ŠOLSKI CENTER CELJE

Splošna in strokovna gimnazija Lava

IGRA LEPO JE BITI MILIJONAR

(raziskovalna naloga)

Mentor:
Mojmir KLOVAR, univ. dipl. inž.

Avtorja:
Andrej MUŽIČ, GL – 4. f
Tomaž KRAMAR, GL – 4. f

Celje, marec 2006

1. Kazalo

Kazalo vsebine

Kazalo slik

2. Povzetek

V raziskovalni nalogi bova predstavila spremenjeno in izboljšano igro Lepo je biti milijonar. Največja izboljšava je »Zgodovina odgovorov«, ki se spreminja glede na to, kako je igralec igral v preteklosti. Ta lastnost naredi najin program drugačen od drugih, bolj življenjski in zanimivejši. Opisala bova težave, ki so se pojavile pri programiranju in njihove rešitve.

3. **Uvod**

3.1 Predstavitev raziskovalnega problema

Že od nekdaj se zanimava za delo z računalniki, zato sva se tudi vpisala na šolo s poudarkom na računalništvu. Najbolj naju zanima programiranje, zato ni bila raziskovalna naloga iz programiranja naključje. Oba pa sva tudi rada gledala oddajo Lepo je biti milijonar. Razmišljala sva, kako bi igro najbolje prilagodila za igranje na računalniku. Navdušila naju je ideja o »Zgodovini odgovorov« namesto »Glasa ljudstva«. To naredi program drugačen od drugih, ki sva jih našla. Najina opcija omogoča, da je statistika ob vsakem igranju drugačna, saj se med igranjem spreminja glede na igralčeve odgovore. Tako naredi »Zgodovina odgovorov« igro zanimivejšo in program bolj življenjski.

3.2 Hipoteze

Želiva napisati program tako, da se z vsakim igranjem spremeni statistika odgovorov glede na to, kako je igralec odgovarjal v prejšnjih igrah. Podatkovna baza bo sestavljena iz treh datotek, ki so ločene glede na težavnost vprašanj, in tako je igranje zanimivejše in resničnejše. Najin program bo vseboval program za urejanje in vpisovanje vprašanj, ki jih bo uporabnik lahko urejal po želji.

3.3 Opis raziskovalnih metod

Najina raziskovalna metoda je bila programiranje v programskem jeziku C++. Pomoč programa je bila ključnega pomena, saj sva si z njo pomagala največ.

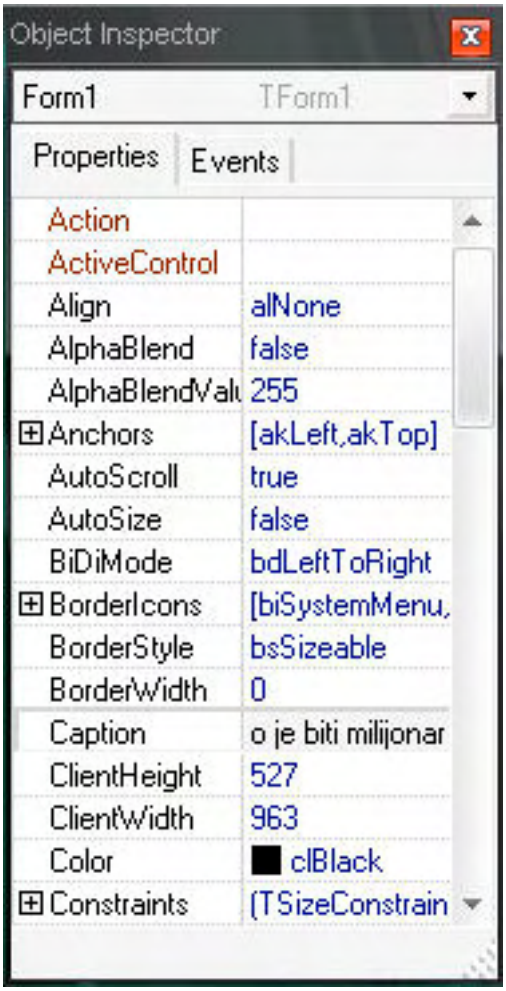
4. OSREDNJI DEL NALOGE

4.1 Rezultati raziskovalne naloge

Problemi

Ugotovila sva, da je možno rešiti vsak problem v programiranju, če ima programer voljo, znanje in vztrajnost. Ta raziskovalna naloga naju je popeljala v svet vizualnega programiranja. Zadovoljstvo po končanem programiranju naju je še bolj navdušilo za študij v tej smeri.

