Osnove mikroprocesorske elektronike

Priprava 14: Igrica – drugič

V sklopu te priprave se boste pripravili na implementacijo jedra igrice – na implementacijo *algoritma*, ki implementira obnašanje in pravila igrice med samim igranjem. To boste storili tako, da se boste najprej seznanili s potekom igrice, nato pa boste preučili še *seznam vseh ključnih funkcij in podatkovnih struktur*, ki jih boste potrebovali za implementacijo igranja igre. Preden pa pričnete s to pripravo, pa boste popravili in dopolnili še nekaj malenkosti pri celotnem projektu.

Priprava – popravki obstoječega projekta z dodatki

Tokrat vam ni potrebno ustvarjati novega projekta, pač pa kar nadaljujte s projektom s prejšnje vaje `VAJA_13-game`. Pri temu projektu storite sledeče:

1. ***Projektu dodajte aplikacijski modul*** `math_utils`, znotraj katerega so implementirane razne matematične funkcije, ki jih potrebuje algoritem igrice.

   Modul med drugim vsebuje funkcije za *generacijo naključnih števil*, kar pa je ključno, če želimo igrico napraviti nepredvidljivo.

   Modul najdete v eFE mapi "predloge". Vključite ga med *aplikacijske* module.

2. ***Projektu dodajte sledeče <u>posodobljene</u> datoteke:***

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

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

   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.

3. ***Obstoječemu aplikacijskemu modulu*** `game` ***dodajte funkcije, s katerimi bomo implementirali igranje igrice.***

   Funkcije najdete v eFE mapi "predloge" v datoteki " `predloga funkcij za modul game.c`". Vsebino skopirajte na smiselno mesto v datoteki `game.c`.

   Funkcija `GamePlay()` naj prepiše obstoječo istoimensko funkcijo.

   Poskrbite tudi za definicijo prototipa nove funkcije

   `GamePlay_Update_Aiming_LED_Indicator`.

   Prototip dodajte znotraj datoteke game.c pod sekcijo "Private definitions", saj gre za funkcijo, ki jo uporabljamo zgolj znotraj modula `game`.

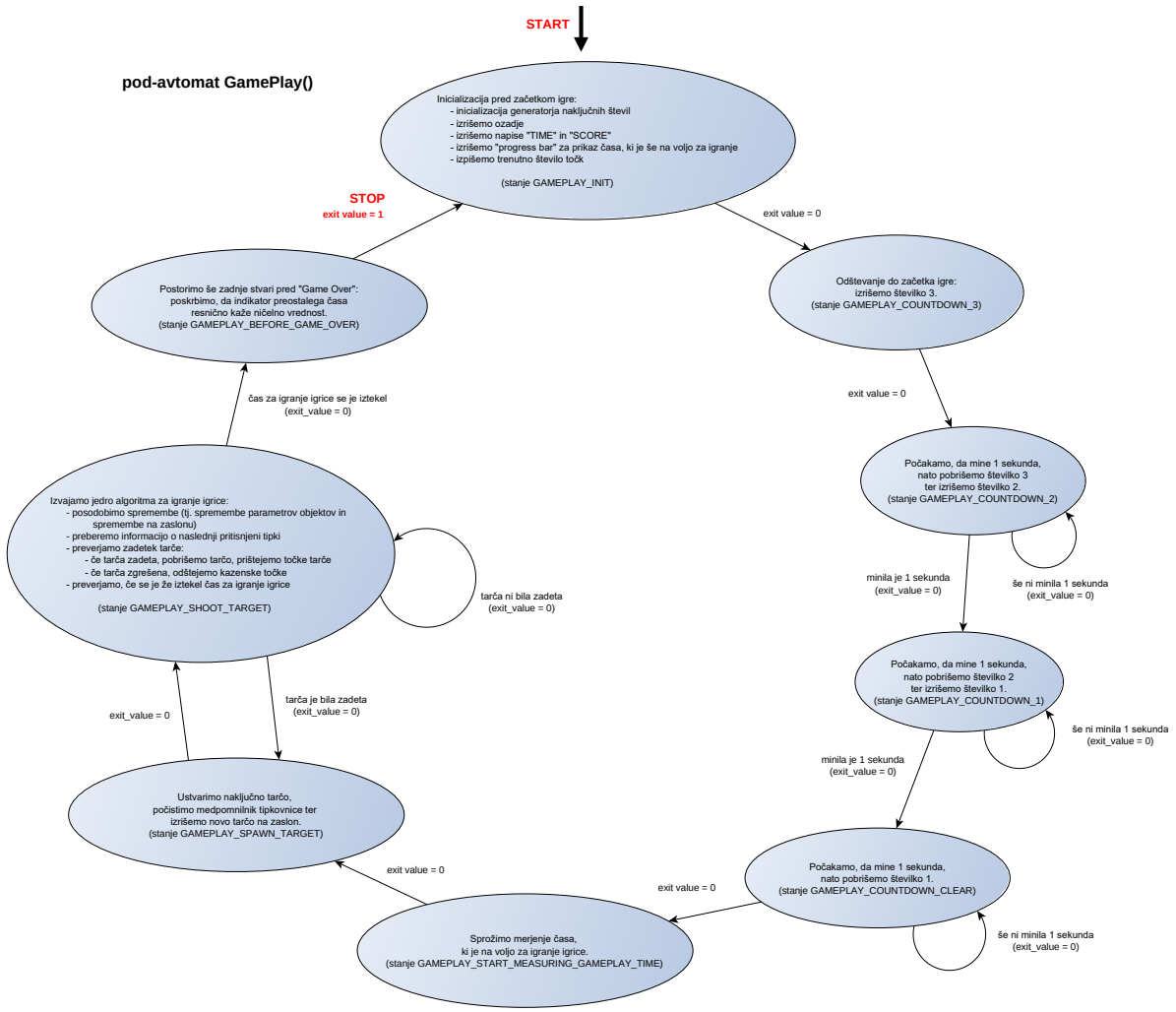
```
// ---------------------- Private definitions ------------------

// ----- Prototipi "privatnih" funkcij ------
void GamePlay_Update_Aiming_LED_Indicator(void);
```

