v konstruktorju, bi v večini primerov lahko na tem mestu zapisali kar celotno vsebino datoteke *Form1.Designer.cs*.

```
namespace PrviOkenskiProgram //imenski prostor, v katerem gradimo naš projekt
{
    //razred Form1 deduje razred Form (osnovni obrazec)
    public partial class Form1 : Form
    {
        public Form1()  //konstruktor razreda Form1
        {
            /*metoda, ki poskrbi za inicializacijo objektov, njihovih
            lastnosti in dogodkov – vsebino te metode si lahko ogledamo
            datoteki Form1.Designer.cs v Solution Explorerju*/
            InitializeComponent();
        }
    }
```

```
}
```

Nadaljujmo sedaj z našim prvim projektom. V glavnem oknu kliknemo na ikono *Start Debugging* (ali pa izberemo *Debug→Start Debugging*, ali pa stisnemo tipko *F5*) in naš projekt se bo prevedel (seveda, če je brez napake, a ker doslej še nismo pisali nobene kode, naj napak ne bi bilo). Kot rezultat prevajanja se na zaslonu pojavi novo okno, v katerem se izvaja naš prvi projekt. Zaenkrat je v tem oknu le pozdravno besedilo, ima pa okno v svojem desnem zgornjem kotu tri sistemske ikone za minimiranje, maksimiranje in zapiranje projekta. Obenem je nekje na disku (znotraj mape, ki smo jo na začetku določili pri kliku na *Save All*, in sicer v mapi *Bin→Debug*) nastala datoteka *PrviOkenskiProgram.exe* – to je t.i. izvršilna datoteka (*executable file*). Ta datoteka (program) predstavlja rezultat našega dela (projekta). To je tudi datoteka, ki jo lahko kadarkoli prenesemo na drug računalnik in jo tam izvajamo kot samostojno aplikacijo (seveda je potrebno včasih, zaradi nastavitev, obenem kopirati še kake druge datoteke, a o tem kasneje). Na ciljnem računalniku pa mora biti seveda nameščeno Microsoftovo okolje (*Framework*).

Okno, v katerem teče naš pravkar prevedeni projekt zaprimo, da se vrnemo nazaj v razvojno okolje. Sedaj lahko zapremo tudi razvojno okolje (ali pa odpremo nov projekt). Še prej s klikom na gumb *Save All* zagotovimo, da je celoten projekt res shranjen!

Če v oknu *Properties* izberemo katerokoli lastnost in nanjo naredimo desni klik, se prikaže okno, v katerem je vrstica *Description*. Če je ta vrstica odkljukana, se nam na dnu okna *Properties* prikaže kratek opis izbrane lastnosti.

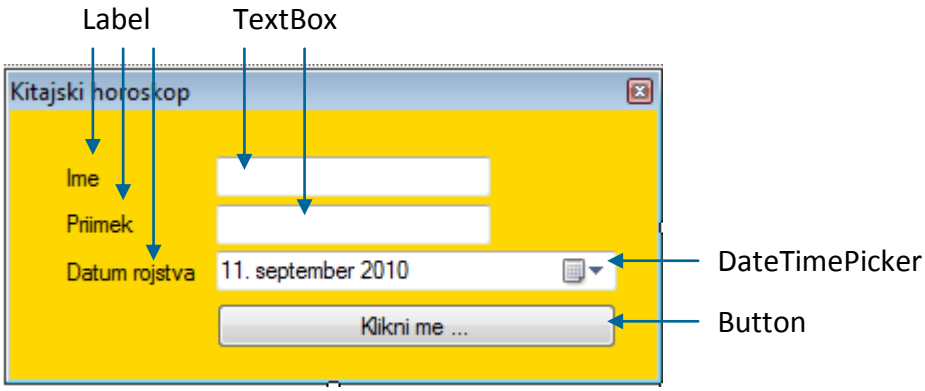
Kitajski horoskop

Radi bi napisali program, ki bo od uporabnika zahteval vnos imena, priimka in rojstnega datuma. Ob kliku na gumb na obrazcu, bo program uporabniku sporočil njegovo znamenje v kitajskem horoskopu!

Ustvarimo torej nov projekt (*File → New Project*), ga poimenujmo in shranimo pod imenom *KitajskiHoroskop*. V oknu *Properties* nastavimo obrazcu naslednje lastnosti:

- *Text*: vpišimo besedilo *Kitajski horoskop*
- *FormBorderStyle*: izberimo lastnost *FixedToolWindow*: ta stil okna označuje, da v bo v naslovni vrstici tega okna le en gumb, to je gumb za zapiranje okna. V času izvajanja programa uporabnik okna ne bo mogel minimirati ali maksimirati, niti ga ne bo mogel pomanjšati ali ga povečati.
- *BackColor*: v spustnem seznamu, ki ga lahko odpremo pri tej lastnosti, izberimo jeziček *Web* in v njem npr. barvo *Gold*.

Na tako pripravljen obrazec nato postavimo nekaj gradnikov (vsi se nahajajo na paleti *Common Controls*) in jih razporedimo tako, kot kaže slika:



Slika 11: Gradniki, ki jih potrebujemo na obrazcu *Kitajski horoskop*.

Oglejmo si gradnike, ki smo jih poleg oznake (label) še uporabili. To so:

- *TextBox*: osnovni namen tega gradnika je, da uporabnik programa vanj vnese neko besedilo.
- *DateTimePicker*: gradnik je nemenjen izbiri datuma.
- *Button*: gumb, ki je namenjen izvajanju nekega dogodka, ko uporabnik nanj klikne.

Vsem trem oznakam in gumbu na obrazcu spremenimo lastnost *Text*, da bo besedilo ustrezalo sliki. Lastnosti ostalih gradnikov ne spreminjamo.

Glavni način uporabe gumba (gradnika button) je ta, da napišemo metodo, ki naj se izvede ob uporabnikovem kliku nanj.

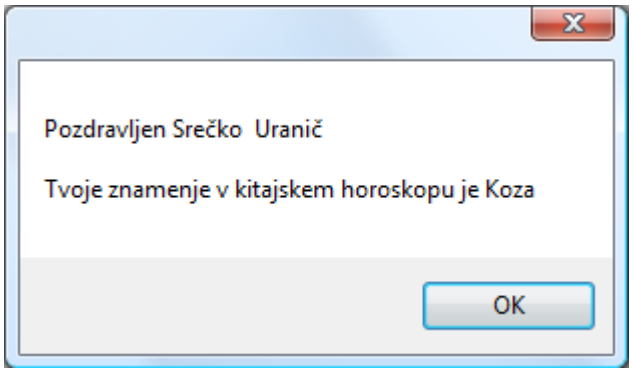
Namreč ena od osnovnih idej pri gradnji okenskih aplikacij je uporaba dogodkovnega programiranja. O tem bomo več povedali v nadaljevanju. Zanekrat povejmo le, da s pomočjo razvojnega okolja za posamezne gradnike napišemo, kaj se bo zgodilo, če pride do določenega dogodka (npr. če klinemo na gumb, če se sopremeni vsebina vnosnega polja …).

Gumb najprej označimo, nato pa v oknu *Properties* kliknemo na ikono *Events* pod vrhom tega okna. Na ta način prikažemo vse že pripravljene dogodke, na katere se gradniki tipa *Button* lahko odzivajo. Imena dogodkov v oknu *Properties* so izbrana tako, da nas asociirajo na njihov pomen. Med dogodki izberemo dogodek *Click* in nanj dvokliknemo. Razvojno okolje nam v datoteki s kodo ustvari ogrodje metode, ki se bo izvedla ob uporabnikovem kliku na gumb. Obenem se v urejevalniškem oknu oblikovni pogled na obrazec spremeni v pogled s kodo obrazca, kurzor pa je že postavljen v telo pravkar ustvarjene metode. Metoda je tipa *void*, njeno ime ustreza namenu. Ima tudi dva parametra, o katerih pa bo več napisanega kasneje. Sedaj moramo le še napisati ustrezne stavke, ki naj se izvedejo ob uporabnikovem kliku na gumb.

Znamenje v kitajskem horoskopu je vezano na leto rojstva. Dobimo ga tako, da preverimo ostanek letnice rojstva pri deljenju z 12: če je le ta enak 0 je znamenje "Opica", če je ostanek 1 je znamenje "Petelin" in tako naprej vse do ostanka 11, ki predstavlja znamenje "Koza".

Potrebovali bomo torej letnico rojstva. Gradnik tipa *DateTimePicker* ima lastnost *Value*. V njej je zapisan celoten datum. Letnico pa nato dobimo s pomočjo lastnosti *Year* torej z *dateTimePicker1.Value.Year*.

Algoritem za določitev znamenja si lahko zamislimo tako, da vsa znamenja shranimo v enodimenzionalno tabelo nizov, indeks v tej tabeli pa predstavlja ostanek pri deljenju letnice z 12, preko katerega pridemo do njegovega imena. Rezultat (niz z znamenjem) prikažemo v sporočilnem oknu *MessageBox*.



Slika 12: Pozdravno sporočilo projekta *Kitajski horoskop*.

Tu ne gre za gradnik, ki bi ga dodali na uporabniški vmesnik, ampak ga ustvarimo v sami kodi. To je v bistvu nek že pripravljen obrazec, namenjen sporočilom uporabniku, prikažemo pa ga z metodo *Show*. Ta metoda ima cel kup različnih načinov uporabe. Na začetku bomo uporabljali le osnovni način: metodi za parameter posredujemo poljuben tekst (niz, ki mora biti seveda v dvojnih narekovajih), ki bo kot sporočilo prikazan v tem oknu. Pri oblikovanju tega teksta se držimo pravil, ki jih že poznamo pri delu z nizi.

Tekst v sporočilnem oknu lahko prikažemo tudi večvrstično tako, da za prehod v novo vrsto uporabimo znakovno konstanto za prehod v novo vrsto (\n ali pa \n\r).

```
//celotna koda dogodka, ki se izvede ob kliku na gumb button1
private void button1_Click(object sender, EventArgs e)
{
    //od uporabnika zahtevamo vnos imena in priimka
    if (textBox1.Text.Length == 0 && textBox2.Text.Length == 0)
        MessageBox.Show("Vnesi ime in priimek!");
    else
    {
        //definicija enodimenzionalne tabele znamenj v kitajskem horoskopu
```

```
        string[] Kitajski = new string[12] { "Opica", "Petelin", "Pes",
"Merjasec", "Podgana", "Bivol", "Tiger", "Zajec", "Zmaj", "Kača", "Konj",
"Koza" };
        /*Iz izbranega datuma rojstva izluščimo letnico: celoten datum določa
lastnost Value gradnika dateTimePicker1, letnico pa nato dobimo s pomočjo
lastnosti Year*/
        int leto = dateTimePicker1.Value.Year;
        //izračunamo ostanek pri deljenju z 12
        int ostanekPriDeljenjuZ12 = leto % 12;
        /*ostanek pri deljenju z 12 določa indeks znamenja v tabeli  vseh
          znamenj kitajskega horoskopa*/
        string znamenje = Kitajski[ostanekPriDeljenjuZ12];
       //še zaključno obvestilo uporabniku
        MessageBox.Show("Pozdravljen "+textBox1.Text+" " +
textBox2.Text+"\n\nTvoje znamenje v kitajskem horoskopu je " + znamenje);
    }
}
```

Če smo se pri izbiri dogodka zmotili in npr. namesto dogodka Click izbrali na nek drug dogodek, lahko ogrodje metode v datoteki s kodo (v našem projektu je to datoteka *Form1.cs*) odstranimo takole: iz imena dogodka razberemo za kateri gradnik gre (če se npr. ime dogodka prične z *button1*, gre prav gotovo za dogodek gradnika *button1*). Preklopimo na oblikovni pogled in izberemo ustrezen gradnik. V oknu *Properties* nato najprej prikažemo vse dogodke (klik na ikono *Events*) in se pomaknemo v vrstico, kjer vidimo ime dogodka, ki ga želimo odstraniti. Ime (v desnem stolpcu) nato enostavno pobrišemo. Ogrodje dogodka(kodo) v datoteki s kodo bo nato odstranilo razvojno okolje samo, seveda pa mora biti telo te metode prazno: v njej ne sme biti niti komentarja.

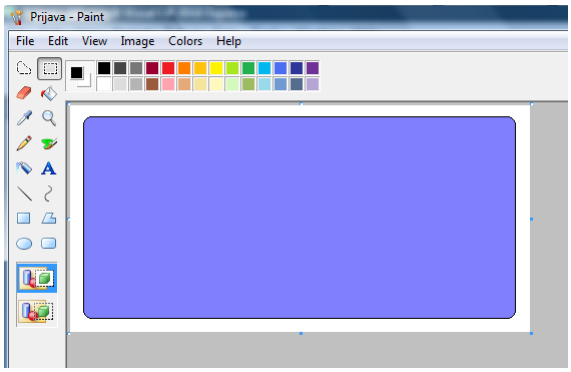
S kakšno metodo se gradnik odzove na določen dogodek povemo tako, da gradnik najprej izberemo, nato pa v oknu *Properties* dvoklinemo v vrstico z ustreznim dogodkom: razvojno okolje bo ob tem pripravilo ogrodje tega dogodka. Vsak gradnik na obrazcu pa ima enega od dogodkov privzetega. To pomeni, da za ustvarjanje ogrodja tega dogodka ni potrebno le-tega izbrati v oknu *Properties*, ampak obstaja tudi krajši način: na obrazcu le dvokliknemo na ta gradnik in ogrodje odzivne metode je pripravljeno. Privzeti dogodek (to je najbolj značilen dogodek, na katerega se gradnik odziva) gradnika *Button* je *Click*, gradnika *TextBox* dogodek *TextChanged*, itd.

Prijavni obrazec

Slej ko prej bomo želeli napisati projekt, v katerega se bo moral uporabnik preko nekega obrazca najprej prijavit (glej Slika 13).

Privzeto je vsak obrazec pravokotne oblike. Kadar pa želimo ustvariti projekt, v katerem bo obrazec poljubne oblike, lahko to storimo tako, da najprej s pomočjo poljubnega grafičnega

urejevalnika najprej pripravimo sliko obrazca (ki mora biti na nekem enobarvnem ozadju). V projektu nato obrazcu to sliko nastavimo kot lastnost *BackgroundImage*, potrebna pa je še nastavitev lastnosti *TansparencyKey*. Ker želimo, da prijavni obrazec ne bo pravokotne oblike, moramo s pomočjo grafičnega urejevalnika (to je lahko kar Slikar) najprej pripraviti ustrezno sliko za ozadje. V našem primeru bomo kar s slikarjem pripravili enobarvno sliko pravokotnika z okroglimi robovi. Da pa bo transparentnost res 100%, sliko shranimo v datoteko tipa *24-bit Bitmap (*.bmp; *.dib)*.

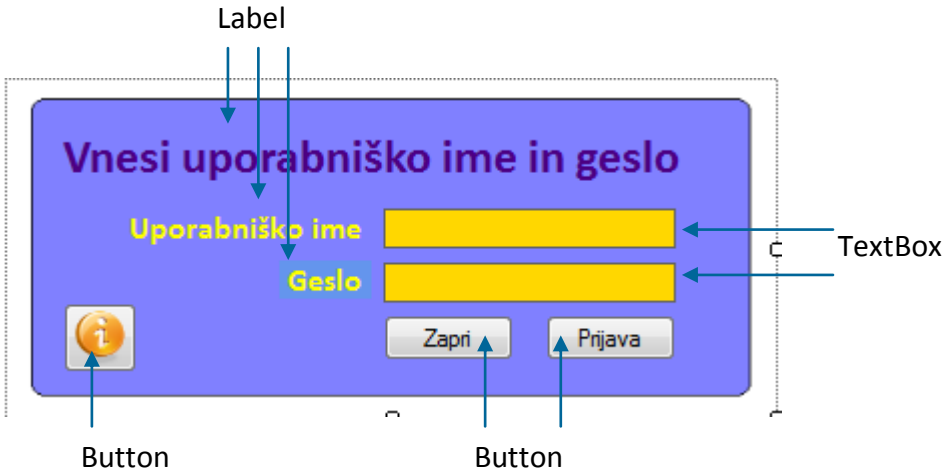


Slika 14: V Slikarju pripravimo ozadje obrazca.

Sliko shranimo in začnimo nov projekt, ki ga poimenujmo *Prijava*. V oknu *Properties* nato obrazcu nastavimo naslednje lastnosti:

- *BackgroundImage*: za ozadje nastavimo pravkar ustvarjeno sliko.
- *BackgroundImageLayout*: *Stretch*(slika naj bo raztegnjena čez celotno ozaddje).
- *TransparencyKey*: nastavimo na *White* (ker je pač slika na belem ozadju).
- *FormBorderStyle*: nastavimo na *None*.

Na obrazec nato postavimo naslednje gradnike:



Slika 15: Gradniki na prijavnem obrazcu.

Če se v urejevalniškem oknu *CodeView* z miško postavimo nad neko spremenljivko ali pa objekt, se pod njim pokaže okvirček z besedilom, ki nam pove tip te spremenljivke ali objekta.

Gradnikom na obrazcu pustimo imena kar privzeta, tako, kot jih določi razvojno okolje. Kadar pa bo na obrazcu več podobnih gradnikov, pa je pametno, da gradnike poimenujemo po svoje (*tBIme*, *tBGeslo*, lIme, *lGeslo*, *bZapri*, *bPrijava*, *bNamig* …)

Gradnikoma *TextBox* nastavimo lastnost *Color* (npr. *Gold*, nikakor pa ne belo barvo, ki smo jo uporabili kot ključ za transparentnost) in lastnost *BorderStyle* (*FixedSingle*). Gradniku *TextBox* za vnos gesla nastavimo še lastnost *PasswordChar* (npr. znak zvezdica). Ko bo uporabnik v polje vnašal znake, se bodo na ekranu prikazovale le zvezdice. Spodni levi gumb na obrazcu je namenjen prikazu sporočilnega okna z namigom o uporabniškem imenu in geslu. Gumbu priredimo še ustrezno sliko (lastnost *BackgroundImage*). Sliko poiščemo v svojem arhivu ali pa s pomočjo spleta, najbolje pa je, da si za naše potrebe vnaprej pripravimo nek nabor slik (*.gif*, *.bmp*, *.jpg, .jpeg*, *.png*, *.wmf*), ki jih bomo potem uvažali v projekte.

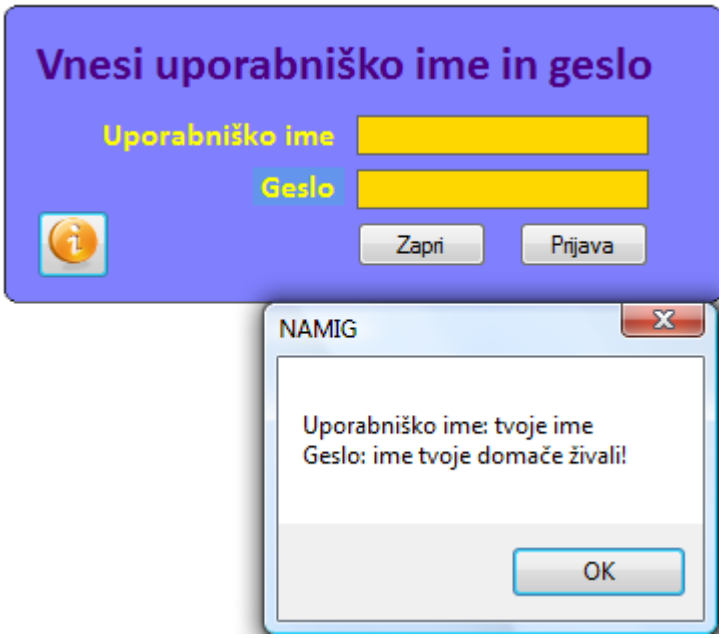
Napisati moramo še kodo za ustrezne dogodke povezane z vsemi tremi gumbi na obrazcu. Tako smo v kodi za klik na gumb *button1* uporabili sporočilno okno s štirimi parametri. Razlaga parametrov je zapisana v komentarju, več o vseh možnih oblikah sporočilnega okna pa zapisano v nadaljevanju.

```
//definicija spremenljivke števila uporabnikovih poskusov vstopa v program
public int poskusov = 0; //uporabniku bomo dovolili le 3 poskuse prijave
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "")
        MessageBox.Show("Nisi vnesel uporabniškega imena!");
    else if (textBox2.Text == "")
        MessageBox.Show("Vnesi geslo!");
    //uporabniško ime smo si zamislili kot "Visual C#", geslo pa "prijava"
    else if ((textBox1.Text == "Visual C#") && (textBox2.Text == "prijava"))
    {
        MessageBox.Show("Geslo pravilno!");//navadno sporočilno okno
        Application.Exit();//zapremo obrazec;
    }
    else
    {
        poskusov++;//povečamo število poskusov
        /*uporabimo sporočilno okno s štirimi parametri: prvi parameter tipa
         niz (string) je izpis v oknu, drugi parameter tipa niz je napis na
         oknu, tretji parameter označuje število in vrsto gumbov, ki se
         prikažejo v oknu, zadnji parameter pa vrsto ikone v tem oknu*/
        MessageBox.Show("Napačno geslo!\n\nŠtevilo dosedanjih poskusov
"+poskusov.ToString()+"\nNa voljo še poskusov: "+(3-poskusov),"Napačno
geslo!",MessageBoxButtons.OK,MessageBoxIcon.Information);
        if (poskusov==3)
```

```
            Application.Exit();
        }
    }
    private void button2_Click(object sender, EventArgs e)
    {
        /*v sporočilnem oknu za informacijo izpišemo namig za uporabniško ime in
          geslo. Zopet uporabimo sporočilno okno s štirimi parametri*/
        MessageBox.Show("Uporabniško ime: tvoje ime\nGeslo: ime tvoje domače
živali!","NAMIG", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    private void button3_Click(object sender, EventArgs e)
    {
        /*obrazec zapremo z metodo Exit razreda Application, lahko pa tudi kar z
          metodo Close. Razlika med metodama je v tem, da Close zapre trenutno
          okno, Exit pa konča program in zapre še ostala okna.*/
        Application.Exit();
    }
```

Seveda je naša aplikacija več ali manj neuporabna, saj ne počne drugega, kot preverja pravilnost gesla. V "pravi" aplikaciji bi seveda po vnosu pravilnega gesla namesto tega, da končamo program, odprli novo okno, ali pa glede na uporabniško geslo odprli točno določen projekt znotraj naše rešitve.

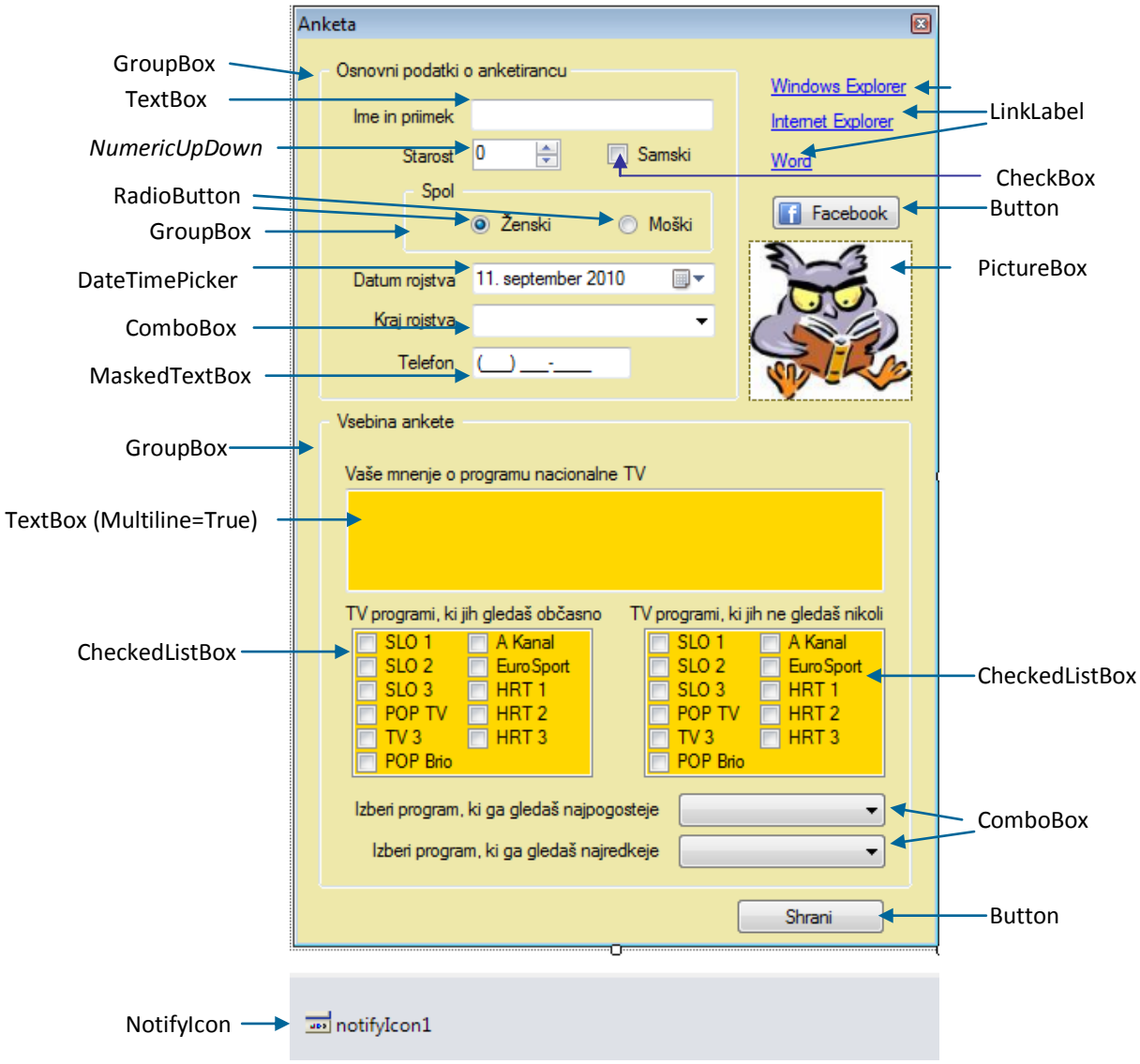
Če projekt zaženemo in kliknemo na spodnji levi gumb, dobimo takole sliko:



Slika 16: Sporočilno okno z nastavitvami na prijavnem obrazcu.

Anketa

Na TV Slovenija so nas prosili, da naj pripravimo obrazec o gledanju TV programov. V ta namen so nam posredovali podatke, ki bi jih želeli izvedeti od TV gledalcev. Rezultate želijo imeti formatirane v ustrezni tekstovni datoteki. Na obrazcu, ki ga bomo za ta namen pripravili, bomo uporabili kopico novih gradnikov: *NumericUpDown*, *CheckBox*, *LinkLabel*, *ComboBox*, *MaskedTextBox*, *PictureBox*, *CheckedListBox*, *NotifyIcon*, *RadioButton* in *GroupBox*. Nekaj o njihovem namenu in najpomembnejših lastnostih si bomo ogledali kar ob izpeljavi tega zgleda. Razporedimo in poimenujmo jih takole:



Slika 17: Gradniki na anketnem obrazcu.

Večina lastnosti gradnikov je privzetih, nekaterim gradnikom pa lastnosti malo popravimo:

- Obrazec *Form1*: lastnost *BackColor* nastavimo na *PaleGoldenrod*, za lastnost *Text* pa zapišimo *Anketa*.
- *LinkLabel*: gradnik ima vlogo povezave (*hyperlink*): njegovemu dogodku *LinkClicked* priredimo zagon poljubnega programa (npr. *Windows Explorer*-ja, *Internet Explorer*-ja, urejevalnika, ..). Gradnikom tega tipa na obrazcu nastavimo lastnost *Text* (tako kot kaže slika) in *VisitedLinkColor* (nastavimo poljubno barvo – ko bo uporabnik kliknil na gradnik, se bo njegova barva ustrezno spremenila).
- *NumericUpDown*: Gradnik je namenjen le vnosu celih števil. Z lastnostma *Max* in *Min* lahko določimo največje in najmanjše število, ki ga uporabnik lahko vnese.
- *GroupBox*: gradnik tega tipa je namenjen združevanju gradnikov, ki po neki logiki spadajo skupaj. V našem primeru ima ta gradnik še poseben pomen pri združevanju radijskih gumbov. Med gumbi, ki so znotraj posameznega obrazca, je lahko naenkrat izbran (ima pikico) le eden. Kadar pa imamo dve (ali več skupin) tovrstnih gumbov, vsako skupino postavimo znotraj združevalnega gradnika. Pravilo, da lahko uporabnik izbere le en radijski gumb, namreč velja znotraj posameznega združevalnega gradnika. Ko ga postavimo na obrazec, ga lahko premaknemo tako, da ga z miško "primemo" za križec v zgornjem levem robu.
- *ComboBox*: Uporabimo ga za vnos poljubnega števila vrstic besedila, med katerimi bo uporabnik lahko izbiral s pomočjo spustnega seznama. Na obrazcu so trije gradniki tega tipa:
  - *ComboBox* za izbiro kraja: ko ga postavimo na obrazec, najprej kliknemo na trikotnik v desnem zgornjem robu tega gradnika. Odpre se okno, v katerem izberemo opcijo *Edit Items…* (ali pa v oknu *Properties* kliknemo na lastnost *Items*) in vnesemo nekaj krajev (vsakega v svojo vrsto, brez ločil!). Lastnost *DropDownStyle* pustimo nespremenjeno (*DropDown*): na ta način uporabniku omogočimo izbiro enega od že vnesenih krajev, poleg tega pa ima možnost, da v prazno polje vpiše nek drug kraj.
  - Dva tovrstna gradnika za izbiro najbolj in najmanj priljubljenega TV programa. S pomočjo lastnosti *Items…* vnesemo nekaj TV programov, vsakega v svojo vrsto, brez ločil. Obema nastavimo še lastnost *DropDownStyle* na *DropDownList*. Na ta način bomo uporabniku dovolili le izbiro enega od vnesenih programov, ne bo pa mogel vpisati svojega.
- *Button*: zgornjemu gumbu poleg lastnosti *Text* (*Facebook*) določimo še sliko za ozadje.
- *MaskedTextBox*: v gradniku lahko nastavimo ustrezno masko, ki je uporabnikom v pomoč pri vnašanju podatkov, npr. za vnos telefonske številke, datuma, časa, …
- *TextBox*: gradniku za vnos mnenja o programu nacionalne TV nastavimo lastnost *Multiline* na *true*. Na ta način lahko v gradnik zapišemo poljubno število vrstic, lastnost *ScrollBars* pa na *Vertical* (nastavimo navpični drsnik). Lastnost *BackColor* tega gradnika naj bo npr. *Gold*.
- *CheckBox*: gradnik uporabniku omogoča izbiro ali brisanje ponujene možnosti.

- *CheckedListBox*: gradnik tega tipa vsebuje poljubno število gradnikov tipa *CheckBox*. V oba gradnika s pomočjo lastnosti *Items* (klik na tripičje ob lastnosti) vnesemo seznam TV programov, lastnost *BackColor* pa naj bo npr. *Gold*.
- *NotifyIcon*: ta gradnik omogoča prikaz ikonce v vrstici *Windows Taskbar* (običajno na spodnjem desnem robu zaslona). Ko gradnik postavimo na obrazec, se pokaže v prostoru pod obrazcem, na dnu urejevalniškega okna. Gradniku moramo nastaviti ustrezno ikono, za kar uporabimo lastnost *Icon*: kliknemo na tripičje ob lastnosti in s pomočjo okna, ki se odpre, poiščemo ustrezno datoteko tipa *.ico*.

Ostane nam še pisanje kode. Vsem trem gradnikom tipa *LinkLabel* nastavimo odzivne metode *LinkClicked*. Gumbu za zagon *Facebook*-a napišemo odzivno metodo dogodka *Click*, gumbu na dnu obrazca prav tako, v njej pa zapišemo kodo, s pomočjo katere bomo uporabnikove vnose zapisali v tekstovno datoteko *Anketa.txt*. Ta se nahaja v mapi *Bin→Debug* znotraj našega projekta.

Ko pišemo odzivno metodo na določen dogodek pogosto enostavno rečemo, da "nastavimo dogodek".

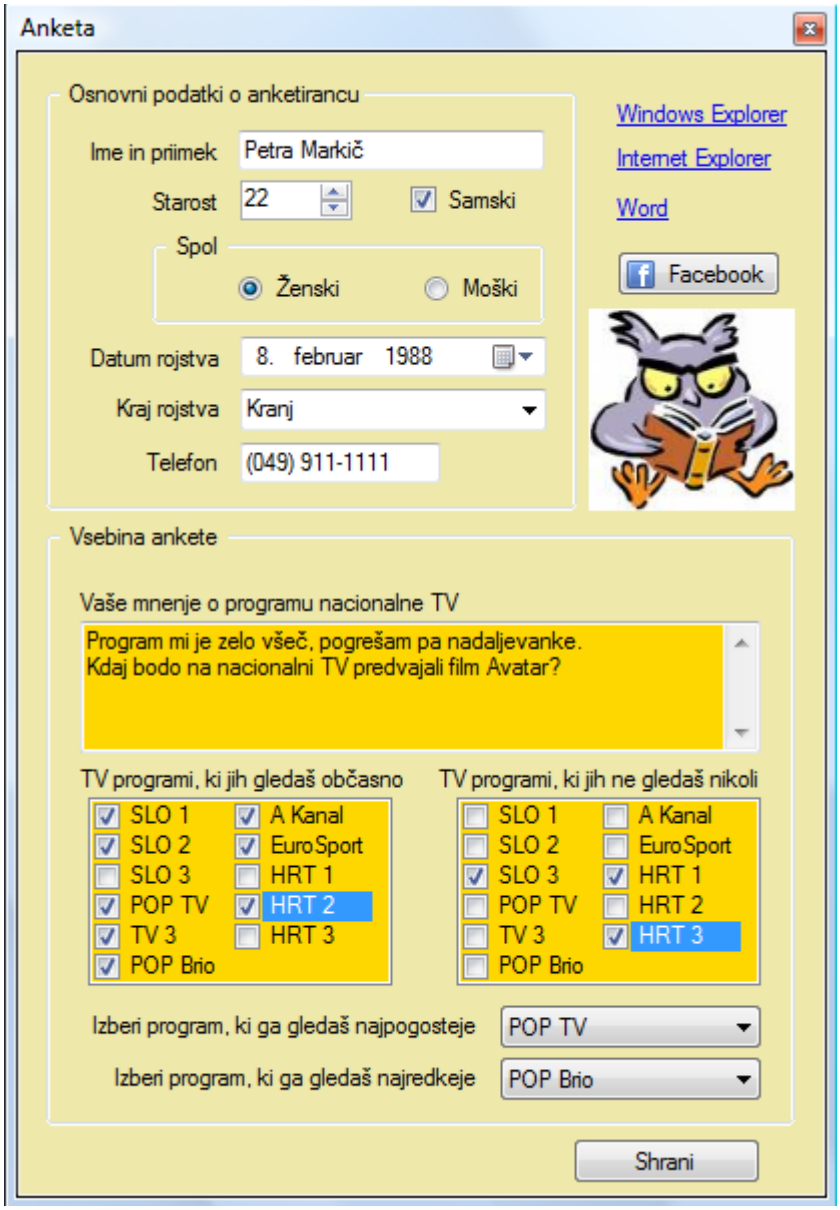
Pri delu se nam bo slej ko prej zgodilo, da bomo v pogledu *Code View* pomotoma pobrisali nek dogodek, ki nam ga je ustvarilo razvojno okolje (bodisi, da smo pomotoma dvokliknili na nek gradnik, ali pa smo v oknu *Properties* dvokliknili na napačen dogodek). Ko projekt nato skušamo prevesti dobimo obvestilo o napaki. Napako odpravimo tako, da v oknu *ErrorList* dvokliknemo na vrstico z napako, nakar nas razvojno okolje postavi v datoteko *.designer.cs* in sicer na mesto napake. Običajno je potrebno vrstico z napako (to je najava nekega dogodka, ki smo ga ravnokar pobrisali) samo pobrisati.

```
//metoda za dostop do Windows Explorerja
private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    try  //varovalni blok za obravnavo prekinitev
    {
        /*označimo, da je bila oznaka že kliknena, zato se ji bo barva
          spremenila. Nastavili smo jo z lastnostjo VisitedLinkColor*/
        linkLabel1.LinkVisited = true;
        //metoda Start nas postavi v mapo z rezultati ankete
        System.Diagnostics.Process.Start("C:\\Anketa\\Datum");
    }
    catch
    {
        MessageBox.Show("Napaka pri odpiranju programa ");
    }
}
//metoda za dostop do Internet Explorerja
```

```
private void linkLabel2_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    try
    {
        linkLabel1.LinkVisited = true;
        /*metoda Start za zagon Internet Explorer-ja in v njem strani
          S sporedi na RTV SLO1*/
        System.Diagnostics.Process.Start("IExplore",
                                         "http://www.napovednik.com/tv/slo1");
    }
    catch
    {
        MessageBox.Show("Napaka pri odpiranju programa ");
    }
}
//metoda za zagon urejevalnika Word
private void linkLabel3_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    try
    {
        System.Diagnostics.Process.Start("Winword.exe","Moji zapiski o
oddajah prejšnejega tedna.doc");//zagon Worda in odpiranje datoteke
    }
    catch
    {
        MessageBox.Show("Napaka pri odpiranju programa ");
    }
}
//metoda za dostop do Facebook-a
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        //metoda Start za odpiranje Facebook - a
        System.Diagnostics.Process.Start("IExplore", "http://sl-
si.facebook.com/");
    }
    catch
    {
        MessageBox.Show("Napaka pri odpiranju programa ");
    }
}

//metoda gumba Shrani za shranjevanje podatkov v tekstovno datoteko
private void button2_Click(object sender, EventArgs e)
{
    /*ime datoteke: ker nismo napisali poti do datoteke, bo le ta shranjena v
      mapi Bin -> Debug tekočega projekta*/
```

```
    string datoteka = "Anketa.txt";
    //podatkovni tok za pisanje v tekstovno datoteko
    StreamWriter pisi = File.AppendText(datoteka);
    string ime=textBox1.Text; //shranimo ime in priimek
    //ker je vrednost v gradniku NumericUpDown tipa Decimal, jo z metodo
    //Convert.Toint32 pretvorimo v celo število
    int starost=Convert.ToInt32(numericUpDown1.Value); //starost
    //status je odvisen od izbire oznake v gradniku CheckBox
    string status = "Poročen";
    if (checkBox1.Checked)
        status="Samski";
    string spol="ženski";//spol je odvisen od izbire radijskega gumba
    if (radioButton1.Checked)
        spol="moški";
    string datum=dateTimePicker1.Value.ToShortDateString();//datum
    string kraj=comboBox1.Text;//rojstni kraj
    string telefon=maskedTextBox1.Text; //telefon
    /*najljubše TV programe zapišemo v niz TVProgramiDa*/
    string TVProgramiDa="";
    for (int i = 0; i < checkedListBox1.CheckedItems.Count;i++ )
    {
          //posamezne programe ločimo z vejico
          TVProgramiDa = TVProgramiDa + checkedListBox1.CheckedItems[i]+',';
    }

    /*najmanj priljubljene TV programe zapišemo v niz TVProgramiNe*/
    string TVProgramiNe = "";
    for (int i = 0; i < checkedListBox2.CheckedItems.Count; i++)
    {
         //posamezne programe ločimo z vejico
         TVProgramiNe = TVProgramiNe + checkedListBox2.CheckedItems[i] + ',';
    }
    string najbolj = comboBox2.Text; //najpogosteje gledani TV program
    string najmanj = comboBox3.Text; //najredkeje gledani TV program
    string komentar = textBox2.Text; //komentar o nacionalni TV
    /*vse podatke zapišemo v datoteko, vmes pa postavimo ločilni znak ; */

    pisi.WriteLine(ime+';'+starost+';'+status+';'+spol+';'+datum+';'+kraj+';'
                   +telefon+';'+TVProgramiDa+';'+TVProgramiNe+';'+najbolj+';'
                   +najmanj+';'+komentar);
    pisi.Close();//zapremo podatkovni tok
    MessageBox.Show("Podatki so shranjeni v datoteki " + datoteka);
    Close();    //zapremo obrazec
}
```

Takole pa je videti obrazec z vnesenimi testnimi podatki:



Slika 18: Anketni obrazec z vnesenimi podatki.

Gradnik *CheckBox* ima lahko tri različna stanja, če je njegova lastnost *ThreeState* nastavljena na *True*. Stanja so *True*, *False* in *Indeterminate* (nedoločeno). Tri stanja so koristna npr. takrat, kadar prikazujemo podatke iz neke relacijske baze. Nekatera polja v bazi podatkov imajo namreč shranjeno vrednost *null*, ki pomeni, da vrednost ni določena ali pa je nepoznana. Kadar želimo to vrednost prikazati tudi v gradniku *CheckBox*, lahko izberemo stanje *Indeterminate*, ki ustreza vrednosti *null* v bazi podatkov.

Samo za vajo si poglejmo še, kako bi podatke, ki so že shranjeni v datoteki *Anketa.txt* prebrali in jih prenesli nazaj na isti obrazec (obraten postopek). Kodo bomo zapisali v metodo, ki se izvede

ob dogodku *Load* obrazca. To pomeni, da ko se obrazec prvič odpre (naloži – Load), bodo v njem že podatki iz prve vrstice te datoteke.

```
private void Form1_Load(object sender, EventArgs e)
{
    try
    {
        //datoteko odpremo za branje s pomočjo objekta tipa StremReader
        StreamReader beri = File.OpenText("Anketa.txt");
        //preberemo vrstico in jo shranimo v spremenljivko tipa niz
        string vrstica = beri.ReadLine();
        beri.Close();//datoteko zapremo
        //s pomočjo metode Split dele vrstice, ki so ločeni z znakom podpičje
        //prenesemo v tabelo nizov
        string[] vsebina = vrstica.Split(';');
        textBox1.Text = vsebina[0]; //v celici z indeksom 0 je ime
        //s pomočjo indeksa dostopamo še do ostalih podatkov v tabeli
        numericUpDown1.Value = Convert.ToDecimal(vsebina[1]);
        if (vsebina[2] == "Samski")
            checkBox1.Checked = true;
        if (vsebina[3] == "Ženski")
            radioButton1.Checked = true;
        else radioButton2.Checked = true;
        dateTimePicker1.Value = Convert.ToDateTime(vsebina[4]);
        comboBox1.Text = vsebina[5];
        maskedTextBox1.Text = vsebina[6];
        textBox2.Text = vsebina[11];
        //v celici z indeksom 7 so TV programi, med seboj ločeni z vejico.
        //s pomočjo metode Split jih prenesemo v tabelo daProgrami
        string[] daProgrami = vsebina[7].Split(',');
        foreach (string program in daProgrami)
        {
            for (int i = 0; i < checkedListBox1.Items.Count; i++)
            {
                //pogledamo vsak CheckBox posebej
                if ((string)checkedListBox1.Items[i] == program)
                {
                    checkedListBox1.SetItemChecked(i, true);
                }
            }
        }
        //še programi, ki jih gledamo občasno
        string[] neProgrami = vsebina[8].Split(',');
        foreach (string program in neProgrami)
        {
            for (int i = 0; i < checkedListBox2.Items.Count; i++)
            {
                //pogledamo vsak CheckBox posebej
                if ((string)checkedListBox2.Items[i] == program)
```

```
                {
                    checkedListBox2.SetItemChecked(i, true);
                }
            }
        }
        comboBox2.Text = vsebina[9];
        comboBox3.Text = vsebina[10];
    }
    catch
    {
       MessageBox.Show("Napaka pri branju datoteke!");
    }
}
```

Lastnosti, metode in dogodki gradnikov, razred *object*

Pojasnili smo že, da je koda, s katero se nastavijo lastnosti gradnikov, ki smo jih postavili na obrazec, zapisana v datoteki *.designer.cs*. Tam je poskrbljeno tudi, da se ustvari povezava med dogodkom in gradnikom – torej, da se nastavi s katero metodo naj gradnik reagira ob določenem dogodku. Podatke v tej datoteki navadno nikoli ne spreminjamo sami, saj za vsebino skrbi razvojno okolje samo. Lastnosti spreminjamo tako, da na obrazcu izberemo ustrezen gradnik, nato pa v oknu *Properties* nastavimo/spremenimo poljubno lastnost. Spremembe v datoteki . *designer.cs* bo nato opravilo razvojno okolje samo. Kar nekaj gradnikov smo že spoznali, prav tako njihove osnovne dogodke. Vseh lastnosti, metod in dogodkov gradnikov je preveč, da bi opisovali vsakega posebej, osnovne oz. najpomembnejše med njimi pa bomo spoznavali na konkretnih primerih.