Ker nama je bilo vizualno programiranje popolnoma novo, sva imela že preglavice z najosnovnejšimi problemi. Na začetku sva hotela program urediti vizualno, šele potem pa vsakemu elementu določiti ustrezno funkcijo. Najin prvi problem se je pojavil pri zamenjavi barve okna, kjer se izvaja igra. Po precej dolgem iskanju sva odkrila, da ima vsak element, ki je na oknu, svoje lastnosti. Tam sva našla tudi odgovor na prvi problem in določila črno ozadje. Ker sva želela ustvariti pomoč za igralce, sva morala povezati 2 okni (formi). Najina težava je bila v povezavi knjižnic, ker se okni med seboj nista poznali. Ta problem nama je rešil mentor, ki nama je preprosto dodal dve knjižnici **(#include <Unit1.h>, #include <Unit2.h>**).



Slika 1 - Prikaz lastnosti izbranega objekta

Naslednji problem je bila izdelava menija, ki nama ni delal velikih težav. Podobno, kot je bilo z menijem, je bilo tudi z vstavljanjem gumbov in napisov (»label«). Porodila se nama je zamisel o prikazu napredka, ki bi se ob vsakem pravilnem odgovoru drugače obarval. Na prikazu »Napredek« pa so napisani zneski igre, ki so razdeljeni na tri stopnje oziroma dele (glede na stopnjo težavnosti vprašanj). Da bi igro vizualno še izboljšala, sva naredila še okvirje okoli vprašanj in odgovorov. Ko sva dokončala osnovo grafičnega dela, sva se lotila programiranja.

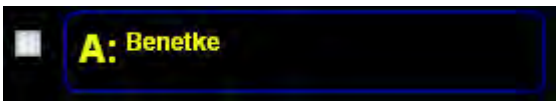
Prvi del programske kode, ki sva ga vpisala, je bil izhod. Potem pa se je pojavila prva resnejša težava. Treba je bilo izdelati odštevalnik časa, saj bi lahko v nasprotnem primeru igralec poiskal odgovor na vprašanje in tako goljufal. Pri tej težavi sva prvič uporabila Borlandovo pomoč (programsko okolje). Po večurnem iskanju po pomoči se nisva uspela prebiti do pravilne rešitve. Znala sva nastaviti uro, ne pa tega, da bi nama program odšteval čas. Najin rešitelj je bil mentor, ki nama je svetoval in popravil najine neuspešne poizkuse. S tem nama je vlil voljo za nadaljevanje. Ker je bilo treba odštevalnik povezati še z gumbom za potrditev odgovora in naslednje vprašanje, sva prilagodila mentorjevo različico in nastavila ustavitev odštevanja ter postavitev odštevalnika na začetek in ponoven zagon odštevanja. Kmalu po tem, ko sva našla med vizualnimi elementi »progress bar«, sva se odločila, da bi poizkusila

najin odštevalnik povezati z »progress barom« in bi s tem izboljšala izgled programa. Čas, ki je na voljo, znaša 1 minuto in pol, zato sva morala »progress baru« določiti korak, ki ga naj izvede vsako sekundo. Z nekaj napora nama je tudi to uspelo. Bistvo je bilo spet v lastnostih, kjer sva to tudi nastavila.

Slika 2 – Prikaz »progress bar« pri 1/3 preteklega časa

Po nasvetu najinega mentorja sva se odločila, da bova spremenila način označevanja vprašanj iz »check box« na »radio button«, saj »radio button« omogoča označitev samo enega izbranega vprašanja, »check box« pa lahko ima enega ali več označenih odgovorov ali pa tudi nobenega. Zato je bila očitno boljša ta izbira.

