Osnove mikroprocesorske elektronike

Vaja 13: Igrica - prvič

Pri tej vaji boste igro oživili in ji dodali prvi in zadnji del celotne funkcionalnosti:

1) ob zagonu igre se bo prikazal *pozdravni zaslon* igre (angl. splash screen) in prižgale se bodo vse LEDice. Izpisal se bo napis "Press any key to continue". Ob pritisku na katerokoli tipko se bodo LEDice ugasnile in pričelo se bo

2) *igranje igre*. Ko se bo čas za igranje iztekel, se bo

3) igra končala in prikazala se bo *zaključna "game over" slika*. Počakamo 3 sekunde, da igralec dojame situacijo. Nato se pa ob pritisku katerekoli tipke oziroma po 10-tih sekundah vrnemo nazaj na začetek igre v pozdravni zaslon.

To boste storili tako, da boste implementirali avtomate `Game()`, `Intro()` ter `GameOver()`. Funkcija pod-avtomata `GamePlay()` je zaenkrat le ta, da izriše ozadje ter čaka na pritisk tipke, preden se zaključi. V sklopu naslednje vaje pa boste implementirali še ta pod-avtomat.

Naloge vaje

1. ***Projektu*** `VAJA_13-game` ***, ki ste ga pripravili tekom priprave na vajo, dodajte sledeče <u>posodobljene</u> datoteke:***

   a) `objects.c` in `objects.h`,
   b) `graphics.c` in `graphics.h`,
   c) `images.h` ter
   d) `game.c` in `game.h`.

   v ustrezno mesto v mapi "`Application`". Nove datoteke najdete v datoteki "`Application.zip`" v eFE mapi "predloga vaja".

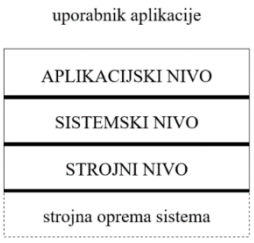
   Datoteke naj kar "povozijo" oziroma prepišejo že obstoječe istoimenske datoteke.

   Tako boste poskrbeli, da vaša aplikacija dobi posodobljen modul za delo z objekti in njihov izris na zaslon, hkrati pa ste z `images.h` vključili tudi nove slike objektov.

   Kot že rečeno, vaša naloga bo, da znotraj modula `game.c` dopolnite implementacijo avtomatov `Game()`, `Intro()` ter `GameOver()`.

2. Znotraj `game.c` ***modula poskrbite, da bo vaša aplikacija vključila <u>sistemske module</u> za delo z***

- LEDicami,
- tipkovnico,
- serijskim vmesnikom SCI,
- "joystickom",
- LCDjem ter
- orodji za merjenje časa



V tem delu programske kode se bo lepo pokazalo, kako je *aplikacija* odvisna zgolj od *sistemskih* modulov, ne pa tudi od modulov *strojnega* nivoja, kar je prav in smiselno (glejte sliko zgoraj)! S premišljenim pristopom ste tako aplikacijo uspešno ločili od nizko-nivojskih modulov vgrajenega sistema (angl. loosely coupled code), kar jo naredi neodvisno od strojne opreme sistema ter zato lažje prenosljivo med različnimi platformami.

3. Znotraj `game.c` ***modula poskrbite, da bo <u>algoritem</u> vaše aplikacije dobil podporo za***

- delo z objekti aplikacije,
- izris objektov na grafični vmesnik – zaslon.

Da spomnimo, kako je zastavljena aplikacija:

1) modul `game.c` bo poskrbel za *implementacijo poteka in pravil* igre, pri čemer bomo *algoritem* igre implementirali v obliki *avtomata stanj* (angl. state machine),
2) modul `objects.c` bo poskrbel za *definicijo* in *manipulacijo* vseh *objektov* (tj. podatkov), ki jih igra potrebuje za svoje delovanje,
3) modul `graphics.c` bo poskrbel za *izris vseh objektov* na uporabniški vmesnik.

4. S pomočjo naštevnih tipov definirajte stanja avtomatov:

a) `GAME_states_t` – stanja avtomata `Game()`,

b) `INTRO_states_t` – stanja avtomata `Intro()`,

c) `GAMEOVER_states_t` – stanja avtomata `GameOver()`.

Pri tem si pomagajte z diagrami prehajanja stanj, ki jih najdete v eFE mapi "flowcharts".

<u>Namig:</u> uporabite "copy-paste" in skopirajte imena stanj kar iz diagramov.

5. ***Dopolnite implementacijo avtomata*** `Game()`.

   Pri tem si seveda pomagajte z diagramom prehajanja stanj avtomata `Game()`.