Lastnosti

Pojem lastnost smo se spoznali že pri osnovah objektnega programiranja, pri pisanju lastnih razredov. Vemo, da s pojmom *lastnost* označujemo nastavitev, ki nam pove nekaj o samem gradniku, npr. obrazcu. Lastnosti gradnikom lahko določamo oz. spreminjamo v fazi načrtovanja projekta (v oknu *Properties*), ali pa med samim delovanjem aplikacije s pomočjo programske kode. Doslej smo jih nastavljali oz. spreminjali v glavnem preko okna *Properties*. Lahko pa jih spremenimo tudi v programsko (nekaj takih primerov smo v dosedanjih vajah že imeli). To lahko storimo tako, da v fazi načrtovanja projekta v urejevalniškem oknu (pogled *Code View*) napišemo ime gradnika, zapišemo piko in razvojno okolje nam s pomočjo sistema *IntelliSense* v okenčku pod besedilom prikaže pomoč pri izbiri ustrezne lastnosti oz. metode. To velja za vse gradnike.

Pri programski spremembi poljubne lastnosti obrazca (vendar pa ne vseh, ker so nekatere lastnosti gradnikov *ReadOnly*), se nanj ne moremo sklicevati preko imena (tako kot vedno v razredih), ampak preko parametra *this*, ki smo ga spoznali že pri razredih in objektih. Prevajalnik namreč vsaki metodi razreda doda še en prameter - kazalec, ki kaže na konkreten objekt (v

našem primeru obrazec) nad katerim deluje metoda. Temu parametru je ime *this*. Inicializira se ob klicu metode in ga uporabljamo za dostop do lastnosti tega objekta.

Napis na obrazcu *Form1* bi torej programsko spremenili takole:

```
this.Text = "Tole je spremenjen napis na vrhu obrazca!";
```

Ta stavek lahko zapišemo v katerikoli odzivni dogodek gradnikov, ali pa v nek dogodek obrazca.

Nekateri gradniki (med njimi tudi *TextBox*) imajo, če jih označimo, v desnem zgornjem kotu posebno puščico. Le-ta omogoča dodatne hitre nastavitve, ki pa se razlikujejo od gradnika do gradnika. Pri gradniku *TextBox* je dodatna hitra nastavitev lastnost *Multiline*. Če odkljukamo to možnost, lahko besedilo v gradniku *TextBox* raztegnemo čez večje število vrstic. Kadar želimo besedilo v tem gradniku programsko zapisati v več vrsticah, uporabljamo za prehod v novo vrsto znakovni konstanti \r\n.

Metode

Objekti (gradniki, tudi obrazci) poznajo tudi celo vrsto metod (pravimo jim tudi objektne metode). S pomočjo objektnih metod objektom (tudi obrazec je objekt) povemo, da naj nekaj storijo, oz. da naj se nekaj zgodi. Že pri osnovah OOP pa smo spoznali, da do metod v splošnem dostopamo preko imena objekta, operatorja pika in imenom metode, oziroma preko parametra *this*, npr.:

```
this.BringToFront();//obrazec postavimo pred vse druge obrazce.
```

Najpomembnejše metode obrazca prikazuje naslednja tabela:

Metoda	Razlaga metode
Activate	Lastnost je uporabna predvsem v primeru, ko imamo znotraj projekta odprtih več obrazcev, ki pa ne smejo biti skriti. Obrazec postane aktiven (se postavi v ospredje) in dobi t.i. "fokus".
BringToFront	Obrazec se postavi v ospredje, pred vse ostale obrazce (nasprotje od *SendToBack*).
Close	Zapiranje obrazca, obrazec se uniči.
Hide	Obrazec postane neviden, oz. se skrije. Je pa še vedno v pomnilniku in ga lahko kadarkoli ponovno prikažemo (z metodo *Show* ali *ShowDialog*).
Refresh	Ažuriranje izgleda obrazca in njegova osvežitev (ponoven izris obrazca).
SendToBack	Obrazec se postavi za vse ostale obrazce (nasprotje od *BringToFront*).
SetBounds	Uporablja se za pozicioniranje obrazca.

Show	Prikaz nemodalnega obrazca (normalno okno).
ShowDialog	Prikaz modalnega obrazca (pogovorno okno). Modalno okno je okno, ki ga moramo nujno zapreti, če želimo nadaljevati delo v enem od ostalih oken.

Tabela 2: Najpomembnejše metode obrazca.

Dogodki

Večina gradnikov vizuelnih aplikacij se odziva na dogodke. Dogodki so v bistvu metode, ki pa se zaženejo le tedaj, ko se zgodi nek dogodek sprožen s strani uporabnika, programske kode ali pa sistema. Metodam, ki se prožijo ob neki uporabnikovi akciji (kliku na gradnik, pritisku tipke na tipkovnici, premiku miške …) pravimo tudi *odzivne metode*. Nabor vseh možnih odzivnih metod je zapisan v oknu *Properties→Events* vsakega gradnika posebej. Kot že vemo iz prejšnjih primerov, določeno odzivno metodo sprožimo tako, da se v oknu *Properties* postavimo v prazno polje določenega dogodka, zapišemo ime metode (kadar želimo metodi prirediti svoje ime), ali pa le dvokliknemo v to polje (v tem primeru bo ime metode določilo razvojno okolje: ime bo sestavljeno iz imena gradnika, ki mu odzivna metoda pripada in imena dogodka – npr *button1_Click*).

Ob odprtju vsakega obrazca se najprej zgodi dogodek *Load*. Odzivno metodo za ta dogodek lahko zato uporabimo za programsko nastavljanje lastnosti obrazca. Pogosto vanj zapišemo tudi začetne vrednosti spremenljivk in objektov. Tega dogodka ne smemo zamenjati z dogodkom *Activated*, saj je med njima velika razlika. Dogodek *Load* se izvede le enkrat, ko so obrazec odpira prvič, dogodek *Activated* pa najprej na samem začetku, takoj za dogodkom *Load*, nato pa vselej ko se obrazec ponovno prikaže (če ga npr. minimiziramo, pa potem zopet prikažemo).

Najpomembnejše dogodke obrazca prikazuje naslednja tabela:

Dogodek	Razlaga, kdaj se dogodek zgodi
Load	Ob nastanku obrazca, ko je obrazec prikazan prvič.
Activated	Ko postane obrazec (ki ga je uporabnik skril npr. z metodo Hide) ponovno aktiven.
Deactivate	Ko aktiven obrazec postane neaktiven (npr. ko se uporabnik premakne na drug obrazec s klikom miške ali pa npr. preko neke bližnjice LinkLabel na neko spletno stran, ali pa je npr. lastnost ActiveForm programsko prirejena drugemu obrazcu).
Shown	Dogodek se zgodi, ko je obrazec prikazan prvič. V primeru, da je bil obrazec šele ustvarjen, se najprej zgodi dogodek *Load*, nato pa šele dogodek *Shown*. Dogodek pa se ne zgodi npr. ob maksimiranju obrazca, ali pa ob ponovnem

	izrisu obrazca in podobno.
FormClosing	Ob zapiranju obrazca, preden se obrazec zapre (z ukazom *Close* ali gumbom za zapiranje). Preden se obrazec zapre, se sprostijo oz. uničijo vse spremenljivke in objekti, ki so bili deklarirani na obrazcu (oz. na splošno v gradniku, ki se zapira).
FormClosed	Ko se obrazec že zapre (z ukazom *Close* ali gumbom za zapiranje).
Paint	Ob ponovnem izrisu obrazca (npr. tedaj, ko je bil obrazec delno zakrit ali pa pomanjšan).
Resize	Ob spremembi velikosti obrazca.

Tabela 3: Najpomembnejši dogodki obrazca.

Preimenovanje obrazcev in ostalih gradnikov in njihovih dogodkov

Včasih se zgodi, da želimo že pripravljen obrazec preimenovati. To lahko storimo na dva načina:

- V oknu *Solution Explorer* izberemo obrazec, ki ga želimo preimenovati (npr. *Form1.cs*) in ga preimenujemo: najprej ga izberemo, kliknemo desni miškin gumb in nato izberemo opcijo *Rename*. Seveda pa ga lahko preimenujemo tudi tako, da v oknu *Solution Explorer* ime obrazca najprej izberemo (ga kliknemo), počakamo nekaj trenutkov, nato pa ga še enkrat kliknemo. Po nekaj trenutkih se prejšnje ime obda z okvirčkom. Nato kar v njem spremenimo ime obrazca (končnice *cs* ni potrebno pisati, doda se avtomatsko). Po spremembi imena nam razvojno okolje v sporočilnem oknu izpiše obvestilo, da smo spremenili ime datoteke in nas vpraša, če želimo preimenovati tudi vse reference, ki se v tem projektu nanašajo na ta obrazec. Izberemo gumb *Da*!

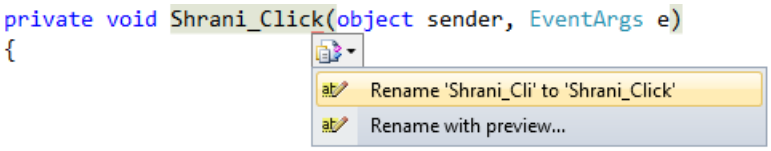
V oknu *Properties* izberemo lastnost *Name* tega obrazca in spremenimo ime. A v tem primeru bomo spremenili le ime temu obrazcu, ne pa tudi reference na ime. Ime obrazca v oknu *Solution Explorer* bo ostalo nespremenjeno. Boljši način za preimenovanja obraza je torej prvi.

Kadarkoli lahko preimenujemo tudi imena gradnikov in odzivnih metod na katere se ti gradniki odzivajo!

Ime gradniku lahko spremenimo tako, da ga najprej izberemo, nato pa v oknu *Properties* spremenimo nastavitev (*Name*) tega gradnika. Priporočljivo je, da gradnikom (pa tudi dogodkom) prirejamo zveneče imena, kar pomeni, da naj nas ime asociira na pomen gradnika. Tako bomo npr. gradniku *TextBox*, ki ga bomo uporabljali za vnos imena dali ime *tBIme* (ali pa npr. *vPIme*) gumbu za shranjevanje naj bo ime *bShrani(ali pa bShrani)*, ipd. Pri poimenovanju gradnikov smo uporabili t.i. *camelCase* konotacijo. Ob spremembi imena bo razvojno okolje avtomatično spremenilo tudi imena vseh referenc (kjerkoli v kodi) na ta gradnik.

- Imena odzivnih metod pa lahko spremenimo neposredno tudi v kodi. Recimo, da bi želeli ime odzivne metode *button1_Click* spremeniti v *Shrani_Click*. Postavimo se v glavo metode, ki ji želimo spremeniti ime in ime spremenimo. Po spremembi imena se pod zadnjim znakom novega imena metode pojavi majhen kvadratek: nanj se postavimo z miško, odpremo prikazan spustni seznam, v njem pa izberemo opcijo *Rename button1_Click to Shrani_Click (*a to moramo storiti takoj, neposredno po spremembi imena, sicer nam razvojno okolje pri tem ne bo več v pomoč!). Vse potrebne spremembe (tudi reference na ta dogodek) bo za nas nato opravilo razvojno okolje.

Poleg konotacije imenovane *camelCase* poznamo še konotacijo imenovano *PascalCase*. Edina razlika med njima je ta, da se imena pri prvi začenjajo z malo črko, pri drugi pa z veliko črko. *Microsoft* priporoča, da za imena spremenljivk uporabljamo konotacijo *camelCase*, za imena metod pa konotacijo *PascalCase*.



Slika 19: Sprememba imena odzivnega dogodka.

Odzivne metode lahko poimenujemo po svoje že pri samem ustvarjanju. Če želimo npr. ustvariti odzivno metodo dogodka *Click* nekega gumba, se v oknu *Properties→Events* najprej postavimo v polje *Click*. Namesto da v to polje dvokliknemo, zapišemo najprej poljubno ime metode (npr *KlikGumba*), ter šele nato dvokliknemo v to polje. Razvojno okolje bo za ime odzivne metode uporabilo naše ime.

Vse odzivne metode obrazca in gradnikov so tipa *void*: nič ne vračajo, le nekaj naredijo. Običajno so zasebne (*private*), kar pomeni da so dostopne le znotraj trenutnega razreda (obrazca, ki ga gradimo).

Parametri odzivnih metod (dogodkov).

Napišimo še nekaj o parametrih odzivnih metod:

- prvi parameter vsake odzivne metode je parameter *sender*, ki je tipa *object*. Vsem razredom v C#, ne glede na vrsto, je skupno, da je njihov skupni osnovni razred *System.Object*. Sem so všteti tudi osnovni in vsi ostali vrednostni tipi (tudi *int*, *float*, *decimal*, …). Vsi razredi imajo torej avtomatično za svojo osnovo (pravimu tudi, da *dedujejo* - več o dedovanju bo napisano v posebnem poglavju) razred *Object*. To pomeni, da avtomatsko že imajo na voljo metode in lastnosti, ki so napisane v tem razredu. Prav tako velja, da so vsi objekti torej sočasno tudi tipa *object*. Vse to so razvijalci jezika s pridom izkoristili tudi pri odzivnih metodah gradnikov. Ne glede na to, preko katerega gradnika (to je objekta izpeljanega iz nekega razreda, npr. objekta

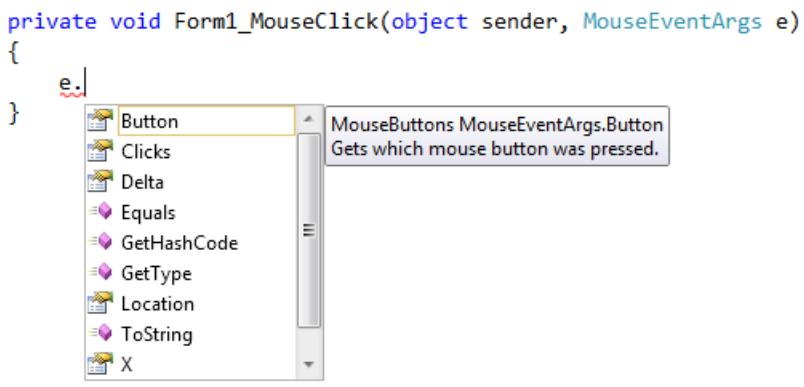
button1 izpeljanega iz razreda *Button*, objekta *textBox1* izpeljanega iz razreda *TextBox*, ipd.) smo v oknu *Solution Explorer* (ali pa z dvoklikom na gradnik) ustvarili ogrodje neke metode, vedno je njen prvi parameter imenovan *sender* (pošiljatelj), ki je splošnega tipa *object*. S pomočjo tega parametra lahko ugotovimo, nad katerim gradnikom se je zgodil določen dogodek. Poznavanje parametra *sender* je nepogrešljivo npr. tedaj, ko je na obrazcu veliko število gumbov in nam ni potrebno pisati odzivnih metod za vsak gumb posebej: enemu od njih napišemo odzivno metodo, ostalim pa v oknu *Properties→Events* ta isti dogodek le priredimo. Znotraj metode pa potem npr. s pomočjo stavka *if* preverjamo ime gradnikov in zapišemo ustrezno kodo.

Razred *object* kot takega (samostojno) uporabljamo le redko, a njegovo poznavanje je nepogrešlivo. Ima le privzeti konstruktor, pozna pa le nekaj metod. Omenimo le tri:

Metoda	Razlaga
Equals	Oceni, ali sta dva objekta enaka ali ne.
GetType	Omogoča dostop do tipa nekega objekta.
ToString()	Pretvorba objekta v niz - *string*.

Tabela 4: Metode razreda *object*.

- Drugi parameter odzivnih metod je dogodkovni parameter (objekt tipa *EventArgs*, ali pa nekega izpeljanega razreda – pojem izpeljanega razreda bo pojasnjen v poglavju o dedovanju) imenovan *e*. To je objekt, ki pogosto vsebuje številne lastnosti s pomočjo katerih lahko pridemo do informacij o samem dogodku, v katerem se nahajamo. Do seznama teh lastnosti najlaže pridemo tako, da v telesu odzivnega dogodka zapišemo oznako parametra, za njim pa piko, nakar nam sistem *IntelliSense* v okvirčku izpiše seznam vseh možnih lastnosti. Pri argumentih tipa *EventArgs* je teh lastnosti malo, pri nekaterih izpeljanih razredih pa precej več. Pri izpeljanem razredu *MouseEventArgs* (dogodku miške) lahko npr ugotovimo, kateri gumb je bil pritisnjen. Več o dogodkih miške pa bo zapisanega v nadaljevanju.



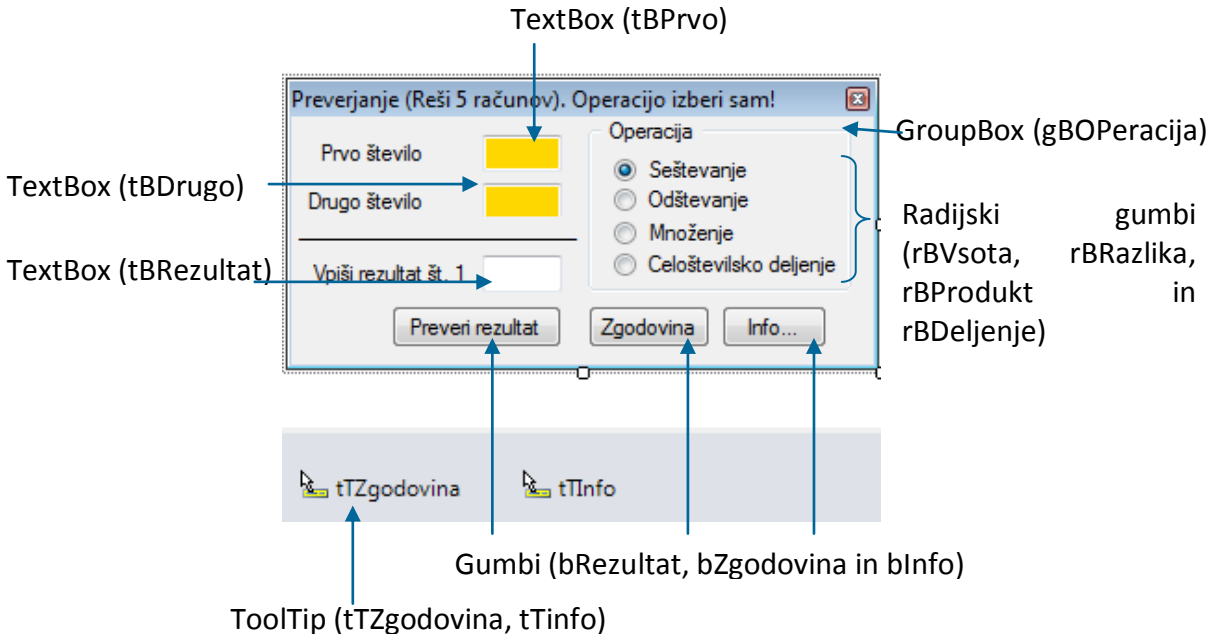
Slika 20: Lastnosti drugega parametra odzivnih metod.

S pomočjo parametra sender lahko preverimo (ali pa določimo) tudi tip objekta (ali pa gradnika, ki nas je poslal v določeno metodo). To lahko storimo s pomočjo operatorjev *is* in *as*, ki bosta razložena v nadaljevanju!

Preverjanje znanja osnovnih računskih operacij

Učitelj matematike v osnovni šoli nas je prosil, naj napišemo program za preverjanje znanja osnovnih računskih operacij. Učenci naj imajo na voljo 5 računov, operacijo (seštevanje, odštevanje, množenje in celoštevilsko deljenje) pa naj izberejo sami. Števila naj bodo naključna, vrednosti pa med 1 in 10. Število pravilnih odgovorov v vsakem poskusu naj se zapisuje v tekstovno datoteko, zraven pa še datum in ura poskusa.

Vse gradnike na obrazcu bomo poimenovali po svoje. Obrazec smo preimenovali *fPreverjanje*, imena ostalih gradnikov pa so razvidna s slike. Gradniki, ki niso poimenovani posebej, so vsi tipa *Label* in imajo privzeta imena - tudi vodoravna črta pod napisom "Drugo število" je tipa label. Njenalastnost *Text* je enaka "_____". Uporabili smo tudi gradnika *ToolTip*. Namen tega gradnika je prikaz pomoči (kratke informacije) v oblačku, če se, (ko je projekt zagnan) z miško postavimo nad gradnik, ki smo mu ta *ToolTip* priredili. V našem primeru smo gradnika vrste *ToolTip* priredili gumboma z napisom *Zgodovina* in *Info*.



Slika 21: Gradniki in njihova imena na obrazcu *FPreverjanje*.

Lastnosti gradnikov na obrazcu so prikazane v naslednji tabeli:

Gradnik	Lastnost	Nastavitev	Opis
Fpreverjanje	FormBorderStyle	FixedToolWindow	Uporabnik velikosti okna ne more spreminjati
tBPrvo	BackColor	Gold	
	TabStop	False	
	TextAlign	Right	Tekst v oknu bo poravnan desno
tBPrvo	BackColor	Gold	
	TabStop	False	
	TextAlign	Right	Tekst v oknu bo poravnan desno
rBVsota	Checked	True	Gumb je na začetku označen
bZgodovina	ToolTip on tTZgodovina	Dosedanji računi	Komentar v oblačku
bInfo	ToolTip on tTInfo	Informacije o programu	Komentar v oblačku
tTZgodovina	IsBaloon	True	Obvestilo bo v oblačku
	ToolTipIcon	Info	V oblačku bo ikona Pomoč
	ToolTipTitle	Pomen gumba	Zapis na vrhu oblačka
tTInfo	IsBaloon	True	Obvestilo bo v oblačku
	ToolTipIcon	Info	V oblačku bo ikona Pomoč
	ToolTipTitle	Pomen gumba	Zapis na vrhu oblačka

Tabela 5: Lastnosti gradnikov na obrazcu z imenom *FPreverjanje*.

V projektu smo uporabili tudi odzivno metodo *Load* obrazca: v njem smo poskrbeli za inicializacijo spremenljivk, ter za klic lastne metode *ZacetnaVrednost*, ki generira naključni števili in jih zapiše v gradnika *tBPrvo* in *tBDrugo*. Gumboma *bZgodovina* in *bInfo* smo priredili isto odzivno metodo z imenom *Info* (ime dogodka smo določili sami). Znotraj dogodka (metode) *Info* s pomočjo parametra *sender* ugotavljamo, kateri od gumbov je bil pritisnjen. Ob kliku na gumb z napisom *Zgodovina* se nam v sporočilnem oknu prikaže vsebina datoteke *Rezultati.txt* (dosedanji rezultati), ob kliku na gumb z napisom *Info* pa informacije o programu! Tudi radijski gumbi krmilijo skupni dogodek z imenom *rBVsota_CheckedChanged* (dogodek se izvede, če spremenimo izbiro radijskega gumba).

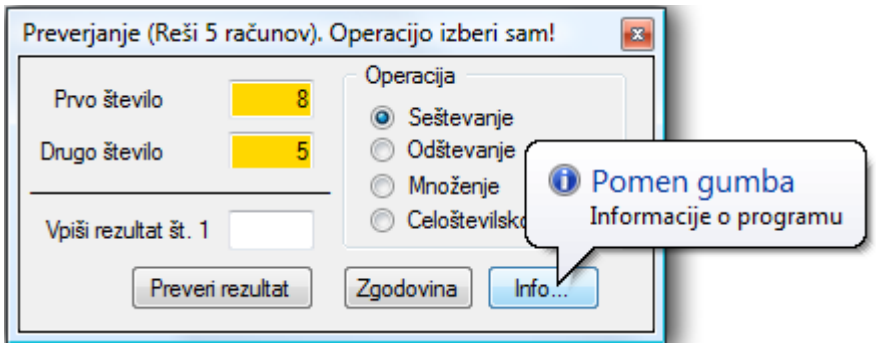
Dogodki, ki so prirejeni posameznim gradnikom so opisani v spodnji kodi obrazca:

```
public partial class FPreverjanje : Form
{
    public FPreverjanje()//konstruktor (privzeti)
    {
        InitializeComponent();
    }
    //javne spremenljivke/objekti
    public int stevilo1, stevilo2, rezultat,stevec,pravilnih;
    Random naklj = new Random();
    //zapomnimo si ime datoteke, saj ga bomo potrebovali večkrat
    public string imeDat = "Rezultati.txt";
    //dogodek Load se zgodi preden se obrazec prvič odpre
    private void FPreverjanje_Load(object sender, EventArgs e)
    {
        //števec vseh in števec pravilnih odgovorov
        stevec = 0; pravilnih = 0;
        //klic metode za generiranje dveh naključnih števil
        ZacetnaVrednost();
    }
    //zasebna metoda za generiranje dveh naključnih števil, ki ju vpišemo v
    //gradnika textBox1 in textBox2
    private void ZacetnaVrednost()
    {
        //ustvarimo dve naključni celi števili med 1 in 10
        stevilo1 = naklj.Next(1, 11);
        stevilo2 = naklj.Next(1, 11);
        //obe števili zapišimo v gradnika textBox1 in taxtBox2
        tBPrvo.Text = stevilo1.ToString();
        tBDrugo.Text = stevilo2.ToString();
        tbRezultat.Clear();//pobrišemo vsebino gradnika textBox3
        //želimo, da gradnik textBox3 postane aktiven gradnik na našem
        //obrazcu – ima fokus
        tbRezultat.Focus();
        stevec++;//povečamo števec računov
        label3.Text = "Vpiši rezultat št. " + stevec.ToString();
    }
    private void bRezultat_Click(object sender, EventArgs e)
    {
        try
        {
            /*v spremenljivko rezultat shranimo uporabnikov odgovor, ki je v
              textBox3*/
            rezultat = Convert.ToInt32(tbRezultat.Text);
            bool OK = false;
            if (rBVsota.Checked)
             /*spremenljivki OK priredimo vrednost true ali false, glede na
              pravilnost oz. nepravilnost primerjanja :
              rezultat == stevilo1 + stevilo2*/
```

```
                OK = (rezultat == stevilo1 + stevilo2);
            /*lahko bi napisali tudi  if (rezultat == stevilo1 + stevilo2)
                                        OK = true;*/
            else if (rBRazlika.Checked)
                OK = (rezultat == stevilo1 - stevilo2);
            else if (rBProdukt.Checked)
                OK = (rezultat == stevilo1 * stevilo2);
            else if (rBDeljenje.Checked)
                OK = (rezultat == stevilo1 / stevilo2);
            //preverimo, če je rezultat OK
            if (OK)
            {
                MessageBox.Show("Odgovor JE pravilen!");
                pravilnih++; //povečamo števec pravilnih odgovorov
            }
            else
                MessageBox.Show("Odgovor NI pravilen!");
            if (stevec == 5)
            {
                string ocena = "";
                switch (pravilnih) //preverimo število pravilnih odgovorov
                {
                    case 0: ocena = "Nezadostno"; break;
                    case 1: ocena = "Nezadostno"; break;
                    case 2: ocena = "Zadostno"; break;
                    case 3: ocena = "Dobro"; break;
                    case 4: ocena = "Prav dobro"; break;
                    default: ocena = "Odlično"; break;
                }
                //Izpis rezultata
                MessageBox.Show("Pravilnih odgovorov: "
                    + pravilnih.ToString()
                    + "\nNepravilnih odgovorov: "
                    + (stevec - pravilnih).ToString()
                    +"\n\nOcena: "+ocena);
                /*rezultat shranimo v tekstovno datoteko Rezultati.txt. Ob
                vsakem rezultatu naj bo zapisan tudi datum in ura reševanja*/
                //metoda AppendText odpre datoteko za dodajanje besedila. Če
                //datoteka še ne obstaja, jo ustvari, datoteka pa ostane
                //odprta za pisanje.
                StreamWriter pisi = File.AppendText(imeDat);
                //rezultat zapišemo v datoteko
                pisi.WriteLine("Datum: " + DateTime.Now.ToShortDateString() +
", Ura: " + DateTime.Now.ToShortTimeString() + ", Pravilnih odgovorov: " +
pravilnih.ToString());
                pisi.Close();//zapremo podatkovni tok
                /*števec vseh odgovorov in števec pravilnih odgovorov
                  postavimo na 0, na vrsti je novih 5 računov*/
                stevec = 0; pravilnih = 0;
            }
```

```
            //klic metode za generiranje novih dveh naključnih števil
            ZacetnaVrednost();
        }
        catch
        {
            MessageBox.Show("Napaka v podatkih!");
        }
    }
    //odzivna metoda, ki se izvede ob kliku na gradnik gBOperacija
    private void gBOperacija_Enter(object sender, EventArgs e)
    {
        tbRezultat.Focus();
    }
    /*Info je odzivna metoda dogodka Click gumbov bZgodovina in bInfo.
      Poimenovali smo jo po svoje!*/
    private void Info(object sender, EventArgs e)
    {
        if (sender == bZgodovina)/*če je bil kliknjen gumb z napisom
                                   Zgodovina*/
        {
            //ime datoteke z dosedanjimi rezultati
            //uporabimo varovalni blok za obdelavo prekinitev
            try
            {
                //celotno vsebino datoteke preberemo naenkrat
                string vsebina=File.ReadAllText(imeDat);
                //in jo izpišemo v sporočilnem oknu
                MessageBox.Show(vsebina);
            }
            catch //v primeru napake naj se izpiše le ustrezno besedilo
            { MessageBox.Show("Napaka pri obdelavi datoteke " + imeDat); }
        }
        else if (sender==bInfo)/*če je bil kliknjen gumb z napisom Info...*/
        {
            //pripravimo informacijo za uporabnika
            string info="V polje 'Vpiši rezultat št. ' vtipkaj rezultat in
klikni gumb 'Preveri rezultat'.";
            //dodajamo nove vrstice
            info+="\nPo petih računih se bo izpisalo število pravilnih
odgovorov in ocena!";
            info+="\nDatum, ura poskusa in rezultati se obenem zapisujejo v
tekstovno datoteko 'Rezultati.txt'.";
            info+="\n\nVsebino te datoteke dobiš s klikom na gumb
'Zgodovina'.";
            //informacije izpišemo v sporočilnem oknu
            MessageBox.Show(info);
        }
    }
    /*če kliknemo na katerega koli od radijskih gumbov, se vsebina polja
      tbRezultat briše*/
```

```
    private void rBVsota_CheckedChanged(object sender, EventArgs e)
    {
        tbRezultat.Clear();
    }
}
```

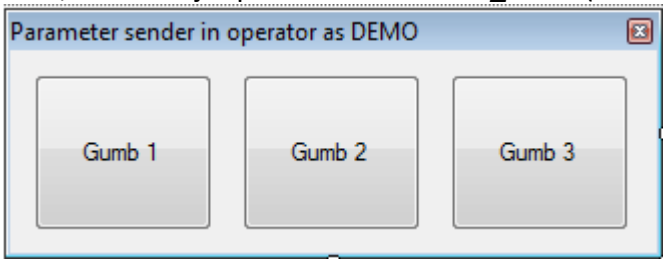


Slika 22: Če se v delujočem programu z miško postavimo na gumb *Info*, se nam prikaže oblaček z ustrezno informacijo!

Operatorja *is* in *as*

S pomočjo parametra *sender* lahko tudi preverjamo tip objekta (gradnika), dostopamo in tudi spreminjamo lastnosti gradnikom, oziroma poganjamo njihove metode. Pomagamo si z operatorjema ***is*** in ***as***.

Parameter *as* omogoča dostop do lastnosti poljubnega objekta ali pa gradnika (preko imena gradnika ali pa preko parametra *sender*). Recimo, da imamo na obrazcu tri gumbe (imena pustimo privzeta: *button1*, *button2* in *button3*). Vsi trije gumbi naj krmilijo isto odzivno metodo *Click*, ki smo jo poimenovali *Gumb_Click* (vsem trem gumbom v oknu *Properties→Events* priredimo/določimo isti odzivno metodo). Ob kliku na kateregakoli od teh treh gumbov želimo spremeniti nekaj njegovih lastnosti, kar dosežemo s pomočjo parametra *sender* in operatorja *as*.



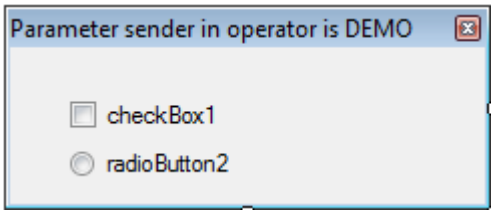
Slika 23: Parameter *sender* in operator *as*.

```
private void Gumb_Click(object sender, EventArgs e)
{    //spremenimo napis na gumbu, ki je bil kliknjen
    (sender as Button).Text = "Skrit";
    //kliknjen gumb naj postane neaktiven (ne moremo ga več "klikniti")
    (sender as Button).Enabled = false;
}
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Lastnosti izbranega gradnika smo torej spremenili brez preverjanje imena gradnika.

S pomočjo parametra *is* pa lahko preverimo tip poljubnega objekta ali pa gradnika (preko imena gradnika, ali pa preko parametra *sender*). Za primer postavimo na obrazec dva gradnika: gradnik *CheckBox* in gradnik *RadioButton* (imeni gradnikov naj bosta privzeti). Obema gradnikom priredimo isti dogodek *ChechedChanged*, ki ga poimenujemo *Sprememba*. S pomočjo parametra *sender*, ter operatorjev *is* in *as* lahko kontroliramo lastnosti obeh gradnikov v isti odzivni metodi.



Slika 24: Parameter *sender* in operator *is*.

```
private void Sprememba(object sender, EventArgs e)
{
    if (sender is CheckBox)//preverimo, če je bil kliknjen gradnik CheckBox

    {
       if ((sender as CheckBox).Checked)//če je izbran
         (sender as CheckBox).Text="Kontrolnik izbran";//napis ob kontrolniku
       else
         (sender as CheckBox).Text="Kontrolnik NI izbran";
    }
    //preverimo, če je bil kliknjen radijski gumb
    if (sender is RadioButton)
    {
        if ((sender as RadioButton).Checked) //če je izbran
            (sender as RadioButton).Text = "Gumb JE izbran";
    }
}
```

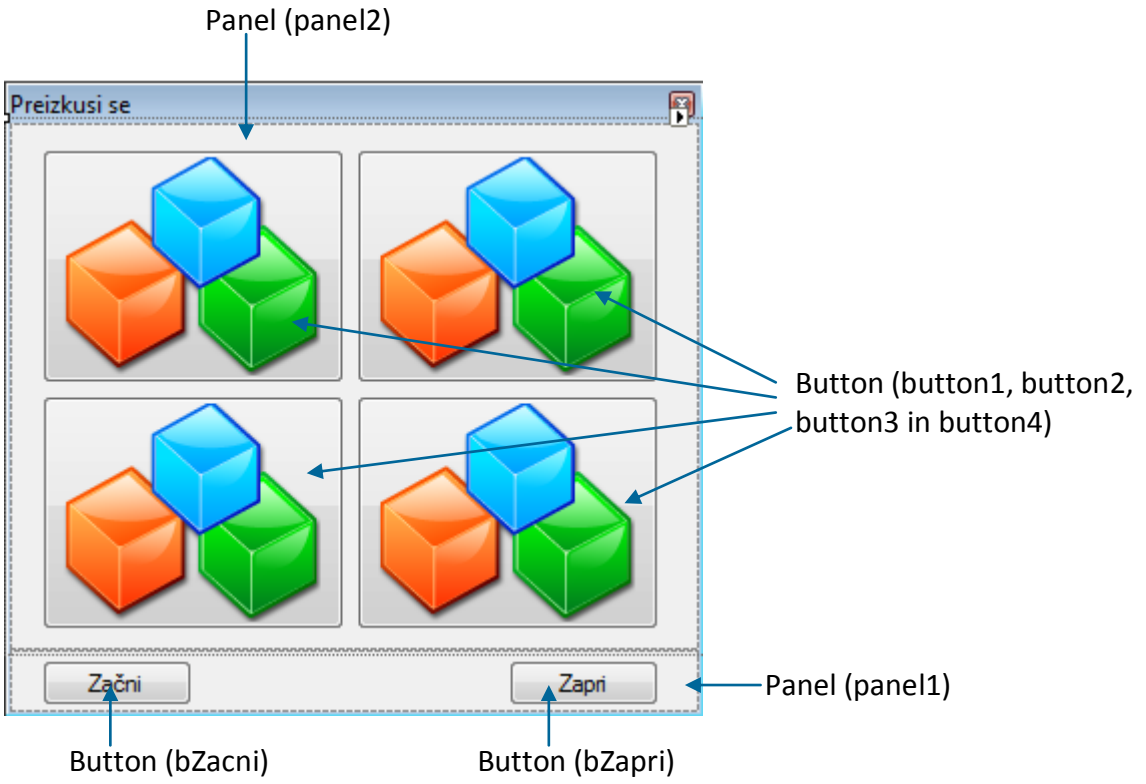
Hitrostno klikanje

Radi bi se preizkusili v hitrostnem klikanju. V ta namen ustvarimo obrazec s štirimi gumbi. Klikniti je možno le na gumb, ki je aktiven. Kateri gumb je aktiven določimo programsko (naključno). Merimo čas od začetka do takrat, ko bomo 20x zaporedoma pravilno kliknili na aktivni gumb. Če gumb zgrešimo, se torej števec zadetkov postavi nazaj na 0.

Lastnosti gradnikov na obrazcu so prikazane v naslednji tabeli:

Gradnik	Lastnost	Nastavitev	Opis
FIgra	FormBorderStyle	FixedToolWindow	Uporabnik velikosti okna ne more spreminjati.
FIgra	StartPosition	Center	Projekt se odpre na sredini zaslona.
panel1	Dock	Bottom	Gradnik je "prilepljen" na dno obrazca.
panel2	Dock	Fill	Gradnik je razširje nad celotni preostali del obrazca.
button1, button2, button3, button4	Image	Poljubna slika	

Tabela 6: Gradniki in njihove lastnosti.



Slika 25: Hitrostno klikanje.

Koda obrazca (gumbom s sliko in obem gradnikom *Panel* priredimo isto odzivno metodo *Gumb_Click*):

```
public partial class FIgra : Form
{
    public FIgra()
    {
        InitializeComponent();
    }
    int stevec;//števec klikov
    //spremenljivki tipa DateTime, ki označujeta začetek in konec klikanja
    DateTime zacetek, konec;
    Random naklj = new Random();//generator naključnih števil
    private void FIgra_Load(object sender, EventArgs e)
    {
        OnemogociGumbe();//vsi štirje gumbi so na začetku onemogočeni
    }
    private void bZacetek_Click(object sender, EventArgs e)
    {
        (sender as Button).Visible = false; //skrijemo gumb
        OnemogociGumbe();
        MessageBox.Show("Pričetek igre. Igraš tako, da klikaš izbrane slike!
                         Igra se konča po 20 izbirah!", "Začetek");
        {
            zacetek = DateTime.Now;
            stevec = 0;//začetna vrednost števca klikov
        }
        OmogociNakljucniGumb();
    }
    private void OmogociNakljucniGumb()
    {
        int izbran = naklj.Next(1,5);//naključno število med 1 in 4
        //omogočimo gumb, ki ustreza naključnemu številu
        if (izbran == 1) button1.Enabled = true;
        else if (izbran == 2) button2.Enabled = true;
        else if (izbran == 3) button3.Enabled = true;
        else if (izbran == 4) button4.Enabled = true;
    }
    private void OnemogociGumbe()
    {
        button1.Enabled = false; //vse štiri gumbe deaktiviramo
        button2.Enabled = false;
        button3.Enabled = false;
        button4.Enabled = false;
    }
    /*metodo Gumb_Click priredimo vsem štirim gumbom s sliko, poleg tega pa
      še obem gradnikom Panel*/
    private void Gumb_Click(object sender, EventArgs e)
    {
```

```
        /*če je uporabnik zgrešil sliko (sliko predstavlja aktivni gumb, vse
          ostalo pa je izven slike) in se je igra že začela*/
        if (!(sender is Button)&& bZacetek.Visible==false)
        {
            MessageBox.Show("Zgrešil si gumb, število uspešnih zadetkov se
                             izniči! ", "Ponovni začetek!");
            stevec = 0;
        }
        else if (sender is Button) //če je bil kliknjen gumb
                {
                    /*preverimo, če je bil kliknjen pravi gumb (tisti, ki ima
                      lastost enabled nastavljeno na true)*/
                    if ((sender as Button).Enabled == true)
                    {
                        //izbrani gumb onemogočimo
                        (sender as Button).Enabled = false;
                        stevec++;//povečamo števec uspešnih klikov
                        if (stevec == 20)
                        {
                            konec= DateTime.Now;//čas,potreben za 20 zadetkov
                            //razlika začetnega in končnega časa
                            TimeSpan porabljenCas = konec - zacetek;
                            MessageBox.Show("Igra je končana!\n\nPotreboval
si " + porabljenCas.TotalSeconds + " sekund");
                        }
                        else
                        {
                            OmogociNakljucniGumb();//prikaz naključnega gumba
                        }
                    }
                }
    }
    private void bZapri_Click(object sender, EventArgs e)
    {
        Close(); //zapremo obrazec
    }
}
```

V zgornjem projektu smo prikazali enostaven primer uporabe operatorjev *is* in *as*. Koda je zato krajša in bolj razumljiva. Obenem smo ponovili še osnovni namen razredov *DateTime* in *TimeSpan*.

Kontrola uporabnikovih vnosov – validacija

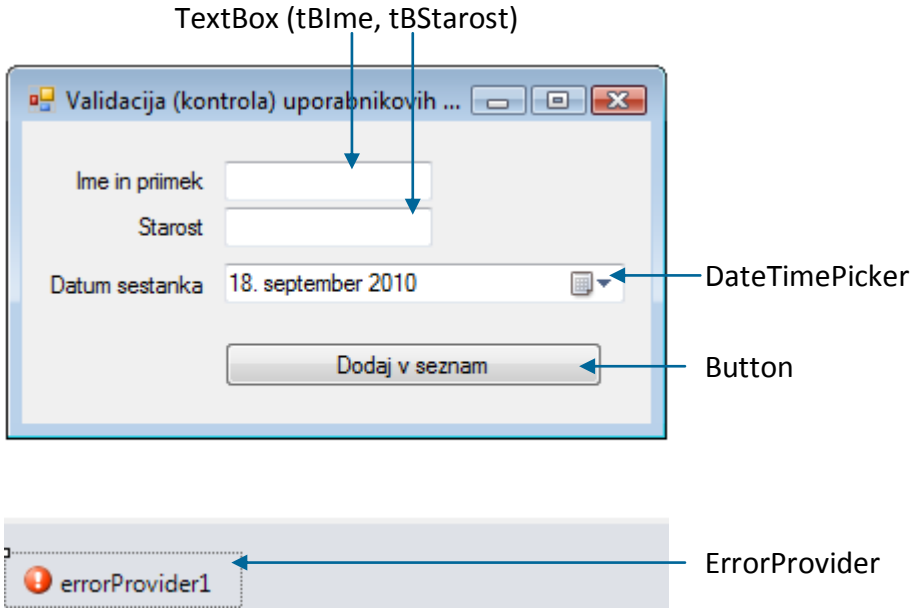
Pred uporabo ali pa pred shranjevanjem podatkov, ki jih uporabnik vnese v gradnike na obrazcu, je potrebno preveriti pravilnost (smiselnost) le-teh. Če podatki niso smiselni, je potrebno na to uporabnika opozoriti in mu pri velikem številu vnosov tudi vizuelno pokazati na gradnik, kjer je vnos nepravilen.

Gradniki, namenjeni uporabnikovim vnosom ali izbiram (npr. *TextBox*, *DateTimePicker*, *RadioButton*, *CheckBox*, *ComboBox*, …) imajo lastnost *CausesValidation*, ki je privzeto nastavljena na *True*. Ta lastnost pomeni, da vsak uporanikov premik iz tega gradnika na nekega drugega sproži dogodek *Validating*. Če to lastnost nastavimo na *False*, se dogodek ne bo izvedel in tako preverjanja ne bo. V ogrodje dogodka *Validating* zapišemo kodo, ali pa klic metode, ki preverja pravilnost vnosa. Dogodek *Validating* lahko zapišemo vsakemu gradniku posebej, na koncu pa še obrazcu, ali pa le obrazcu (odvisno od namena obrazca).

Obvestilo, kakšen vnos je potreben, pa lahko zapišemo tudi v polje *error on errorPrivider* (a le, če smo na obrazec postavili gradnik *ErrorProvider*): v tem primeru se bo že po prikazu obrazca ob gradniku prikazala ikona (privzeto klicaj na rdečem, okroglem ozadju). Če se z miško postavimo na to ikono, bomo v okvirčku dobili obvestilo, ki smo ga zapisali v polje *error on errorProvider*.

Napoved sestanka

Ustvarimo obrazec, ki bo od uporabnika zahteval vnos imena in priimek, njegovo starost in datum nekega sestanka. Vsi trije podatki so obvezni, poleg tega mora biti uporabnik polnoleten, izbrani datum pa ne sme biti na soboto ali nedeljo. Kontrolo vnosov bomo realizirali s pomočjo metod gradnika *ErrorProvider*, ki ga vežemo na obrazec. Besedilo o napaki zapišemo v metodo *SetError* tega gradnika, ki ima dva parametra: prvi parameter je ime gradnika, za katerega pišemo odzivno metodo, drugi pa sporočilo o napaki!.



Slika 26: Kontrola uporabnikovih vnosov.

Gradniki in njihove lastnosti

Gradnik	Lastnost	Nastavitev	Opis
FValidacija	FormBorderStyle	FixedToolWindow	Uporabnik velikosti okna ne more spreminjati
tBStarost	TextAlign	Right	Vnesena starost bo desno poravnana.
errorProvider1	ContainerControl	FValidacija	Nadrejeni gradnik (običajno, pa tudi v našem primeru, je to obrazec), na katerem bo prikazana ikona z opozorilom, da je vnos napačen.

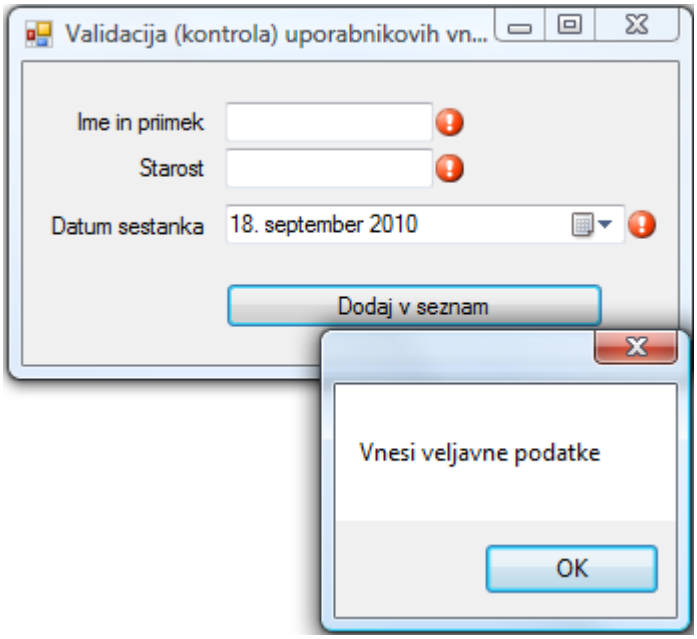
Tabela 7: Gradniki na obrazcu *FValidacija*.