Slika 3 – »Radio button«



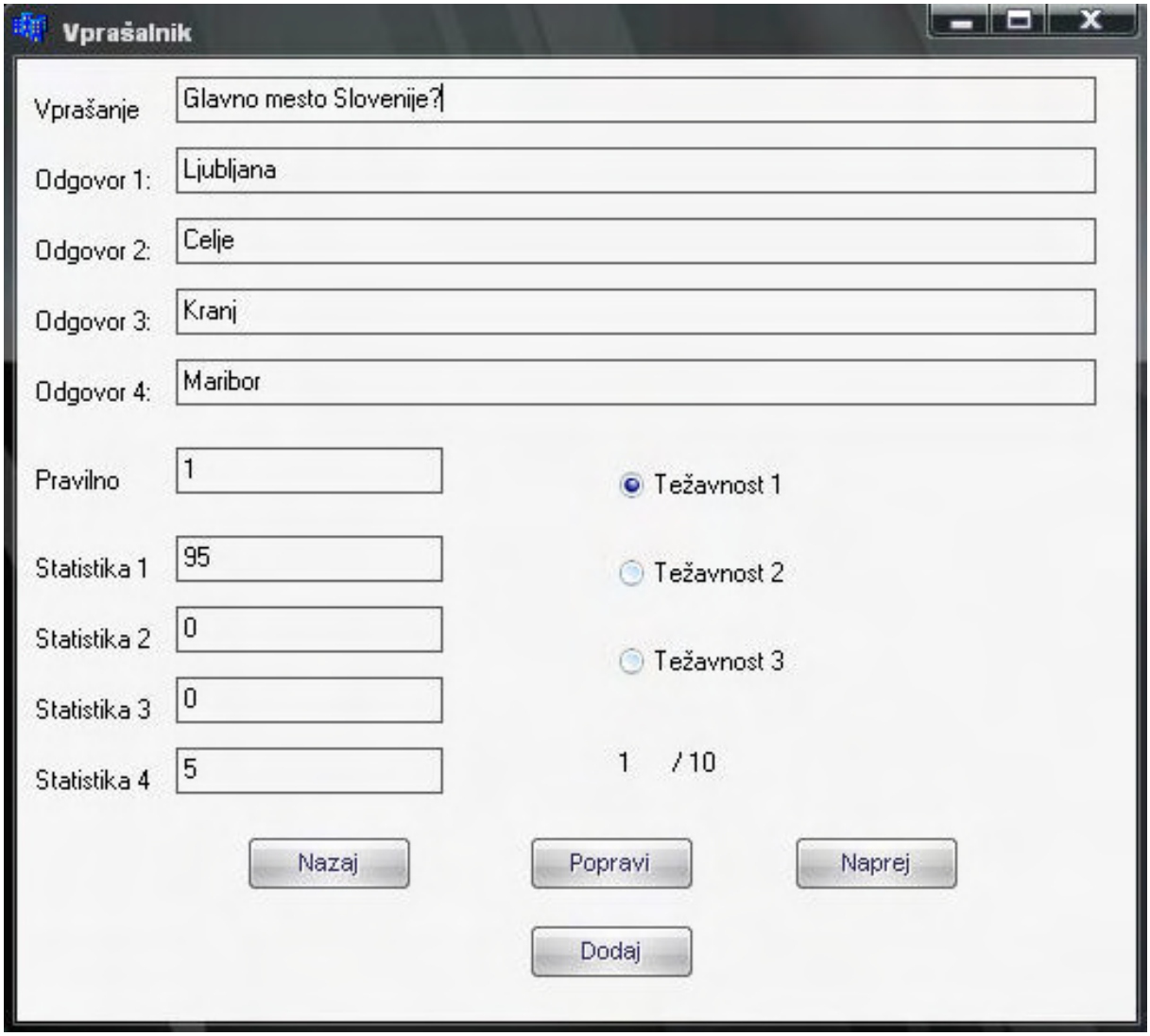
Slika 4 – »Check box«

Kot privzeto pa sva nastavila označen odgovor A, saj sva se s tem rešila možnosti, da igralec ne bi izbral nobenega odgovora.

Podatkovna baza

Podatkovno bazo sva razdelila na tri težavnosti (glede na to, na katero vprašanje po vrsti odgovarja igralec). Tako sva uporabila tri različne datoteke z vprašanji. Naslednje in najtežje vprašanje se nama je pojavilo pri branju iz datoteke in pomikanju po njej (izbira vprašanj). Mentor nama je svetoval, da ustvariva strukturo, ki bo vsebovala vprašanje, štiri odgovore, pravilni odgovor in zgodovino odgovorov. Popolnoma enako strukturo sva naredila v programu za pisanje vprašanj, ki sva ga morala napisati. Ker sta bili strukturi popolnoma enaki, sta oba programa poznala zapis in iz njega ustrezno razbrala podatke in jih ustrezno ločila med seboj (Slika 5). Dolžino strukture sva preverila z ukazom **sizeof** in ugotovila, da je velika 292 bitov. Nastavila sva funkcijo naključnih števil (**random**), ki je odvisna od števila vprašanj, ki jih program prebere na začetku in shrani ta podatek. To naključno število sva pomnožila s 292 in tako dobila premik. Premik sva nastavljala z ukazom **fsetpos**. Program je nato prebral z ukazom **fread** ustrezno vprašanje. Po končani igri pa se je sprememba statistike zapisala z ukazom **fwrite**. Da se ne bi vprašanja ponavljala, sva nastavila 5 spremenljivk (3 krat po 5 vprašanj – tri stopnje težavnosti vprašanj in 5 vprašanj v eni težavnosti) in