   Pri implementaciji boste poskrbeli za sledeče stvari:

   a) definirali boste *statično* spremenljivko `state`, kjer se bo pomnilo trenutno stanje avtomata `Game()`,

   b) definirali boste pomožno spremenljivko `exit_value`, kjer se bo pomnila vrnjena vrednost pod-avtomatov `Intro()`, `GamePlay()` in `GameOver()`,

   c) s pomočjo "switch-case" stavka boste *implementirali algoritem avtomata* `Game()` tako, da boste za vsako stanje avtomata implementirali programsko kodo, ki izvede vsa potrebna opravila posameznega stanja,

   d) poskrbeli boste tudi za implementacijo "varovalke", ki poskrbi za javljanje napake v primeru, da avtomat nekako zaide v nedefinirano stanje.

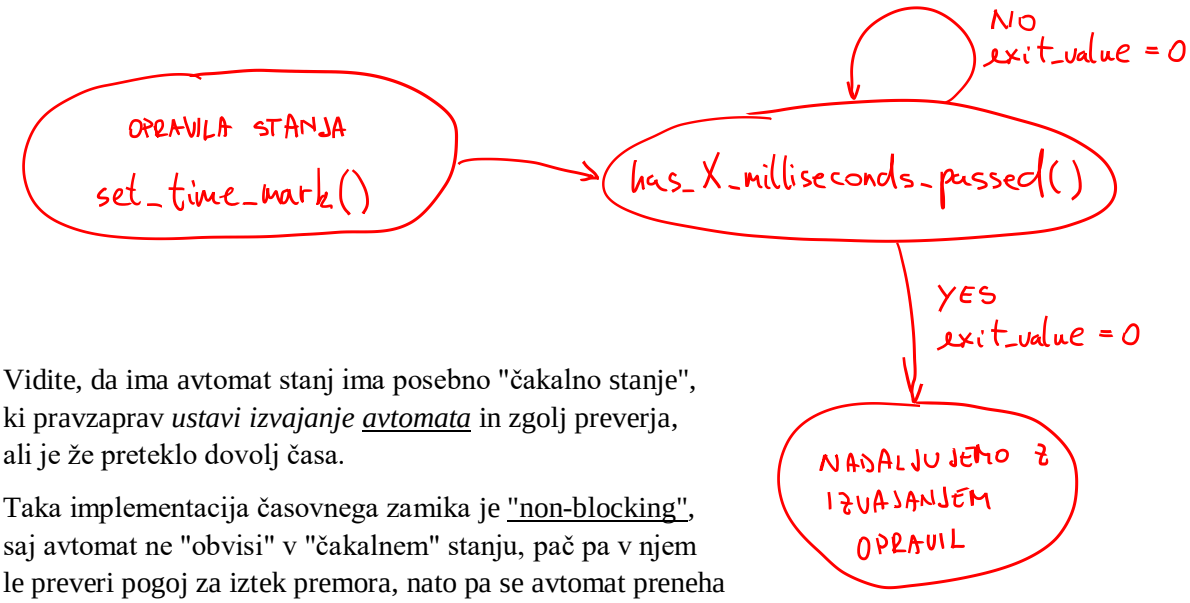
6. ***Implementirajte pod-avtomat*** `Intro()`.

   Pri tem se zgledujte po implementaciji avtomata `Game()` ter sledite komentarjem v predlogi.

   V premislek: pri implementaciji pod-avtomata `Intro()` boste seveda uporabljali funkcionalnost, ki vam jo nudijo *sistemski* moduli. Uporabljati pa boste morali tudi funkcije za delo z objekti (modul `objects.c`) ter funkcije za izris objektov na zaslon (`graphics.c`). Vse potrebne funkcije za izris objektov ste spoznali že v pripravi na vajo. Objekti pa so sedaj novi, zato bo potrebno nekoliko pobrskati po vsebini datoteke `objects.c` in *iz obstoječe kode razbrati*, s katerimi objekti boste upravljali v aplikaciji. Na ta način se urite v interpretaciji programske kode, kar je zelo uporabna veščina, saj boste v praksi pogosto delali s kodo, ki ne bo vaša.

Časovna zakasnitev z avtomatom stanj

V sklopu pod-avtomata `Intro()` boste morali implementirati tudi časovno zakasnitev s pomočjo avtomata stanj. Idejo za implementacijo časovne zakasnite prikazuje skica spodaj.



Vidite, da ima avtomat stanj ima posebno "čakalno stanje", ki pravzaprav *ustavi izvajanje avtomata* in zgolj preverja, ali je že preteklo dovolj časa.

Taka implementacija časovnega zamika je "non-blocking", saj avtomat ne "obvisi" v "čakalnem" stanju, pač pa v njem le preveri pogoj za iztek premora, nato pa se avtomat preneha izvajati do naslednjega ponovnega zagona avtomata.

7. ***Implementirajte pod-avtomat*** `GameOver()`**.**

   Pri tem se zgledujte po implementaciji pod-avtomata `Intro()`.