```
public partial class FValidacija : Form
{
    public FValidacija()
    {
        InitializeComponent();
    }
    //metoda vrne true, če je ime vneseno
    private bool KontrolaImena()
    {
        bool bStatus = true;
        if (tBIme.Text == "")
        {
            /*Če uporabnik ne bo vnesel imena, se bo ob imenu pokazala rdeča
oznaka s klicajem. Če se bomo z miško postavili na to oznako, se pod njim v
okvirčku pojavi besedilo, ki ga zapišemo kot drug parameter metode SetError*/
            errorProvider1.SetError(tBIme, "Vnesi svoje ime");
            bStatus = false;
        }
        else
            /*vnos je pravilen, kar povemo s tem, da je besedilo
            napake prazen niz*/
            errorProvider1.SetError(tBIme, "");
        return bStatus;
    }

    //metoda vrne true, če je vnesena starost OK
    private bool KontrolaStarosti()
    {
        bool bStatus = true;
        if (tbStarost.Text == "")
```

```
        {
            errorProvider1.SetError(tbStarost, "Vnesi starost");
            bStatus = false;
        }
        else
        {
            errorProvider1.SetError(tbStarost, "");
            try
            {
                int temp = int.Parse(tbStarost.Text);
                errorProvider1.SetError(tbStarost, "");
                if (temp < 18)
                {
                    errorProvider1.SetError(tbStarost, "Za udeležbo na
sestanku moraš biti star najmanj 18 let");
                    bStatus = false;
                }
                else
                {
                    errorProvider1.SetError(tbStarost, "");
                }
            }
            catch
            {
                errorProvider1.SetError(tbStarost, "Starost mora biti
številka");
                bStatus = false;
            }
        }
        return bStatus;
    }

    //metoda vrne true, če je izbrani datum pravilen
    private bool KontrolaDatuma()
    {
        bool bStatus = true;
        //Če je izbrani datum sobota ali nedelja, opozorimo uporabnika
        if ((dateTimePicker1.Value.DayOfWeek ==DayOfWeek.Sunday) ||
(dateTimePicker1.Value.DayOfWeek ==DayOfWeek.Saturday))
            {
            errorProvider1.SetError (dateTimePicker1, "Sestanek ne more biti
organiziran med vikendom. Izberi prosim dan med tednom!");
            bStatus = false;
            }
        else
            errorProvider1.SetError (dateTimePicker1, "");
        return bStatus;
    }

    //odzivna metoda Validating gradnika textBox1
```

```
    private void textBox1_Validating(object sender, CancelEventArgs e)
    {
        KontrolaImena();//klic metode za kontrolo vnosa imena
    }

    //odzivna metoda Validating gradnika textBox2
    private void textBox2_Validating(object sender, CancelEventArgs e)
    {
        KontrolaStarosti();//klic metode za kontrolo starosti
    }

    //odzivna metoda Validating gradnika dateTimePicker1
    private void dateTimePicker1_Validating(object sender, CancelEventArgs e)
    {
        KontrolaDatuma();//klic metode za kontrolo datuma
    }

    //metoda za kontrolo vseh gradnikov naenkrat
    private void ValidateForm()
    {
        bool bValidName = KontrolaImena();
        bool bValidAge = KontrolaStarosti();
        bool bValidTestDate = KontrolaDatuma();
        if (bValidName && bValidAge && bValidTestDate)
            MessageBox.Show("Prijava na sestanek uspešna!");
            /*Na to mesto bi v resnici zapisali kodo, ki bi podatke o
              sestanku zapisala npr. v datoteko*/
        else
            MessageBox.Show("Vnesi veljavne podatke");
    }

    /*Še kontrola ob kliku na gumb Dodaj v seznam. Sestanek bo organiziran le
      v primeru, da so vsi podatki na obrazcu vneseni pravilno*/
    private void Shrani_Click(object sender, EventArgs e)
    {
        ValidateForm();//klic metode za kontrolo vseh gradnikov na obrazcu
    }
    private void Form1_Activated(object sender, EventArgs e)
    {
        tbStarost.Text = tBIme.Text;
    }
}
```

V projektu smo prikazali le "masko" za vnos podatkov o nekem sestanku, dejansko dodajanje v seznam sestankov pa bi morali še napisati, npr. v nekem nadrejenem obrazcu, to je obrazcu iz katerega smo odprli obrazec *FValidacija*.



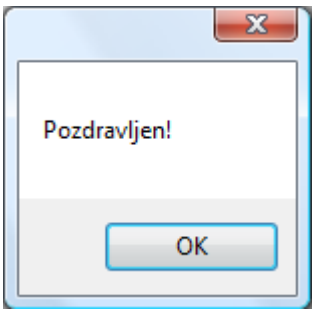
Slika 27: Če so vnosi napačni (oz. vnosa ni), se ob gradniku pojavi ikona z opozorilom.

Sporočilno okno MessageBox

Sporočilno okno *MessageBox* je namenjeno posredovanju sporočil. Obenem je to tudi pogovorno okno namenjeno uporabnikovim odločitvam. Okno lahko prikaže sporočila v številnih variantah. V dosedanjih primerih smo ga uporabljali le v njegovi najenostavnejši obliki, z enim samim parametrom. Prikažemo ga s pomočjo metode *Show*. Obstaja kar 21 različnih načinov (preobtežitev) uporabe te metode. Ogledali pa si bomo le najpomembnejše.

- *MessageBox.Show (string)* - Prikaz osnovnega sporočila uporabniku znotraj sporočilnega okna:

```
MessageBox.Show("Pozdravljen!");
```

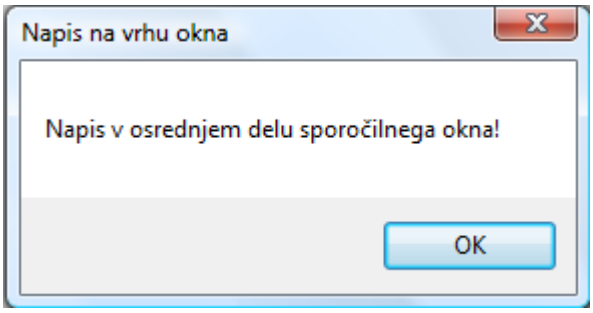


Slika 28: Osnovna uporaba sporočilnega okna *MessageBox*.

V oknu se pokaže obvestilo, ki smo zapisali v oklepaju metode *Show*, pod besedilom pa je gumb z napisom *OK*. Ob kliku na ta gumb se sporočilno okno zapre.

- *MessageBox.Show (string, string)* - Prikaz poljubnega teksta v sporočilnem oknu in še poljubnega teksta v naslovni vrstici okna.

```
MessageBox.Show("Napis v osrednjem delu sporočilnega okna!", "Napis na vrhu okna");
```



Slika 29: Sporočilno okno z dvema parametroma tipa string.

- *MessageBox.Show (string, string, MessageBoxButtons)* - Sporočilno okno s tekstom v oknu, napisom na oknu in odločitvenimi gumbi.

Prva dva parametra sta dva niza (tip *string*). Prvi parameter predstavlja napis v oknu, drugi pa napis na oknu. Tretji parameter je tipa *MessageBoxButtons* in določa število in vrsto gumbov, ki bodo prikazani v oknu. To je v bistvu naštevni tip s konstantami, ki določajo kateri gumbi se bodo prikazali v sporočilnem oknu. Napisi na gumbih so odvisni od jezikovne variante operacijskega sistema. Vse možne konstante in njihova razlaga, so zbrane v naslednji tabeli:

MessageBoxButtons	Razlaga (SLO verzija operacijskega sistema in ANJ verzija operacijskega sistema)
AbortRetryIgnore	Sporočilno okno vsebuje gumbe *Prekini*, *Poskusi znova*, in *Prezri* (*Abort*, *Retry*, *Ignore*).
OK	Sporočilno okno vsebuje gumb *V redu* (OK).
OKCancel	Sporočilno okno vsebuje gumba *V redu* in *Prekliči* (*OK*, *Cancel*).
RetryCancel	Sporočilno okno vsebuje gumba *Poskusi znova*, in *Prekliči* (Retry, *Cancel*).
YesNo	Sporočilno okno vsebuje gumba *Da* in *Ne* (*Yes*, *No*).
YesNoCancel	Sporočilno okno vsebuje gumbe *Da*, *Ne* in *Prekliči* (*Yes*, *No*,

	Cancel)

Tabela 8: Vrednosti naštevnega tipa *MessageBoxButtons.*

MessageBox je razred, metoda *Show* pa statična metoda tega razreda, ki je naštevnega tipa (*enum*) *DialogResult*. Vrednost, ki jo metoda *Show* vrne je torej tipa *DialogResult*, odvisna pa je od uporabnikovega klika na določen gumb. Od tega pa je seveda odvisno nadaljevanje aplikacije (za kar moramo poskrbeti z ustrezno kodo). Vrednosti tipa *DialogResult* so zbrane v naslednji tabeli:

DialogResult	Razlaga
Abort	Prekini (Abort).
Cancel	Prekliči (Cancel).
Ignore	Prezri (Ignore).
No	Ne (No).
OK	V redu (OK).
Retry	Poskusi znova (Retry).
Yes	Da (Yes).

Tabela 9: Vrednosti tipa *DialogResult*.

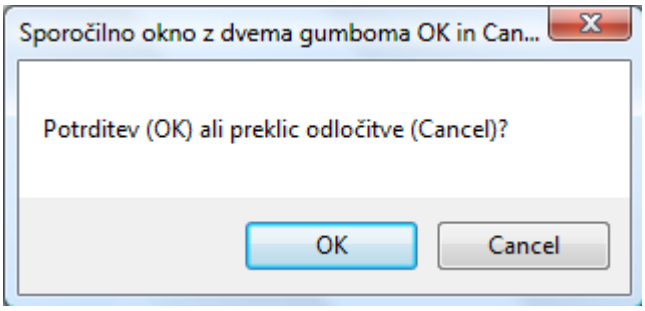
Kadar sporočilno okno vsebuje več kot en gumb, moramo seveda predvideti klik na kateregakoli od teh gumbov in ustrezno reagirati v aplikaciji. Tule je primer uporabe parametra *MessageBoxButtons* z gumboma *OK* in *Cancel*:

```
/*ker metoda Show vrne vrednost tipa DialogResult jo lahko uporabimo
takole*/

DialogResult odlocitev = MessageBox.Show("Potrditev (OK) ali preklic
odločitve (Cancel)?", "Sporočilno okno z dvema gumboma OK in Cancel!",
MessageBoxButtons.OKCancel);

if (odlocitev == DialogResult.OK)
   //stavki, ki se izvedejo, če je uporabnik kliknil gumb OK
else
   //stavki, ki se izvedejo, če je uporabnik kliknil gumb Cancel
```

ali krajše (in lepše) takole:

```
if (MessageBox.Show("Potrditev (OK) ali preklic odločitve (Cancel)?",
"Sporočilno okno z dvema gumboma OK in Cancel!",
MessageBoxButtons.OKCancel) == DialogResult.OK)
    //stavki, ki se izvedejo, če je uporabnik kliknil gumb OK
```

```
else
    //stavki, ki se izvedejo, če je uporabnik kliknil gumb Cancel
```



Slika 30: Uporaba parametra *MessageBoxButtons* v sporočilnem oknu.

- *MessageBox.Show (string, string, MessageBoxButtons, MessageBoxIcon)* - Sporočilno okno s tekstom v oknu, napisom na oknu, odločitvenimi gumbi in *ikono* v oknu.

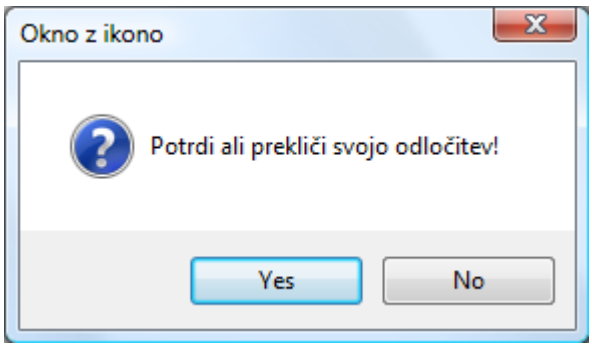
V sporočilnem oknu lahko prikažemo tudi eno od vnaprej pripravljenih ikon (vprašaj, klicaj, napaka, opozorilo). Prvi trije parametri metode so enaki kot v prejšnjem primeru, četrti parameter pa je tipa *MessageBoxIcon*. To je naštevni tip s konstantami, ki določajo katera ikona se bo prikazala v sporočilnem oknu. Vse možne konstante, skupaj z razlago so zbrane v naslednji tabeli:

Oznaka ikone	Razlaga
Asterisk	Sporočilno okno vsebuje grafični simbol z malo črko i v sredini.
Error	Sporočilno okno vsebuje grafični simbol z znakom X na rdeči podlagi.
Exclamation	Sporočilno okno vsebuje grafični simbol z znakom ! (klicaj) na rumeni podlagi.
Hand	Sporočilno okno vsebuje grafični simbol z znakom X na rdeči podlagi.
Information	Sporočilno okno vsebuje grafični simbol z malo črko i.
None	Sporočilno okno ne vsebuje grafičnega simbola.
Question	Sporočilno okno vsebuje grafični simbol z znakom ? (vprašaj) na modri podlagi.
Stop	Sporočilno okno vsebuje grafični simbol z znakom X na rdeči podlagi.
Warning	Sporočilno okno vsebuje grafični simbol z znakom ! (klicaj) na rumeni podlagi.

Tabela 10: Vrednosti naštevnega tipa *MessageBoxIcon*.

```
//klic sporočilnega okna z gumboma Yes, NO in ikono z vprašajem
if (MessageBox.Show("Potrdi ali prekliči svojo odločitev!", "Okno z
ikono", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.Yes)
```

```
    //stavki, ki se izvedejo, če uporabnik klikne gumb Yes
else
    //stavki, ki se izvedejo, če uporabnik klikne gumb No
```



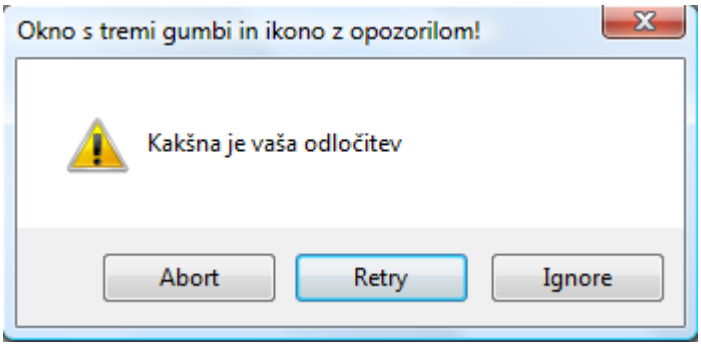
Slika 31: Sporočilno okno z ikono.

- *MessageBox.Show(string, string, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton)*

Kadar je v sporočilnem oknu več kot en gumb, lahko vnaprej določimo privzeti gumb, to je gumb, ki je vnaprej označen (aktiven). Peti parameter (aktivni gumb v oknu) je v v tem primeru parameter tipa *MessageBoxDefaultButton*, ki predstavlja oznako enega izmed treh možnih gumbov v sporočilnem oknu. Gumb, naveden kot peti parameter, bo izbrani gumb (bo torej aktiven in ima fokus), ko se pogovorno okno prikaže.

Če torej želimo na potek nadaljevanja programa vplivati preko takega sporočilnega okna, bi zapisali

```
//Sporočilno okno s tremi gumbi, izbrani (aktivni) gumb je gumb Retry
switch (MessageBox.Show("Kakšna je vaša odločitev", "Okno s tremi gumbi
in ikono z opozorilom!", MessageBoxButtons.AbortRetryIgnore,
MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2))
{
    case DialogResult.Abort: /*stavki, ki se izvedejo, če uporabnik
                               klikne gumb Abort*/
        break;
    case DialogResult.Retry: /*stavki, ki se izvedejo, če uporabnik
                               klikne gumb Retry*/
        break;
    case DialogResult.Ignore: /*stavki, ki se izvedejo, če uporabnik
                                klikne gumb Ignore*/
        break;
}
```



Slika 32: Sporočilno okno s tremi gumbi, ikono in izbranim aktivnim gumbom.

- *MessageBox.Show (string, string, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions)*

Sporočilno okno s tekstom v oknu, napisom na oknu, odločitvenimi gumbi, vrsto ikone, privzetim gumbom in *opcijami*. Šesti parameter (možnosti v oknu) je parameter tipa *MessageBoxOptions*. To je oznaka ene izmed štirih posebnih možnosti, s katerimi povemo, kako naj se prikaže sporočilo v sporočilnem oknu. Seznam vseh možnosti, ki jih lahko zapišemo kot šesti parameter je v naslednji tabeli:

Oznaka Parametra	Razlaga
DefaultDesktopOnly	Sporočilno okno je prikazano na aktivnem namizju.
RightAlign	Tekst sporočila je desno poravnan.
RtlReading	Nastavitev določa, da je vsebina sporočilnega okna prikazana od desne proti levi (besedilo je na levi, ikona pa na desni.
ServiceNotification	Sporočilno okno je prikazano na aktivnem namizju.

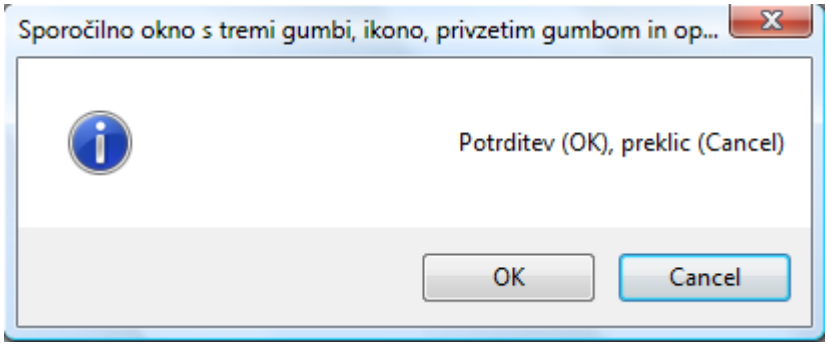
Tabela 11: Tabela možnih nastavitev v oknu *MessageBox*

S kodo

```
if (MessageBox.Show("Potrditev (OK), preklic (Cancel)", "Sporočilno
okno s tremi gumbi, ikono, privzetim gumbom in opcijo",
MessageBoxButtons.OKCancel,MessageBoxIcon.Information,MessageBoxDefault
Button.Button2,MessageBoxOptions.RightAlign)==DialogResult.Cancel)
{  //stavki, ki se izvedejo, če je uporabnik kliknil gumb OK}
else
{  //stavki, ki se izvedejo, če je uporabnik kliknil gumb Cancel}
```

torej dosežemo, da se okno pojavi v taki obliki:

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.



Slika 33: Sporočilno okno z desno poravnavo teksta.

- *MessageBox.Show (string, string, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions,string FileHelpPath)*

Sporočilno okno s tekstom v oknu, napisom na oknu, odločitvenimi gumbi, vrsto ikone, privzetim gumbom, opcijami in dodatnim gumbom za pomoč oz. ime datoteke s pomočjo.

Če namesto sedmega parametra zapišemo le besedico *true*, bo v sporočilnem oknu prikazan gumb z napisom *Help* (oz. Pomoč). Ob kliku na ta gumb se bo izvedel dogodek *HelpRequested*, ki pa ga moramo seveda prej pripraviti. Za vajo v ta namen pripravimo dogodek obrazca *Form1_HelpRequested*, v katerega zapišimo le klic enostavnega sporočilnega okna.

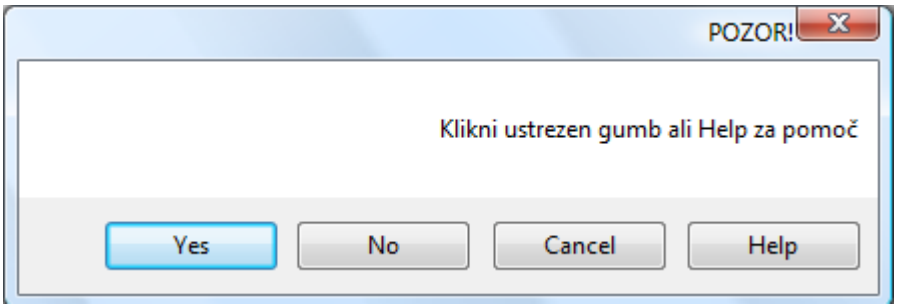
```
private void Form1_HelpRequested(object sender, HelpEventArgs hlpevent)
{
    MessageBox.Show("Pomoč...");
}
```

Če se v metodi *Show* sporočilnega *MessageBox* ne želimo navesti katerega od parametrov (razen prvih dveh, ki sta tipa niz), namesto njih zapišemo vrednost 0. Takrat tega parametra ne upoštevamo oz. upoštevamo privzete vrednosti

Klic sporočilnega okna lahko sedaj zapišemo takole:

```
/*v metodi testiramo, kateri gumb za zapiranje sporočilnega okna je
  kliknil uporabnik*/
private void button1_Click(object sender, EventArgs e)
{
    switch (MessageBox.Show("Klikni ustrezen gumb ali Help za pomoč",
"POZOR!", MessageBoxButtons.YesNoCancel, 0, 0,
MessageBoxOptions.RightAlign, true))
    {
        case DialogResult.Yes: /*stavki, ki se izvedejo, če uporabnik
                                 klikne gumb Yes*/
```

```
            break;
        case DialogResult.No: /*klik na gumb No*/
            break;
        case DialogResult.Cancel: /*klik na gumb Prekliči*/
            break;
    }
}
```



Slika 34: Prikaz gumba *Help* (Pomoč).

Kot sedmi parameter običajno napišemo ime neke datoteke s pomočjo. V sporočilnem oknu bo v tem primeru prikazan dodaten gumb z napisom *Help (*oz. *Pomoč*, če imamo slovenski operacijski sistem*)*. Ob kliku na ta gumb se prikaže okno z vsebino datoteke tipa *.chm*, katere ime smo zapisali kot sedmi parameter.

```
//POZOR: datoteka Help.chm mora obstajati
switch (MessageBox.Show("Klikni ustrezen gumb ali Help za pomoč",
"POZOR!", MessageBoxButtons.YesNoCancel,0,0,0,
@"c:\Ikone\Ikone\standard-software-icons\Help.chm"))
{
    case DialogResult.Yes: /*stavki, ki se izvedejo, če uporabnik
                             klikne gumb Yes*/
                           break;
    case DialogResult.No: /*klik na gumb No*/
                          break;
    case DialogResult.Cancel: /*klik na gumb Cancel*/
                              break;
}
```

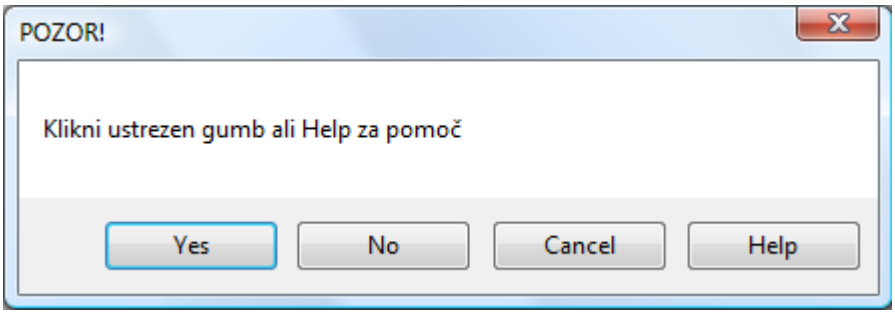
Pa še eno pomembno opozorilo, čeprav o programskem dodajanju in odvzemanju odzivov na dogodke v tej literaturi še ni bilo govora: v kolikor smo na obrazcu, v okviru katerega izvajamo to sporočilno okno, že ustvarili odzivni dogodek *HelpRequested* (npr *Form1_HelpRequested*), moramo pred prikazom sporočilnega okna odziv na ta dogodek začasno odstraniti s stavkom

```
this.HelpRequested -= new
System.Windows.Forms.HelpEventHandler(this.Form1_HelpRequested);
```

Če tega ne storimo, sta uporabnikovem kliku na gumb *Help* prirejena dva enaka dogodka. Posledica tega je hkratno odpiranje sporočilnega okna in še okna z vsebino datoteke s pomočjo.

Po zaprtju sporočilnega okna ne smemo pozabiti dogodka *HelpRequested* vključiti nazaj s stavkom

```
this.HelpRequested += new
System.Windows.Forms.HelpEventHandler(this.Form1_HelpRequested);
```



Slika 35: Prikaz gumba in datoteke s pomočjo.

- *MessageBox.Show (string , string, MessageBoxButtons , MessageBoxIcon , MessageBoxDefaultButton, MessageBoxOptions, string, HelpNavigator)*

  Sporočilno okno s tekstom v oknu, napisom na oknu, odločitvenimi gumbi, vrsto ikone, privzetim gumbom, opcijami, nizom s katerim podamo ime datoteke s pomočjo in parametrom s katerim povemo kateri element datoteke s pomočjo bo prikazan.

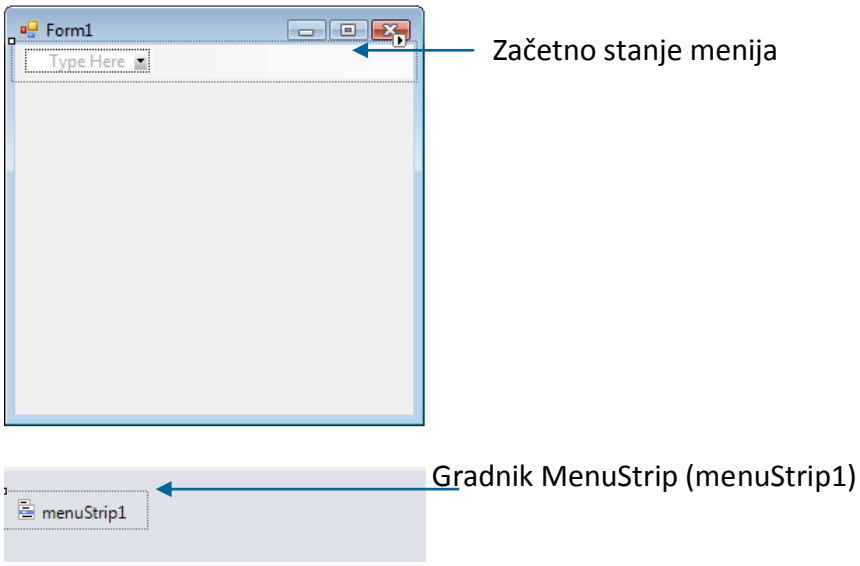
Opisanih in na primerih razloženih je bilo le nekaj najpogosteje uporabljenih oblik metode *Show* razreda *MessageBox*.

Delo z enostavnimi in sestavljenimi meniji, orodjarna

Meniji so standardni elementi okenskih programov. V menijih zapisane vrstice predstavljajo ukaze in gume, ki jih lahko aktiviramo z miško ali pa s tipkovnico. Poznamo dve vrsti menijev: glavnega, ki ga navadno postavimo pod gornji rob obrazca in lebdečega, ki ga odpre klik desnega gumba miške na določenem predmetu. Gradnika, namenjena oblikovanju menijev, sta *MenuStrip* – glavni meni in *ContextMenuStrip* – lebdeči meni.

Glavni meni – MenuStrip (nadgradnja)

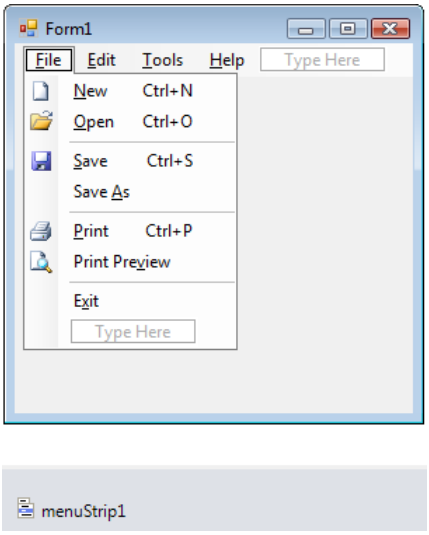
Gradnik *MenuStrip*, ki ga potrebujemo za ustvarjanje glavnega menija, se nahaja v oknu *Toolbox* v skupini gradnikov *Menus & Toolbars*. To je nevizuelni gradnik: ko ga postavimo na obrazec se namesti v polje pod obrazcem, na vrhu obrazca pa se pokaže začetno stanje menija.



Slika 36: Ustvarjanje glavnega menija – na obrazec postavimo gradnik *MenuStrip*.

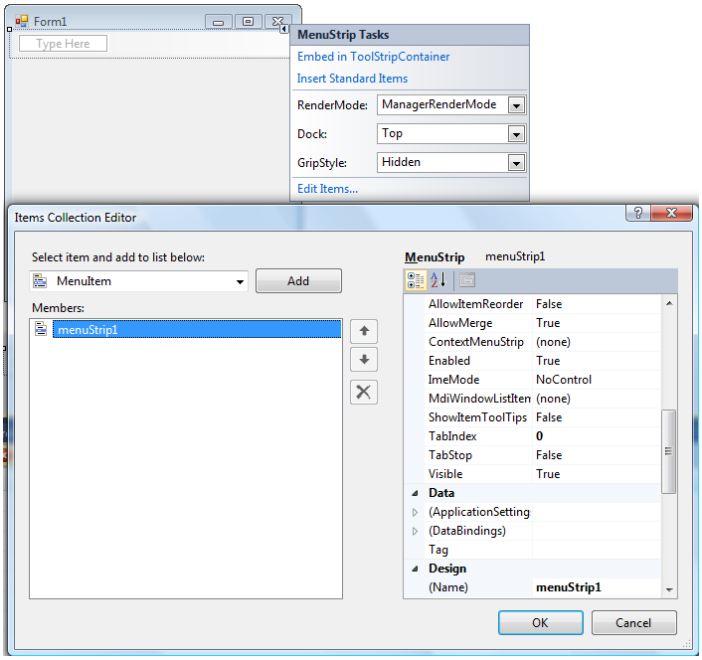
Z oblikovanjem menija pričnemo na več načinov:

- kliknemo v okno menija z napisom *Type Here* in zapišemo tekst, ki ga želimo imeti v meniju. Tekst menija nato po želji dodajamo v horizontalni ali pa v vertikalni smeri. Urejevalnik menija je videti kot pravi meni, le da ima nakazana prazna mesta, kamor je mogoče pisati nove postavke. Več o menijih in o sestavljenih menijih bo prikazano v posebnem poglavju;
- kliknemo na puščico v zgornjem desnem robu začetnega menija: odpre se pogovorno okno *MenuStrip Tasks*, kjer pa imamo zopet na voljo več možnosti. Najpomembnejše so
  - *Embed in ToolStrip Container*: ob izbiri se bo v hipu zgradil celotni meni z najpogostejšimi opcijami, ki se v menijih uporabljajo, a meni bo zgrajen znotraj gradnika *ToolStripContainer*. Takega načina ustvarjanja menija zaenkrat ne bomo uporabljali;
  - *InsertStandard Items*: v tem primeru se bo v hipu zgradil celotni meni z najpogostejšimi opcijami, ki se v menijih uporabljajo, obenem pa bodo v meniju tudi sličice, separatorji …) ;



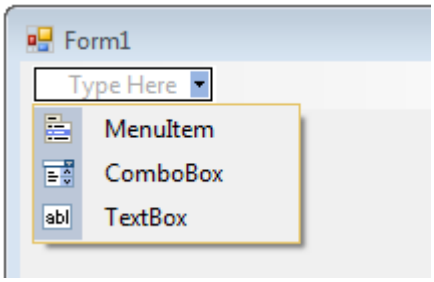
Slika 37: Glavni meni s standardnimi opcijami.

- *Edit Items*: posamezne postavke menija, tako v vertikalni, kot v horizontalni smeri, bomo oblikovali sami s pomočjo *okna Items Collection Editor*;



Slika 38: Ročno oblikovanje menija s pomočjo urejevalnika menija.

Preden začnemo pisati besedilo nove postavke menija, lahko na desni strani postavke odpremo spustni seznam in izberemo kakšen tip postavke želimo (*MenuItem* - običajna vrstica menija, *ComboBox* - spustni seznam, ali pa *TextBox* - okno z besedilom).

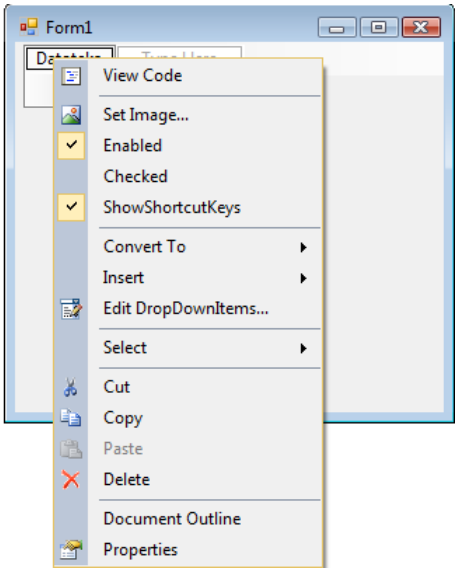


Slika 39: Pri ustvarjanju menija lahko izbiramo med tremi možnostmi.

Vsaka postavka v meniju ima svojo kartico lastnosti, ki so prikazane v oknu *Properties*. Posamezno postavko menija najpogosteje predstavlja *MenuItem*. Najbolj značilna lastnost menijske postavke je njen napis (lastnost *Text*), iz katere *Visual C#* tudi izpelje ime postavke (lastnost *Name*). Če smo npr. za neko postavko napisali ime *Konec*, je njeno programsko ime *KonecToolStripMenuItem*.

Če pred ime postavke menija, ali pa pred katerikoli znak v menijski postavki, zapišemo znak &, bo ta znak v meniju podčrtan. To pomeni, da lahko do te postavke dostopamo tudi s kombinacijo tipk *Alt+Znak*.

Z desnim klikom nad posamezno postavko menija na obrazcu se prikaže *PopUp* meni s številnimi opcijami.



Slika 40: *Pop Up* meni za oblikovanje menija.

Pomen posameznih postavk:

- *View Code*: preklop v urejevalniško okno s kodo;
- *Set Image:* izbira slike (ikone), ki naj se prikaže ob postavki menija;
- *Enabled*: postavka menija omogočena ali onemogočena: če je lastnost onemogočena, se dogodek, ki je tej postavki prirejen, ne bo izvedel;
- *Chechedd*: če jo označimo se na levi strani prikaže kljukica: postavka menija se lahko sedaj odziva različno glede na to, ali je postavka odkljukana ali ne;
- *ShowShortcutKeys*: vklop/izklop prikaza kombinacije tipk za dostop do te postavke menija preko tipkovnice (v tem primeru moramo v oknu *Properties* gradniku določiti lastnost *ShortcutKeys* in izbrati ustrezno kombinacijo tipk);
- *Convert To*: pretvorba postavke menija v neko drugo obliko (vsaka postavka v meniju je lahko prikazana na štiri načine: kot *MenuItem*, *ComboBox*, *Separator* ali kot *TextBox*);
- *Insert*: vrivanje nove postavke menija;
- *Edit DropDownItems…*: urejanje postavk v vertikalni smeri (podmenija lahko ustvarimo tudi s kombinacijo tipk *Ctrl +* →);
- *Select*: izbira te opcije nam ponudi seznam gradnikov, v katerem izberemo, kateri od njih naj bo izbran;
- *Delete*: brisanje že narejene postavke;
- *Cut*: izreži izbrano opcijo menija;
- *Copy*: kopiranje izbrane opcije menija;
- *Delete*: brisanje izbrane opcije menija;
- *Document Outline*: prikaz drevesne strukture celotnega menija v novem oknu, ki se običajno prikaže pod oknom *Toolbox*;
- *Properties*: prikaz lastnosti izbrane opcije menija.

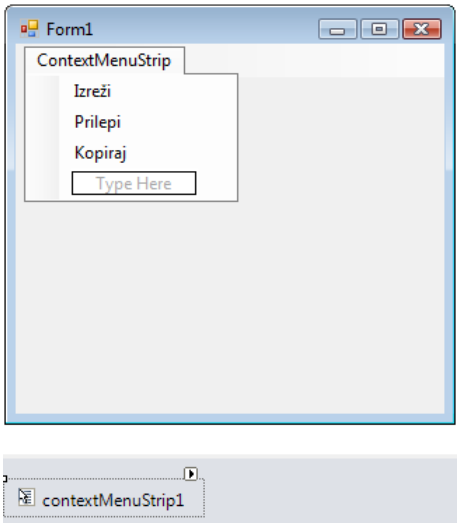
Ločilno črto (separator) v meniju lahko naredimo tudi tako, da namesto vnosa imena neke postavke zapišemo le znak pomišljaj ''-''.

Vsaki postavki v meniju priredimo odzivno metodo tako, da jo najprej izberemo, nato v oknu *Properties* kliknemo hitri gumb za prikaz dogodkov (*Events*) in končno izberemo ustrezni dogodek. Običajno bo to dogodek *Click* ali pa *DoubleClick*. Dvokliknemo v prazno polje ustreznega dogodka, da nam C# pripravi ogrodje metode, nato pa končno zapišemo ustrezne stavke odzivne metode. Ime metode bo v tem primeru privzeto (npr. *toolStripMenuItem1_Click*). Dogodku pa seveda lahko priredimo tudi poljubno ime (bodisi tako, da pred dvoklikom v oknu *Propertires→Events* zapišemo ime dogodka, ali pa v urejevalniškem oknu čez staro ime zapišemo novo ime in spremembo potrdimo v celotnem projektu).

Lebdeči (Pop Up) meni - ContextMenuStrip

Lebdeči meni se v delujoči aplikaciji pokaže, ko na nek gradnik kliknemo z desnim gumbom miške. Priredimo ga lahko kateremukoli gradniku, ki smo ga že postavili na obrazec. To storimo tako, da ime lebdečega menija izberemo kot lastnost *ContextMenuStrip* izbranega gradnika

(npr. gumba, vnosnega polja …). Seveda pa moramo prej lebdeči meni pripraviti. Izdelava je podobna izdelavi glavnega menija. Razlika je le tem, da uporabimo gradnik *ContextMenuStrip*, ki se tako kot gradnik *MenuStrip* nahaja v oknu *Toolbox* v skupini gradnikov *Menus & Toolbars*. Tudi ta gradnik je nevizuelni gradnik. Ko ga postavimo na obrazec, se namesti v polje pod obrazcem, na vrhu obrazca pa se pokaže začetno stanje menija.



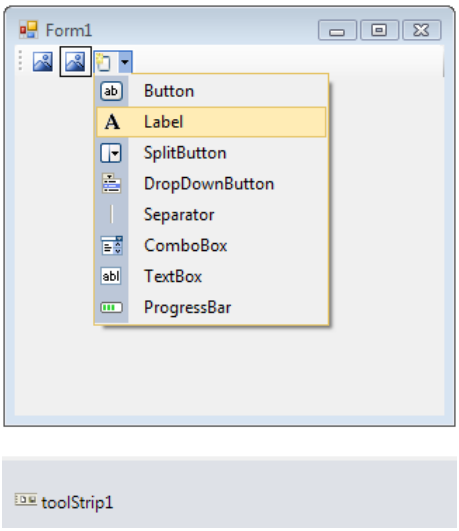
Slika 41: Ustvarjanje lebdečega menija.

Lebdeči meni urejamo tako kot glavni meni: v prazno okno vpišemo besedilo postavke, nato pa se premaknemo navzdol, če želimo imeti naslednjo opcijo v navpični smeri, ali pa desno, če želimo v tej vrstici začeti nov podmeni. Vsaki postavki v lebdečem meniju priredimo odzivni dogodek tako, da jo najprej izberemo, nato v oknu *Properties* kliknemo hitri gumb za prikaz dogodkov (*Events*) in končno izberemo ustrezni dogodek. Običajno bo to dogodek *Click* ali pa *DoubleClick*. Dvokliknemo v prazno polje ustreznega dogodka, da nam *C#* pripravi ogrodje metode, nato pa končno zapišemo ustrezne stavke odzivne metode.

Orodjarna - ToolStrip

Orodjarna ali orodna vrstica (letvica) je vsebnik – gradnik, ki vsebuje druge gradnike. V oknu *Toolbox* je to gradnik *ToolStrip*, nahaja pa se v skupini gradnikov *Menus & Toolbars*. Na letvico lahko postavljamo gradnike tipa *Button*, *Label*, *SplitButton*, *DropDownButton*, *Separator*, *ComboBox*, *TextBox* in *ProgresBar*. Nove gradnike v orodjarno dodajamo tako, da odpremo spustni seznam na desni strani letvice in izberemo vrsto gradnika.

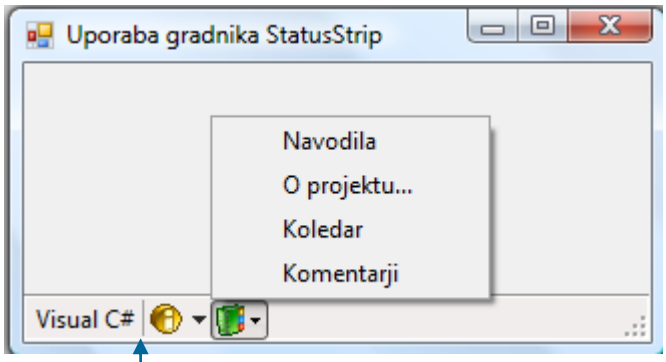
Novo postavko tipa *Button* v orodjarni lahko najhitreje dobimo tako, da nemsto odpiranja spustnega seznama nanj le kliknemo. Tako nastalim gumbom lahko poljubno spreminjamo lastnosti (npr. sličico na gumbu, …) in prirejamo odzivne metode (najpogosteje dogodek *Click*). Posamezne gradnike znotraj orodjarne lahko kadarkoli odstranimo (pobrišemo) ali pa jih preuredimo (primemo z miško in prenesemo na drugo pozicijo v orodjarni).



Slika 42: Oblikovanje orodjarne.

Gradnik StatusStrip

StatusStrip je gradnik, ki se običajno prikazuje na dnu obrazca. Namenjen je oblikovanju statusne vrstice na dnu okna. V gradniku ustvarjamo predalčke: to storimo tako, da odpremo spustni seznam znotraj tega gradnika in izberemo eno od ponujenih opcij (*StatusLabel* – napis, *ProgressBar*, *DropDownButton* in *SplitButton*). Predalčkom tipa *StatusLabel* običajno še priredimo lastnost *BorderSides* – *Right*, da dobimo robove. Delo s posameznimi predalčki je zelo podobno oblikovanju glavnega menija. Širina predalčkov (*size*) je lahko poljubna, le lastnost *AutoSize* posameznega predalčka moramo prej nastaviti na *False*. Kadar je predalček tipa *StatusLabel*, mu lahko določimo še dve posebni lastnosti: lastnost *isLInk* (če jo nastavimo na *True*, se obnaša kot bližnjica, npr. do spletne strani), in lastnost *Spring* (če jo nastavimo na *True*, se predalček s oznako razširi tako, da z ostalimi predalčki zavzema celotno širino obrazca).



StatusStrip s tremi predalčki tipa StatusLabel, SplitButton in DropDownButton

Slika 43: Gradnik *StatusStrip* s tremi predalčki.

Do posameznega predalčka gradnika *StatusStrip* lahko dostopamo tudi programsko. Vsi skupaj sestavljajo zbirko predalčkov (*Items*), do posameznega pa dostopamo preko indeksa, npr. takole:

```
//prikaz teksta v prvem predalčku (indeks je 0) statusne vrstice
statusStrip1.Items[0].Text = "TŠC Kranj!";
```

Dialogi – okenska pogovorna okna

Poleg sporočilnega okna *MessageBox* vsebuje razvojno okolje *Visual C#* še kar nekaj koristnih in zelo uporabnih pogovornih oken. Standardna pogovorna okna najdemo v oknu *Toolbox* v skupini *Dialogs***.** Če kateregakoli od teh gradnikov (dialogov) postavimo na obrazec se njegova "slika" pokaže v polju pod obrazcem. Tam ga potem lahko kadarkoli izberemo, ter mu prirejamo lastnosti in dogodke. S tem, ko pa smo ga postavili na obrazec, smo v projekt vključili objekt ustreznega pogovornega okna. Imena tako ustvarjenih objektov so privzeta: (*colorDialog1*, *fontDialog1*, *openDialog1*…), do njihovih lastnosti in dogodkov pa tako kot pri vseh objektih dostopamo s pomočjo operatorja pika.

Osnovna pogovorna okna, ki jih pozna *Visual C#*, so naslednjih tipov:

- *ColorDialog*,
- FolderBrowserDialog,
- *FontDialog*,
- *OpenDialog* in
- *SaveDialog*.

Vsa pogovorna okna naštetih tipov odpiramo (prikažemo) z metodo *ShowDialog*. Okna so *modalna*, kar pomeni, da vračajo vrednost naštevnega tipa *DialogResult* (tako kot smo to že videli pri sporočilnem oknu *MessageBox*).

ColorDialog – pogovorno okno za izbiro barve

Pogovorno okno *ColorDialog* je namenjeno izbiri barve. Če uporabnik v tem oknu izbere poljubno barvo in okno zapre z gumbom *OK* (*V redu*), se izbrana barva shrani v lastnost okna *Color*. Na ta način lahko uporabniku omogočimo, da po svojih željah nastavi barvo, ki jo bo v programu uporabil na različne načine (lahko bo to vrednost nekega polja v bazi, ozbrana barva lahko povzroči spremembo barve ozadja, od izbire barve je lahko odvisna tudi razvejitev programa…).

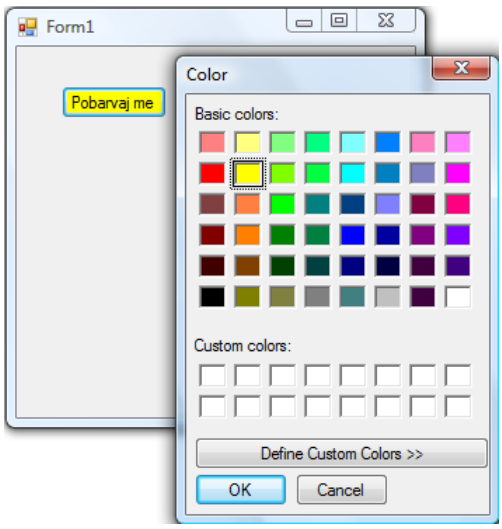
Lastnost	Razlaga
AlowFullOpen	Omogočen/onemogočen gumb za mešanje barv v oknu.
AnyColor	Če lastnost postavljena na *True* bodo v množici bazičnih barv prikazane vse možne barve.
Color	Predzbrana barva gradnika.
SolidColorOnly	Barve v pogovornem oknu bodo/ne bodo omejene le na prave/pristne barve (osnovne barve brez senc in poltonov) .

Tabela 12: Najpomembnejše lastnosti pogovornega okna *ColorDialog*.

Naslednji primer prikazuje, kaj se zgodi ob dogodku *Click* gumba z imenom *bPobarvaj*. Ob kliku na ta gumb se je odprlo pogovorno okno *ColorDialog* in če uporabnik v tem oknu izbere poljubno barvo in okno zapre s klikom na gumb *OK* (*V redu*), se gumb pobarva z izbrano barvo.

To dosežemo z naslednjo odzivno metodo.

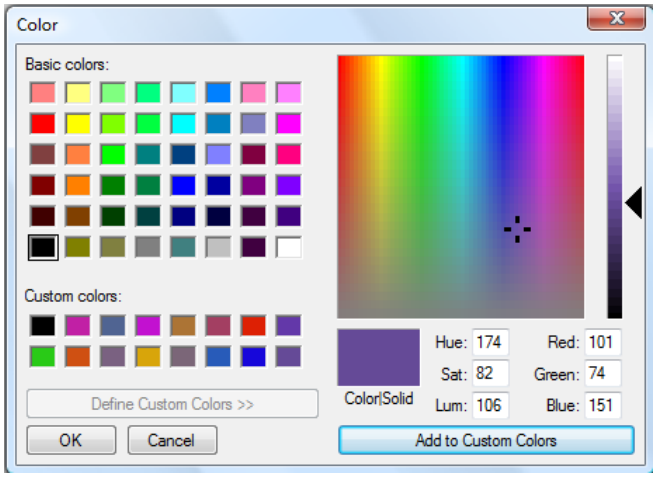
```
private void bPobarvaj_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog()==DialogResult.OK)
            (sender as Button).BackColor = colorDialog1.Color;
            /*lahko pa bi zapisali tudi takole:  bPobarvaj.BackColor =
              colorDialog1.Color; */
}
```



Slika 44: Pogovorno okno *ColorDialog*.

Posebnost okna za izbiranje barv je njegova tabelarična lastnost *CustomColors* (barve po meri), kamor lahko uporabnik med izvajanjem programa shrani 16 svojih, iz osnovnih barv narejenih oz. "namešanih" barv. Odpremo ga s klikom na gumb *Define Custom Colors*. Okno se razširi in v

razširjenem oknu lahko z dvoklikom na barvni paleti najprej izberemo ustrezno barvo, nato pa jo s klikom na *Add to Custom Colors* dodamo na paleto *Custom Colors*. Izbrane barve so v tej paleti shranjene ves čas delovanja programa.



Slika 45: Ustvarjanje lastne barve palete v pogovornem oknu *ColorDialog*.

Vsak dialog (vsako pogovorno okno) lahko definiramo in ga prikažemo tudi povsem programsko, ne da bi prej na obrazec postavili ustrezen gradnik. To dosežemo z dinamičnim zaseganjem pomnilnika (ustvarjanjem novega objekta ustreznega razreda) za konkreten dialog. Za *ColorDialog* bi to naredili npr. takole:

```
private void button1_Click_1(object sender, EventArgs e)
{
   //najprej ustvarimo nov objekt tipa ColorDialog. Ime objekta naj bo cD1
   ColorDialog cD1=new ColorDialog();
   /*dialog odpremo z metodo ShowDialog in preverimo če je uporabnik
     kliknil gumb V redu*/
   if (cD1.ShowDialog()==DialogResult.OK)
      (sender as Button).BackColor = cD1.Color;

      /*Če je bil kliknjen gumb V redu, pobarvamo pošiljatelja (v našem
        primeru gumb button1). Parameter sender smo uporabili tudi v programu
        Hitrostno klikanje*/
}
```

Naslednji primer prikazuje, kako dinamično ustvarimo nov barvni dialog in nato z njegovo pomočjo določimo barvo gradnika *TextBox* (ime gradnika *textBox1*) – kodo zapišemo npr. ob dogodku *Click* gumba *button2*.

```
private void button2_Click(object sender, EventArgs e)
{
    ColorDialog MyDialog = new ColorDialog();//Ustvarimo nov dialog
    MyDialog.AllowFullOpen = false; // Onemogočimo gumb za mešanje barv
```

```
    MyDialog.ShowHelp = true; // Uporabniku omogočimo dostop za gumb Pomoč.
    /*Dialogu ColorDialog nastavimo začetno barvo na trenutno barvo
      gradnika textBox1.*/
    MyDialog.Color = textBox1.ForeColor;
    /*Če uporabnik v ColorDialog klikne gumb V redu, spremenimo barvo v
      gradniku textBox1 */
    if (MyDialog.ShowDialog() == DialogResult.OK)
        textBox1.ForeColor = MyDialog.Color;
}
```

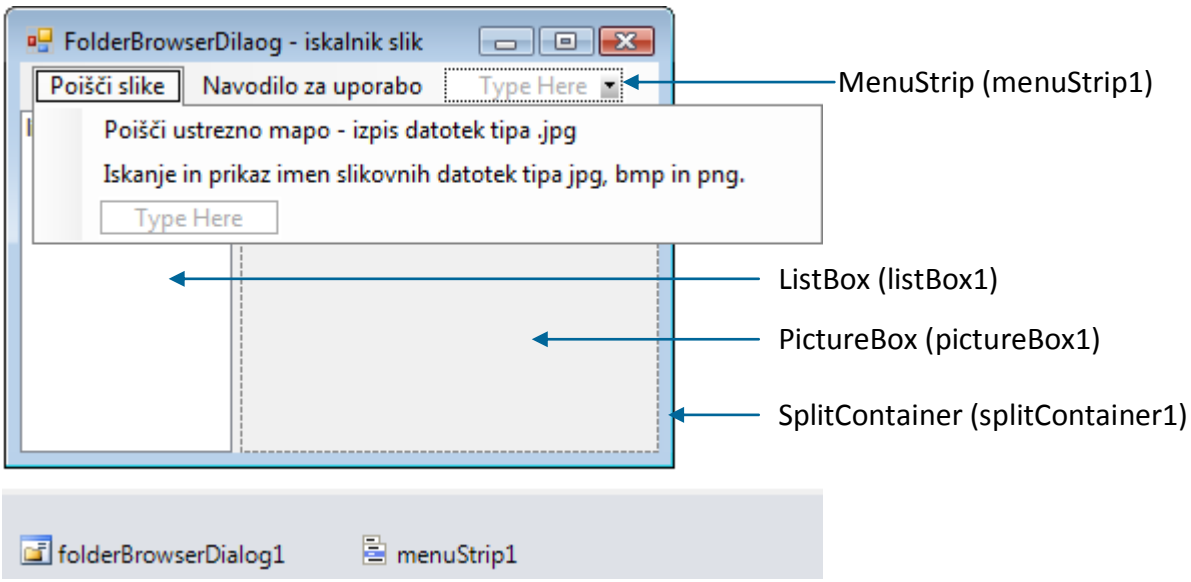
FolderBrowserDialog – pogovorno okno za raziskovanje map

Pogovorno okno *FolderBrowserDialog* omogoča uporabniku premikanje po mapah računalnika in ustvarjanje novih map. Dialog je nekakšen komplement dialogu *OpenFileDialog*, ki pa se uporablja za raziskovanje in določanje imen datotek. Okno odpremo z metodo *ShowDialog*, omogoča pa izbiro poljubne mape. Izbiro potrdimo s klikom na gumb *OK* (*V redu***)**. To pomeni, da okno vrne vrednost *DialogResult.OK.*

Lastnost	Razlaga
Description	Nastavitev poljubnega teksta (naslova), ki se izpiše nad drevesno strukturo map v oknu.
SelectedPath	Če že pred odpiranjem dialoga tu vnesemo vrednost, s tem določimo mapo, ki bo izbrana, ko se bo dialog odprl. Ob zaključku pa ta lastnost vsebuje ime izbrane mape.
ShowNewFolderButton	V oknu je privzeto tudi gumb, ki omogoča ustvarjanje novih map. Če lastnost *ShowNewFolderButton* postavimo na *false*, tega gumba ni.

Tabela 13: Glavne lastnosti pogovornega okna *FolderBrowserDialog*.

Uporabo *FolderBrowserDialog* dialoga prikažimo na primeru preprostega iskalnika slik. Na obrazec postavimo gradnik *MenuStrip* in v njem ustvarimo postavke kot jih prikazuje spodnja slika. Dodajmo še *SplitContainer* – ta gradnik vsebuje dve plošči (dva panela), ki ju lahko med izvajanjem poljubno širimo in ožimo. Na levi panel nato postavimo gradnik *ListBox* (*Dock* = *Fill*), v desnega pa *PictureBox* (*Dock* = Fill, *SizeMode*=*StretchImage*). Gradnik *ListBox* že s svojim imenom nakazuje, da se uporablja za izpisovanje seznamov, rezultatov poizvedb, ipd. Na obrazec postavimo še gradnik *FolderBrowserDialog*, s pomočjo katerega bomo iskali mapo s slikami.



Slika 46: Gradniki na obrazcu za prikaz pogovornega okna *FolderBrowserDialog*.

Uporabili smo gradnik *PictureBox*. Namen tega gradnika je prikaz poljubne slike.

S pomočjo pogovornega okna *FolderBrowserDialog* in dogodkov *Click* glavnega menija, bomo v oknu *ListBox* prikazali enkrat imena vseh slikovnih datotek tipa *jpg*, v drugem primeru pa slikovnih datotek tipov *jpg*, *bmp* in *png*. Uporabili bomo metodo *GetFiles* razreda *Directory*. Posredujemo ji dva parametra: mapo, ki jo preiskujemo in tipa datotek, ki jih iščemo v tej mapi. Metoda vrne seznam vseh datotek, ki ga priredimo tabeli nizov.

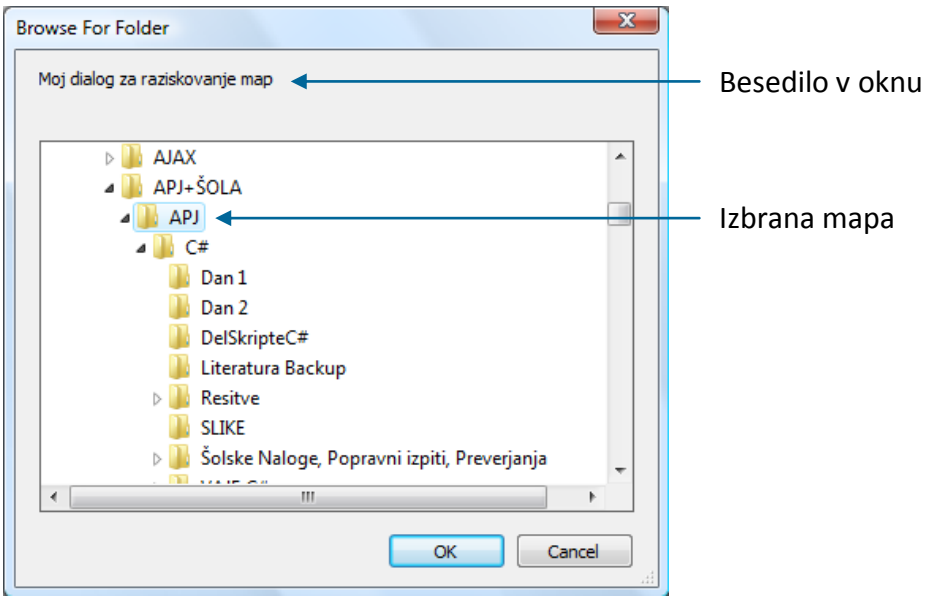
```
private void SlikeJPG_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        string[] fileArray;
        //imena vseh datotek tipa jpg izbrane mape zapišemo v tabelo nizov
        fileArray = Directory.GetFiles(folderBrowserDialog1.SelectedPath,
"*.jpg");
         foreach (string imeDat in fileArray)//slike dodamo v gradnik ListBox
            listBox1.Items.Add(imeDat);
    }
}
private void VseSlike_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        string[] fileArray;
        //imena vseh datotek izbrane mape zapišemo v tabelo nizov
        fileArray = Directory.GetFiles(folderBrowserDialog1.SelectedPath,
"*.*");
        foreach (string imeDat in fileArray) //obdelamo tabelo
```

```
        {
            //črke v imenih datoteke spremenimo v velike črke
            string ime = imeDat.ToUpper();
            //če gre za datoteko s ustrezno končnico, jo dodamo v seznam
            if (ime.EndsWith(".JPG") || ime.EndsWith(".BMP") ||
ime.EndsWith(".PNG"))
                listBox1.Items.Add(imeDat);
        }
    }
}
private void NavodiloZaUporabo_Click(object sender, EventArgs e)
{
    MessageBox.Show("V meniju 'Poišči slike' najprej poišči mapo s
slikami.\nSeznam slikovnih datotek se nato prikaže v gradniku ListBox.\n\nOb
kliku na vrstico ListBox-a se prikaže ustrezna slika!\n\nMejo med seznamom
slik in sliko lahko premikaš levo-desno.\n\nCeloten obrazec lahko poljubno
povečaš ali zmanjšaš.", "Navodilo za uporabo", MessageBoxButtons.OK,
MessageBoxIcon.Information);
}
private void listBox1_Click(object sender, EventArgs e)
{
    try
    {
        pictureBox1.Image = Image.FromFile(listBox1.SelectedItem.ToString());
    }
    catch
    { }
}
```

Oglejmo si še primer dinamičnega ustvarjanja *FolderBrowserDialog*-a in programske spremembe nekaterih nastavitev. Dialog smo poimenovali *MojDialog*.

```
private void button3_Click(object sender, EventArgs e)
{
    FolderBrowserDialog MojDialog = new FolderBrowserDialog();
    //Določimo tekst v oknu, tik nad prikazom map
    MojDialog.Description = "Moj dialog za raziskovanje map";
    //Določimo privzeto ime mape, ki bo izbrana ob prikazu dialoga
    MojDialog.SelectedPath = "c:\\Program Files\\Common Files";
    //Onemogočimo gumb za ustvarjanje novih map
    MojDialog.ShowNewFolderButton = false;

    if (MojDialog.ShowDialog() == DialogResult.OK)
    {    /*stavki, ki naj se izvedejo če uporabnik izbiro mape potrdi s
         klikom na gumb OK (V redu)*/
    }
}
```

Pri navajanju privzete poti v lastnosti *SelectedPath* moramo zapisati znaka \\ , ali pa pred začetkom niza zapisati znak @. Že od prej namreč vemo, da enojni znak \ v nizu pomeni ubežni niz.



Slika 47: *FolderBrowserDialog* z nekaterimi programskimi prednastavitvami.

FontDialog – pogovorno okno za izbiro pisave.

Pogovorno okno *FontDialog* služi za izbiro pisave. Če uporabnik pisavo izbere in okno zapre s klikom na gumb *V redu***,** okno vrne vrednost *DialogResult.OK*, izbrana pisava pa se shrani v lastnost okna *Font*.

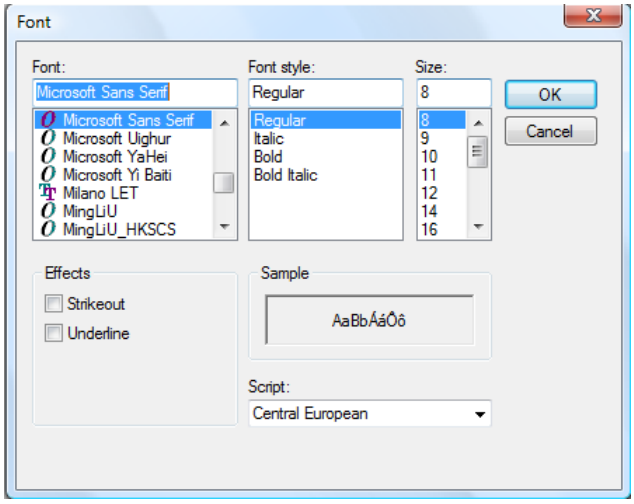
Naslednji primer pa prikazuje, kako s pomočjo dialoga za izbiro pisave spremenimo pisavo gradnika *TextBox* (ime gradnika je *textBox1*). Prikazana sta dva načina: v prvem primeru upoštevamo vse uporabnikove nastavitve v oknu *FintDialog*, v drugem (v obliki komentarja) pa le nekatere (ime, velikost in stil pisave). Poudarimo, da s tem le določimo ime datoteke. Same datoteke pa s tem še ne odpremo. Nekatere lastnosti pogovornega okna *FontDialog* so zbrane v naslednji tabeli:

Lastnost	Razlaga
Color	Izbrana barva pisave.
Font	Izbrana vrste pisave.
MaxSize	Nastavitev največje velikosti pisave, ki jo uporabnik še lahko izbere.
MinSize	Nastavitev najmanjše velikosti pisave, ki jo uporabnik še lahko izbere.
ShowApply	V oknu bo oz. ne bo gumba *Uporabi*.

ShowColor	V oknu bo oz ne bo možno izbiranje barve pisave.
ShowEfects	V oknu bo oz. ne bo prikazana postavka *Učinki* (prečrtanje, podčrtavanje in izbira barve).
ShowHelp	V oknu bo oz. ne bo gumba *Pomoč*.

Tabela 14: Najpomembnejše lastnosti pogovornega okna *FontDialog*.

```
if (fontDialog1.ShowDialog() == DialogResult.OK)
{
     textBox1.Font = fontDialog1.Font;
     /*lahko tudi textBox1.Font = new Font(fontDialog1.Font.Name,
       fontDialog1.Font.Size ,fontDialog1.Font.Style);*/
}
```



Slika 48: Pogovorno okno *FontDialog* za izbiro pisave.

OpenFileDialog - pogovorno okno za odpiranje datotek

Pogovorno okno *OpenFileDialog* je namenjeno izbiri datoteke, ki jo nameravamo odpreti. Okno odpremo z metodo *ShowDialog***.** Če uporabnik datoteko izbere in okno zapre s klikom na gumb *OK* (*V redu*)**,** okno vrne vrednost *DialogResult.OK*, ime izbrane datoteke pa se shrani v lastnost okna *FileName*.

Lastnost	Razlaga
AddExtension	Lastnost določa, ali naj pogovorno okno avtomatično doda datoteki končnico, v primeru, da jo je uporabnik pozabil napisati.
CheckFileExists	Lastnost določa, ali naj pogovorno okno izpiše obvestilo v primeru, da je uporabnik navedel ime datoteke, ki ne obstaja.
CheckPathExists	Lastnost določa, ali naj pogovorno okno izpiše obvestilo v primeru,

	da je uporabnik navedel pot do datoteke, ki ne obstaja.
DefaultExt	Nastavitev privzete končnice datoteke.
FileName	Ime datoteke, ki naj bo izbrana v pogovornem oknu, po izhodu pa tu dobimo "rezultat".
Filter	Omejitev le na določene vrste datotek (npr. tekstovne, ..).
InitialDirectory	Nastavitev privzetega imenika.
Multiselect	Lastnost določa, ali lahko v pogovornem oknu hkrati izberemo več datotek
ReadOnlyChecked	Lastnost določa kakšna je privzeta nastavitev za možnost *ReadOnly*, ki uporabnika opozarja, da izbrano datoteko lahko odpre le branje. Nastavitev opcije je smiselna le v primeru, da je lastnost *ShowReadOnly* nastavljena na *True*.
ShowReadOnly	Lastnost določa, ali naj se v pogovornem okno prikaže stikalo *Samo za branje*.
Title	Naslov pogovornega okna.

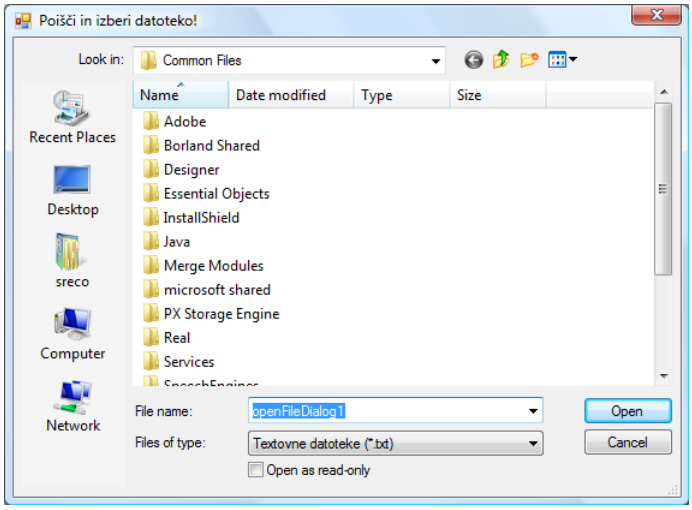
Tabela 15: Lastnosti pogovornega okna *OpenFileDialog*.

Naslednji primer prikazuje, kako s pomočjo *OpenFileDialoga* ime neke datoteke zapišemo v polje *TextBox* (ime gradnika je *textBox1*).

```
/*Pred odpiranjem okna lahko nastavimo tudi filter, s katerim povemo kakšne
vrste datoteke želimo prikazati v oknu*/
openFileDialog1.Filter = "Textovne datoteke (*.txt)|*.txt|Moje datoteke
(*.MOJ)|*.MOJ";

/*s stikalo Read Only lahko uporabnika obvestimo, da bo izbrano datoteko
lahko le pregledoval, ne pa tudi ažuriral*/
openFileDialog1.ShowReadOnly = true;
//POZOR: dva znaka \\ za dostop do podmape pišemo zaradi tolmačenja znaka \ v
//nizu
openFileDialog1.InitialDirectory = "c:\\Program Files\\Common
Files";//Privzeti imenik
//Določimo lahko tudi naslov pogovornega okna
openFileDialog1.Title = "Poišči in izberi datoteko!";
//Pogovorno okno odpremo z metodo ShowDialog
if (openFileDialog1.ShowDialog() == DialogResult.OK) textBox1.Text =
openFileDialog1.FileName;
```

V prikazanem primeru smo nastavili tudi lastnost *Filter*, s katero povemo, katere vrste datotek bodo v oknu prikazane. Z ločilno črto "|" delimo filter v dva dela: prvi del pomeni besedilo v spustnem seznamu za izbiro tipa datoteke, drugi del pa so dejanski tipi datotek. Nastavimo lahko več filtrov hkrati. Z lastnostjo *ShowReadOnly* lahko uporabnika tudi obvestimo, da

datotek ne bo mogel spreminjati. Z lastnostjo *InitialDirectory* smo dosegli, da se pri odpiranju dialoga prikaže vsebina točno določene mape. Lastnost *Title* določa napis na vrhu okna *OpenFileDialog* (privzeti zapis bo sicer *Open*).



Slika 49: Pogovorno okno *OpenFileDialog*.

SaveFileDialog - pogovorno okno za shranjevanje datotek

Pogovorno okno *SaveFileDialog* je namenjeno pomoči pri shranjevanju datotek. Če uporabnik datoteko izbere in okno zapre s klikom na gumb *OK* (*V redu*), okno vrne vrednost *DialogResult.OK*, ime izbrane datoteke pa se shrani v lastnost okna *FileName*.

S pogovornima oknoma *OpenFileDialog* in *SaveFileDialog* datoteke ni mogoče ne odpreti ne shraniti. Pogovorni okni samo omogočata izbiro datoteke, ki jo nameravamo odpreti oziroma shraniti. Za odpiranje in zapiranje datoteke moramo poskrbeti z ustrezno programsko kodo.

Lastnost	Razlaga
AddExtension	Lastnost določa, ali naj se datoteki, v primeru da jo je uporabnik pozabil napisati, avtomatično doda končnica. Če je vrednost nastavljena na *True*, se imenu datoteke (seveda le v primeru, da lastnost filter ni nastavljena), avtomatično doda končnica, zapisna v *DefaultExt*.
CheckFileExists	Lastnost določa, ali naj se izpiše obvestilo v primeru, da je uporabnik navedel ime datoteke, ki ne obstaja.
CheckPathExists	Lastnost določa, ali naj se izpiše obvestilo v primeru, da je uporabnik navedel pot do datoteke, ki ne obstaja.
CreatePrompt	Nastavitev dovoljenja za ustvarjanje datoteke, če uporabnik navede ime datoteke, ki še ne obstaja.

DefaultExt	Nastavitev privzete končnice datoteke.
FileName	Ime datoteke, ki jo izberemo v pogovornem oknu, oz. ime datoteke, ki smo ga zapisali sami. Po zapiranju okna tu dobimo "rezultat".
Filter	Omejitev le na določene vrste datotek (npr. tekstovne, ..).
InitialDirectoty	Nastavitev privzetega imenika.
OverwritePrompt	Lastnost določa, ali naj nas program opozori, če za ciljno datoteko izberemo datoteko, ki že obstaja.
Title	Naslov pogovornega okna.

Tabela 16: Tabela lastnosti pogovornega okna *SaveFileDialog*.

V naslednjem primeru bo prikazana še uporaba gradnika *RichTextBox*. Besedilo, ki ga uporabnik zapiše v ta gradnik, je lahko poljubno oblikovano. Primer prikazuje, kako v datoteko zapišemo celotno vsebino gradnika *RichTextBox* (ime gradnika je *richTextBox1*). Pri tem si za izbiro datoteke pomagamo z gradnikom *SaveFileDialog*.

```
private void button2_Click(object sender, EventArgs e)
{
    saveFileDialog1.OverwritePrompt = true; /*program naj nas opozori, če
                                            datoteka že obstaja*/
    //Določimo privzeti imenik
    saveFileDialog1.InitialDirectory = "c:\\Programi";
    saveFileDialog1.Title = "Shrani datoteko!"; //Naslov pogovornega okna
    //Nastavitev filtra
    saveFileDialog1.Filter = " Moje rtf datoteke (*.rtf)|*.rtf";
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        richTextBox1.SaveFile(saveFileDialog1.FileName,
RichTextBoxStreamType.RichText);
}
```

V prikazanem primeru smo s pomočjo lastnosti *OverwritePrompt* najprej poskrbeli, da nas program pri shranjevanju v datoteko opozori, če ta že obstaja. Lastnost *Title* določa napis na vrhu okna *SaveFileDialog* (privzeti zapis bo sicer *Save As*).

Predvajalnik glasbe in videa

Visual C# zna delati tudi z glasbenimi in video datotekami. Ustvarimo nov projekt in ga poimenujmo *Glasba*. Spoznali bomo razred *SoundPlayer* za delo z glasbenimi datotekami in gradnik *WindowsMediaPlayer* za predvajanje glasbe in videa.

Na obrazec postavimo gradnik *MenuStrip* z izbirami:

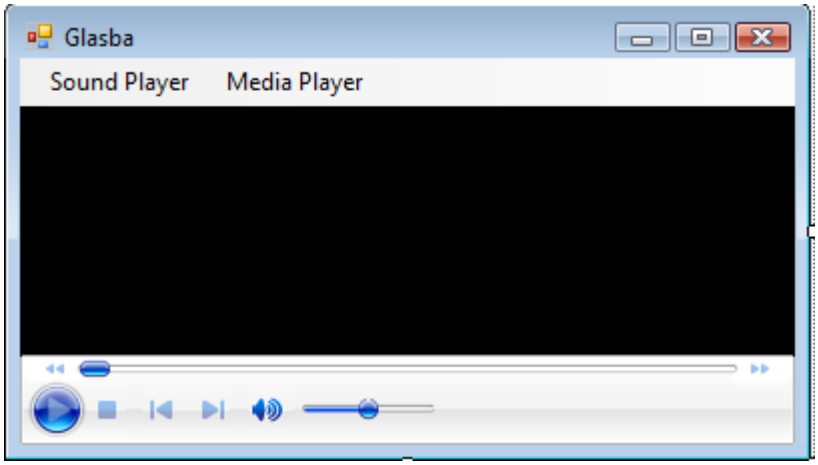
- SoundPlayer
  - Zaigraj privzeto skladbo
  - Zaigraj mojo skladbo
  - Poišči glasbo
  - Zaključi predvajanje
- MediaPlayer
  - Moja glasba
  - Poišči glasbo
  - Video

Za prvo postavko menija bomo potrebovali razred *SoundPlayer* (objekti tipa *SoundPlayer* so namenjeni predvajanju glasbe tipa *wav*). V sekciji *using* moramo dodati stavek

```
using System.Media;
```

Drugo postavko menija *SoundPlayer* bomo realizirali s pomočjo gradnika *WindowsMediaPlayer*. V okno *ToolBox* dodajmo že pripravljen gradnik *Windows Media Player* takole: preklopimo na pogled *Design* → *ToolBox* → *General* → desni klik miške → *Choose Items* → *COM Components*, poiščemo *WindowsMediaPlayer*, ga odkljukamo in izbiro potrdimo s klikom na gumb *OK*. V paleti *General* se pojavi nov gradnik *Windows Media Player*, ki ga postavimo na obrazec. Gradniku nastavimo še lastnost *Dock=Fill*.

Prvo postavko menija *SoundPlayer* (*Zaigraj privzeto skladbo*) izvedemo tako, da *Solution Explorerju* izberimo *Glasba→Properties*. V oknu, ki se prikaže, izberimo *Resources→Audio* in s potegom miške (*Drag & Drop*) v to okno povlecimo poljubno datoteko tipa .wav (v našem primeru je to datoteka *AmericanPie.WAV*).



Slika 50: Obrazec za predvajanje glasbe in videa.

Druga postavka menija *SoundPlayer* je podobna prvi, le da smo datoteko tipa *.wav* naprej prekopirali v točno določeno mapo diska C, od koder jo nato predvajamo. S prvo in drugo

postavko menija smo tako le prikazali, kako lahko določeno skladbo (ali pa seveda zvok) vključimo v projekt na dva načina. Na računalniku moramo v ta namen najprej poiskati dve glasbeni datoteki in potem ustrezno nastaviti vrednost spremenljivki *PotDoGlasbe*. Pri takih projektih bi bilo torej smiselno programiranje ustreznih nastavitev naše aplikacije. Podatke o poteh do podatkov (datotek, slik, zvokov, …) bi zapisali v posebno datoteko, nastavili pa bi jih pri izdelavi namestitvenega programa.

Tretja postavka menija *SoundPlayer* prikazuje, kako lahko neko skladbo poiščemo s pomočjo dialoga *OpenFileDialog*.

Prva postavka menija *MediaPlayer* prikazuje, kako lahko glasbeno datoteko, ki se nahaja v določeni mapi, predvajamo s pomočjo *MediaPlayer*-ja, druga in tretja postavka pa, kako lahko glasbeno oz. video datoteko poiščemo s pomočjo objekta tipa *OpenFileDialog*. Ime izbrane datoteke moramo zapisati v lastnost URL našega predvajalnika.

```
public partial class Form1 : Form
{
    //V sekcijo using najprej vključimo stavek using System.Media;
    //objekti, ustvarjeni iz razreda SoundPlayer, igrajo le glasbo tipa wav!
    public SoundPlayer mojPredvajalnikGlasbe = new SoundPlayer();
    public Form1()
    {
        InitializeComponent();
    }
    private void ZaigrajPrivzetoSkladboToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        /*V Solution Explorerju izberimo našo rešitev (Glasba)->Properties.
          V oknu, ki se prikaže, izberimo Resources-->Audio. S potegom
          miške (Drag & Drop) nato v to okno povlecimo poljubno datoteko
          tipa .wav. Okno Properties nato lahko zapremo.*/
          mojPredvajalnikGlasbe.Stream = Properties.Resources.American_Pie;
          //z metodo PlayLooping začnemo predvajanje, ko pa se skladba konča
          //se predvajanje avtomatično prične znova
          mojPredvajalnikGlasbe.PlayLooping();
    }
    private void ZaigrajMojoSkladboMapiToolStripMenuItem_Click(object sender,
EventArgs e)
    {
         string potDoGlasbe = @"C:\Glasba\Lolita.wav";
         mojPredvajalnikGlasbe.SoundLocation = potDoGlasbe;
         mojPredvajalnikGlasbe.Play();
    }
    private void PoiščiGlasboToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        OpenFileDialog opf = new OpenFileDialog();
        opf.Filter = "Glasbene datoteke|*.wav";
```

```
        if (opf.ShowDialog() == DialogResult.OK)
        {
            mojPredvajalnikGlasbe.SoundLocation = opf.FileName;
            mojPredvajalnikGlasbe.Play();
        }
    }
    private void ZaključiPredvajanjeToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        //zaustavitev predvajalnika
        mojPredvajalnikGlasbe.Stop();
    }
    private void MojaGlasba_Click(object sender, EventArgs e)
    {
        //določimo pot in ime skladbe za predvajanje v MediaPlayer-ju
        string potDoGlasbe = @"C:\Glasba\Krompir.mp3";
        WindowsMediaPlayer.settings.autoStart = true;
        WindowsMediaPlayer.URL = potDoGlasbe;
        WindowsMediaPlayer.Visible = true;
    }
    private void PoiščiGlasboToolStripMenuItem2_Click(object sender,
EventArgs e)
    {
        /*s pomočjo OpenFileDialoga poiščemo skladbo za predvajanje v
          MediaPlayer-ju*/
        OpenFileDialog opf = new OpenFileDialog();
        //dovolimo iskanje le glasbenih datotek tipa wav in mp3
        opf.Filter = "Glasbene datoteke(*.wav *.mp3)|*.wav;*.mp3";
        if (opf.ShowDialog() == DialogResult.OK)
        {
            WindowsMediaPlayer.settings.autoStart = true;
            // izbrano glasbeno datoteko zapišemo v lastnost URL
            WindowsMediaPlayer.URL = opf.FileName;
            WindowsMediaPlayer.Visible = true;
        }
    }
    private void VideoToolStripMenuItem_Click(object sender, EventArgs e)
    {
        OpenFileDialog opf = new OpenFileDialog();
        //dovolimo iskanje le video datotek tipa WMV in AVI
        opf.Filter = "Video datoteke(*.WMV *.AVI)|*.WMV;*.AVI";
        if (opf.ShowDialog() == DialogResult.OK)
        {
            WindowsMediaPlayer.settings.autoStart = true;
            //izbrano video datoteko zapišemo v lastnost URL
            WindowsMediaPlayer.URL = opf.FileName;
            WindowsMediaPlayer.Visible = true;
        }
    }
}
```

Preprosti urejevalnik besedil

Izdelajmo svoj preprosti urejevalnik besedil. Besedilo bomo lahko urejali in oblikovali, shranjevali v tekstovne datoteke, možno pa bo tudi odpiranje že obstoječih tekstovnih datotek.

Na obrazec zaporedoma postavimo gradnike *MenuStrip*, *ToolBox*, *StatusStrip*. V srednji del postavimo še *Panel* (*Dock=Fill*), nanj pa gradnik *TextBox* (*Dock=Fill*). Glavni meni oblikujmo tako, da bo vseboval opcije:

- Datoteka
  - Nova
  - Odpri
  - Shrani
  - Shrani kot…
  - Izhod
- Urejanje
  - Razveljavi
  - Izreži
  - Kopiraj
  - Prilepi
  - Izberi vse
- Oblikovanje
  - Pisava
  - Barva pisave
  - Poravnava besedila
    - Levo
    - Sredinsko
    - Desno
  - Ozadje
- Pogled
  - Statusna vrstica

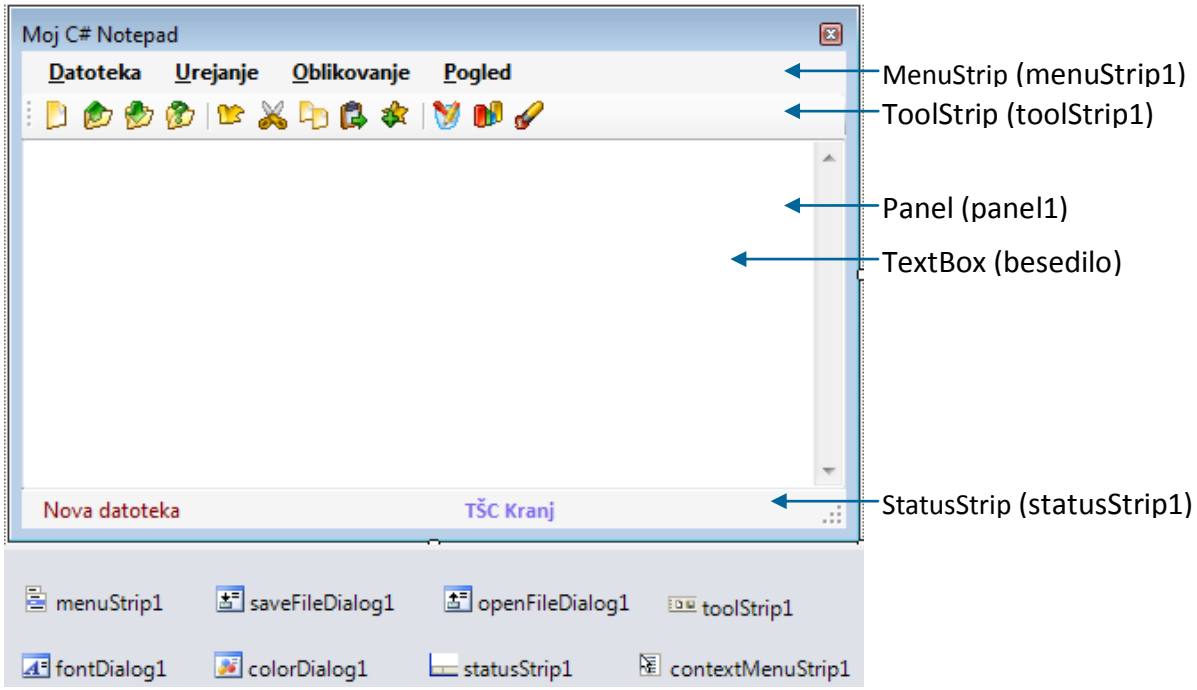
V orodjarno dodajmo 12 gumbov in jim priredimo primerne slike tipa *bmp*. Eden od možnih virov za sličice je npr. *http://qicons.cubeactive.com/*.

Spomnimo se, da gumbe v orodjarno dodajamo z desnim klikom na gradnik tipa *ToolStrip*, za katerim si izberemo *Button*. Dogodki, ki jih bomo priredili gumbom v orodjarni, se bodo ujemali z vrstnim redom postavk glavnega menija.

Naredimo še dva predalčka v gradniku *StatusStrip*. Prvemu določimo lastnost *AutoSize* na *False*, nato pa spremenimo *Size→Width* na 100. S tem bomo dosegli zadostno širino za tekst, ki ga želimo v predalčku prikazati. Drugemu nastavimo lastnost *Spring* na *True*, da se razširi čez

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

celotno preostalo širino gradnika *StatusStrip*). Prvi predalček bo hranil ime datoteke, ki jo trenutno urejamo. Če pa datoteka še ni shranjena, bo v njem napis *Nova datoteka* (takšno je tudi začetno besedilo).



Slika 51: Gradniki *Visual C#* urejevalnika.

V projekt dodajmo še lebdeči meni (*ContextMenuStrip*) z naslednjimi opcijami:

- Izreži
- Kopiraj
- Prilepi
- Izberi vse
- Razveljavi
- Pisava
- Barva pisave
- Ozadje

Imena dogodkov se ujemajo z njihovim namenom. Zato v spodnji kodi ne bo težko ugotoviti, kateri opciji glavnega menija, lebdečega menija ali pa gumbom v orodjarni pripadajo.

```
//Oznaka, da gre za novo datoteko, ki še ni shranjena
private string nova = "Nova datoteka";
public Form1()//Konstruktor obrazca
{
    InitializeComponent();
    //oznaka, da gre za novo, še neimenovano datoteko
```

```
    toolStripStatusLabel1.Text=nova;
}
//Nova datoteka
private void novaToolStripMenuItem_Click(object sender, EventArgs e)//Nova
datoteka
{
    ShraniSpremembe("Nov dokument");
    besedilo.Clear();
    toolStripStatusLabel1.Text = nova;
}
//metoda za preverjanje sprememb
private void ShraniSpremembe(string napis)
{
    //preverimo, če je besedilo v urejevalniku spremenjeno in še neshranjeno
    if (besedilo.Modified)
    {
        if (MessageBox.Show("Shranim spremembe?", napis,
MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            /*če je uporabnikova odločitev DA,se vsebina gradnika besedilo
              shrani v datoteko*/
            if (toolStripStatusLabel1.Text != nova)
            {
                try
                {
                    //zapis v datoteko
                    File.WriteAllText(toolStripStatusLabel1.Text,
besedilo.Text);
                }
                catch
                { MessageBox.Show("Napaka pri pisanju v datoteko!"); }
            }
            else VDatoteko();//klic metode za shranjevanje v datoteko
        }
    }
}
//metoda za shranjevanje v datoteko
private void VDatoteko()
{
    try
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            //preverimo, če datoteka s tem imenom že obstaja
            if (File.Exists(saveFileDialog1.FileName))
            {
                if (MessageBox.Show("Datoteka s tem imenom že obstaja! Ali
naj jo prepišem?", "POZOR!", MessageBoxButtons.YesNo) == DialogResult.Yes)
                {
                    File.WriteAllText(saveFileDialog1.FileName,
```

```
                                        besedilo.Text);
                    toolStripStatusLabel1.Text = saveFileDialog1.FileName;
                }
            }
            else
            {
                File.WriteAllText(saveFileDialog1.FileName, besedilo.Text);
                toolStripStatusLabel1.Text = saveFileDialog1.FileName;
            }
        }
    }
    catch
    { MessageBox.Show("Napaka pri pisanju v datoteko!"); }
}
```

Pokazali smo, kaj vse je potrebno postoriti pred odpiranjem nove datoteke. Za preverjanje sprememb in shranjevanje v datoteko smo napisali dve novi metodi. Sledi pa nekaj odzivnih metod za odpiranje že obstoječe datoteke in prenos vsebine te datoteke v gradnik *besedilo*, ter za shranjevanje oz. zaključek programa.