shranjevala naslove že uporabljenih vprašanj. Tukaj nama je delala preglavice napaka, da je program prebral število vprašanj za eno večje, kot pa je bilo v resnici. Tako najin sistem do odkritja te napake ni delal.
Ko sva imela ta način branja vprašanj dokončan, nama je mentor svetoval, da prebereva na začetku čisto vsa vprašanja iz vseh treh datotek v tri polja. Zato sva morala določene stvari popraviti. Prednost tega novega načina je predvsem v hitrosti programa, saj se vprašanja naložijo v glavni pomnilnik, ki pa je hitrejši od vhodno izhodne naprave (trdega diska).



Slika 5 – Program za pisanje in urejanje vprašanj

Program za vpis vprašanj sva morala popraviti, saj je omogočal samo dodajanje vprašanj in ne popravljanja. Naredila sva še premik po polju naprej in nazaj, tako lahko uporabnik pregleduje vprašanja in jih po želji popravlja. Možen je tudi premik med že prej omenjenimi težavnostmi. Ob zaprtju programa se vse spremembe v polju zapišejo še v datoteke.

Izzivi

Iz »radio button« je bilo še treba razbrati, kateri odgovor je bil označen in potrjen. Označen odgovor sva razbrala glede na to, kateri »radio button« ima lastnost **checked** true ali false (če je true, potem je izbran ta »radio button«, če pa je false, pa kateri od drugih). Tu sva uvedla novo spremenljivko **s.** Vsak »radio button« predstavlja svoj odgovor (»radio button« 1 = A, »radio button« 2 = B …). V primeru, da je bil izbran »radio button« 1, sva s pomočjo **switch** stavka določila vrednost spremenljivki **s** na 1 (s = 1) , za izbran drugi odgovor, ki ga predstavlja »radio button« 2, pa spremenljivko s na 2 (s = 2). Tako sva naredila še za ostali dve možnosti. To spremenljivko sva nato primerjala s pomočjo stavka **if** s številko pravilnega odgovora (to sva prebrala iz datoteke skupaj z odgovorom in vprašanji). Če se ujemata, je odgovor pravilen in igra se lahko nadaljuje. Če ne, pa izpiše, da je odgovor napačen in ustrezen dosežen znesek. Tedaj je igra zaključena (vsi gumbi nastavijo lastnost **enabled** na false). To pa pomeni, da vsi gumbi postanejo neaktivni. Tako mora uporabnik začeti igro znova.

Primer ugotavljanja pravilnega odgovora:

```
if(RadioButton1->TabStop= =true)
   {
   s=1;
   polje.v1[a].stat1++;
   }
 if(s==prav)Label2->Caption="Odgovor je pravilen!";
 else
  {
  Button1->Enabled=false;
  Button2->Enabled=false;
  Button3->Enabled=false;
  Button4->Enabled=false;
  Button5->Enabled=false;
  Label2->Caption="Odgovor je napačen!";
  Label33->Visible = true;
  if(varovalo<=5)Label33->Caption="Več sreče prihodnjič!";
  else if(varovalo<=10)Label33->Caption="Osvojili ste 50.000 SIT!";
      else Label33->Caption="Osvojili ste 1.000.000 SIT!";
  }
```

Naslednji izziv je bila »**Polovička«**. »Polovička« je morala biti izbrana po naključnem izboru (**random** funkcija). Pri tem pa je bilo treba paziti, da ne bi program izločil pravilnega odgovora. To sva naredila tako, da sva vsako generirano število (od 1 do 4) primerjala s pravilnim. Generirano število je moralo biti različno od pravilnega in se ni smelo ponoviti dvakrat, saj bi v nasprotnem primeru polovička izločila samo en napačen odgovor ali pa pravilnega.