8. ***Ko boste implementirali avtomate*** `Game()`, `Intro()` in `GameOver()`, ***preizkusite delovanje vaše aplikacije.***

   To storite tako, da funkcijo avtomata `Game()` vključite v neskončno `while(1)` zanko znotraj vašega glavnega programa v `main.c`,

```
while (1)
{
    Game();
}
```

   Tako zagotovite, da bo glavni program neprenehoma izvajal avtomat stanj, znotraj katerega ste implementirali vašo aplikacijo – igro.

   Ne pozabite v `main.c` vključiti še zglavne .h datoteke modula `game`.

   V poglavju spodaj si lahko pogledate diagram, ki vam bo morda pomagal razjasniti, kako pravzaprav poteka izvajanje igrice s pomočjo avtomata stanj znotraj `while(1)` zanke.

Dodatna pojasnila

Kako poteka izvajanje igre s pomočjo avtomata stanj?

Da bi postalo bolj jasno, kako se pravzaprav izvaja avtomat stanj, si lahko ogledate spodnji premislek in skico primera, ko avtomat `Game()` preide iz začetnega stanja, kjer se izvaja `Intro()`, v stanje, kjer se izvaja igranje igrice `GamePlay()`.

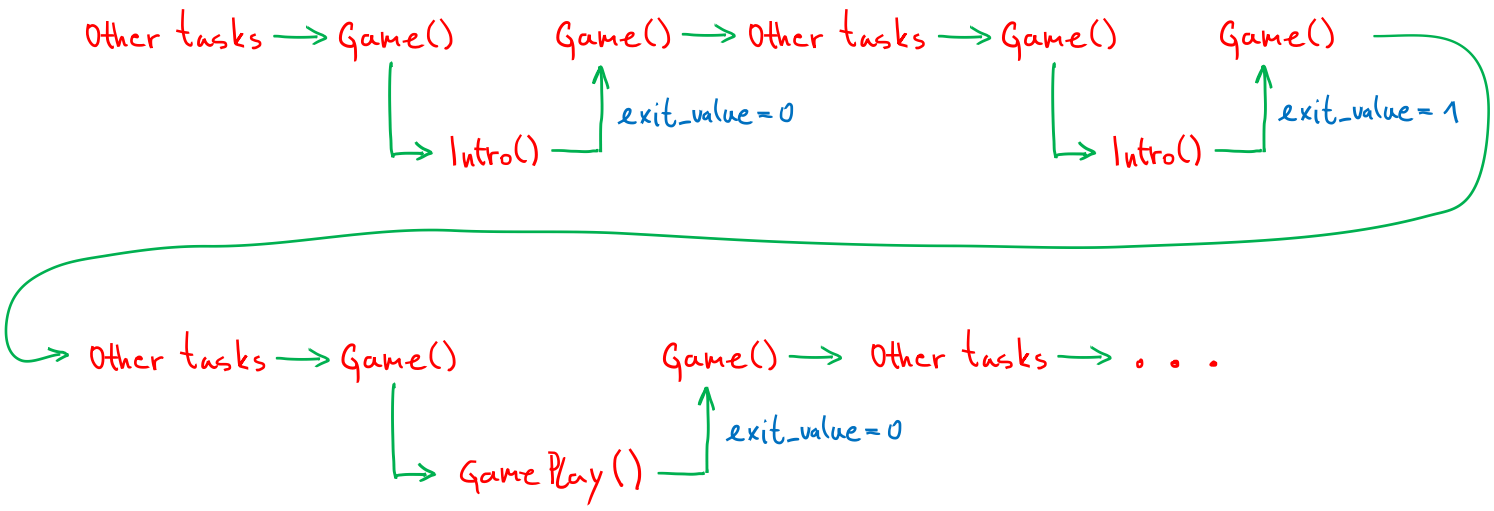
Programsko kodo za izvajanje igrice znotraj `while(1)` zanke

```
while (1)
{
    Game();
}
```

bi lahko posplošili tako, da glavni program poleg izvajanja igre izvaja tudi krajša preostala opravila:

```
while (1)
{
    Other_Tasks();
    Game();
}
```

Sedaj si pa delovanje te kode oglejte na primeru, ko nad-avtomat `Game()` preide iz začetnega stanja, kjer se izvaja opravilo `Intro()`, v stanje, kjer se izvaja igranje igrice `GamePlay()`.



Opazite lahko, da se avtomat `Game()` *izvaja vsake toliko časa periodično*. Avtomat `Game()` poskrbi, da se glede na stanje igrice izvaja ustrezni pod-avtomat. Sprva se izvaja pod-avtomat `Intro()`, vse dokler s pritiskom na katerokoli tipko ne preidemo v naslednje stanje – v igranje igre, ki je izvedena s pod-avtomatom `GamePlay()`. Prehod v naslednje stanje se zgodi, ko pod-avtomat `Intro()` vrne vrednost `exit_value = 1` in s tem sporoči nad-avtomatu `Game()`, da je opravil vsa svoja opravila in zato lahko (zaenkrat) zaključi svoje izvajanje.