```
//Odpri datoteko
private void odpriToolStripMenuItem_Click(object sender, EventArgs e)
{
    ShraniSpremembe("Shranjevanje");
    besedilo.Clear();
    toolStripStatusLabel1.Text = nova;
    //datoteko bo poiskal oz. vpisal ime uporabnik
    openFileDialog1.FileName = "";
    try //odpiranje nove datoteke
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            string myString = File.ReadAllText(openFileDialog1.FileName);
            besedilo.Text = myString;
            toolStripStatusLabel1.Text = openFileDialog1.FileName;
        }
    }
    catch {  }
}
//Shrani datoteko
private void shraniToolStripMenuItem_Click(object sender, EventArgs e)
{
    ShraniSpremembe("Shranjevanje");
}
//Shrani kot
private void toolStripMenuItem3_Click(object sender, EventArgs e)
{
    VDatoteko(); //shranjevanje v datoteko
}
```

```
//Izhod
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    ShraniSpremembe("Izhod");
    Application.Exit();//Zaključek programa
}
```

Urejevalnik bo vseboval tudi nekatere možnosti, ki jih vsebuje večina urejevalnikov (razveljavi, izreži, kopiraj, prilepi). Pri kopiranju in lepljenju si pomagamo z metodami razreda *ClipBoard*, ki nam pomagajo pri pridobivanju in shranjevanju podatkov na odložišče sistema *Windows* (začasno odlagališče podatkov, kamor sistem *Windows* shranjuje vse, kar smo označili in za to izbrali ukaz *Kopiraj* ali *Izreži*).

```
//Undo
private void razveljavi_Click(object sender, EventArgs e)
{
    /*preverim, če je razveljavitev možna: v tem primeru, ima lastnost
        CanUndo gradnika TextBox vrednost True*/
    if (besedilo.CanUndo == true)
        besedilo.Undo();
}
//Izreži
private void izrezi_Click(object sender, EventArgs e)
{
    Clipboard.Clear();//brisanje vsebine odložišča
    /*če izbrano besedilo vsebuje vsaj en znak (lastnost SelectionLength
    je večja od 0), ga shranim v odložišče*/
    if (besedilo.SelectionLength > 0)
        //lastnost SelectedText označuje izbrano besedilo gradnika TextBox
        Clipboard.SetText(besedilo.SelectedText);
    besedilo.SelectedText = "";
}
//Kopiraj
private void kopiraj_Click(object sender, EventArgs e)
{
    Clipboard.Clear(); //čiščenje odložišča
    if (besedilo.SelectionLength > 0)//če je izbran poljubnen tekst
        Clipboard.SetText(besedilo.SelectedText); //ga kopiramo v odložišče
}
//Prilepi
private void prilepi_Click(object sender, EventArgs e)
{
    /*po lepljenju želimo postaviti kazalnik miške na konec vstavljenega
        Besedila. Pri tem si pomagamo z lastnostma SelectionStart (indeks
        začetnega znaka označenega besedila) in SelectionLength (število
znakov
        označenega besedila)*/

    int trenutni = besedilo.SelectionStart+besedilo.SelectionLength;
```

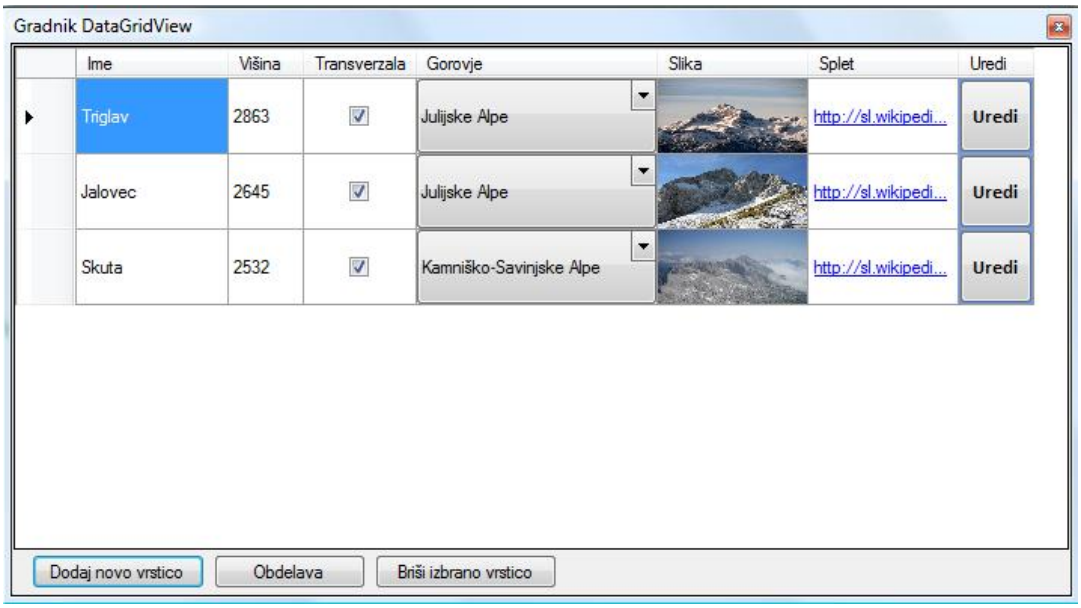
```
    besedilo.Text=besedilo.Text.Insert(besedilo.SelectionStart,
Clipboard.GetText());
    besedilo.SelectionStart = trenutni;
}
//Izberi vse
private void izberiVse_Click(object sender, EventArgs e)
{
    besedilo.SelectAll();//označim celoten tekst
}
//Pisava
private void pisava_Click(object sender, EventArgs e)
{
    fontDialog1.ShowDialog();//dialog za izbiro pisave
    besedilo.Font = fontDialog1.Font;//uporabimo izbrani font
}
//Barva pisave
private void barvaPisave_Click(object sender, EventArgs e)
{
    colorDialog1.ShowDialog();//dialog za barve
    besedilo.ForeColor = colorDialog1.Color;//uporabimo izbrano barvo
}
//Leva poravnava
private void levaPoravnava_Click(object sender, EventArgs e)
{
    besedilo.TextAlign = HorizontalAlignment.Left;
}
//Sredinska poravnava
private void sredinskaPoravnava_Click(object sender, EventArgs e)
{
    besedilo.TextAlign = HorizontalAlignment.Center;
}
//Desna poravnava
private void desnaPoravnava_Click(object sender, EventArgs e)
{
    besedilo.TextAlign = HorizontalAlignment.Right;
}
//Nastavitev ozadja
private void ozadje_Click(object sender, EventArgs e)
{
    colorDialog1.ShowDialog();
    besedilo.BackColor = colorDialog1.Color;
}
//prikaz oz. skrivanje statusne vrstice
private void statusnaVrsticaToolStripMenuItem_Click(object sender, EventArgs
e)
{
    if (statusnaVrsticaToolStripMenuItem.Checked)
    {
        statusStrip1.Visible = true;
        statusnaVrsticaToolStripMenuItem.Checked = false;
```

```
    }
    else
    {
        statusStrip1.Visible = false;
        statusnaVrsticaToolStripMenuItem.Checked=true;
    }
}
```

Gradnik DataGridView

Gradnik *DataGridView* je namenjen tabelaričnemu prikazu podatkov. Vsebuje stolpce in vrstice, podatke vanj pa lahko vnesemo programsko, ali pa ga preko lastnosti *DataSource* povežemo z nekim izvorom podatkov. O povezovanju z izvorom podatkov bomo zapisali več kasneje, na tem mestu pa poglejmo, kako naredimo stolpce, ter kako podatke v ta gradnik dodajamo, jih ažuriramo in brišemo programsko.

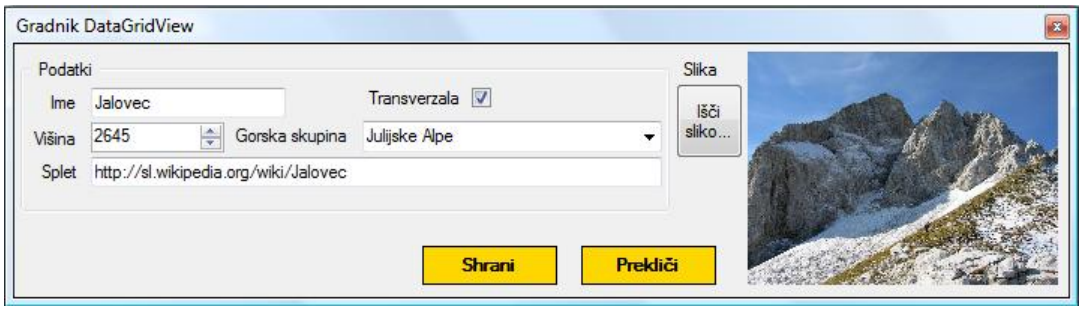
Gradnik *DataGridView*, njegove lastnosti in metode bomo najhitreje spoznali kar na konkretnem primeru. Sestavili bomo tabelarični prikaz podatkov in slik o nekaterih naših gorah.



Slika 52: Prikaz začetnega obrazca projekta *DataGridViewDemo*.

Naš program bo omogočal tudi dodajanje novih podatkov (vrstic), ter brisanje že obstoječih. Ob kliku na gumb *Obdelava* pa bomo v sporočilnem oknu izpisali skupno število vrhov, ki so v transverzali, ter kolikšna je povprečna višina vseh gora v tabeli.

Ob kliku na gumb na desni strani vsake vrstice, se bo tabela s podatki začasno skrila, prikazala pa se bo plošča z gradniki za ažuriranje podatkov izbrane vrstice, tako kot kaže slika.



Slika 53: Prikaz obrazca v primeru ažuriranja podatkov.

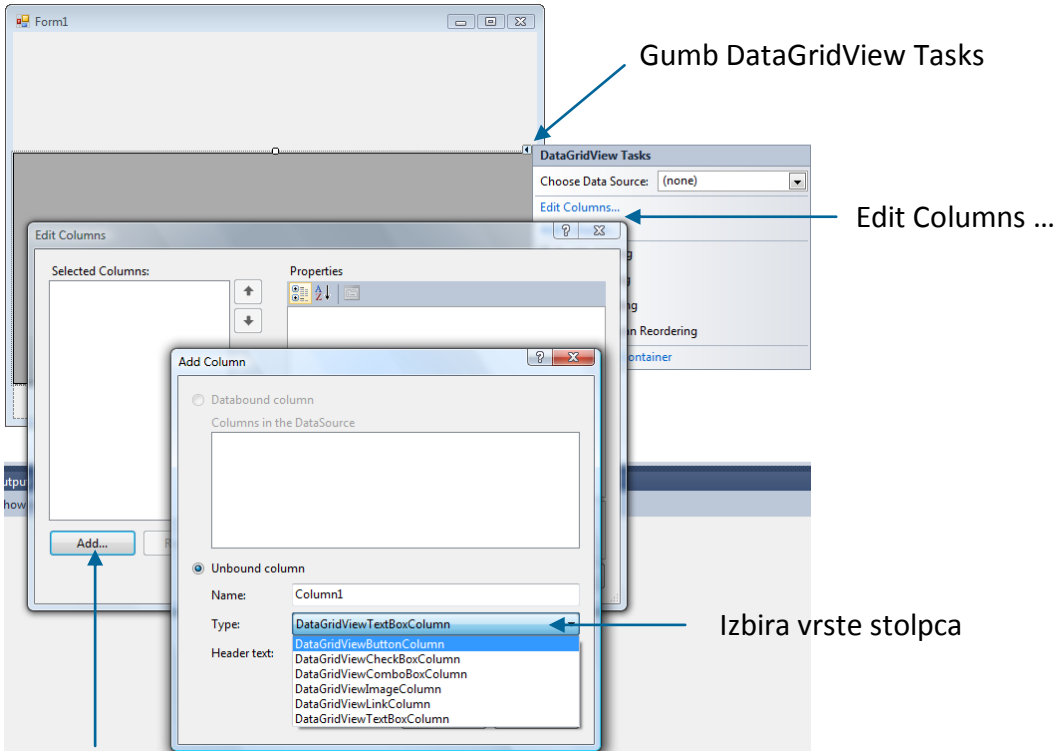
Ker v našem primeru gradnika *DataGridView* ne bomo povezovali z izvorom podatkov, bomo imena stolpcev, v katerih bomo prikazovali podatke, ustvarili že pri načrtovanju aplikacije. Na obrazec najprej postavimo gradnik *Panel* in mu nastavimo lastnost *Dock* na *Bottom*. Nanj dodajmo dva gumba z napisoma *Dodaj novo vrstico* in *Urejanje*.

Nad ploščo postavimo še gradnik *DataGridView*. Ta se nahaja se na paleti *Data*. Da bo poravnan z dnom plošče, mu lastnost *Dock* nastavimo na *Bottom*. Nato pa kliknemo gumb *DataGridView Tasks* v zgornjem desnem kotu gradnika. V oknu, ki se odpre, izberemo *EditColumns*, nato pa s klikom na gumb *Add* dodajamo stolpce.

Polje *Name* predstavlja ime stolpca znotraj našega programa, lastnost *HeaderText* pa napis na vrhu stolpca. Izbrati moramo še vrsto stolpca (*Type*):

- *DataGridViewButtonColumn*: v celici bo prikazan gumb;
- *DataGridViewCheckBoxColumn*: v celici bo prikazan gradnik *CheckBox*;
- *DataGridViewComboBoxColumn*: v celici bo prikazan gradnik *ComboBox*;
- *DataGridViewImageColumn*: v celici bo prikazana slika;
- *DataGridViewLinkColumn*: v celici bo prikazana povezava (*Link*) do spletne strani ali do poljubne aplikacije;
- *DataGridViewTextBoxColumn*: celica bo namenjena za hranjenje oz vnos besedila.

Za vajo ustvarimo 7 stolpcev: *Ime* in *Visina* (*DataGridViewTextBoxColumn)*, *Transverzala* (*DataGridViewCheckBoxColumn*), *Gorovje* (*DataGridViewComboBoxColumn*), *Slika* (*DataGridViewImageColumn*), *Splet* (*DataGridViewLinkColumn*) in *Urejanje* (*DataGridViewButtonColumn*).



Slika 54: Ustvarjanje imen stolpcev gradnika *DataGridView*.

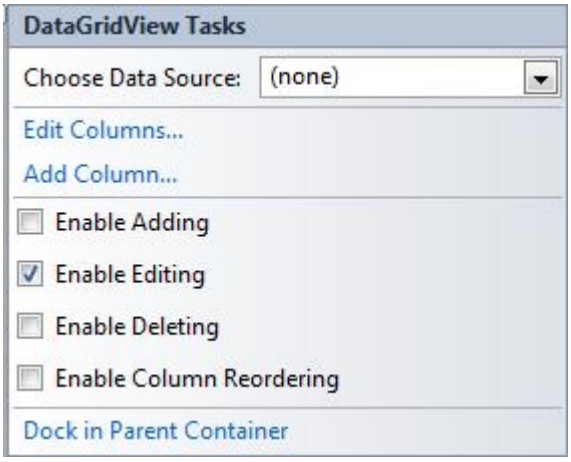
Tako ustvarjenim stolpcem določimo še nekaj osnovnih lastnosti. Odpremo urejevalnik stolpcev (ikona *DataGridView Tasks* v zgornjem desnem delu gradnika *DataGridView→Edit Columns*), nato izberemo ustrezen stolpec in določimo naslednje lastnosti:

Ime stolpca	Lastnost	Nastavitev	Opis
Ime	DefaultCellStyle→ Alignment	MiddleLeft	Vsebina celice bo poravnana levo
Visina	DefaultCellStyle→ Alignment	MiddleRight	Vsebina celice bo poravnana desno
	Width	50	Širina stolpca
Transverzala	DefaultCellStyle→ Alignment	MiddleCenter	Sredinska poravnava vsebine celice
Transverzala	Width	75	Širina stolpca
Gorovje	DefaultCellStyle→ Alignment	MiddleLeft	MiddleCenter

	Items	Jugozahodni del Julijske Alpe Kamniško-Savinjske Alpe Karavanke Pohorje in severovzhodni del	Vsebina gradnika
	Width	160	Širina stolpca
Slika	ImageLayout	Stretch	V celici bo prikazana celotna slika
	DefaultCellStyle→ Alignment	MiddleCenter	Vsebina celice bo poravnana sredinsko
Uredi	DefaultCellStyle→ Alignment	MiddleCenter	Vsebina celice bo poravnana sredinsko
	Text	Uredi	Na gumbu bo napis Uredi
	UseColumnTextForButtonValue	True	Napis na gumbu bo tak kot je zapisan v lastnosti Text

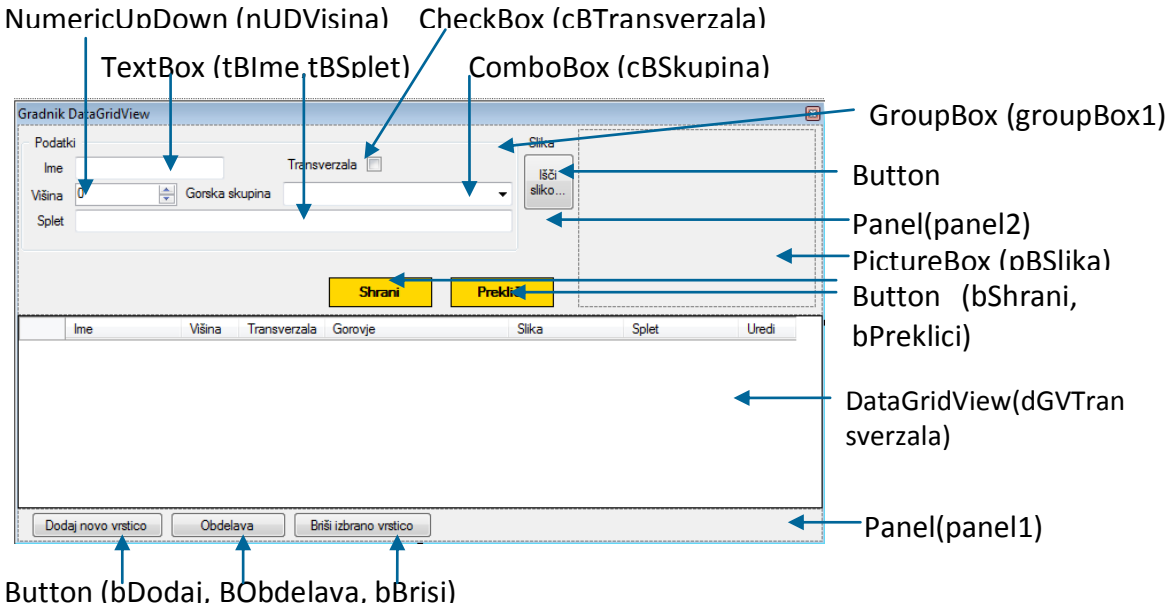
Tabela 17: Lastnosti stolpcev gradnika *DataGridView*.

Urejevalnik stolpcev nato zapremo, v oknu *DataGridVew Tasks* pa odkljukajmo le opcijo *Enable Editing* (dovoljeno urejanje neposredno v posamezni celici). Ostale nastavitve pomenijo, da ni dovoljeno dodajanje vrstice neposredno v gradniku *DataGridView*, prav tako pa ni dovoljeno neposredno brisanje vrstic.



Slika 55: Nastavitev v oknu *DataGridView Tasks*.

Nad gradnik *DatagridView* postavimo gradnik *Panel* (*Dock=Fill*), nanj pa še nekaj gradnikov, ki jih že poznamo. Namenjeni bodo urejanju in dodajanju novih postavk. Projekt bomo zastavili tako, da bo uporabnik na začetku videl le postavke v gradniku *DataGridView* (zgornji panel z gradniki za urejanje bo skrit).



Slika 56: Gradniki projekta *DataGridViewDemo*.

V mapo *Bin→Debug* našega projekta skopirajmo tri slike naših vrhov, nato pa v konstruktorju obrazca poskrbimo, da bo na začetku v gradniku že nekaj vrstic podatkov. Stavki, zapisani v konstruktorju, se bodo namreč izvedli ob ustvarjanju obrazca, preden se bo ta prvič prikazal. Posamezno vrstico dodamo s pomočjo metode *Rows.Add()*, ki ima toliko parametrov, kot je stolpcev v tabeli.

```
//dodajanje nove vrstice v gradnik DataGridView
dGVTransverzala.Rows.Add("Triglav", 2863, true, "Julijske Alpe", slika,
"http://sl.wikipedia.org/wiki/Triglav");
```

Če se bo uporabnik odločil za urejanje oz. dodajanje novih zapisov, bo viden le panel z gradniki za vnos oz. ažuriranje podatkov izbrane vrstice gradnika *DataGridView*. Skrivanje in prikazovanje panela ter gradnika *DataGridView* najlaže dosežemo z metodama *BringToFront*() in *SendToBack*() teh dveh gradnikov (a le v primeru, da smo gradnikoma nastavili lastnost *Dock*). Dodajmo še osnovne metode za delo z gradnikom *DataGridView* (dodajanje, dostop do izbrane vrstice in celice, obdelava vseh vrstic, brisanje). Novo vrstico dodamo s pomočjo metode *Rows.Add()*, vsebino izbane celice tekoče vrstice pa dobimo preko objekta *CurrentRow.Cells[ime].Value* (*ime* je tukaj ime stolpca, namesto imena stolpca pa lahko zapišemo tudi njegov indeks).

Razložimo še dogodka, s katerima lahko kontroliramo uporabnikove vnose in se tem zavarujemo pred napačnimi vnosi podatkov. To sta dogodka *CellValidating* in *CellEndEdit*.

Namen metode *CellValidating* je zagotoviti, da bo uporabnik v določeno polje vnesel neko veljavno vrednost, npr. nenegativno celo število. Drugi parameter te metode, parameter *e*, je objekt tipa *DataGridViewCellValidatingEventArgs*. Ta objekt pozna tudi lastnost, s katero lahko ugotovimo, katera celica je bila ažurirana: *e.ColumnIndex* vsebuje številko stolpca, *e.RowIndex* pa številko vrstice. Pri tem upoštevamo, da imata prva vrstica in prvi stolpec indeks enak 0.

Pri obdelavi dogodka *CellValidating* lahko uporabimo metodo *int.TryParse*, ki je zelo koristna v primerih preverjanja, ali lahko nek niz pretvorimo v celo število. Metoda vrne *True*, če je pretvarjanje uspešno, obenem pa v svoj drugi parameter (v našem primeru *zacasna*), ki je klican po referenci, zapiše pretvorjeno vrednost. Če je pretvarjanje neuspešno (uporabnik je vnesel npr. niz, ki vsebuje črke, ali pa je vnesena vrednost negativna), se v lastnost *ErrorText* trenutne vrstice v tabeli zapiše ustrezno obvestilo o napaki. Na začetku te vrstice se prikaže ikona s klicajem. Če na ikono postavimo kazalnik miške, se pod njo izpiše obvestilo o napaki (v našem primeru tekst "*Vrednost mora biti nenegativno število*"). Z naslednjim stavkom (*e.Cancel = true;*) pa poskrbimo, da se uporabnik nikakor ne more premakniti v drugo celico vse dotlej, dokler v celico ne vnese pravilnega podatka, ali pa celoten vnos prekliče s tipko *Esc*.

Za gradnik *DataGridView* zapišimo še odzivno metodo za dogodek *CellEndEdit*. Ta metoda se izvede, ko je uporabnikov vnos veljaven in se uporabnik premakne v drugo celico. Koda je naslednja:

```
dGVTransverzala.Rows[e.RowIndex].ErrorText = "";
```

S tem stavkom enostavno pobrišemo vsa obvestila o napaki zaradi nepravilnega vnosa podatkov v trenutno izbrano celico.

Pojasnimo še namen dogodka *DataError* gradnika *DataGridView*. Ta dogodek zajame prav vse napake, ki nastanejo pri preverjanju uporabnikovega vnosa v katerokoli celico (predvsem velja to za celice, katerih vneseno vsebino nismo preverjali preko imena celice. Vsebina ustrezne metode poskrbi za splošno obvestilo uporabniku, obenem pa prepreči, da bi se uporabnik kljub napačnemu vnosu premaknil iz trenutne celice.

```
public partial class Form1 : Form
{
    /*spremenljivka urejanje določa, ali je gradnik DataGridView v fazi
      urejanja (true) ali dodajanja (false)*/
    bool urejanje;
    public Form1() //konstruktor obrazca Form1
    {
        InitializeComponent();
        //na začetku je panel skrit
        panel2.Hide();
        //DataGridView razširimo čez celoten obrazec
```

```
        dGVTransverzala.Dock = DockStyle.Fill;

        //določimo višino posameznih vrstic (zaradi slik)
        dGVTransverzala.RowTemplate.Height = 50;
        //v DataGridView z metodo Items.Add dodamo tri vrstice
        Image slika = Image.FromFile("Triglav.JPG");
        dGVTransverzala.Rows.Add("Triglav", 2863, true, "Julijske Alpe",
slika, "http://sl.wikipedia.org/wiki/Triglav");
        slika = Image.FromFile("Jalovec.JPG");
        dGVTransverzala.Rows.Add("Jalovec", 2645, true, "Julijske Alpe",
slika, "http://sl.wikipedia.org/wiki/Jalovec");
        slika = Image.FromFile("Skuta.JPG");
        dGVTransverzala.Rows.Add("Skuta", 2532, true, "Kamniško-Savinjske
Alpe", slika, "http://sl.wikipedia.org/wiki/Skuta");
    }
    /*ob kliku na celico v gradniku DataGridView se najprej zgodi dogodek
      CellClick, za njim pa še dogodek CellContentClick. Samo za informacijo:
      dogodek CellClick se zgodi pri vsakem kliku v notranjost celice,
      CellContentClick pa le, če kliknemo na neko vsebino v celici */
    private void dGVTransverzala_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        urejanje = true;
        if (e.ColumnIndex == this.dGVTransverzala.Columns["Uredi"].Index)
        {
            //vsebino posameznih celic zapišemo v ustrezne gradnike
            tBime.Text =
dGVTransverzala.CurrentRow.Cells["Ime"].Value.ToString();
            nUDVisina.Value =
Convert.ToInt32(dGVTransverzala.CurrentRow.Cells["Visina"].Value);
            cBTransverzala.Checked =
Convert.ToBoolean(dGVTransverzala.CurrentRow.Cells["Transverzala"].Value);
            Image slika = (Image)
(dGVTransverzala.CurrentRow.Cells["Slika"]).Value;
            pBSlika.Image = slika;
            tBSplet.Text =
dGVTransverzala.CurrentRow.Cells["Splet"].Value.ToString();
            cBSkupina.Text =
dGVTransverzala.CurrentRow.Cells["Gorovje"].Value.ToString();
            //prikažemo panel in s tem skrijemo DataGridView
            panel2.Show();
            panel1.Visible = false;
            this.Height = 190;
        }
    }
    private void bShrani_Click(object sender, EventArgs e)
    {
        if (urejanje)
        {
            try
```

```
            {
                /*vrednosti gradnikov na panelu Panel2 prenesemo v
                  vrstico gradnika DataGridView*/
                dGVTransverzala.CurrentRow.Cells["Ime"].Value = tBime.Text;
                dGVTransverzala.CurrentRow.Cells["Visina"].Value =
                   nUDVisina.Value;
                dGVTransverzala.CurrentRow.Cells["Transverzala"].Value =
                   cBTransverzala.Checked;
                dGVTransverzala.CurrentRow.Cells["Splet"].Value =
                   tBSplet.Text;
                dGVTransverzala.CurrentRow.Cells["Gorovje"].Value =
                   cBSkupina.Text;
                dGVTransverzala.CurrentRow.Cells["Slika"].Value =
                  pBSlika.Image;
            }
            catch
            {  MessageBox.Show("Napaka pri shranjevanju!"); }
            finally
            {
                /*po urejanju skrijemo Panel2*/
                panel2.Hide();
                //prikaz panela z gumboma za urejanje in dodajanje
                panel1.Visible = true;
                urejanje = false;//oznaka, da je urejanje končano
            }
        }
        else
        {
            /*najprej preverimo, če je uporabnik vnesel vse podatke
            nato pa jih shranimo v gradnik DataGridView*/
            dGVTransverzala.Rows.Add(tBime.Text, nUDVisina.Value,
cBTransverzala.CheckState, cBSkupina.Text, pBSlika.Image, tBSplet.Text);
            panel1.Visible = true;
            urejanje = false;
            /*skrijemo Panel2*/
            panel2.Hide();
        }
        this.Height = 400;
    }
    private void bPreklici_Click(object sender, EventArgs e)
    {
        /*po urejanju skrijemo Panel2*/
        panel2.Hide();
        //prikažemo panel z gumboma za urejanje in dodajanje
        panel1.Visible = true;
        urejanje = false;
        this.Height = 400;
    }
    private void bIsciSliko_Click(object sender, EventArgs e)
    {
```

```
        try
        {
            //ustvarimo nov objekt tipa OpenFileDialog
            OpenFileDialog OPF = new OpenFileDialog();
            //v oknu bodo prikazane le slikovne datoteke
            OPF.Filter = "Slike|*.jpg;*.gif;*.bmp;*.pmg";
            //okno zapremo s sklikom na gumb Open
            if (OPF.ShowDialog() == DialogResult.OK)
            {
                //izbrano sliko prenesemo v gradnik
                Image slika = Image.FromFile(OPF.FileName);
                pBSlika.Image = slika;
            }
        }
        catch
        { MessageBox.Show("Napaka!"); }
    }
    private void dGVTransverzala_CellContentClick(object sender,
DataGridViewCellEventArgs e)
    {
        //preverim, če je bila kliknjena prava celica
        if (e.ColumnIndex == this.dGVTransverzala.Columns["Splet"].Index)
            System.Diagnostics.Process.Start("IExplore",
dGVTransverzala.CurrentRow.Cells["Splet"].Value.ToString());
    }
    private void bDodaj_Click(object sender, EventArgs e)
    {
        panel2.Show();
        panel1.Visible = false;
        //pobrišemo dosedanje vrednosti gradnikov
        tBime.Clear(); nUDVisina.Value = 0; tBSplet.Clear();
        cBSkupina.Text = "";
        cBTransverzala.Checked = false;
        pBSlika.Image = null;
        urejanje = false;
        this.Height = 190;
    }
    //odzivna metoda gumba za obdelavo tabele
    private void button1_Click(object sender, EventArgs e)
    {
        //ugotovimo, koliko je skupno število vrhov, ki so v transverzali
        int skupaj = 0;
        //ugotovimo še, kolikšna je povprečna višina vseh gora
        int visina = 0;
        for (int i = 0; i < dGVTransverzala.Rows.Count; i++)
        {
            if
(Convert.ToBoolean(dGVTransverzala.Rows[i].Cells["Transverzala"].Value))
                skupaj++;
            //prištevamo vrednost v celici Visina
```

```
visina=visina+Convert.ToInt32(dGVTransverzala.Rows[i].Cells["Visina"].Value);
        }
        MessageBox.Show("Skupaj gora v transverzali: "+skupaj+"\nPovprečna
višina vseh gora: "+Math.Round((double)visina/dGVTransverzala.Rows.Count,2)+"
m");
    }
    private void dGVTransverzala_CellValidating(object sender,
DataGridViewCellValidatingEventArgs e)
    {
        int zacasna;
        dGVTransverzala.Rows[e.RowIndex].ErrorText = "";
        /*če smo naredili spremembe v stolpcu Visina, je potrebna
          validacija*/
        if ((dGVTransverzala.Columns[e.ColumnIndex].Name == "Visina"))
        {
            /*metoda TryParse pretvarja niz v celo število. Ima dva
parametra: prvi je tipa niz, drugi pa celo število(klic po referenci). Če
pretvorba uspe, metoda vrne true, rezultat pretvorbe pa je shranjen v drugem
parametru*/
            if (!int.TryParse(e.FormattedValue.ToString(), out zacasna) ||
zacasna < 0)
            {
                /*v primeru napake oblikujemo ustrezno sporočilo: potrebno
                  pa je napisati dogodek CellEndEdit*/
                dGVTransverzala.Rows[e.RowIndex].ErrorText = "Vrednost mora
biti nenegativno število";
                /*uporabnik se nikakor ne more premakniti v drugo celico
                  vse dotlej, dokler v celico ne vnese veljavnega podatka*/
                e.Cancel = true;
            }
        }
    }

    private void dGVTransverzala_CellEndEdit(object sender,
DataGridViewCellEventArgs e)
    {
        //ko je napaka odpravljena izbrišemo sporočilo o napaki
        dGVTransverzala.Rows[e.RowIndex].ErrorText = "";
    }
    //odziva metoda gumba za brisanje vrstice gradnika ataGridView
    private void button2_Click(object sender, EventArgs e)
    {
        if (MessageBox.Show("Brišem izbrano
vrstico?","Brisanje?",MessageBoxButtons.OKCancel,MessageBoxIcon.Question)==Di
alogResult.OK)
        {
            DataGridViewRow row = dGVTransverzala.CurrentRow;
            dGVTransverzala.Rows.Remove(row);
        }
```

```
    }
}
```

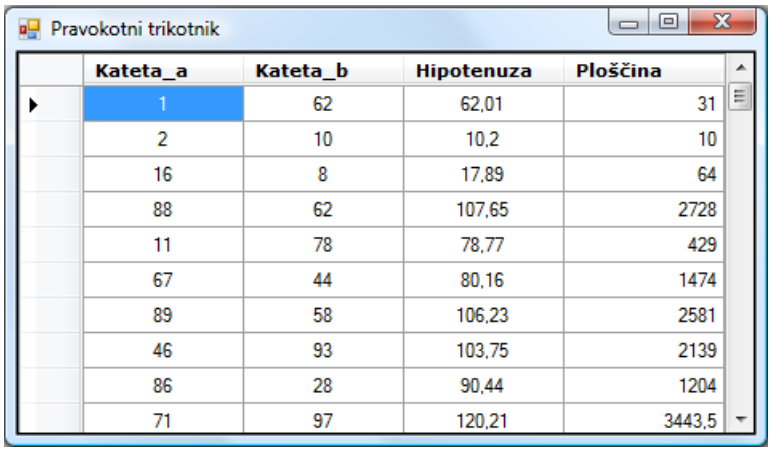
Pokažimo še, kako lahko gradnik *DataGridView* povežemo z nekim izvorom podatkov. O povezavi z neko tabelo iz baze bomo več napisali pri bazah podatkov, na tem mestu pa bomo pokazali povezavo z neko tipizirano zbirko.

V matematiko pogosto delamo s pravokotnimi trikotniki. Sestavimo razred *PravokotniTrikotnik* z dvema poljema (kateti pravokotnega trikotnika), metodo za izračun ploščine trikotnika, ter lastnostmi za izračun hipotenuze in za nastavljanje vrednosti katet.

```
public class PravokotniTrikotnik
{
    double a, b;  //kateti sta zasebni polji
    public PravokotniTrikotnik(double a, double b) //konstruktor
    {
        this.Kateta_a = a;
        this.Kateta_b = b;
    }
    //Lastnosti
    public double Kateta_a //lastnost za prvo kateto
    {
        get { return a;}
        set
        {   //če bomo za a skušali nastaviti negativno vrednost, ostane a=0
            if (value > 0) a = value;
        }
    }
    public double Kateta_b //lastnost za drugo kateto
    {
        get { return b; }
        set
        {   //če bomo za b skušali nastaviti negativno vrednost, ostane b=0
            if (value >= 0) b = value;
        }
    }
    public double Hipotenuza //lastnost za hipotenuzo
    {
        get { return Math.Round(Math.Sqrt(a * a + b * b), 2);}
    }

    public double Ploščina //lastnost za ploščino pravokotnega trikotnika
    {
        get { return a * b / 2.0;}
    }
}
```

V konstruktorju obrazca ustvarimo 100 objektov tipa *PravokotniTrikotnik* (kateti naj bosta naključni celi števili med 1 in 100) in objekte dodajmo v tipizirano zbirko *trikotniki*. Zbirko nato povežimo z gradnikom *DataGridView* (ime gradnika je *DGV*), ki ga pred tem postavimo na prazen obrazec.

```
//Deklaracija tipizirane zbirke pravokotnih trikotnikov
List<PravokotniTrikotnik> trikotniki = new List<PravokotniTrikotnik>();
//generator naključnih števil
Random naklj = new Random();

//Konstruktor obrazca
public Form1()
{
    InitializeComponent();
    /*v zanki ustvarimo 100 objektov tipa PravokotniTrikotnik in jih dodamo
     v zbirko*/
    for (int i = 0; i < 100; i++)
    {
        //kateti naj bosta naključni celi števili med 1 in 100
        int katetaA=naklj.Next(1,101);
        int katetaB=naklj.Next(1,101);
        PravokotniTrikotnik p = new PravokotniTrikotnik(katetaA, katetaB);
        //objekt p dodamo v zbirko trikotniki
        trikotniki.Add(p);
    }
    /*objekt DataGridView (ime gradnika je DGV in je že na obrazcu),
      povežemo z tipizirano zbirko Trikotniki*/
    DGV.DataSource = trikotniki;
    //nekatere oblikovne lastnosti gradnika DGV lahko nastavimo programsko
    DGV.Columns[0].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
    DGV.Columns[1].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
    DGV.Columns[2].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
    DGV.Columns[3].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
    //oblikujmo font za glave stolpcev
    DataGridViewCellStyle columnHeaderStyle = new DataGridViewCellStyle();
    columnHeaderStyle.ForeColor = Color.Red;
    columnHeaderStyle.Font = new Font("Verdana", 8, FontStyle.Bold);
    DGV.ColumnHeadersDefaultCellStyle = columnHeaderStyle;
    //določimo še širino obrazca
    this.Width = DGV.Columns[0].Width * 5;
    DGV.ReadOnly = true;//urejanje ni možno
}
```



Kateta_a	Kateta_b	Hipotenuza	Ploščina
1	62	62,01	31
2	10	10,2	10
16	8	17,89	64
88	62	107,65	2728
11	78	78,77	429
67	44	80,16	1474
89	58	106,23	2581
46	93	103,75	2139
86	28	90,44	1204
71	97	120,21	3443,5

Slika 57: Prikaz vsebine tipizirane zbirke v gradniku *DataGridView*.

Stolpci gradnika *DGV* se pri povezavi ustvarijo avtomatično, njihova imena pa so enaka lastnostim razreda *PravokotniTrikotnik*. V zgornjem primeru so vsi stolpci *ReadOnly*, kar pomeni, da uporabnik vrednosti celic neposredno ne more spreminjati. S tem se izognemo napakam pri spreminjanju podatkov o obeh katetah (stolpca *Hipotenuza* in *Ploščina* pa tako in tako ne moremo spreminjati, ker predstavljata lastnosti).

Dogodki tipkovnice, miške in ure

Dogodki tipkovnice

Obrazci in večina gradnikov v okolju Windows obravnavajo vnose preko tipkovnice z obdelavo dogodkov ki jih proži tipkovnica. V oknu *Properties→Events* teh gradnikov obstajata dva dogodka, ki se zgodita, ko uporabnik pritisne tipko na tipkovnici in en dogodek ko spusti tipko na tipkovnici. Dogodki tipkovnice so predstavljeni in razloženi v naslednji tabeli:

Dogodek Tipkovnice	R a z l a g a
KeyDown	Dogodek, ki se izvede ob pritisku na katerokoli tipko. Zgodi se samo enkrat.
KeyPress	Dogodek se zgodi ob pritisku na tipko, ki predstavlja nek znak iz množice vseh znakov. Če uporabnik tipko drži pritisnjeno, se zgodi večkrat.

KeyUp	Dogodek se zgodi, ko uporabnik spusti tipko.

Tabela 18: Dogodki tipkovnice.

Ko uporabnik pritisne tipko, se najprej preverja, kateri dogodek se bo izvedel. To pa je odvisno od tega, ali je pritisnjena tipka (ali pa kombinacija tipk) nek ASCII znak (črke, številke, *Enter*, *BkSp, Shift A*, *AltGr W*…), ali pa gre za kako drugo tipko na tipkovnici (npr. *Tab*, *CapsLock*, *Shift, Ctrl, Insert, Delete, PgUp, PgDn* …). Če kombinacija tipk, ki jih uporabnik pritisne, ustreza nekemu znaku, se najprej zgodi dogodek *KeyPress*, za njim pa še dogodek *KeyDown*. Če ne gre za znakovno tipko (npr. tipka *Shift*), se zgodi le dogodek *KeyDown*.

Vrstni red dogodkov, ki jih proži tipkovnica je torej naslednji:

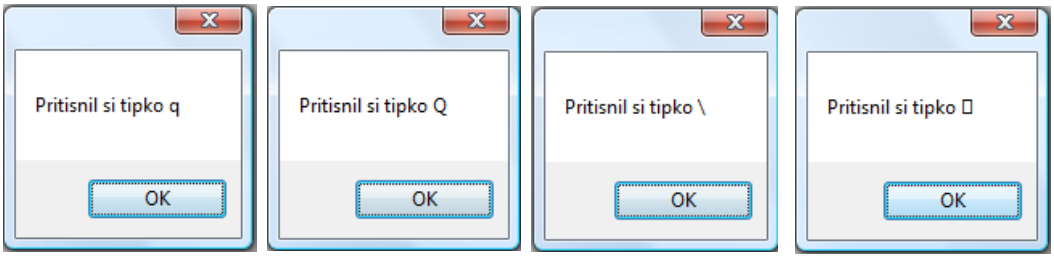
- *KeyPress* – izvede se ob pritisku na tipko, ki predstavlja nek znak iz množice vseh znakov. Dogodek se bo sprožil če npr. uporabnik pritisne tipko *'Q'*, če uporabnik pritisne kombinacijo tipk *Shift + 'Q'*, če npr. uporabnik pritisne kombinacijo tipk *AltGr + 'Q'*, če pritisne tipko *Ctrl+'Q'* … Vsaka od napisanih kombinacij namreč predstavlja nek znak, ki ga dobimo s pomočjo tipkovnice.
- *KeyDown* – izvede se ob pritisku na katerokoli tipko.
- *KeyUp* – izvede se, ko uporabnik spusti katerokoli tipko.

Za vsako tipko na tipkovnici lahko preko drugega parametra odzivnega dogodka tipkovnice dobimo vse podatke o tej tipki ali pa kombinaciji tipk:

- Dogodek *KeyPress* ima drug parameter tipa *KeyPressEventArgs*. Objekt tega razreda nam pove, katero tipko ali pa katero kombinacijo tipk je pritisnil uporabnik. Oznako tipke, ali pa kombinacijo tipk lahko izpišemo v sporočilnem oknu s pomočjo lastnosti KeyChar: ta predstavlja znak, ki ustreza pritisnejeni tipki.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{     MessageBox.Show("Pritisnil si tipko " + e.KeyChar);}
```

Spodnja slika prikazujejo sporočilna okna, ko smo zaporedoma pristisnili tipko *'Q'*, kombinacijo tipk *Shift + 'Q'*, kombinacijo tipk *AltGr + 'Q'* in kombinacijo *Ctrl+'Q'*.



Slika 58: Sporočilna okna s prikazom različnih kombinacij pritiska tipke Q.

- Dogodek *KeyDown* ima drug parameter tipa *KeyEventArgs*. To je razred, ki vsebuje nekaj zelo koristnih metod, s katerimi lahko kontroliramo uporabnikov pritisk na katerokoli tipko na tipkovnici. Hrani tudi podatka o imenu tipke in o kodi ASCII tega znaka. Podatka dobimo s pomočjo lastnosti *KeyCode* in *KeyValue*. Njuni vrednosti lahko npr. izpišemo v sporočilnem oknu:

```
Private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    MessageBox.Show("Pritisnil si tipko "+ e.KeyCode + "- ASCII: = "
    + e.KeyValue);
}
```

  S pomočjo drugega parametra in lastnosti *Handled*, lahko dogodek predčasno zaključimo in na ta način kontroliramo uporabnikove vnose preko tipkovnice. S tem stavkom enostavno povemo, da so se vsi dogodki, povezani s to tipko, že zaključili, zato se pritisnjena tipka v gradniku sploh ne prikaže.

```
e.Handled = true; /dogodek je zaključen, vsi nadaljnji stavki v telesu
dogodka se ne bodo izvedli*/
```

- Dogodek *KeyUp* ima drug parameter tipa *KeyEventArgs*. Z njegovo pomočjo dobimo podatek, katera tipka je bila spuščena. Z lastnostjo *Handled* lahko tudi v tem dogodku letega predčasno zaključimo, ali ga celo preprečimo.

```
private void tbStarost_KeyUp(object sender, KeyEventArgs e)
{
    MessageBox.Show("Spustil si tipko " + e.KeyCode);
}
```

Opisane dogodke pogostokrat uporabljamo za kontrolo uporabnikovih vnosov podatkov. Tako lahko npr. uporabniku preprečimo vnos znakov, ki niso številke, v tisto vnosno polje, ki naj bi predstavljalo nek znesek, ali pa vnos numeričnih podatkov v vnosno polje, ki naj bi predstavljalo zaporedje črk abecede, ipd.