Malo lažji izziv je bilo »**Podaljšanje časa«**. V tem primeru sva samo odštela 30 sekund od položaja »progress bara«. Nastavila sva, da je »progress bar« dolg 90 enot, vsako sekundo se poveča »progress bar« za eno enoto (skupaj je to minuto in pol). Nastavila sva tudi, da uporabnik ne more izkoristiti te pomoči pred iztekom ene minute. Ni namreč logično, da bi jo izkoristil prej. Po izteku časa se uporabniku izpiše na zaslonu »Zmanjkalo vam je časa!«. S tem je igra končana, saj gumbi za nadaljnje igranje osivijo in jih ni mogoče več klikniti.
Primer:

```
{
if(Button3->Enabled == true)
  if (ProgressBar1->Position >= DODAJ_CAS)
     {ProgressBar1->Position = ProgressBar1->Position -30;
      Button3->Enabled = false;}
}
```

Preostala nama je še »**Zgodovina odgovorov«**. Tukaj je bilo treba narediti sprotno spreminjanje statistike. Glede na to, kako je igralec odgovarjal (statistika se poveča za en odgovor, ne glede na to, ali je izbran pravilen ali napačen odgovor - **polje.v1[a].stat1++**). Potem pa ob izhodu to statistiko program shrani v datoteko.

Slika 6 – Celoten program

Zgornja slika prikazuje program v teku. Ko uporabnik požene program, mora klikniti v meniju »Igra« in potem »Nova«. Takoj se izpiše prvo vprašanje. Uporabnik oziroma igralec

klikne na odgovor, za katerega se je odločil, in nato klikne gumb »Potrdi odgovor«. Za tem pa, seveda če je bil odgovor pravilen, gumb »Naslednje vprašanje«. Tako se igra nadaljuje, dokler igralec ne izbere napačnega odgovora ali pa ne klikne gumba »Vzamem denar«. Med igro lahko igralec uporabi katerokoli od treh pomoči, vendar samo enkrat. Znesek, ki ga je igralec že osvojil, se sproti primerno obarva v »Napredku«.

4.2 Razprava

Najin program sva želela čim bolj približati znani televizijski igri Lepo je biti milijonar, vendar pa ob tem tudi narediti nekaj novega, inovativnejšega in zanimivejšega. Poleg tega ponujava še učinkovit program za pisanje vprašanj. Tako si lahko vsak po želji dodaja vprašanja in jih posodablja. Izboljšava v primerjavi z televizijsko igro in ostalimi podobnimi igrami je »Zgodovina odgovorov«, saj naredi igro bolj življenjsko. V televizijski igri »Zgodovina odgovorov« seveda ni mogoča, ker se tam vprašanja ne ponavljajo. Podatkovna baza pa seveda ne more biti tako obsežna, da se vprašanja nikoli ne bi ponovila. Še ena razlika med televizijsko in računalniško igro je »Klic v sili«. Tega v računalniški igri ni mogoče izvesti, zato sva se odločila za možnost podaljšanja časa namesto »Klica v sili«. Večina ostalih programov, ki sva jih našla na internetu, so imeli »Klic v sili«, ampak se nama je zdel nesmiseln, ker igralec ne more poklicati nekoga v igri, ampak je odgovor že naprej podan.

5. Zaključek

Najine hipoteze so se potrdile. Naredila sva program, ki sva si ga zamislila na začetku. Z najinim programom sva zadovoljna. Vsebuje vse novosti in spremembe, ki sva jih želela uvesti v najino igro. Pri programiranju sva naletela na marsikatero težavo in jo tudi uspešno rešila. Pri tem sva se naučila veliko novega o programskem jeziku C++ in o načinu dela z njim. Program sva razdelila svojim sošolcem in znancem, ki sedaj veselo igrajo in testirajo svoje sposobnosti pri odgovarjanju na vprašanja. Program nameravava naložiti tudi na internet, kjer bo na voljo vsem, ki se želijo sprostiti ob igranju igre Lepo je biti milijonar.

6. Viri in literatura

- Pomoč v Borlandu C++ Builderju 6

- REISDORPH, Kent in Ken HENDERSON. (1997). Teach Yourself Borland C++ Builder in 21 Days. [CD ROM]. Indianapolis.

- NEZNANI AVTOR. (2006). Branje iz datoteke in spreminjanje nizov. [Online]. [Citirano 1.3.2006; 15:45]. Dostopno na naslovu: http://www.slo-tech.com/script/forum/izpisitemo.php?threadID=206624#neprebrano

- PANDYALAKAL, Vijay Mathew. (2001). Simple Database. [Online]. [Citirano 2.2.2006; 19:38]. Dostopno na spletnem naslovu: http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=2642&lngWId=3

- MUŽIČ, A. (2005 - 2006). Zvezek za Računalniške sisteme in omrežja (4. letnik tehniške gimnazije). Celje: Šolski center Celje.

7. Zahvala

Iskreno se zahvaljujeva mentorju Mojmiru Klovarju za pomoč pri snovanju in programiranju programa.