Kadar torej želimo specifične vnose, ali pa uporabniku dovoliti uporabo le določenih tipk, oz. mu preprečiti nezaželene vnose, lahko to storimo s pomočjo dogodka *KeyPress*! Tule je nekaj primerov odzivnih dogodkov vrste *KeyPress*, ko uporabnik vnaša podatke v gradnik tipa *TextBox*.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    //Uporabniku dovolimo le vnos znakov angleške abecede:
    if ((e.KeyChar < 'A') || (e.KeyChar > 'Z')) e.Handled = true;
}
```

Zopet smo uporabili drugi parameter odzivne metode in lastnost Handled: vsi dogodki, povezani s to tipko so zaključeni. Celotni zgornji stavek moramo torej razumeti takole: če pritisnjena tipka ni velika črka med A in Z, potem velja, kot da nismo pritisnili nobene tipke.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    //Uporabniku dovolimo le vnos znakov D ali N:
    if ((e.KeyChar != 'D') && (e.KeyChar != 'N')) e.Handled = true;
}
```

Če torej nismo pritisnili velike črke D oziroma velike črke N, potem velja, kot da nismo pritisnili nobene tipke.

Pa še primer kode odzivnega dogodka, s katerim uporabniku dovolimo le vnos števk in decimalne vejice, dovolimo pa mu tudi brisanje že napisanih znakov (torej uporabo tipke *BackSpace*).

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    /*Uporabniku dovolimo le vnos števk in decimalne vejice, dovolimo pa mu
      tudi brisanje že napisanih znakov ( (char)8 je tipa BackSpace)*/
    if (((e.KeyChar < '0') || (e.KeyChar > '9'))
        && (e.KeyChar != ',') && (e.KeyChar != (char)(8))) e.Handled = true;
}
```

Omenjene dogodke bomo uporabili pri programiranju naslednjega zgleda.

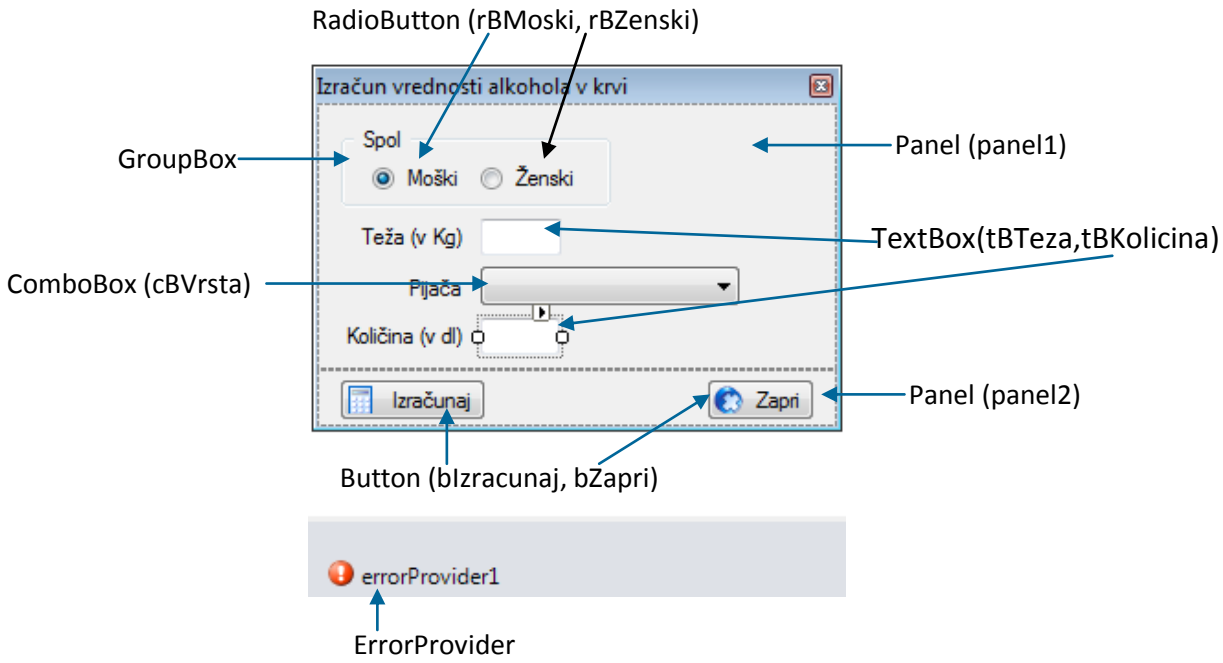
Izračun stopnje alkohola v krvi

Približen čas izločanja alkohola iz telesa je povprečno od 0,1 do 0,15 g za vsak kg teže osebe na uro. Če imamo podatek o koncentraciji alkohola v krvi, lahko izračunamo koncentracijo alkohola v krvi. Poenostavljena formula za približen izračun koncentracije je: $c = m / (TT)*r$, pri čemer je:

c = koncentracija alkohola v krvi (gramov oz promilov etanola na kg krvi)
m = masa popitega čistega alkohola izražena v gramih
TT = telesna masa izražena v kilogramih
r = porazdelitveni faktor (za moške 0,7, za ženske 0,6)

Podatke o stopnji alkohola v posameznih vrstah pijače imamo zbrane v tekstovni datoteki *Alkohol.txt*. Napišimo program, ki bo glede na spol, vrsto pijače, količino zaužite pijače in teže izračunal, kolikšna je približna koncentracija alkohola v krvi potem, ko popijemo to vrsto pijače (in smo bili prej povsem trezni). Pri tem naj bo uporabniški vmesnik tak, kot ga prikazuje slika.

Projekt poimenujmo *Alkohol*, obrazec pa *FAlkohol*. Nanj postavimo naslednje gradnike:



Slika 59: Gradniki na obrazcu *FAlkohol*.

Gradnik	Lastnost	Nastavitev	Opis
FAlkohol	FormBorderStyle	FixedToolWindow	Uporabnik velikosti okna ne more spreminjati.
panel2	Dock	Bottom	Spodnji panel je prilepljen na dno obrazca *FAlkohol*.
panel1	Dock	Fill	Zgornji panel je raztegnjen čez vso preostalo površino obrazca *FAlkohol*.
rBMoski	Checked	True	Radijski gumb je izbran.
tBTeza	TextAlign	Right	Teža bo desno poravnana.
tBKolicina	TextAlign	Right	Količina bo desno poravnana.
cBVrsta	DropDownStyle	DropDownList	Uporabnik ne bo mogel vpisati svoje vrednosti, ampak le izbral eno od obstoječih.
bIzracunaj	Image	Poljubna slika	

bIzracunaj	ImageAlign	MiddleLeft	Slika bo na gumbu poravnana na levi strani.
bIzracunaj	TextAlign	MiddleRight	Besedilo bo na gumbu poravnano na desni strani.
bZapri	Image	Poljubna slika	
bZapri	ImageAlign	MiddleLeft	Slika bo na gumbu poravnana na levi strani.
bZapri	TextAlign	MiddleRight	Besedilo bo na gumbu poravnano na desni strani.

Tabela 19: Lastnosti gradnikov obrazca *FAlkohol*.

V projektu smo ustvarili razred *pijaca* z dvema poljema (vrsta pijače in stopnja alkohola). Dogodek *Load* obrazca smo uporabili za prenos podatkov iz tekstovne datoteke *Alkohol.txt* (ta se nahaja v mapi *Bin→Debug* našega projekta) v tipizirano zbirko objektov tipa *pijaca* (vrsta zbirke je *List*) z imenom *Pijace*. Potem še gradnik *ComboBox* (imenovan *cBVrsta*) s pomočjo zbirke *Pijace* napolnimo z vrstami pijač (opisali smo ga že v programu *Anketa*).

Ker je najbolj tipičen podatek, ki ga uporabnik vnese ob zagonu programa, njegova teža, poskrbimo, da bo vnosno mesto v tem tekstovnem polju. Za to bomo poskrbeli tako, da bomo v v odzivni metodi obrazca na dogodek *Shown* poskrbeli, da bomo fokus postavili na to vnosno polje. Dogodek *Shown* se zgodi takoj za dogodkom *Load*, ko so objekti na obrazcu že ustvarjeni.

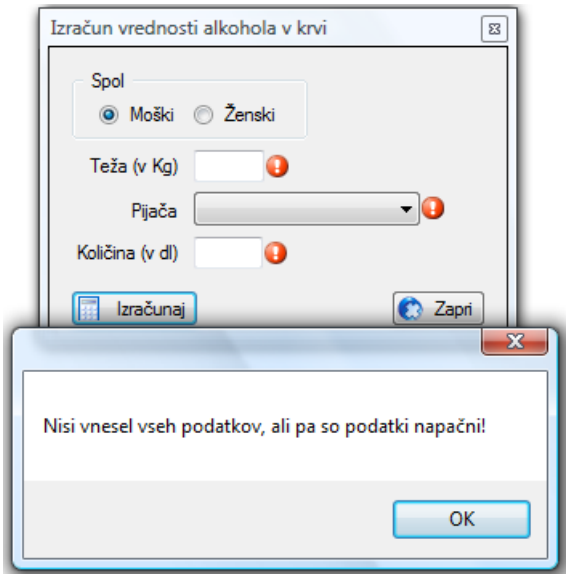
```
public FAlkohol()
{
    InitializeComponent();
}
/*razred Pijaca: vsak objekt tega tipa bo hranil podatke o vrsti pijače
  in stopnji alkohola v njej*/
public class Pijaca
{
    public string vrsta;
    public double stopnja;
    //konstruktor
    public Pijaca(string vrsta, double stopnja)

    {
        this.vrsta = vrsta;
        this.stopnja = stopnja;
    }
    public string Vrsta //lastnost/property
    {
        get { return vrsta; }
```

```
        set { vrsta = value; }
    }
    public double Stopnja //lastnost/property
    {
        get { return stopnja; }
        set { stopnja = value; }
    }
}
List<Pijaca> pijace = new List<Pijaca>();//deklaracija tipizirane zbirke
/*preden se obrazec prikaže na ekranu, napolnimo tako zbirko kot ComboBox
   s podatki iz datoteke*/
private void FAlkohol_Load(object sender, EventArgs e)
{
    try
    {
        //podatke iz tekstovne datoteke prenesemo  tabelo objektov
        StreamReader beri = File.OpenText("Alkohol.txt");
        string vrstica = beri.ReadLine();//beremo vrstico
        while (vrstica != null)  //če branje uspešno
        {
            /*podatka o vrsti pijače in stopnji alkohola sta v vrstici
              razmejena z znakom '|'. Ločimo ju z metodo Split*/
            string[] zac = vrstica.Split('|');
            //ustvarimo nov objekt tipa Pijaca
            pijaca nova=new pijaca(zac[0],Convert.ToDouble(zac[1]));
            pijace.Add(nova); //in ga damo  zbirko
            vrstica = beri.ReadLine();//beremo vrstico
        }
        beri.Close();//zapremo podatkovni tok/datoteko
        //gradnik cBVrsta povežemo s podatkovnim izvorom – zbirko pijace
        cBVrsta.DataSource = pijace;
        //določimo polje, ki se bo prikazovalo v spustnem seznamu
        cBVrsta.DisplayMember ="Vrsta";//Vrsta=lastnost razreda Pijaca
        //Določimo polje, ki ga bomo uporabili kot vrednost
        cBVrsta.ValueMember ="Stopnja";//Stopnja=lastnost razreda Pijaca
    }
    catch
    {   MessageBox.Show("Napaka pri delu z datoteko!"); }
}
/*dogodek Shown se zgodi takoj za dogodkom Load, ko so objekti na obrazcu
  že ustvarjeni*/
private void FAlkohol_Shown(object sender, EventArgs e)
{
   tBTeza.Focus();//Na začetku bo aktiven gradnik tBTeza
}
private void bZapri_Click(object sender, EventArgs e)
{
   Application.Exit();//zaključek projekta, obrazec se zapre
}
```

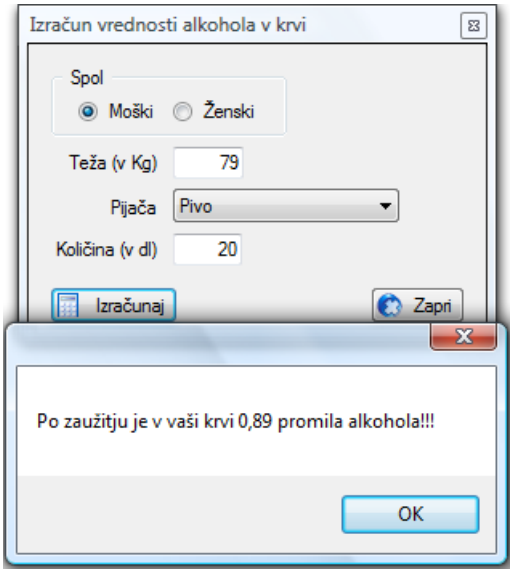
Uporabnikov vnos teže in količine kontroliramo s pomočjo dogodka *KeyPress* (dovolimo le vnos števk, decimalne vejice, dovolimo pa tudi brisanje že napisanih znakov). S pomočjo gradnika *ErrorProvider* pa še preverjamo, ali je uporabnik sploh kaj vnesel oz. ali je izbral vrsto pijače. Celotno kodo smo zapakirali še v varovalni blok in oblikovali osnovno sporočilo o napaki! Zaradi enostavnosti v razredu ni kontrole pravilnosti stanj (vrednosti polj vrsta in stopnja), saj predpostavljamo, da njihovo pravilnost program oz. kontrolira uporabniški vmesnik.

```
private void tB_KeyPress(object sender, KeyPressEventArgs e)
{
    /*če vneseni znak ni števka (0 do 9), ali decimalna vejica, ali pa
      tipka BackSpace (char 8), je dogodek zaključen*/
    if (((e.KeyChar < '0') || (e.KeyChar > '9')) && (e.KeyChar != ',') &&
(e.KeyChar != (char)(8)))
        e.Handled = true;
}
private void bIzracunaj_Click(object sender, EventArgs e)
{
    /*kontrola vnosa teže se izvede avtomatično, ker je gradnik tBTeza ob
      zagonu aktiven gradnik (ima fokus)*/
    tB_Validating(cBVrsta, null);//kontrola izbire vrste pijače
    tB_Validating(tBKolicina, null);//kontrola vnosa količini
    try
    {
        /*izračun stopnje alkohola v krvi: za izbrani tekst v gradniku
ComboBox poiščemo stopnjo alhohola v zbirki pijace*/
        int indeks = cBVrsta.SelectedIndex;
        double stopnja = Convert.ToDouble(cBVrsta.SelectedValue);
        double teza = Convert.ToDouble(tBTeza.Text);
        double skupno_gramov=(Convert.ToDouble(tBKolicina.Text))*stopnja;
        double alkohola;
        if (rBMoski.Checked)  //Izbran radijski gumb Moški
            alkohola = skupno_gramov / (teza) * 0.6;
        else alkohola = skupno_gramov / (teza) * 0.7;/*Ženske*/
        MessageBox.Show("Po zaužitju je v vaši krvi " +
Convert.ToString(Math.Round(alkohola, 2)) + " promila alkohola!!!");
    }
    catch
    { MessageBox.Show("Ni vseh podatkov, ali pa so podatki napačni!"); }
}

/*metoda za kontrolo vnosa številke: izvede se ob dogodku Validating obeh
 TextBox-ov in gradnika ComboBox na obrazcu*/
private void tB_Validating(object sender, CancelEventArgs e)
{
    if (sender == cBVrsta)
    {
```

```
        /*če uporabnik ni izbral vrste pijače, je indeks -1 (privzeta
          vrednost), se pojavi ikona z opozorilom*/
        if (cBVrsta.SelectedIndex == -1)
            errorProvider1.SetError(cBVrsta, "Izberi vrsto pijače!");
        else /*odstranimo ikono z opozorilom*/
           errorProvider1.SetError((cBVrsta), "");
    }
    else if (sender is TextBox)
    {
        if ((sender as TextBox).Text == "")
             errorProvider1.SetError((sender as TextBox), "Vnesi številko
večjo od 0!");
        else /*odstranimo ikono z opozorilom*/
            errorProvider1.SetError((sender as TextBox), "");
    }
}
```



Slika 60: Obvestilo o napačnem vnosu!



Slika 61: Rezultat pravilnega vnosa.

Dogodki, ki jih sproži miška

Pri obdelavi dogodkov, ki jih proži miška, nas običajno zanima položaj miškinega kazalca in stanje njenih gumbov.

Ko uporabnik premakne miško, operacijski sistem poskrbi, da se miškin kazalec na zaslonu premakne na ustrezno mesto. Kazalec predstavlja eno samo točko (*pixel*), ki ji operacijski sistem sledi in prepoznava kot položaj tega kazalca. Ob premiku miške ali pa pritisku gumba se zgodi eden od dogodkov miške. Trenutni položaj miške lahko tako npr. ugotovimo s pomočjo lastnosti *Location* parametra tipa *MouseEventArgs* (dobimo relativen položaj miške, to je položaj miške

znotraj gradnika, v katerem se nahajamo), ali pa npr. z uporabo lastnosti *Position* razreda *Cursor* (dobimo absoluten položaj miške, to je položaj glede na celoten obrazec).

```
/*Pozicija kazalca miške (relativen, glede na gradnik) z uporabo parametra e
tipa MouseEventArgs*/
MessageBox.Show("Lokacija kazalca miške: x = " + e.Location.X.ToString() + ",
y= " + e.Location.Y.ToString());
/*Pozicija kazalca miške (absolutno, glede na obrazec) z uporabo razreda
Cursor*/
MessageBox.Show("Lokacija kazalca miške: x =
"+Cursor.Position.X.ToString()+", y= "+Cursor.Position.Y.ToString());
```

S pomočjo informacije, ki jo dobimo o položaju miške tako lahko izvedemo točno določeno operacijo, ki je odvisna od tega, kje in kdaj smo kliknili z miško.

Temeljni problem, kako obdelati neko potezo narejeno z miško, je pravilna obdelava miškinih dogodkov. V naslednji tabeli so prikazani vsi dogodki, ki jih miška proži, zraven pa je razlaga, kdaj do njih pride.

Dogodek Miške	R a z l a g a
Click	Dogodek se zgodi ko kliknemo miškin gumb To je tik preden se zgodi dogodek *MouseUp*. Ta dogodek obdelamo tipično tedaj, ko nas zanima samo uporabnikov klik na določen gradnik. Če je gradnik izbran (ima fokus), se kot klik na ta gradnik šteje tudi pritisk na tipko <*Enter*>.
MouseClick	Dogodek se zgodi, ko uporabnik z miško klikne nek gradnik, a ne tudi ob pritisku na tipko <*Enter*> . Ta dogodek uporabimo tedaj, ko želimo ob kliku na nek gradnik dobiti podatke o sami miški. Dogodek *MouseClick* se zgodi za dogodkom *Click*.
DoubleClick	Dogodek se zgodi, ko na nek gradnik dvokliknemo. Uporabimo ga le tedaj, ko res želimo preveriti, ali je uporabnik dvokliknil nek gradnik.
MouseDoubleClick	Dogodek se zgodi, ko uporabnik z miško dvoklikne na določen gradnik. Ta dogodek uporabimo tedaj, ko želimo ob dvokliku na nek gradnik dobiti podatke o sami miški.
MouseDown	Dogodek se zgodi, ko je kazalnik miške nad določenim gradnikom in uporabnik pritisne enega od miškinih gumbov.
MouseEnter	Dogodek se zgodi, ko miškin kazalec doseže rob nekega gradnika, oziroma samo površino tega gradnika – odvisno od vrste gradnika.
MouseHoover	Dogodek se zgodi, ko se z miško zaustavimo in za trenutek obmirujemo nad nekim gradnikom.

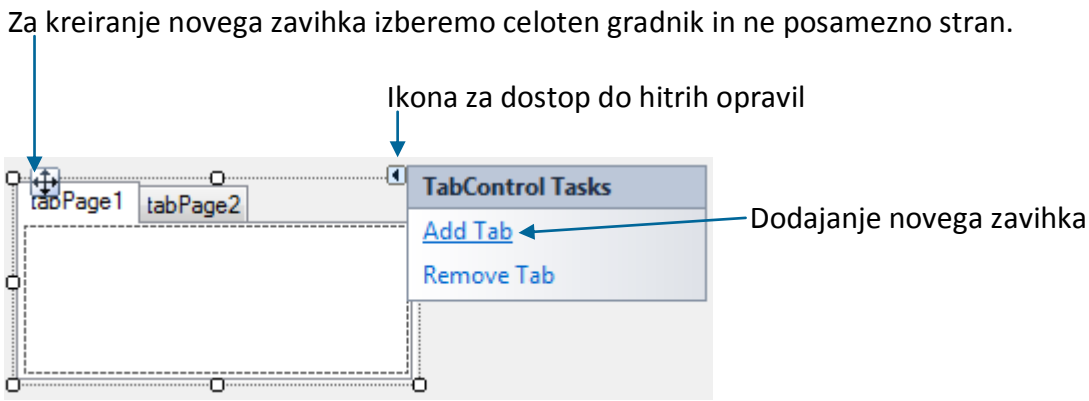
MouseLeave	Dogodek se zgodi, ko miškin kazalec zapusti rob določenega gradnika, oz. ko miškin kazalec zapusti površino nekega gradnika – odvisno od vrste gradnika.
MouseMove	Dogodek se zgodi, ko se kazalnik miške, medtem ko smo nad nekom gradnikom, premakne.
MouseUp	Dogodek se zgodi, ko je kazalec miške nad določenim gradnikom in uporabnik spusti miškin gumb.
MouseWheel	Dogodek se zgodi, ko uporabnik zavrti miškin kolešček, medtem ko je izbran ustrezen gradnik (ima fokus).

Tabela 20: Dogodki miške.

Za vajo ustvarimo nov projekt *DogodkiMiške*, v katerem bomo prikazali dogodke miške. Spoznali bomo tudi uporabo zavihkov in poenostavljen prikazovalnik slik. Za potrebe vaje v oknu *Code View* najprej definirajmo tabelo nekaterih barv in ustvarimo generator naključnih števil

```
Random naklj = new Random();
//tabela barv
Color[] Barve = {
Color.Chocolate,Color.YellowGreen,Color.Olive,Color.DarkKhaki,Color.Sienna,Co
lor.PaleGoldenrod,Color.Peru,Color.Tan,Color.Khaki,Color.DarkGoldenrod,Color.
Maroon,Color.OliveDrab, Color.DarkOrchid };
```

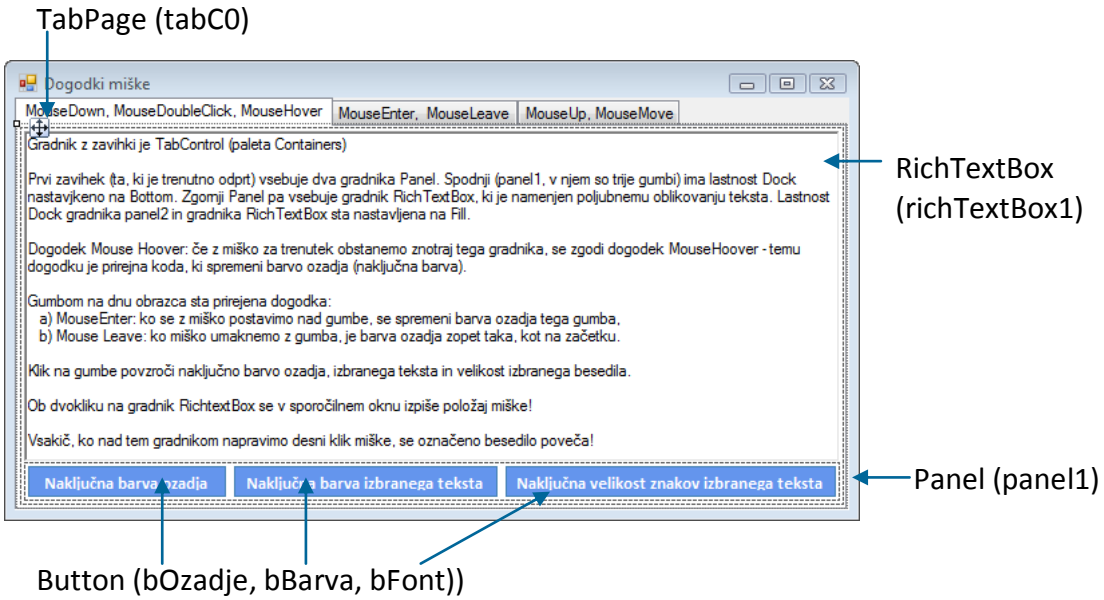
Na obrazec postavimo gradnik *TabControl*: ta gradnik omogoča ustvarjanje poljubnega števila zavihkov. Vsak zavihek lahko vsebuje svoje gradnike, ki so povsem neodvisni od gradnikov na ostalih zavihkih. Gradniku *TabControl* najprej nastavimo lastnost *Dock* na *Fill*, da zapolni celoten obrazec. Zavihke nato tvorimo tako, da kliknemo na ikono v zgornjem desnem delu gradnika in izberemo *Add Tab*.



Slika 62: Ustvarjanje novega zavihka gradnika TabControl.

Dva zavihka sta izdelana že ko postavimo gradnik na obrazec, mi pa ustvarimo še enega. Celoten gradnik poimenujmo *tBDogodkiMiske*, posamezne zavihke pa *tabC0*, *tabC1* in *tabC2*. Najprej izberemo posamezen zavihek (kliknemo na zavihek), nato kliknemo na površino tega zavihka in končno glede na zavihek zapišemo v lastnost (*Name)* ime tega zavihka. Posameznim zavihkom določimo še lastnost *Text* tako, kot kaže slika. Ta lastnost določa, kaj je napisano na zavihku. Napisi na zavihkih predstavljajo dogodke miške, ki jih želimo predstaviti na posameznem zavihku. Na površino prvega zavihka postavimo gradnik *Panel* (*panel1*, lastnost *Dock* je *Bottom*), nato pa še gradnik *RichTextBox* (lastnost *Dock* je *Fill*). Kliknimo v lastnost *Text* tega gradnika in zapišimo besedilo (komentar), tako kot je na sliki. Gradnik *RichTextBox* je namenjen pisanju teksta, ki ga lahko poljubno oblikujemo.

V gradnik *Panel*, ki je na dnu prvega zavihka postavimo še tri gumbe z imeni *bOzadje*, *bBarva* in *bFont*. Vsem trem gumbom priredimo lastnost *BackColor→CornFlowerBlue*, *ForeColor→*White, ime fonta naj bo *Calibri* velikosti 10, lastnost *FlatStyle* pa naj bo *Flat*. Ta zadnja opredeljuje izgled gumba.



Slika 63: Objektii na prvem zavihku gradnika *TabControl*.

Gumbom zaporedoma priredimo ista odzivni metodi *bOzadje_MouseEnter* in *bOzadje_MouseLeave* (vsem tem gumbom se bo spremenilo ozadje, če bomo vanje vstopili ali izstopili z miško). Spomnimo se, da smo pri projektu *Preverjanje znanja osnovnih računskih operacij* razložili, kako lahko isto odzivno metodo uporabimo nad dogodki različnih gradnikov. Gumbom nato zaporedoma priredimo še odzivne metode *bOzadje_Click*, *bBarva_Click* in *bFont_ Click*.

```
private void bOzadje_MouseEnter(object sender, EventArgs e)
{
      /*ozadje gumba dobi novo barvo*/
```

```
      (sender as Button).BackColor = Color.Orange;
}

private void bOzadje_MouseLeave(object sender, EventArgs e)
{
     /*ozadje gumba dobi novo barvo*/
     (sender as Button).BackColor = Color.CornflowerBlue;
}

private void bOzadje_Click(object sender, EventArgs e)
{
    //barva ozadja bo ena izmed barv iz tabele Barve
    richTextBox1.BackColor = Barve[naklj.Next(0, Barve.Length)];
}

private void bBarva_Click(object sender, EventArgs e)
{
    //tri naključna cela števila med 0 in 254 za mešanje barv
    byte r = Convert.ToByte(naklj.Next(0, 255));
    byte g = Convert.ToByte(naklj.Next(0, 255));
    byte b = Convert.ToByte(naklj.Next(0, 255));
    /*naključno barvo besedila dosežemo z metodo FromArgb razreda Color*/
    richTextBox1.SelectionColor = Color.FromArgb(r, g, b);
}

private void bFont_Click(object sender, EventArgs e)
{
    /*naključno število tipa float med 10 in 20*/
    float velikost=(float)(naklj.NextDouble()*10+10);
    /*ustvarimo nov objekt tipa Font: uporabimo enega izmed konstruktorjev
      razreda Font: parametri konstruktorja so ime fonta, velikost in stil */
    Font novi = new Font(richTextBox1.Font.Name, velikost, FontStyle.Bold);
    /*richTextBox1.Font = novi;  //Takale bi bila npr.nastavitev novega fonta
      za celoten RichTextBox1*/
    richTextBox1.SelectionFont = novi;//izbrani tekst dobi nov font
}
```

Gradniku *richTextBox1* priredimo odzivne metode za dogodke *MouseDown*, *MouseDoubleClick* in *MouseHover* (za dogodek MouseHover v oknu *Properties→Events→*MouseHover izberemo že prej napisano odzivno metodo *bOzadje_Click*).

```
private void richTextBox1_MouseDown(object sender, MouseEventArgs e)
{
    /*preverimo, če je bil pritisnjen desni gumb*/
    if (e.Button == MouseButtons.Right)
    {
         /*označeno besedilo gradnika richTextBox1 se poveča za ena*/
```

```
            richTextBox1.SelectionFont = new
Font(richTextBox1.SelectionFont.Name, richTextBox1.SelectionFont.Size + 1,
FontStyle.Italic);
        }
}

private void richTextBox1_MouseDoubleClick(object sender, MouseEventArgs e)
{
    MessageBox.Show("Položaj oz. koordinate miške\n\nDvokliknjena je bila
tipka : "+e.Button+"\nOddaljenost od levega roba: " + e.X + "\nOddaljenost od
vrha obrazca: " + e.Y);
}
```

Preklopimo sedaj na drugi zavihek. Kot smo že napovedali, bomo na tem zavihku delali slikami. Potrebujemo torej prostor za 9 slik.

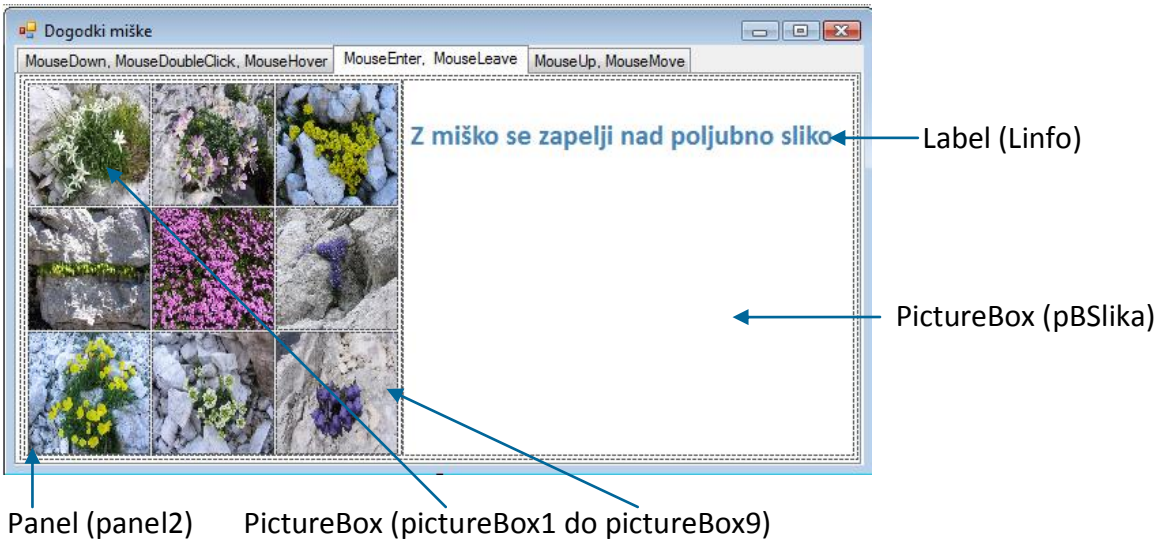
Na zavihek najprej postavimo gradnik *Panel* in ga poimenujmo *panel2*. Ker mu lastnost *Dock* nastavimo na *Left*, bomo s tem dosegli, da bo gradnik prilepljen na levem delu zavihka. V ta gradnik nato postavimo 9 enakih gradnikov *PictureBox* z imeni *picureBox1* do *pictureBox9*. Vsi imajo lastnost *SizeMode* nastavljeno na *StretchImage*. S tem dosežemo, da bo v gradniku prikazana, povečana ali pa pomanjšana, celotna slika, ne glede na to, če bo uporabnik morda spremenil velikost okna. Velikost gradnika naj bi npr. 90 x 90. Kljub temu velja omeniti, da je priporočljivo, da si vsaj v osnovi pripravimo slike, ki so ustrezne velikosti. Programi za delo s slikami namreč povečevanje/pomanjševanje slik opravijo praviloma kvalitetneje, kot so metode, vgrajene v C#, saj gre za specializirane programe, namenjene prav temu.

Na desno stran drugega zavihka zatem postavimo še en gradnik *PictureBox* in ga poimenujmo *pBSlika*, lastnost *Dock* naj bo *Fill*, lastnost *SizeMode* pa naj bo zopet *StretchImage*. V gradnike *pictureBox1* do *pictureBox9* sedaj uvozimo devet različnih slik. Končno v gradnik *pictureBox9* postavimo še gradnik *Label* z napisom kot je na sliki.

Gradnikom *pictureBox1* do *pictureBox9* priredimo isti odzivni metodi dogodkov *MouseEnter* in *MouseLeave*. Namen teh dveh metod je, da ko se z miško postavimo na katerokoli sliko, se le-ta povečana prikaže v desnem delu zavihka (obenem pa se skrije oznaka z napisom), ko pa z miško sliko zapustimo, je desna stran zavihka zopet taka kot prej (oznaka z napisom se zopet prikaže).

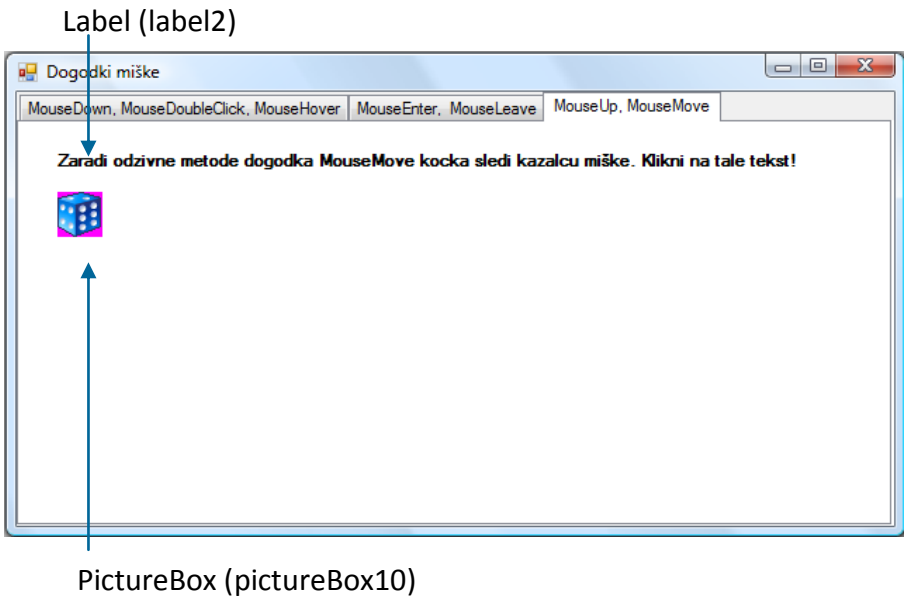
```
private void pictureBox1_MouseEnter(object sender, EventArgs e)
{
    lInfo.Visible=false;
    pbSlika.Image = (sender as PictureBox).Image;
}

private void pictureBox1_MouseLeave(object sender, EventArgs e)
{
    pbSlika.Image = null;
    lInfo.Visible = true;
}
```

Slika drugega zavihka:



Slika 64: Objekti na drugem zavihku gradnika *TabControl*.

Preklopimo še na tretji zavihek. V njem bomo Nanj postavimo le dva gradnika: gradnik *Label* in gradnik *PictureBox* (lastnost *SizeMode* naj bo *StretchImage*).



Slika 65: Objekta na tretjem zavihku gradnika *TabControl*.

Gradniku *tabC2* (tretjemu zavihku) priredimo odzivno metodo dogodka *MouseMove* in ga poimenujmo *tm_MouseMove*, gradniku *Label* pa dogodek *MouseUp*. S pomočjo dogodka

MouseMove in parametrom tipa *MouseEventArgs*, ki vsebuje tudi lastnosti *X* in *Y* (to sta koordinati miške) smo dosegli potovanje slike skupaj z miškinim kazalcem.

```
private void label2_MouseUp(object sender, MouseEventArgs e)
{
    /*fontu oznake priredimo nov objekt tipa Font: uporabimo enega izmed
      konstruktorjev razreda Font: s parametri konstruktorja povemo, v katero
      družino ta font spada, velikost (tipa Float) in stil */
    label2.Font = new Font(FontFamily.GenericSansSerif,20.0F,FontStyle.Bold);
    label2.Text = "Ko si spustil tipko, se je zgodil dogodek
        \r\nMouseUp\r\n\r\nTrenutni čas: "+DateTime.Now.ToLongTimeString();
}
private void tm_MouseMove(object sender, MouseEventArgs e)
{
    //ob premiku miške, bo slika sledila miški!
    pictureBox10.Location = new Point(e.X, e.Y);
}
```

Point v C# je razred, ki predstavlja točko z njenima koordinatama v dvodimenzionalnem koordinatnem sistemu. Konstruktorju objekta smo posredovali dve koordinati, ki predstavljata trenutni položaj miškinega kazalca. Ta nova točka tako postane novi položaj ikone kocke v gradniku *PictureBox*.

Labirint

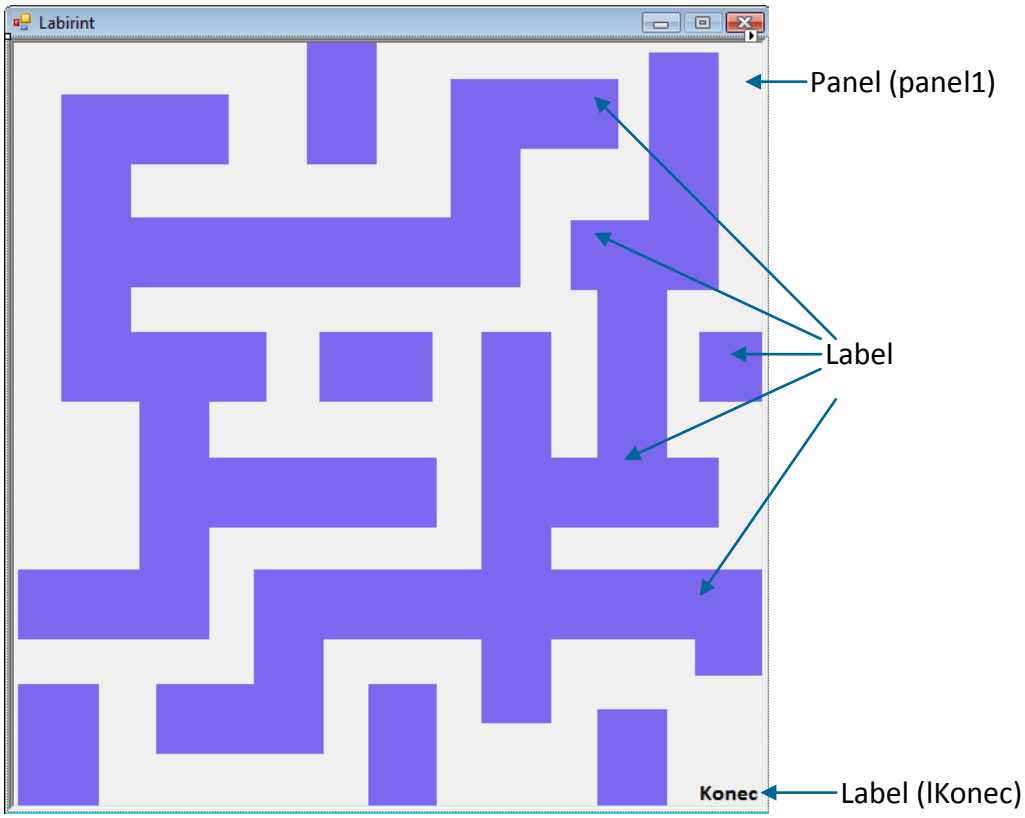
Izdelajmo preprosto igro *Labirint*. Uporabnik bo moral kazalnik miške premakniti od levega zgornjega roba do spodnjega desnega roba obrazca, pri čemer ne sme zadeti stene labirinta. Zamislili smo si, da bo igra videti tako, kot prikazuje Slika 66: Sestavni deli labirinta. Na sliki smo že označili gradnike, ki bodo potrebni. Kot vidimo, tokrat izjemoma ne bomo potrebovali nobenega novega.

Lastnost *Text* obrazca zapišimo *Labirint* in obrazcu določimo velikost, kakršno menimo, da bomo potrebovali za naš labirint. Ker bomo lastnost *FormBorderStyle* nastavili na *Fixed3D* in lastnost *MaximazeBox* na *False*, uporabnik velikosti okna ne bo mogel spreminjati, niti ga maksimirati.

Na obrazec postavimo gradnik *Panel* in mu lastnost *Dock* nastavimo na *Fill*. S tem ga raztegnemo ga čez celotno površino. Ker mu lastnost *BorderStyle* nastavimo na *Fixed3D*, bodo robovi obrazca dobro vidni.

Določiti moramo še, kje je izhod iz labirinta. V ta namen na obrazec postavimo še gradnik *Label*. Poimenujmo (lastnost *(name)*) ga kar *lKonec* in mu lastnost *Text* spremenimo v *Konec*. Oznako premaknimo v spodnji desni rob obrazca, vendar se ne sme dotikati njegovih robov.

Labirint moramo še ustrezno oblikovati. V ta namen si pripravimo gradnik tipa *Label*, ki ga bomo oblikovali v obliko polnega pravokotnika. Zato mu pobrišimo lastnost *Text* in ga pobarvamo tako, da mu določimo lastnost *BackColor* (v vaji je to *MediumStatBlue*). Ko mu poskušamo s pomočjo vlečenja spremeniti velikost, vidimo, da ne gre. Razlog je v tem, da je lastnost *AutoSize* privzeto nastavljena na *True*. To pomeni, da se velikost oznake avtomatsko spreminja glede na to, koliko prostora potrebujemo za prikaz besedila na oznaki (lastnost *Text*). Ko mu lastnost *AutoSize* nastavimo na *False*, našo "opeko" lahko nastavimo na poljubno velikost.



Slika 66: Sestavni deli labirinta.

S pomočjo tehnike *Copy – Paste* iz teh oznak oblikujmo svoj labirint, pri čemer posamezne oznake poljubno povečujemo oz. jih pomanjšujemo. Paziti moramo le na to, da bo prazen prostor med posameznimi oznakami dovolj velik, da ne bo igra pretežka. Seveda pa moramo poskrbeti tudi za to, da v labirintu obstaja vsaj ena pot od začetka do konca.

Ostane nam le še zapis ustreznih dogodkov. Uporabnik bo moral s kazalnikom miške priti od začetka labirinta (v zgornjem levem kotu obrazca) do konca. Najprej napišimo metodo *PremikNaZacetek*, ki bo kazalnik miške na začetku postavila v zgornji levi kot obrazca. Pred pisanjem metode ustvarimo še dva objekta tipa *SoundPlayer*: kadarkoli bomo zadeli opeko, se bo oglasil zvočnik z nekaj toni *"chord.wav"*. Ob uspešnem zaključku igre pa se bo oglasil zvok *"tada.wav"*. Več o tem gradniku smo si že ogledali v projektu *Predvajalnik glasbe in videa*.

```
//Objekt tipa SoundPlayer, ki se oglasi vselej, ko igralec zadene ob opeko
System.Media.SoundPlayer zadetekOpeke = new
System.Media.SoundPlayer(@"C:\Windows\Media\chord.wav");
//Objekt tipa SoundPlayer, ki se oglasi ko igralec uspešno zaključi igro
System.Media.SoundPlayer zakljucniZvok = new
System.Media.SoundPlayer(@"C:\Windows\Media\tada.wav");
```

Sedaj pripravimo metodo, ki bo povzročila premik kazalca miške na mesto, ki je 10 točk oddaljeno od zgornjega roba in 10 točk od levega roba.

```
private void PremikNaZacetek()
{
    zadetekOpeke.Play();//Začetni zvok
    /*Objekt tipa Point v C# predstavlja že narejen razred, ki predstavlja
      točko z dvema koordinatama v dvodimenzionalnem koordinatnem sistemu*/
    Point zacetnaTocka = panel1.Location;
    zacetnaTocka.Offset(10, 10); //Premik točke
    Cursor.Position = PointToScreen(zacetnaTocka); //Začetni položaj miške
}
```

S *panel1.Location* smo ugotovili, kje na zaslonu je levi zgornji vogal panela, kjer je labirint. Z metodo *Offset* smo potem to točko premaknili 10 točk desno in 10 točk dol, saj je koordinatni sistem tak, da koordinate x naraščajo v desno, koordinate y pa navzdol.

S *Cursor.Position* nadziramo položaj miške. Metoda *PointToScreen* izračuna koordinate točke, torej ravno točko, kamor želomo postaviti miškin kazalec.

Metodo *PremikNaZacetek()* bomo poklicali pri zagonu projekta (v konstruktorju obrazca)

```
public Form1()
{
    InitializeComponent();
    //Klic metode: začetna pozicija miške
    PremikNaZacetek();
}
```

in vselej, ko uporabnik zadene ob neko opeko. Zadetek opeke je dogodek *MouseEnter*, ki ga priredimo vsem opekam na obrazcu (vsem oznakam). To storimo tako, da najprej izberemo poljubno opeko, v oknu *Properties→Events* v polje *MouseEnter* zapišemo ime dogodka *Opeka_MouseEnter*, nato dvokliknemo na to ime in zapišemo ustrezno kodo – to bo le klic že napisane metode *PremikNaZacetek*.

```
private void Opeka_MouseEnter(object sender, EventArgs e)
{
    /*Ko kazalec miške zadene opeko ali pa vstopi v gradnik panel,
      pokličemo metodo PremikNaZacetek()*/
    PremikNaZacetek();
```

```
}
```

Dogodek *Opeka_MouseEnter* moramo sedaj prirediti vsem opekam na obrazcu. Najlažje tako, da v meniju *Edit* razvojnega okolja kliknemo *Select All*. S tem smo izbrali vse objekte, tudi oznako *lKonec*, ki pa je ne želimo imeti med izbranimi. Kot običajno v okolju *Windows* stisnemo tipko *Ctrl* in kliknemo na oznako z napisom *Konec*. Tej oznaki bomo kasneje priredili dogodek za zaključek igre. Izbrane imamo torej vse gradnike, razen oznake z napisom *Konec*. V oknu *Properties→Events* v polju *MouseEnter* sedaj izberemo dogodek *Opeka_MouseEnter*, ki smo napisali že prej. Priredi se vsem izbranim objektom.

Ostane še dogodek *MouseEnter* oznake *lKonec*. V sporočilnem oknu igralcu čestitamo za uspešen zaključek, še prej pa naj se oglasi zvočnik s fanfarami!

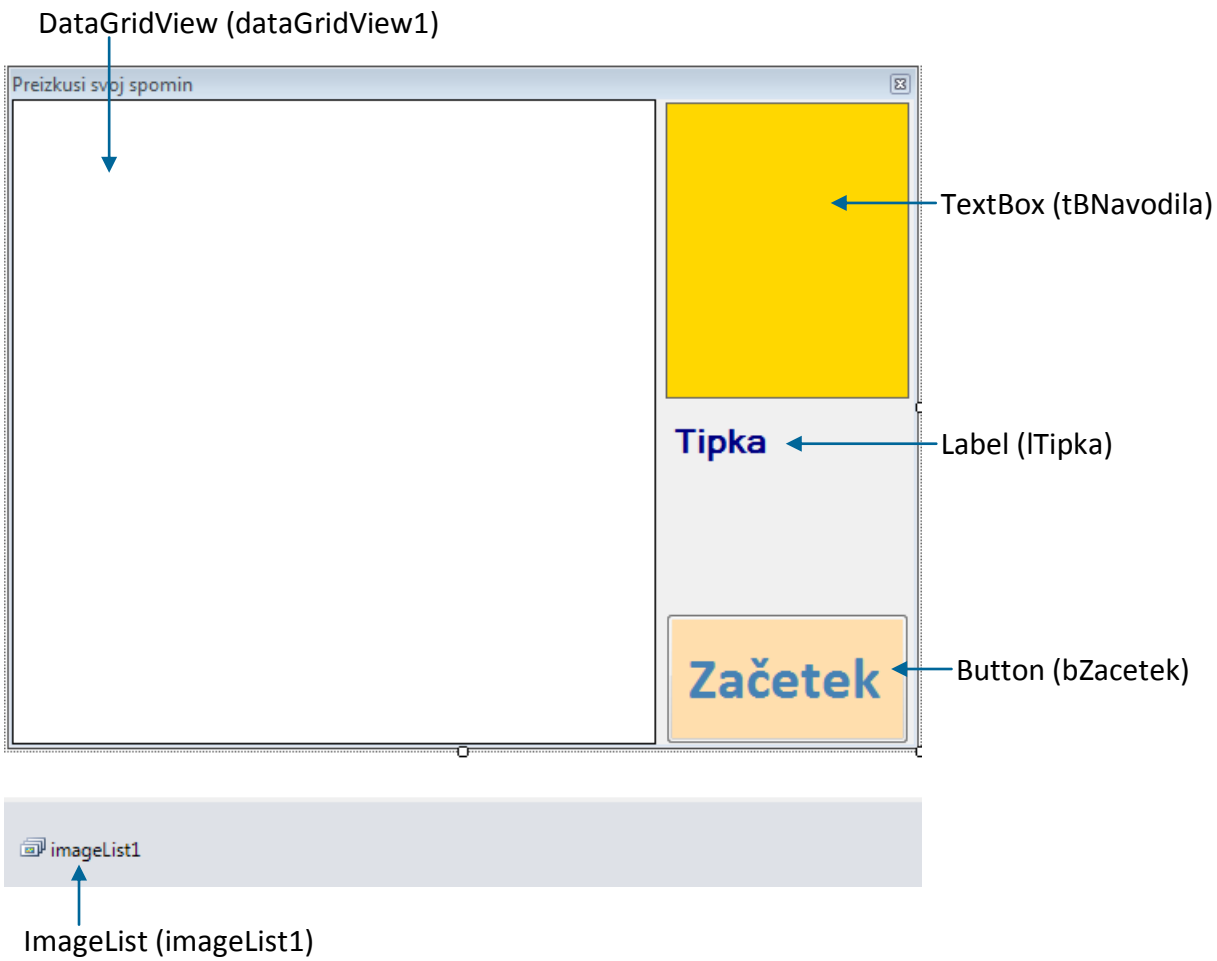
```
private void lKonec_MouseEnter(object sender, EventArgs e)
{
    //fanfare na koncu in še obvestilo o zaključku igre
    zakljucniZvok.Play();
    MessageBox.Show("Čestitke!");
    Close();
}
```

Spomin – igra za preizkus spomina

Spomnimo se igre, ki smo se je igrali v zgodnji mladosti. Na voljo imamo 16 podvojenih sličic. Na začetku so sličice razporejene naključno in obrnjene narobe. V čim manj korakih skušamo odkriti pare tako, da obrnemo po dve sličici. Če sta enaki, ju pustimo odkriti, sicer pa ju obrnemo nazaj. Za to igro napišimo ustrezen program, v katerem bomo uporabili gradnik *DataGridView*.

V tem projektu bomo vrstice s podatki (slike) dodajali in spreminjali programsko. Pokazali bomo, kako dostopamo do posamezne vrstice in posamezne celice tega gradnika. Slike, ki jih bomo potrebovali, bomo hranili v gradniku tipa *ImageList*, da bomo do njih lahko dostopali preko indeksa. Ko gradnik *ImageList* postavimo na obrazec, se ta umesti na dno urejevalniškega okna, pod obrazcem. Gre namreč za nevizuelni gradnik, ki ga na obrazcu neposredno ne vidimo, saj je njegov namen le hranjenje slik. Slike vanj dodamo tako, da ga izberemo, nato pa v zgornjem desnem kotu kliknemo na puščico. V oknu, ki se odpre izberemo velikost slik (*ImageSize*) npr. 256 x 256, nato pa kliknemo na *Choose images*: v pogovornem oknu (*Image Collecton Editor*), ki se pri tem odpre, nato s klikom na gumb *Add* v ta seznam uvozimo 8 različnih slik, ki jih bomo uporabili v gradniku *DataGridView*, deveto sliko (njen indeks bo 8), pa pripravimo v poljubnem grafičnem urejevalniku (lahko kar v *Slikarju*). Ta slika naj bo le ustrezne

velikosti (256 x 256) in naj ima le belo ozadje – nič drugega! Uporabili jo bomo za začetne slike celic gradnika *DataGridView* . Načrt uporabniškega vmesnika bo sledeč.



Slika 67: Gradniki na obrazcu *FSpomin*.

V vnosnem polju *tBNavodila* bomo prikazali osnovna navodila. Slike bomo odkrivali s klikom miške, z oznako *lTipka* pa bomo uporabniku sporočali, katera tipka miške je na vrsti. Na obrazcu je še gumb, s katerim bomo igro zagnali.

Lastnosti gradnikov na obrazcu so prikazane v naslednji tabeli:

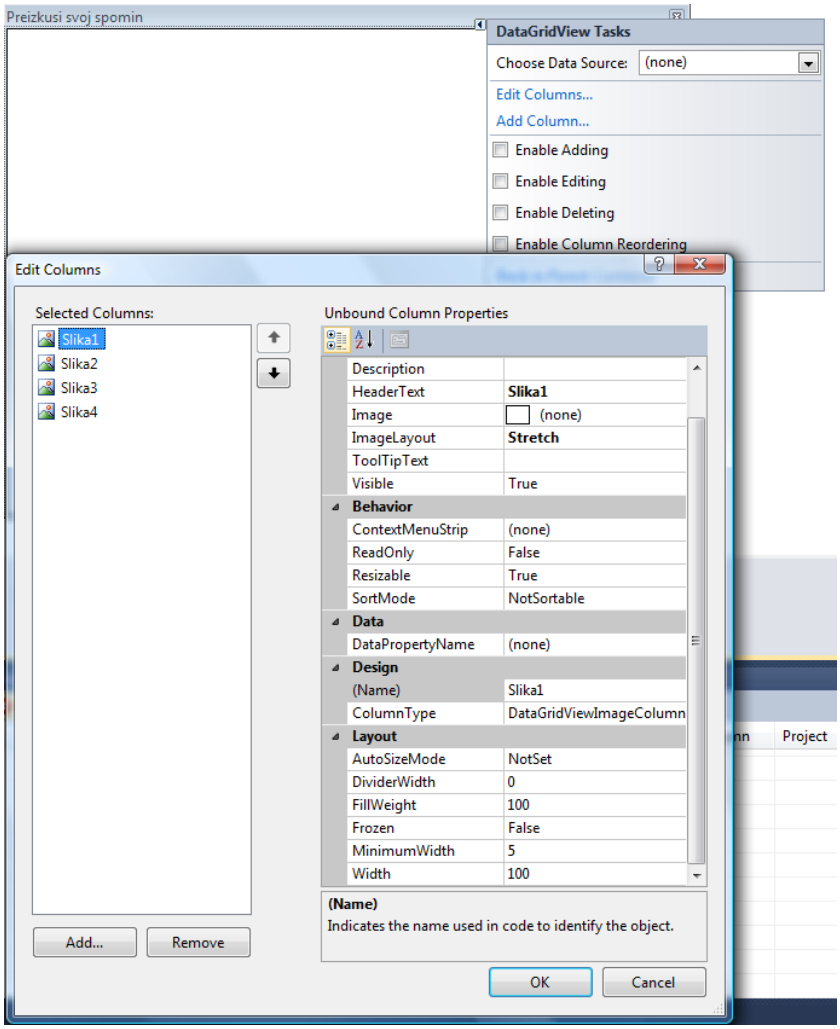
Gradnik	Lastnost	Nastavitev	Opis
Form1	FormBorderStyle	FixedToolWindow	Uporabnik velikosti okna ne more spreminjati

	Name	FSpomin	Ime obrazca
tBNavodila	BackColor	Gold	Barva ozadja
	BorderStyle	FixedSingle	Okvir okoli besedila
	MultiLine	True	Večvrstično besedilo
	ForeColor	Indigo	Barva besedila
bZacetek	BackColor	NavajoWhite	Barva ozadja
		StellBlue	Barva besedila
dataGridView1	BackGroundColor	White	Barva ozadja
	AllowUserToAddRows	False	Uporabnik ne more dodajati vrstic
	AllowUserToDeleteRows	False	Uporabnik ne more brisati vrstic
	AllowUserToResizeColumns	False	Uporabnik ne more spreminjati velikosti vrstic
	AllowUserToResizeRows	False	Uporabnik ne more spreminjati velikosti stolpcev
	ColumnHeadersVisible	False	Imena stolpcev niso vidna
	Columns	S klikom na gumb *Add* dodamo 4 stolpce: stolpcem priredimo lastnost *Name* in *HeaderText* takole: *Slika1*, *Slika2*, *Slika3* in *Slika4*, za lastnost *Type* pa izbremo *DataGRidViewImageColumn* (stolpec s sliko). Vsem stolpcem določimo še lastnost *ImageLayout* – *Stretch*. Na ta način se bo velikost slike avtomatsko prilagodila velikosti celice gradnika *DataGridView*.	
	Dock	Left	Gradnik bo zapolnil celotno levo stran obrazca
	Enabled	False	Gradnik na začetku ni aktiven
	RowHeadersVisible	False	Glave vrstic niso vidne

SCrollBars	Vertical	Navpični drsnik

Tabela 21: Lastnosti gradnikov obrazca *FSpomin*.

Izdelavo in poimenovanje stolpcev gradnika *DataGridView* , ter določanje njihovih lastnosti prikazuje naslednje slika. Vsi stolpci imajo enake lastnosti, le imena so različna. Zato tudi tu nastavljanje hitreje opravimo tako, da s pomočjo tipke *Ctrl* izberemo vse hkrati in jim nastavimo skupne lastnosti.



Slika 68: Izdelava stolpcev v gradniku *DataGridView* in nastavitev lastnosti.

V primeru, da smo sliko postavili v datoteko virov (kako to storimo smo pokazali v projektu *Predvajalnik glasbe in videa*), pa smo to storili pomotoma oz. jo želimo od tam odstraniti, to na tem mestu ni mogoče – odstranimo jo tako, da v *Solution Explorerju* poiščemo datoteko *Resources.resx*. Dvokliknemo na to datoteko in prikaže se nam njena vsebina - odstranimo vse nezaželene vnose.

Slike v gradniku *DataGridView* bomo prikazovali z desnim klikom miške, ugibali pa jih bomo z levim klikom. Za naključno razporeditev slik bomo poskrbeli s pomočjo tabele celih števil, katere vsebina bo predstavljala indekse slik, ki bodo v posameznih celicah gradnika *DataGridView*. Potrebovali tudi spremenljivki tipa *DateTime*, s pomočjo katerih bomo merili čas, ki ga bo tekmovalec potreboval za uspešno opravljeno nalogo. V konstruktrju obrazca poskrbimo še za začetni izgled igre, dodajmo pa še dve odzivni metodi, s katerima spreminjamo izgled gumba za začetek igre.

```
Public partial class Fspomin : Form
{
    int[] tabStevil;//deklaracija tabele naključnih slik
    bool prikaziSliko;
    int indeksDrugeSlike; //indeks druge slike v tabeli celih števil
    int vPrva, sPrva;//številka vrstice in stolpca za prvo sliko
    int zadetihParov;//število zadetih parov
    Random naklj=new Random();//generator naključnih števil
    DateTime zacetek, konec;
    public Fspomin()
    {
        InitializeComponent();
        //položaj okna bo na začetku na sredini zaslona
        this.StartPosition = FormStartPosition.CenterScreen;
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        this.Height = 427;//začetna širina obrazca
        //začetni stil obrazca
        this.FormBorderStyle = FormBorderStyle.FixedToolWindow;
        dataGridView1.Width = 402;//začetna širina gradnika DataGridView
        //začetna višina stolpca gradnika DataGridView
        dataGridView1.RowTemplate.Height = 100;
        lTipka.Text = "";//skrijem oznako za oznako tipke
        //vse slike so nazačetku bele: bela slika ima indeks 8
        for (int i=0;i<4;i++) //nove vrstice z belimi slikami (ozadjem)
            dataGridView1.Rows.Add(imageList1.Images[8],
imageList1.Images[8], imageList1.Images[8], imageList1.Images[8]);
        tBNavodila.Text="KRATKA NAVODILA:\r\n\r\nOdkriti moraš 8 parov enakih
slik. Z desno tipko odkriješ prvo sliko, nato pa z levo ugibaš njen
par!\r\n\r\nZa začetek klikni gumb 'Začni'! ";
        tBNavodila.ReadOnly = true;
        /*onemogočimo klikanje na gradnik DataGridView dokler ni kliknjen
         gumb Začetek*/
        dataGridView1.Enabled = false;
    }
    private void bZacetek_MouseEnter(object sender, EventArgs e)
    {
        bZacetek.BackColor = Color.DarkSlateBlue;
        bZacetek.ForeColor = Color.White;
    }
```

```
    private void bZacetek_MouseLeave(object sender, EventArgs e)
    {
        bZacetek.BackColor = Color.NavajoWhite;
        bZacetek.ForeColor = Color.SteelBlue;
    }
}
```

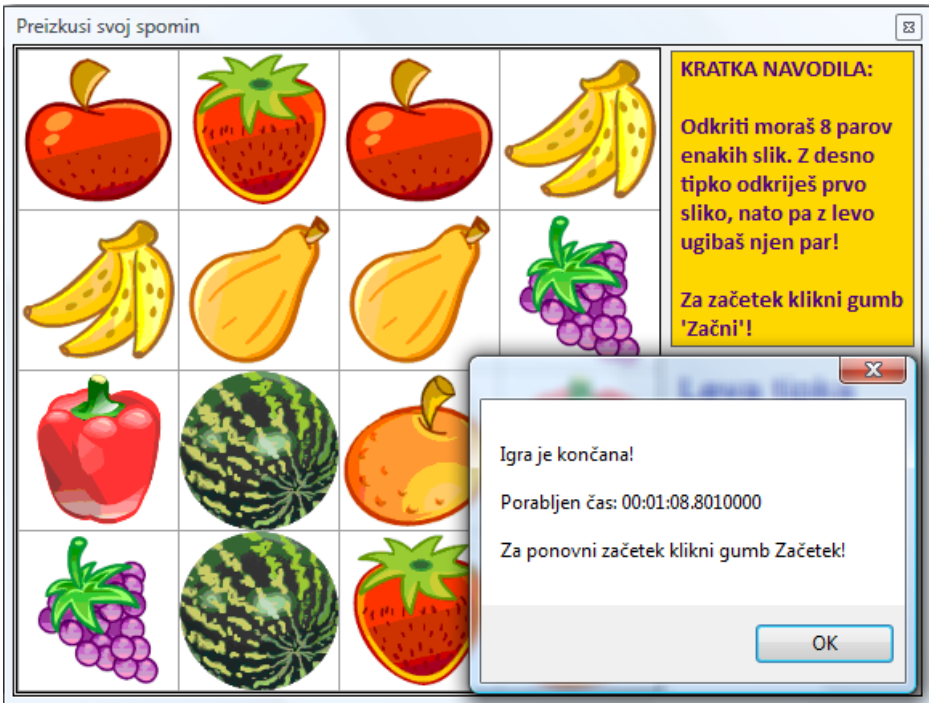
Odzivna metoda ob kliku na gumb začetek poskrbi za naključno razporeditev slik.

```
private void bZacetek_Click(object sender, EventArgs e)
{
    RazporediSlike();
}
private void RazporediSlike()
{
    prikaziSliko = true;/*indikator da je na vrsti izbira slike (desna
                          tipka)*/
    /*inicializiramo tabelo 16 naključnih števil (število se ponovi 2x)*/
    tabStevil = new int[16];
    for (int i = 0; i < 16; i++)
        tabStevil[i] = -1; //tabelo napolnimo z števili -1
    for (int i=0;i<8;i++)
    {
        int stCelice1; //določimo številko prve še proste celice
        do
        {
            stCelice1=naklj.Next(16);//naključno celo število med 0 in 15
        } while (tabStevil[stCelice1]!=-1);
        int stevilo;
        do  /*naključno številko med 0 in 7, ki še ni v nobeni celici*/
        {
            stevilo = naklj.Next(8);/*naključno število med vključno 0 in
                                      vključno 7*/
        } while (tabStevil.Contains(stevilo));
        tabStevil[stCelice1] = stevilo;
        int stCelice2;//določimo številko druge še proste celice
        do
        {
           stCelice2 = naklj.Next(16);//naključno celo število med 0 in 15
        } while (tabStevil[stCelice2]!=-1);
        tabStevil[stCelice2]=stevilo;//število zapišemo v tabelo
    }
    dataGridView1.Enabled = true; //klik na gradnik DataGridView je mogoč
    zadetihParov = 0;//število zadetkov je na začetku 0
    lTipka.Text = "Desna tipka";//besedilo na na oznaki
    bZacetek.Visible = false;//skrijemo gumb Začetek
    dataGridView1.Focus();//gradnik DataGridView je aktiven gradnik
    zacetek = DateTime.Now;//trenutni čas
}
```

Slike bomo odkrivali z desnim oziroma levim klikom miške, zato potrebujemo odzivno metodo dogodga *CellMouseClick*.

```
//desni klik v okno prikaže prvo sliko
private void dataGridView1_CellMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
{
    if (e.Button == MouseButtons.Right)//pritisnjen desni gumb miške
    {
        if (prikaziSliko)//če je na vrsti desna tipka
        {
            vPrva = e.RowIndex; //shranimo indeks vrstice za prvo sličico
            sPrva = e.ColumnIndex;//še indeks stolpca za prvo sličico
            dataGridView1.Rows[vPrva].Cells[sPrva].Value =
                 imageList1.Images[(tabStevil[ vPrva * 4 + sPrva])];
            prikaziSliko = false;/*indikator, da je na vrsti ugibanje
                                   slike (leva tipka)*/
            lTipka.Text = "Leva tipka";/*navodilo uporabniku, katero
                                         tipko naj pritisne*/
        }
    }

    /*če je na vrsti levi gumb in če je ta pritisnjen in slika v celici
       Ni bela (indeks slike je 8) */
    else if (!prikaziSliko && e.Button == MouseButtons.Left &&
              dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value !=
              imageList1.Images[8])
    {
        //glede na izbiro celice izračunamo indeks druge slike
        indeksDrugeSlike = e.RowIndex * 4 + e.ColumnIndex;
        /*v izbrani celici prikažemo sliko, ki ji ustreza glede na
          številko v tabeli tabStevil*/
        dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value =
          imageList1.Images[(tabStevil[indeksDrugeSlike])];
        //indikator da je na vrsti izbira slike (desna tipka)
        prikaziSliko = true;
        //Preverimo, če sta sliki enaki
        if (tabStevil[vPrva * 4 + sPrva] == tabStevil[indeksDrugeSlike])
        {
            zadetihParov++;//sliki sta enaki, povečamo števec zadetkov
            if (zadetihParov == 8)//če so vsi pari odkriti
            {
                konec = DateTime.Now; //trenutni čas
                MessageBox.Show("Igra je končana!\n\nPorabljen čas: "+(konec-
          zacetek).ToString()+"\n\nZa ponovni začetek klikni gumb Začetek!");
                //Pobrišemo vsebino gradnika DataGridView
                dataGridView1.Rows.Clear();
                //Vse slike so na začetku bele
                for (int i = 0; i < 4; i++)
```

```
                    dataGridView1.Rows.Add(imageList1.Images[8],
    imageList1.Images[8], imageList1.Images[8], imageList1.Images[8]);/*nova
                                                                    vrstica */
                RazporediSlike();//nova razporeditev slik
                bZacetek.Visible = true;//skrijemo gumb
            }
        }
        //če izbrani sličici nista enaki, ju moramo pobisat
        else
        {
            MessageBox.Show("Poskusi znova!","Izbira
               napačna",MessageBoxButtons.OK,MessageBoxIcon.Warning);
            //skrijem prvo sliko: slika je zopet bela
            dataGridView1.Rows[vPrva].Cells[sPrva].Value =
               imageList1.Images[8];
            //skrijemo drugo sliko: slika je zopet bela
            dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value =
               imageList1.Images[8];
            lTipka.Text = "Desna tipka";
        }
    }
}
```



Slika 69: Končni izgled programa *Spomin*!

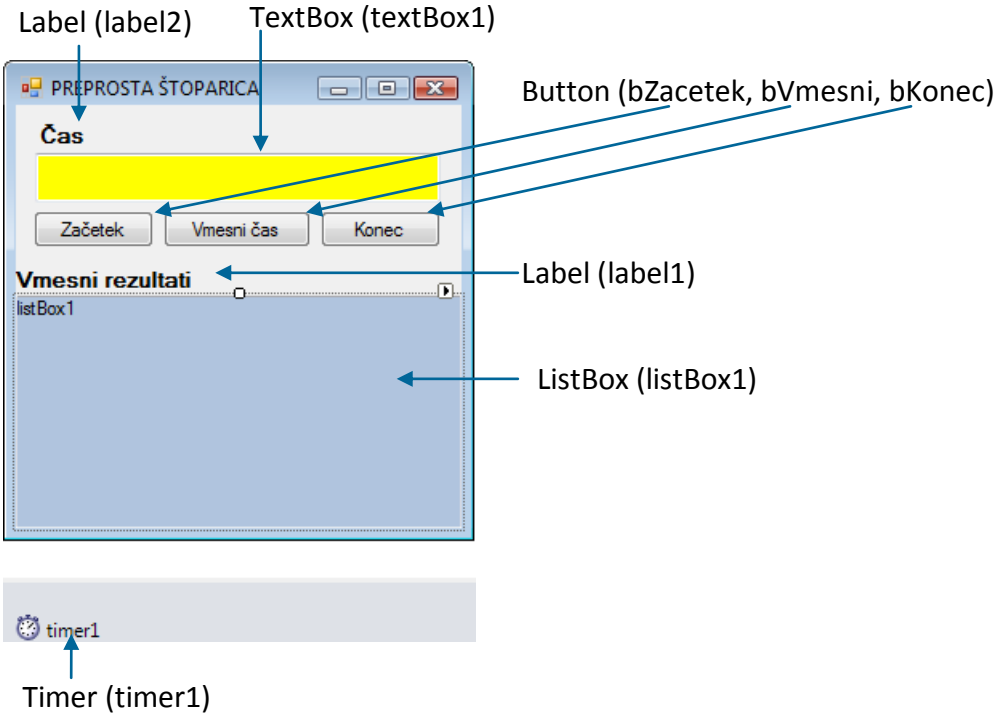
Dogodki, ki jih sproži ura (gradnik *Timer*) in metode za delo s časom

Gradnik *Timer* (programska ura) je namenjen generiranju dogodkov v določenih časovnih intervalih. V oknu *Toolbox* se nahaja v skupini *Components*. Ta gradnik je eden izmed številnih nevizuelnih gradnikov. Ko ga postavimo na obrazec, se avtomatično namesti v posebno polje pod samim obrazcem. Gre pač za gradnik, ki na samem obrazcu ne bo viden, bo pa opravljal neko funkcijo.

Gradnik ima za razliko od ostalih gradnikov le nekaj lastnosti, od katerih sta poleg imena pomembni le dve: to sta lastnosti *Enabled* in *Interval*. Lastnost *Enabled* je privzeto postavljena na *False*. Če hočemo torej gradnik postaviti v operativno funkcijo, mu moramo lastnost *Enabled* postaviti na *True*. Z lastnostjo *Interval* pa nastavimo časovni interval v milisekundah, ki bo pretekel med posameznima dogodkoma, ki jih bo sprožil ta gradnik. Vrednost 1000 npr. pomeni vsako sekundo, vrednost 100 eno desetinko sekunde ...

Gradniku *Timer* lahko v oknu *Properties→Events* priredimo odziv na en sam dogodek. To je dogodek *Tick*, ki se zgodi vsakokrat, ko preteče čas, podan z lastnostjo *Interval*.

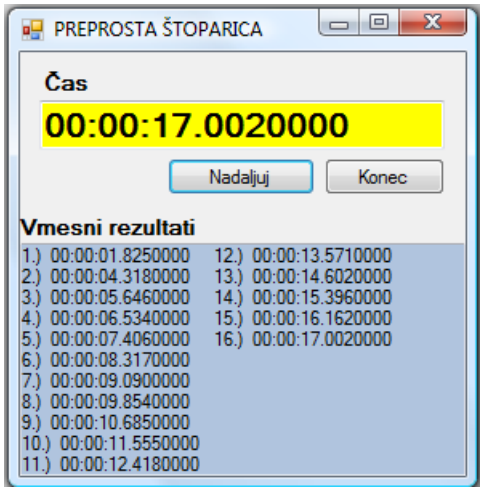
Zgradimo projekt, v katerem bomo predstavili preprosto štoparico. V projekt moramo vključiti nevizuelni gradnik *Timer*. Izgled obrazca naj bo takle:



Slika 70: Gradniki na obrazcu za prikaz štoparice.

Za merjenje časa bomo uporabili dva objekta tipa *DateTime* z imenoma *Zacetek* in *Konec*. Inicializirali ju bomo ob kliku na gumba *Zacetek* in *Konec*. Klik na gumb *Začetek* predstavlja

začetek merjenja časa, ob kliku na gumb *Vmesni čas* se pretečeni čas zapiše v gradnik *ListBox*, obenem pa se prikazovanje časa v gradniku *TextBox* zaustavi za toliko časa, dokler ponovno ne pritisnemo tega gumba. Ob kliku na gumb *Konec* se merjenja časa dokončno zaustavi.



Slika 71: Prikaz delovanja štoparice.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    DateTime Start; //objekt, ki bo hranil čas začetka merjenja
    DateTime End;   //objekt, ki bo hranil končni čas
    //objekt tipa TimeSpan potrebujemo za merjenje časovnega intervala
    TimeSpan izmerjeniCas;
    private void Form1_Load(object sender, EventArgs e)
    {
        bVmesni.Visible = false;
        bKonec.Visible = false;
    }
}
```

V odzivni metodi dogodka Load smo poskrbeli, da sta vmesni in končni čas na začetku nevidna

Potrebujemo še logično spremenljivko, ki bo med delovanjem programa določala, ali naj čas v gradniku *textBox1* teče ali pa ne. Odzivna metoda dogodka *Tick* gradnika *Timer* bo poskrbela za prikaz pretečenega časa. Gradniku *Timer* pustimo lastnost *Enabled* kar privzeto, to je *False*.

```
/*spremenljivka casTece določa,ali v gradniku texBox1 čas teče ali ne*/
bool casTece;
int stevec = 1;//števec vmesnih časov
//odzivna metoda timerja
private void timer1_Tick(object sender, EventArgs e)
```

```
{
    /*ob vsakem dogodku tick gradnika Timer se osveži vsebina gradnika
      textBox1*/
    if (casTece)
        textBox1.Text=Convert.ToString(DateTime.Now-Start);
}
```

Napišime še odzivne metode vseh treh gumbov. Ob kliku na gumb *Zacetek* poskrbimo za začetne vrednosti objektov in sprožimo *Timer*. Kliku na gumb *Konec* zaustavi *Timer* in prikaže končni čas, vmesne čase pa sporočamo in jih shranjujemo ob kliku na srednji gumb.

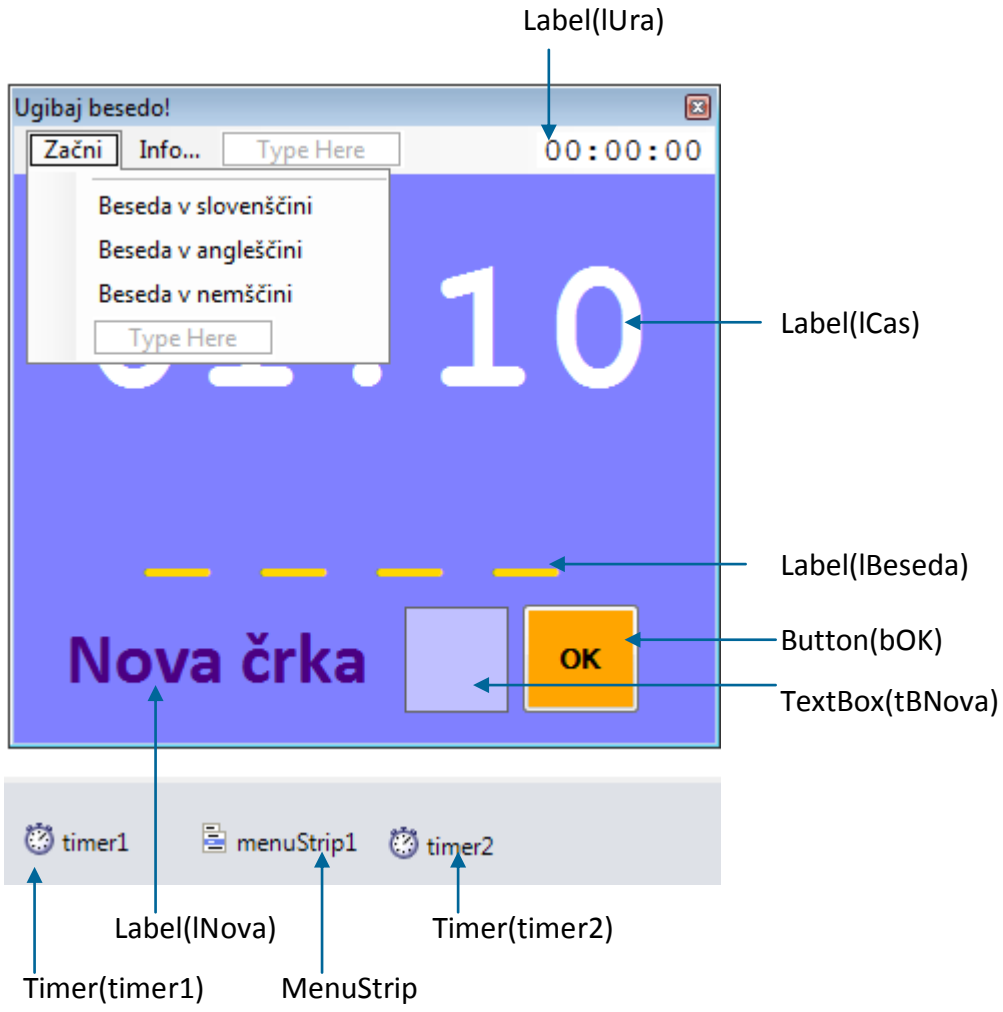
```
//klik na gumb Začetek
private void bZacetek_Click(object sender, EventArgs e)
{
    Start = DateTime.Now;    //začetek merjenja časa
    casTece = true; //v gradniku textBox1 se prikazuje izmerjeni čas
    timer1.Enabled = true;   //poženemo timer
    bVmesni.Visible = true;  //prikažemo gumb za vmesne rezultate
    bKonec.Visible = true;   //prikažemo gumb za končni rezultat
    bZacetek.Visible = false;//skrijemo gumb za začetek
    //na začetku vedno pobrišemo prejšnje rezultate
    listBox1.Items.Clear();
}
//klik na gumb Konec
private void bKonec_Click(object sender, EventArgs e)
{
    End = DateTime.Now; //shranimo trenutni čas
    timer1.Enabled = false;    //zaustavimo timer (merjenje časa)
    //Končni čas zapišemo tudi v ListBox1
    listBox1.Items.Add("Končni čas: " + textBox1.Text);
    bVmesni.Visible = false;  //skrijemo gumb za vmesne rezultate
    bKonec.Visible = false;   //skrijemo gumb za končni rezultat
    bZacetek.Visible = true;  //prikažemo gumb za začetek
}
//klik na gumb Vmesni čas
private void bVmesni_Click(object sender, EventArgs e)
{
    if(casTece==true)
    {
        casTece = false; //zaustavimo prikazovanje časa v gradniku textBox1
        bVmesni.Text = "Nadaljuj";
        //v listBox1 zapišemo vmesni rezultat
        listBox1.Items.Add(stevec + ".)  " + textBox1.Text);
        stevec++;
    }
    else
    {   //nadaljujemo s prikazom časa v gradniku textBox1
        casTece = true;
        bVmesni.Text = "Vmesni čas";
```

```
    }
}
```

Ugibaj besedo

Ugibaj besedo je igra, pri kateri si prijatelj(v tem primeru naš računalnik) izmisli neko besedo, mi pa moramo v določenem času ugotoviti, katera beseda je to. Da bo zadeva bolj zanimiva, bo naš program trojezičen – torej bomo lahko ugibali slovenske, angleške ali nemške besede. Kateri jezik bo izbran, bo ena od možnosti, ki si jo bo iz menija izbral uporabnik.

Kot vedno, si tudi sedaj najprej pripravimo videz uporabniškega vmesnika. Prikazan je na sliki Slika **72:** Gradniki na obrazcu fUgibajBesedo



Slika 72: Gradniki na obrazcu f*UgibajBesedo*

V projektu bomo uporabili dva gradnika *Timer*. Prvi gradnik *Timer* bomo uporabili za iztekajoči se čas, drugega pa za prikaz tekočega časa. V projektu bomo uporabili tudi gradnik *MenuStrip*, s katerim bomo ustvarili preprosti meni. Spomnimo se, da smo s tem gradnikom delali v razdelku *Delo z enostavnimi in sestavljenimi meniji, orodjarna*.

Obrazec poimenujmo *fUgibajBesedo* in nanj postavimo glavni meni (gradnik MenuStrip), ter dva gradnika tipa *Timer*. Gradniku *timer1* priredimo dogodek, ki bo vsako sekundo odšteval čas, prikazan na osrednji oznaki na obrazcu, gradniku *timer2* pa dogodek, ki bo na oznaki v zgornjem desnem robu obrazca vsako sekundo osveževal trenutni čas.

Gradnik	Lastnost	Nastavitev	Opis
Form1	FormBorderStyle	FixedToolWindow	Uporabnik velikosti okna ne more spreminjati
	Name	fUgibajBesedo	Ime obrazca
	StartPosition	CenterScreen	Obrazec se bo odprl na sredini zaslona
lCas	Font	Courier New, Bold, 72	Pisava
	BackColor	128:128;255	Barva ozadja
	ForeColor	White	Bela barva znakov
	Text	01:10	Začetni čas
lBeseda	Font	Courier New, Bold, 36	Pisava
	Text	- - - -	Začetni tekst
tBNova	BackColor	192;192;255	
	BorderStyle	FixedSingle	Enojni okvir okoli polja z besedilom
	CharacterCasing	Upper	Avtomatska pretvorba vnesenih črk v velike črke
	Font	Calibri, Bold, 28	Pisava
	ForeColor	Navy	
	MaxLength	1	Možen je vnos le enega znaka
	TextAlign	Center	Znak bo poravnan na sredini kvadratka

timer1	Interval	1000	Časovni interval je 1 sekunda
timer2	Interval	100	Časovni interval je destinka sekunde
lUra	Font	Courier New, 12	Pisava
	Text	00:00:00	Začetni tekst (ura)
	Anchor	Top,Right	Oznaka je pripeta v zgornji desni rob obrazca
	BackColor	ControlLightLight	Barva ozadja

Tabela 22: Lastnosti gradnikov na obrazcu f*UgibajBesedo*.

V konstruktorju tega obrazca zapišimo kodo, s katero obrazec pred odpiranjem tako pomanjšamo, da je na začetku viden le zgornji del. Opcija *Info…* glavnega menija izpiše sporočilno okno z navodili za uporabo programa.

Pogojni operator '?' je edini operator v C#, ki potrebuje tri sestavne dele: pogoj in dva stavka, ločena s podpičjem. (*pogoj* ? vrednost1 : vrednost2) Logika operatorja je naslednja: če je izpolnjen pogoj, ki se nahaja na levi strani operatorja, vrni vrednost, ki se nahaja na levi strani podpičja (*vrednost1*), sicer pa vrednost na desni strani podpičja (*vrednost12*).

Na začetku iz datoteke *Besede.txt* (ta se mora nahajati v isti mapi kot izvršilna datoteke) preberemo vse besede in jih shranimo v tabelo besede. Pri delu z datoteko v program vključimo tudi varovalni blok.

```
public partial class fUgibajBesedo : Form
{
    string[] besede; //tabela, v katero bomo shranili besede iz datoteke
    public fUgibajBesedo()
    {
        InitializeComponent();
        this.Height = 200;//začetna višina obrazca
        //zaradi dela z datoteko uporabimo varovalni blok
        try
        {
           //vse besede iz datoteke Besede.txt shranimo v tabelo besede
           string[] besede = File.ReadAllLines("Besede.txt");
        }
        catch { MessageBox.Show("Napaka pri delu z datoteko!"); }
    }
    string pomozna="",beseda;
    int sekunde; //spremenljivka, s katero bomo merili čas
    int steviloKorakov = 0;
```

```
    private void timer1_Tick(object sender, EventArgs e)
    {
        sekunde--;
        string TimeInString = "";
        int min = sekunde / 60; //celoštevilsko deljenje, dobimo minute
        int sek = sekunde % 60; //preostanek so sekunde
        if (sekunde >= 0)
        {
            /*če je število minut manjše od 10, nizu dodamo še ničlo,
             sicer min le spremenim v niz*/
            TimeInString = ((min < 10) ? "0" + min.ToString() :
min.ToString());
            /*če je število sekund manjše od 10, pred niz sek dodamo še
             ničlo, sicer niz sek le spremenim v nz*/
            TimeInString += ":"+((sek<10)?"0"+sek.ToString():sek.ToString());
            //dobljeni niz, ki predstavlja tekoči čas prikažem na oznaki
            lCas.Text = TimeInString;
        }
        else
        {
            timer1.Stop();//ko se čas izteče se timer zaustavi
            MessageBox.Show("Žal je čas potekel!\n\nIskana beseda je bila
"+beseda+"\n\nOb kliku na OK se bo program zaprl!");
            Close();
        }
    }
    private void timer2_Tick(object sender, EventArgs e)
    {
        //prikaz tekoče ure v zgornjem desnem delu obrazca
        lUra.Text = DateTime.Now.ToLongTimeString();
    }
    private void infoToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("V meniju klikni 'Začni' in izberi jezik, v katerem
boš ugibal besedo.\n\nProgram izbere naključno besedo iz datoteke Besede.txt,
ki se nahaja v isti mapi, kot program.\n\nČas, ki ga imaš na voljo je odvisen
od izbire jezika!");
    }
}
```

Ko uporabnik izbere eno od treh ponujenih opcij za ugibanje besede, program iz tabele *besede* izbere naključno vrstico in iz nje besedo v ustreznem jeziku. V vsaki vrstici te datoteke je namreč beseda napisana v treh jezikih, vmes pa so znaki podpičje. Najprej je beseda v slovenščini, nato v angleščini in na koncu še v nemščini. Glede na število črk v tej besedi program izračuna velikost obrazca in na sredino obrazca izpiše toliko črtic, kot je dolžina besede.

```
private void Ugibaj(object sender, EventArgs e)
{
```

```
    Random naklj = new Random();
    //iz tabele vzamemo naključno celico
    string celica = besede[naklj.Next(0, besede.Length)];
    /*v nizu celica so 3 besede (v SLO, ANG in NEM), ločene s
      podpičjem. Z metodo Split jih shranimo v tabelo tabBesed*/
    string[] tabBesed = celica.Split(';');
    if (sender == besedaVSlovenščiniToolStripMenuItem)
    {
        beseda = tabBesed[0].ToUpper();//vse črke naj bodo velike
        sekunde = 70;//na voljo imamo 70 sekund
        this.Text = "Ugibaj besedo v slovenščini!";
    }
    else if (sender == besedaVAngleščiniToolStripMenuItem)
    {
        beseda = tabBesed[1].ToUpper();//vse črke naj bodo velike
        sekunde = 80;//na voljo imamo 80 sekund
        this.Text = "Ugibaj besedo v angleščini!";
    }
    else
    {
        beseda = tabBesed[2].ToUpper();//vse črke naj bodo velike
        sekunde = 90;//na voljo imamo 90 sekund
        this.Text = "Ugibaj besedo v nemščini!";
    }
    pomozna = "";
    /*spremenljivka pomozna naj na začetku vsebuje toliko
     presledkov, kot je dolžina niza beseda*/
    for (int i = 0; i < beseda.Length; i++)
        pomozna += " ";
    Prikazi();//klic metode, ki pokaže že ugotovljene znake
    /*gradnike postavimo na sredino okna, ki ne sme biti ožje od 350 px*/
    if (lBeseda.Width < 350)
        this.Width = 350;
    else  //obrazec je za 10 px širši kot je širina besede
        this.Width = lBeseda.Width + 10;
    /*ker je dolžina beseda naključna, moramo vse gradnike postaviti
      na sredino obrazca*/
    lCas.Left = (this.Width - 337) / 2;
    lBeseda.Left = 10 + (this.Width - lBeseda.Width) / 2;
    lNova.Left = (this.Width-(lNova.Width+tBNova.Width + bOK.Width + 5)) / 2;
    tBNova.Left = lNova.Left + lNova.Width + 5;
    bOK.Left = lNova.Left + lNova.Width + bOK.Width + 5;
    tBNova.Focus(); //gradnik za vnos črke postane aktiven
    this.Height = 360; //višina obrazca
    this.CenterToScreen();//okno naj bo na sredini zaslona
    timer1.Start(); //začnemo meriti (odštevati) čas
}
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

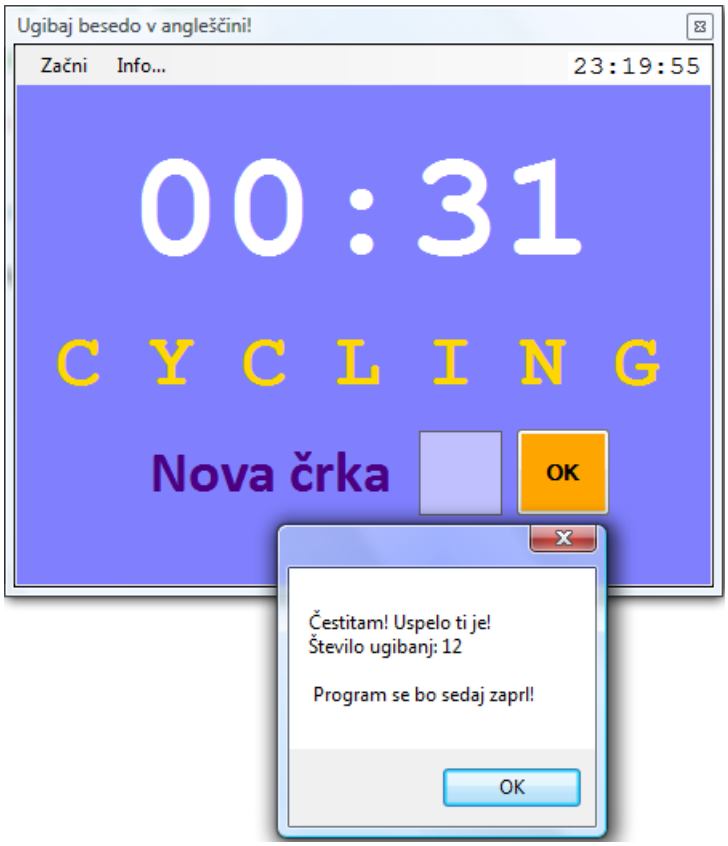
Pod črticami je na sredini obrazca vnosno polje za vnos črke. Možen je vnos le enega znaka, to pa je lahko le črka angleške abecede (dogodek *KeyPress*). Na obrazcu je še gumb za potrditev vnosa. Odzivna metoda tega gumba je namenjena preverjanju pravilnosti vnesene črke in prikazu doslej uganjenih črk skrite besede. V ta namen bomo napisali svojo metodo z imenom *Prikazi*.

```
//odzivna metoda za kontrolo vnesenih znakov
private void tBNova_KeyPress(object sender, KeyPressEventArgs e)
{
    //dovoljene tipke so le tipke z znaki
    if (!(e.KeyChar >= 'A' || e.KeyChar <= 'Z'))
        e.Handled = true;
    else bOK.Focus();//gradnik za vnos črke ostane aktiven
}
//odzivna metoda dogodka Click gumba za potrditev vnesenega znaka
private void bOK_Click(object sender, EventArgs e)
{
    if (tBNova.Text.Length > 0)//preverimo, če je uporabnik vnesel črko
    {
        steviloKorakov++;
        /*vnos shranimo v spremenljivko tipa char(znak)*/
        char znak = Convert.ToChar(tBNova.Text);
        string zac = "";//začasna spremenljivka
        /*preverimo, če se ta črka nahaja v iskani besedi: če se, jo
        (jih) umestim na pravilno mesto*/
        for (int i = 0; i < beseda.Length; i++)
        {
            if (beseda[i] == znak)
                zac = zac + znak;
            else
                zac = zac + pomozna[i];
        }
        /*spremenljivka pomozna vsebuje doslej pravilno ugotovljene
        znake, ostali znaki so presledki*/
        pomozna=zac;
        Prikazi();//klic metode, ki pokaže že ugotovljene znake
        tBNova.Clear(); //pobrišem črko
        tBNova.Focus(); //aktiven gradnik je gumb OK
        if (pomozna==beseda)
        {
            timer1.Enabled = false;//zaustavim čas odštevanja
            MessageBox.Show("Čestitam! Uspelo ti je!\nŠtevilo ugibanj:
"+steviloKorakov+"\n\n Program se bo sedaj zaprl!");
            Close();
        }
    }
}
//metoda, ki prikaže že ugotovljene znake
private void Prikazi()
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
{
    string zac = "";//začasen string
    for (int i = 0; i < beseda.Length; i++)
    {
        if (pomozna[i]!=' ')//če je dosedanji znak presledek
            //spremenljivki zac dodam ta znak in še presledek
            zac=zac+beseda[i]+" ";
        else //sicer pa dodam podčrtaj in presledek
            zac=zac+"_"+" ";
    }
    lBeseda.Text = zac;//niz zac prikažem na oznaki
}
```

Še prikaz delujočega projekta



Slika 73: Ugibanje besede.

Program bi lahko nadgradili tudi tako, da bi uporabniku omogočili, da izbira datoteko, kjer so napisane besede. Prav tako pa bi mu lahko dodali še enostaven urejevalnik besed v datoteki s katerim bi kar programsko dodajali (in brisali) besede. Z doselj naučenim to zagotovo ne bo predstavljajo prehudega izziva.

Dinamično ustvarjanje okenskih gradnikov in njihovih dogodkov

Dinamično ustvarjanje okenskih gradnikov

Okenske objekte, ki smo jih uporabljali v naših aplikacijah, smo doslej vselej ustvarjali statično, to je v fazi načrtovanja. V oknu *ToolBox* smo izbrali določen gradnik, ga postavili na določeno mesto na obrazcu in s tem ustvarjali nove okenske objekte. Pri pisanju aplikacij pa se pogosto zgodi, da ne vemo v naprej, koliko gradnikov (gumbov, oznak, vnosnih polj…) bomo potrebovali, ali pa bi radi nek okenski objekt ustvarili šele v fazi izvajanja programa – dinamično. Gradniki, ki se nahajajo na paletah v oknu *ToolBox*, so v bistvu razredi, zato lahko iz njih tvorimo objekte kadarkoli. Vemo, da novo *primerek* (nov objekt) nekega razreda ustvarimo s pomočjo rezervirane besede *new* (to smo spoznali že pri učenju osnov jezika *C#*. Na enak način lahko tvorimo tudi nove objekte tipa *Button*, *Label*, *TextBox*, *ComboBox*…

```
Button gumb = new Button();          //nov objekt tipa Button
Label mojaLabela = new Label();      //nov objekt tipa Label – oznaka
TextBox sporocilo = new TextBox();   //Nov objekt tipa TextBox
ComboBox cBImena = new ComboBox();   //Nov objekt tipa ComboBox
```

S stališča programerja je uporaba besedice *new* pogosto že kar avtomatična – omogoči pač izdelavo novega objekta. Vendar pa je ustvarjanje objekta v resnici dvofazni proces. Operacija *new* mora najprej *alocirati* (zasesti) nekaj prostega pomnilnika na kopici – na ta del ustvarjanja novega objekta nimamo prav nobenega vpliva. Drugi del operacije *new* pa je v tem, da mora zasedeni pomnilnik pretvoriti v objekt – mora *inicializirati* objekt. To drugo fazo operacije *new* pa seveda lahko kontroliramo z uporabo konstruktorja. V zgornjih primerih smo nove objekte izdelali s pomočjo privzetih konstruktorjev, ki nimajo parametrov. Ko je nov objekt ustvarjen, lahko do njegovih članov (polj, metod, ..) dostopamo z uporabo operatorja pika, npr.:

```
sporocilo.Text = "Tekstovno sporočilo!";
gumb.BackColor = Color.Red;
mojaLabela.Visible = false;
```

Za vsak vizuelni objekt, ki ga ustvarimo dinamično, pa moramo še določiti, na kateri gradnik ga želimo postaviti in kakšen je njegov položaj na tem gradniku. Če je obrazec še prazen, in postavljamo nanj dinamično ustvarjen gradnik, to storimo takole

```
//nov objekt tipa Button – ime objekta je gumb
Button gumb = new Button();
```

```
gumb.Parent = this; //gumb postavimo na trenutni na obrazec
gumb.Top = 200; //oddaljenost gumba od vrha obrazca
gumb.Left = 200; //oddaljenost gumba od levega roba obrazca
gumb.BringToFront();//gumb postavimo v ospredje
```

Ključni stavek pri tem je

```
gumb.Parent = this;
```

s katerim določimo, na katerem gradniku leži naš objekt gumb. Ker smo znotraj razreda *Form1*, nam *this* seveda označuje prav ta obrazec, ki nastaja.

Oglejmo si še primer, ko želimo dinamično izdelan objekt postaviti na nek že obstoječi gradnik na obrazcu. Recimo, da imamo na obrazcu že gradnik Panel z imenom panel1, nanj pa želimo dinamično postaviti nov objekt tipa *TextBox*.

```
//ustvarimo nov objekt tipa TextBox – ime mu je tB
TextBox tB = new TextBox();
/*objekt gumb postavimo na gradnik Panel (plošča) z imenom panel1: gradnik
panel1 mora seveda že obstajati!!!*/
tB.Parent = panel1; //objekt tB je postavljen na ploščo z imenom panel1
tB.Top = 20; //oddaljenost od vrha plošče
tB.Left = 20; //oddaljenost od levega roba plošče
tB.BringToFront();
```

Če so na gradniku, na katerem ustvarjamo nov objekt, še drugi gradniki, z metodo *BringToFront* poskrbimo, da bo nov objekt postavljen v ospredje (na ta način zagotovo ne bo skrit za kakim drugim gradnikom!).

Dinamično ustvarjanje okenskih dogodkov

Vsi dinamično ustvarjeni okenski gradniki že poznajo dogodke, ki jih pri statično ustvarjenih objektih najdemo v oknu *Properties→Events*. Njihova implementacija je pri statičnih gradnikih enostavna: v fazi načrtovanja projekta v oknu *Properties→Events* dvokliknemo na ustrezni dogodek in razvojno okolje nam ustvari ogrodje odzivne metode, v katero le še zapišemo ustrezno kodo. Pri dinamično ustvarjenih okenskih gradnikih pa moramo to storiti programsko. To storimo v dveh korakih:

- Najprej moramo poimenovati novo odzivno metodo za dogodek dinamično ustvarjenega gradnika in jo uvrstiti v seznam ostalih metod, ki obdelujejo dogodke. Tem metodam pravimo tudi obdelovalci dogodkov - *event handlers*.

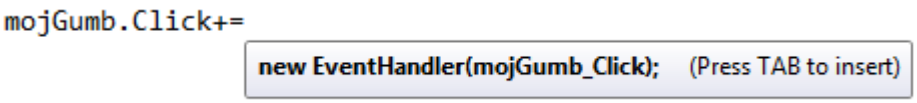
```
/*dinamično ustvarjenemu gradniku mojGumb določimo ime odzivne metode
dogodka Click: odzivno metodo poimenujmo mojGumb_Click*/
mojGumb.Click += new EventHandler(mojGumb_Click);
```

Popolnoma isto se zgodi tudi pri ustvarjanju odzivnih metod gradnikom, ki jih na obrazec postavljamo pri gradnji projekta. A v tem primeru je razvojno okolje tisto, ki poskrbi za dodajanje metode v seznam, kar se odrazi v datoteki *.designer.cs*

- Odzivno metodo *mojGumb_Click* moramo nato še napisati. Tako kot vse odzivne metode, mora tudi ta vsebovati dva parametra: parameter *sender* (pošiljatelj) in parameter tipa *EventArgs* (parameter s pomočjo katerega lahko pridemo do dodatnih informacij o dogodku).

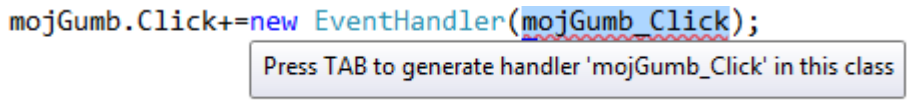
```
public void mojGumb_Click(object sender, EventArgs e)
{
    //stavki, ki naj se izvedejo ob kliku na dinamično ustvarjen gumb
    MessageBox.Show("Ups! KLiknil si me!");
}
```

Pri dinamičnem ustvarjanju okenskih dogodkov pa nam v veliki meri priskoči na pomoč tudi razvojno okolje. Ko namreč začnemo pisati najavo novega dogodka, se v okvirčku pokaže zelo koristna pomoč (zapis *mojGumb.Click+=…* je krajša oblika zapisa *mojGumb.Click=mojGumb.Click + …)*:



Slika 74: Pomoč pri dinamičnem ustvarjanju okenskega dogodka.

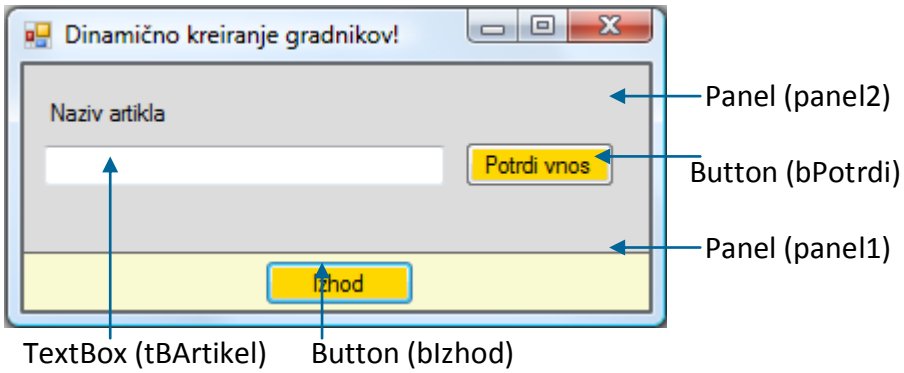
Razvojno okolje ponudi možnost, da najavo novega dogodka zaključimo s pritiskom na tipko tabulator (*Tab*). In še več: če tipko *Tab* pritisnemo, se najava dogodka zapiše do konca (metoda dobi privzeto ime, v našem primeru *mojGumb_Click*), razvojno okolje pa nam ponudi tudi možnost zapisa ogrodja metode, ki jo proži ta dogodek:



Slika 75: Razvojno okolje nam ponudi možnost zapisa ogrodja odzivnega dogodka.

Če torej tipko *Tab* pritisnemo še enkrat, nam razvojno okolje zapiše glavo in ogrodje odzivne metode, mi pa moramo poskrbeti le še za njeno vsebino.

Za vajo ustvarimo obrazec, na katerem bomo vse gradnike (dve plošči, oznako, vnosno polje in dva gumba), ter njihove lastnosti ustvarili dinamično. Dinamično ustvarimo tudi tri dogodke: dogodek *Load* obrazca in dva dogodka *Click* gumbov, ki sta na obrazcu. Končna oblika obrazca naj bo taka:



Slika 76: Dinamično ustvarjeni okenski gradniki na obrazcu.

Vse gradnike bomo ustvraili v metodi *Load*, torej preden se obrazec sploh prikaže. Tako bomo takoj videli vse, tudi dinamično ustvarjene gradnike. Ker v sami kodi ni nič novega, jo takoj navedimo v celoti.

```
public Form1()
{
    InitializeComponent();

    /*Ustvarjanje odzivne metode dogodka Load obrazca Form1 - ime metode bo
      Form1_Load*/
    this.Load += new EventHandler(Form1_Load);
    this.Text = "Dinamično ustvarjanje gradnikov!"; //napis na obrazcu
    this.Width = 330;
    this.Height = 160;
}

private void Form1_Load(object sender, EventArgs e)
{
    //Prva plošča
    Panel panel1 = new Panel();
    panel1.Parent = this; //plošča je postavljena na obrazec
    panel1.Dock = DockStyle.Bottom;  //plošča je prilepljena na dno obrazca
    panel1.Height = 30; //višina lošče
    panel1.BackColor = Color.LightGoldenrodYellow; //barva ozadja
    //plošča ima enojni okvir
    panel1.BorderStyle = BorderStyle.FixedSingle;

    //Druga plošča
    Panel panel2 = new Panel();
    panel2.Parent = this;    //plošča je postavljena na obrazec
    //plošča je raztegnjena čez preostali del obrazca
    panel2.Dock = DockStyle.Fill;
    panel2.BackColor = Color.Gainsboro;  //barva ozadja
```

```
    //plošča ima enojni okvir
    panel2.BorderStyle = BorderStyle.FixedSingle;
    //Oznaka
    Label mojaLabela = new Label();
    mojaLabela.Parent = panel2;
    mojaLabela.Text = "Naziv artikla"; //Napis na oznaki
    mojaLabela.Top = 17;  //oddaljenost od zgornjega roba obrazca
    mojaLabela.Left = 10; //oddaljenost od levega roba obrazca
    //Vnosno polje
    TextBox tBArtikel = new TextBox();
    tBArtikel.Parent = panel2;//TextBox  je postavljen na ploščo panel2
    tBArtikel.Width = 200; //širina vnosnega polja
    tBArtikel.Top = 40;
    tBArtikel.Left = 10;
    //Prvi gumb je ob vnosnem polju
    Button bPotrdi = new Button();

    bPotrdi.Parent = panel2; //Gumb  je postavljen na ploščo panel2
    bPotrdi.Top = 38;
    bPotrdi.Left = 220;
    bPotrdi.Text = "Potrdi vnos";
    bPotrdi.BackColor = Color.Gold;  //barva ozadja
    /*Napoved dogodka Click gumba bPotrdi - ime odzivne metode bo
      bPotrdi_Click*/
    bPotrdi.Click += new EventHandler(bPotrdi_Click);
    Button bIzhod = new Button(); //Drugi gumb je na spodnji plošči
    bIzhod.Parent = panel1; //Gumb je postavljen na ploščo panel1
    bIzhod.Top = 3;
    bIzhod.Left = 120;
    bIzhod.Text = "Izhod";
    bIzhod.BackColor = Color.Gold;  //barva ozadja
    //Napoved dogodka Click gumba bIzhod - ime odzivne metode bo bIzhod_Click
    bIzhod.Click += new EventHandler(bIzhod_Click);
}

public void bPotrdi_Click(object sender, EventArgs ee)
{
    /*stavki, ki se izvedejo ob kliku na gumb Potrdi
      npr. zapis artikla v datoteko ...*/
}

public void bIzhod_Click(object sender, EventArgs ee)
{
    if (MessageBox.Show("Zaključek programa?", "OPOZORILO!",
        MessageBoxButtons.OKCancel) == DialogResult.OK)
        Close();
}
```

Povzetek

V poglavju so zapisane le osnove izdelave okenskih aplikacij. V primerih smo uporabili večino standardnih gradnikov, spoznali njihove osnovne lastnosti in dogodke. Vemo pa tudi to, kako ustvarimo nove vizuelne objekte in njihove odzivne metode. Pri gradnji aplikacij pa lahko uporabimo tudi številne brezplačne gradnike, ki so dostopni preko spleta in s katerimi bodo naši projekti vizuelno privlačnejši in sodobnejši. Ponudnikov takih gradnikov je veliko npr. https://www.devexpress.com/Products/Free/NetOffer/#winforms

Obdelali smo tudi enega najkompleksnejših gradnikov *DataGridView*, ki ga bomo potrebovali tudi v zadnjem delu učbenika, pri prikazovanju vsebine tabele iz baze podatkov. Naučili smo se tudi uporabljati že izdelana pogovorna okna, ter spoznali dogodke tipkovnice, miške in ure. Na koncu poglavja smo še spoznali, da lahko objekte gradnikov okenskih aplikacij ustvarjamo tudi dinamično, kar razvijalcem projektov omogoča izdelavo bolj kompleksnih rešitev z manj pisanja kode.

Ker so vsi gradniki primerki objektov iz pripravljenih razredov, zapisani primeri omogočajo, da bo objektni pristop k programiranju postal bolj domač. Znanje pa bo potrebno utrditi tako, da boste napisali kar največ uporabnih programov.

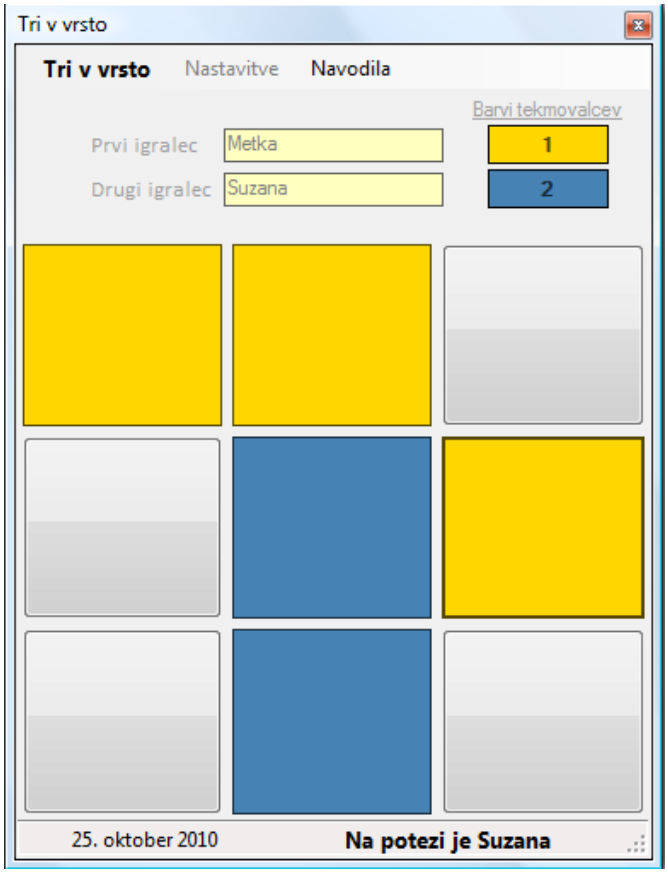
Pridobljeno znanje bomo nadgradili z izdelavo večokenskih aplikacij in vpogledom v naslednja ključna pojma objektno orientiranega programiranja – *dedovanje* in *polimorfizem*.

Kot zaključek spoznavanja osnov izdelave okenskih aplikacij pa sedaj rešimo naš uvodni izziv – program za igranje igrice Tri v vrsto. Seveda pa se bomo tudi tu spoznali s čim novim, uporabili kakšen nov gradnik. A osnovno znanje potrebno za naš program smo si doselej že pridobili.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Tri v vrsto

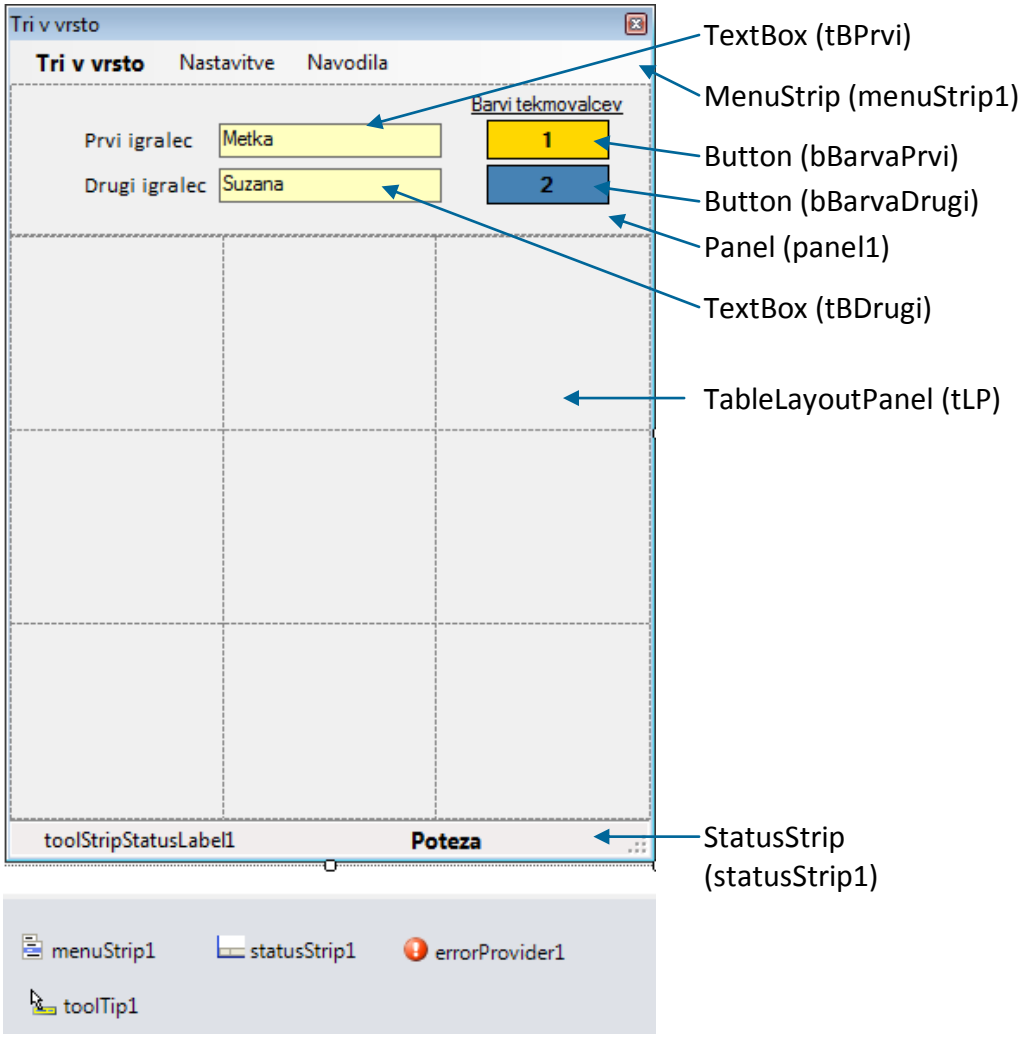
Večino znanja, ki smo ga pridobili pri dosedanjih projektih, torej uporabimo za rešitev igrice *Tri v vrsto*.



Slika 77: Program *Tri v vrsto* med igro.

Ob pripravi programa bomo spoznali gradnik *TableLayoutPanel*, ki predstavlja nekakšno grafično različico dvodimenzionalne tabele. V tem gradniku bomo dinamično ustvarili devet gumbov (objektov tipa *Button*), jim dinamično določili nekatere lastnosti (velikost, barvo ozadja, stil), dinamično pa bomo ustvarili tudi odzivne metode dogodkov *Click*, *MouseEnter* in *MouseLeave*. Seveda bi lahko rešitev naredili tudi tako, da bi uporabili devet statičnih objektov izpeljanih iz gradnika *Button*, a pregled nad njimi bi bil v tem primeru manj pregleden, koda pa zaradi tega daljša. Za vsakokratno preverjanje zmagovalca in preverjanje, ali je igra končana,

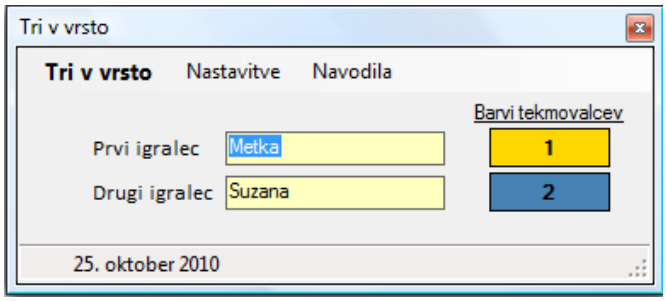
bomo napisali tudi nekaj lastnih metod. V rešitvi bomo uporabili tudi kontrolo uporabnikovih vnosov, njihova pravilnost pa bo pogoj za začetek igre. Oba igralca bosta s pomočjo dialoga za izbiro barve izbrala svojo igralno barvo.



Slika 78: Gradniki obrazca *Tri v vrsto*.

Na prazen obrazec postavimo gradnike v naslednjem zaporedju: *MenuStrip* in *Panel* (*Dock=Top,* nanj pa postavimo tri oznake, dve vnosni polji in dva gumba), *StatusStrip* (na njem sta dva predalčka: prvemu nastavimo lastnost *AutoSize* na *False* in mu določimo širino 150, drugemu pa nastavimo lastnost *Spring* na *True*, lastnost *Text* pa *Poteza*), na koncu pa še gradnik *TableLayoutPanel* (*Dock=Fill*). Gradnik privzeto že vsebuje dva stolpca in dve vrstici: poimenujmo ga *tLP* in mu dodajmo še po en stolpec in eno vrstico (kliknemo na puščico v zgornjem desnem kotu gradnika, nato pa *Add Column* in *Add Row*). Stolpcem in vrsticam nastavimo enake širine in višine (klik na lastnost *Columns* oz. *Rows*, nato pa v oknu, ki se odpre, vsaki postavki posebej določimo lastnost *Percent* na 33.33 procentov!).

V konstruktor obrazca bomo zapisali kodo tako, da bo ob zagonu projekta viden le zgornji del obrazca.



Slika 79: Začetna velikost obrazca *Tri v vrsto*.

Od uporabnika programa namreč najprej zahtevamo, da vnese imeni obeh igralcev, ki imata tudi možnost izbire svoje barve. Izbrana barva (ali pa privzeta barva) je prikazana v okvirčkih s številkama obeh tekmovalcev. V glavnem meniju je tudi opcija *Navodila*, ki v sporočilnem oknu prikaže kratka navodila za samo igro. Na dnu obrazca je statusna vrstica s tekočim datumom. Igro pričnemo s klikom na *Tri v vrsto*.

Med samo igro je v statusni vrstici ves čas navodilo, kateri igralec je trenutno na vrsti (glej napis desno spodaj na sliki Slika **77:** Program *Tri v vrsto* med igro. Ob zmagi enega izmed igralcev (tri enake barve v vrsti, stolpcu ali diagonali), se v sporočilnem oknu izpiše ustrezno obvestilo. Le-to se izpiše tudi v primeru neodločenega rezultata (vsa okna so pobarvana, zmagovalca pa ni!). Ko uporabnik zapre sporočilno okno, se avtomatično prične nova igra.

V konstruktorju obrazca ustvarimo 9 gumbov in jih dodamo v gradnik *TableLayoutPanel*. Za vsakega od gumbov napovemo še odzivne metode za tri dogodke.

```
bool igralec = true;//igro začne prvi igralec
ColorDialog cD = new ColorDialog();//nov objekt za dialog izbire barve
//konstruktor obrazca
public Form1()
{
    InitializeComponent();
    //v okencu gradnika StatusStrip prikažemo datum
    toolStripStatusLabel1.Text = DateTime.Now.ToLongDateString();
    //izdelava gumbov
    for (int i = 0; i < 9; i++)
    {
        Button button = new Button();//nov objekt tipa Button
        button.Text = ""; //gumb je brez napisa
        //gumb dodamo v gradnik TableLayoutPanel
        tLP.Controls.Add(button);
        /*do vsakega gumba v gradniku tLP dostopamo s pomočjo lastnosti
          Controls, oglatega opklepaja in indeksa tega gumba (indeksi se
          začenjajo z 0!)*/
```

```
        //gumb zasede celotno celico tabele
        tLP.Controls[i].Dock = DockStyle.Fill;
        //upravljalcu dogodkov dodajmo dogodek Click posameznega gumba
        tLP.Controls[i].Click += new System.EventHandler(OK_Click);
        //upravljalcu dogodkov dodajmo dogodek MouseEnter posameznega gumba
        tLP.Controls[i].MouseEnter += new System.EventHandler(OK_Enter);
        //upravljalcu dogodkov dodajmo dogodek MouseLeave posameznega gumba
        tLP.Controls[i].MouseLeave += new System.EventHandler(OK_Leave);
    }
    this.Height = 160;//Začetna velikost obrazca
    /*vsebina predalčka gradnika StatusStrip, ki označuje kateri igralec je
      na potezi, je na začetku prazna*/
    toolStripStatusLabel2.Text = "";
}
```

Za uspešen začetek igre morata igralca v vnosni polji vpisati svoja imena. Ob kliku na možnost *Tri v vrsto* glavnega menija se igra lahko začne.

```
private void tB_Validating(object sender, CancelEventArgs e)
{
    KontrolaImena();//metoda za preverjanje vnosa imen obeh tekmovalcev
}
private bool KontrolaImena()
{
    bool bStatus = true;
    if (tBPrvi.Text == "")
    {
        /*Če uporabnik ne bo vnesel imena, se bo ob imenu pokazala rdeča
          oznaka s klicajem. Če se bomo z miško postavili na to oznako, se
          pod njim v okvirčku pojavi besedilo, ki ga zapišemo kot drug
          parameter metode SetError */
        errorProvider1.SetError(tBPrvi, "Vnesi ime prvega igralca");
        bStatus = false;
    }
    else errorProvider1.SetError(tBPrvi, "");/*ko je ime vneseno, odstranimo
                                               sporočilo o napaki*/
    if (tBDrugi.Text == "")
    {
        errorProvider1.SetError(tBDrugi, "Vnesi ime drugega igralca");
        bStatus = false;
    }
    else errorProvider1.SetError(tBDrugi, "");
    return bStatus;//metoda vrne false, če imena tekmovalcev niso vnesena
}
//odzivna metoda za začetek igre
private void triVVrstoToolStripMenuItem_Click(object sender, EventArgs e)
{
    //če sta imeni tekmovalcev vneseni, lahko začnemo z igro
```

```
    if (KontrolaImena())
    {
        panel1.Enabled = false;
        this.Height = 500;
        //med samo igro ni možno spreminjati barve posameznega tekmovalca
        nastavitveToolStripMenuItem.Enabled = false;
        Zacetek();
    }
}
//lastna metoda za določanje začetnega stanja
private void Zacetek()
{
    igralec = true;//začenja prvi igralec
    toolStripStatusLabel2.Text = "Na potezi je " + tBPrvi.Text;
    //vsem gumbom gradnika TablePanelLayout določimo začetno barvo in stil
    for (int i = 0; i < 9; i++)
    {
        (tLP.Controls[i] as Button).FlatStyle = FlatStyle.System;
        tLP.Controls[i].BackColor = Color.FromArgb(0,0,0,0);
    }
}
```

Igralca lahko določita svoji barvi gumbov:

```
//odzivna metoda za izbiro barve prvega igralca
private void barvaPrvegaIgralcaToolStripMenuItem_Click(object sender,
EventArgs e)
{
    /*če prvi igralec izbere barvo in klikne gumb OK, mu določimo novo
      barvo*/
    if (cD.ShowDialog() == DialogResult.OK)
    {
        if (bBarvaDrugi.BackColor != cD.Color)
            bBarvaPrvi.BackColor = cD.Color;
        else MessageBox.Show("Barvi obeh tekmovalcev ne smeta biti enaki!");
    }
}
//odzivna metoda za izbiro barve drugega igralca
private void barvaDrugegaIgralcaToolStripMenuItem_Click(object sender,
EventArgs e)
{
    */če drugi igralec izbere barvo in klikne gumb OK, mu določimo novo
      Barvo*/
    if (cD.ShowDialog() == DialogResult.OK)
    {
        if (bBarvaPrvi.BackColor != cD.Color)
            bBarvaDrugi.BackColor = cD.Color;
        else MessageBox.Show("Barvi obeh tekmovalcev ne smeta biti enaki!");
    }
```

```
}
```

Dodamo še odzivno metodo postavki *Navodila* glavnega menija. V sporočilnem oknu pojasnimo pravila igre.

```
private void navodilaToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("Vnesi imeni obeh tekmovalcev, v meniju 'Nastavitve'
izberi barvo posameznega tekmovalca, nato pa v meniju klikni 'Tri v
vrsto'!\r\n\r\nBarvo igralca lahko izbereš tudi s klikom na gumb 1 oz
2!","Kratka navodila",0,MessageBoxIcon.Information);
}
```

Odzivne metode za vstop in izstop miškinega kazalca v območje gumba, ter metoda za klik na gumb, so enake za vse gumbe v gradniku *TableLayoutPanel*. Pri tem bmo uporabili parameter *sender* odzivnih metod, ki smo ga spoznali v poglavju *Parametri odzivnih metod*.

```
//dogodek se izvede, ko z miško vstopimo v območje nekega gumba
public void OK_Enter(object sender, EventArgs ee)
{
    //ko z miško vstopimo v območje gradnik, se oblika gumba spremeni
    (sender as Button).FlatStyle=FlatStyle.Popup;
}
//dogodek se izvede, ko z miško zapustimo območje nekega gumba
public void OK_Leave(object sender, EventArgs ee)
{
    /*če z miško zapustimo gumb, ki še ni pobarvan, bo njegova oblika taka,
    kot pred vstopom*/
    if ((sender as Button).BackColor==Color.FromArgb(0,0,0,0))
        (sender as Button).FlatStyle=FlatStyle.System;
}

public void OK_Click(object sender, EventArgs ee)
{
    if ((sender as Button).BackColor == Color.FromArgb(0, 0, 0, 0))
    {
        Color barva;
        if (igralec)//če gre za prvega igralca
        {
            /*barva za prvega igralca je določena z ozadjem gumba
              bBarvaPrvi*/
            barva = bBarvaPrvi.BackColor;
            (sender as Button).BackColor = barva;
            /*v predalček gradnika StatusStrip zapišemo, da je na potezi
              drugi igralec*/
            toolStripStatusLabel2.Text = "Na potezi je " + tBDrugi.Text;
        }
        else
```

```
        {
            /*barva za drugega igralca je določena z ozadjem gumba
              bBarvaDrugi*/
            barva = bBarvaDrugi.BackColor;
            (sender as Button).BackColor = barva;
            /*v predalček gradnika StatusStrip zapišemo, da je na potezi
              prvi igralec*/
            toolStripStatusLabel2.Text = "Na potezi je " + tBPrvi.Text;
        }
        //preverimo, če je zmagal igralec, ki je bil pravkar na potezi
        if (PreveriRezultat(barva))
        {
            /*izpraznimo vsebino predalčka gradnika StatusStrip, ki
             označuje kateri igralec je na potezi*/
            toolStripStatusLabel2.Text = "";
            //preverimo, za katerega igralca gre in izpišemo zmagovalca
            if (igralec)preverimo, za katerega igralca gre
                MessageBox.Show("KONEC - Zmagovalec je " + tBPrvi.Text);
            else MessageBox.Show("KONEC - Zmagovalec je " + tBDrugi.Text);
            //določimo začetno velikost obrazca (skrijemo spodnji del)
            this.Height = 160;//Začetna velikost obrazca
            //omogočimo dostop do gradnikov na plošči panel1
            panel1.Enabled = true;
            /*izpraznimo vsebino predalčka gradnika StatusStrip, ki
              označuje kateri igralec je na potezi*/
            toolStripStatusLabel2.Text = " ";
            //omogočimo izbiro barv obeh igralcev
            nastavitveToolStripMenuItem.Enabled = true;
        }
        /*preverimo še, če so vsa polja že polna, kar pomeni, da je rezultat
          neodločen*/
        else if (Neodloceno())
        {
            /*izpraznimo vsebino predalčka gradnika StatusStrip, ki
              označuje kateri igralec je na potezi*/
            toolStripStatusLabel2.Text = "";
            MessageBox.Show("Rezultat je neodločen!", "Igra je končana", 0,
MessageBoxIcon.Information);
            this.Height = 160;//Začetna velikost obrazca
            //omogočimo dostop do gradnikov na plošči panel1
            panel1.Enabled = true;
            //omogočimo izbiro barv obeh igralcev
            nastavitveToolStripMenuItem.Enabled = true;
        }
        else igralec = !igralec;//zamenjamo igralca, ki je na vrsti
    }
}
```

Po vsakem kliku gumba smo glede na igralca pobarvali kliknjen gumb in takoj preverili stanje igre. Najprej smo s sklicem logične metode *PreveriRezultat* preverili, če je igra končana in imamo zmagovalca. V kolikor zmagovalca ni, vsi gumbi pa so poklikani, s pomočjo metode *Neodloceno* preverimo morebiten neodločen rezultat. Tule sta obe metodi.

```
//metoda vrne True, če imamo zmagovalca
private bool PreveriRezultat(Color barva)
{
    /*Preverimo vse možne kombinacije za tri v vrsto. Metoda vrne true, če
      imamo zmagovalca*/
    if ((tLP.Controls[0].BackColor==barva &&
tLP.Controls[1].BackColor==barva&&tLP.Controls[2].BackColor==barva)||
        (tLP.Controls[3].BackColor == barva && tLP.Controls[4].BackColor ==
barva && tLP.Controls[5].BackColor == barva)||
        (tLP.Controls[6].BackColor == barva && tLP.Controls[7].BackColor ==
barva && tLP.Controls[8].BackColor == barva)||
        (tLP.Controls[0].BackColor == barva && tLP.Controls[3].BackColor ==
barva && tLP.Controls[6].BackColor == barva)||
        (tLP.Controls[1].BackColor == barva && tLP.Controls[4].BackColor ==
barva && tLP.Controls[7].BackColor == barva)||
        (tLP.Controls[2].BackColor == barva && tLP.Controls[5].BackColor ==
barva && tLP.Controls[8].BackColor == barva)||
        (tLP.Controls[0].BackColor == barva && tLP.Controls[4].BackColor ==
barva && tLP.Controls[8].BackColor == barva)||
        (tLP.Controls[2].BackColor == barva && tLP.Controls[4].BackColor ==
barva && tLP.Controls[6].BackColor == barva))
        return true;
    else return false;
}
/*metoda vrne True, če je igra končana (vsa polja so poklikana) in zmagovalca
ni*/
private bool Neodloceno()
{
    //če so že vsa polja izpolnjena metoda vrne true: rezultat je neodločen
    bool neodl = true;
    for (int i=0;i<9;i++)
        if (tLP.Controls[i].BackColor==Color.FromArgb(0,0,0,0))
            neodl=false;
    return neodl;
}
```

LITERATURA IN VIRI

- Uranič S. Praktično programiranje (online). Kranj: TŠC, 2010. (citirano 1. 2. 2011). Dostopno na naslovu: http://uranic.tsckr.si/C%23/Prakticno%20programiranje.pdf
- Sharp, J. *Microsoft Visual C# 2005 Step by Step*. Washington: Microsoft Press, 2005.
- Murach, J., in Murach, M. *Murach's C# 2008*. Mike Murach @ Associates Inc. London, 2008.
- Petric, D. *Spoznavanje osnov programskega jezika C#*, diplomska naloga. Ljubljana: UL FMF, 2008.
- Jerše, G., in Lokar, M. *Programiranje II*, *Objektno programiranje*. Ljubljana: B2, 2008.
- Jerše, G., in Lokar, M. *Programiranje II*, *Rekurzija in datoteke*. Ljubljana: B2, 2008.
- Kerčmar, N. *Prvi koraki v Javi, diplomska naloga*. Ljubljana: UL FMF, 2006.
- Lokar, M. *Osnove programiranja: programiranje – zakaj in vsaj kaj*. Ljubljana: Zavod za šolstvo, 2005.
- Uranič, S. *Microsoft C#.NET*, (online). Kranj: TŠC, 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http://uranic.tsckr.si/C%23/C%23.pdf
- Uranič, S. *Microsoft Visual C#.NET*, (online). Kranj: TŠC, 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http://uranic.tsckr.si/VISUAL%20C%23/VISUAL%20C%23.pdf
- *C# Practical Learning 3.0*, (online). 2008. (citirano 1.2.2011). Dostopno na naslovu: http://www.functionx.com/csharp/
- Lokar, M., et.al. *Projekt UP – Kako poučevati začetni tečaj programskega jezika, sklop interaktivnih gradiv*, (online). 2008. (citirano 1. 1. 2011). Dostopno na naslovu: http://up.fmf.uni-lj.si/
- Lokar, M. *Wiki C#*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu http://penelope.fmf.uni-lj.si/C_sharp
- Coelho, E. *Crystal Clear Icons*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu: http://commons.wikimedia.org/wiki/Crystal_Clear
- Mayo, J. *C# Station Tutorial*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx
- *C# Station*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx
- *CSharp Tutorial*, (online). 2008. (citirano 1.12.2011). Dostopno na naslovu: http://www.java2s.com/Tutorial/CSharp/CatalogCSharp.htm
- *FunctionX Inc*, (online). 2008. (citirano 1.2.20111). Dostopno na naslovu: http://www.functionx.com/csharp/
- *SharpDevelop, razvojno okolje za C#*, (online). 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http: // www.icsharpcode.net/OpenSource/SD/.

- WIKI DIRI 06/07, (online). 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http://penelope.fmf.uni-lj.si/diri0607/index.php/Kategorija:Naloge