Priprava – preučitev algoritma za igranje igrice `GamePlay()`

Vaša naloga pri vajo bo ta, da *implementirate algoritem za igranje igrice* ob pomoči že definiranih *funkcij za manipulacijo objektov* igrice (tj. podatkovnih struktur aplikacije) ter *funkcij za izris objektov na zaslon* (tj. modul za delo z grafičnim vmesnikom). Poglejmo si torej, kako naj bi zgledalo igranje igrice:

> ko po uvodnem "splash screen" zaslonu igralec pritisne katerokoli tipko, se nato izriše prostor (slika), kjer bo igralec streljal tarče. V spodnjem statusnem delu zaslona lahko igralec vidi indikator preostalega časa in število točk, ki jih je dosegel. Na zaslonu se odšteva čas do začetka igranja: 3 – 2 – 1, nato se igra prične. Na zaslonu se pojavljajo naključne tarče, ki se odbijajo od roba prostora. Igralec jih mora sestreliti tako, da z "joystickom" postavi namerek na tarčo ter s pritiskom na katerokoli tipko ustreli. Če je bil strel uspešen, tarča izgine in se pojavi nova. Če je strel zgrešil, se igralcu odšteje točke. Zgrešene strele se torej kaznuje. V spodnjem statusnem delu zaslona se sproti izrisuje čas, ki je še ne voljo za igranje ter število doseženih točk. Ko mine 20 sekund, se igra zaključi in pridemo do "game over" zaslona.

Zgoraj opisano obnašanje igrice bolj natančno prikazuje diagram prehajanja stanj za algoritem igranja igrice `GamePlay()`, kjer so bolj podrobno podana opravila, ki jih je potrebno opraviti v vsakem od stanj igrice. Preučite ta diagram, da se vam izčisti potek algoritma, ki ga boste implementirali.



Priprava – preučitev funkcij za implementacijo algoritma `GamePlay()`

Na zgornjem diagramu imate podrobno opisan *postopek*, ki ga mora implementirati algoritem za igranje igrice. Sedaj si pa še poglejmo, katere funkcije *modula za manipulacijo objektov* igrice `OBJ` ter *modula za grafični izris* objektov `GFX` boste potrebovali pri implementaciji algoritma. Potrebovali bomo tudi funkcije modula `MATH` ter pomožne funkcije modula `game`.

Preučite torej spodnje sezname ključnih funkcij in se seznanite s pomenom teh funkcij.

Funkcije modula `GFX`

`GFX_draw_gfx_object()`	funkcija izriše grafični objekt brez upoštevanja plasti in transparentnega dela grafičnega objekta
`GFX_draw_one_gfx_object_on_background()`	funkcija izriše en sam grafični objekt na ozadje, upoštevajoč transparentni del grafičnega objekta
`GFX_draw_two_gfx_objects_on_background()`	funkcija izriše dva grafična objekta na ozadje, upoštevajoč plast (angl. layer) na katerem se nahaja grafični objekt
`GFX_clear_gfx_object_on_background()`	funkcija pobriše en sam grafični objekt na ozadju
`GFX_update_moving_gfx_object_location()`	funkcija posodobi lokacijo premikajočega grafičnega objekta, upoštevajoč hitrost objekta
`GFX_display_text_object()`	funkcija izriše tekstovni objekt (npr. tekst, ki vsebuje število doseženih točk)
`GFX_display_progress_bar()`	funkcija izriše "progress bar" indikator

Funkcije modula `OBJ`

`OBJ_spawn_target()`	funkcija ustvari naključno tarčo: naključno izbere vrsto tarče, njeno pozicijo in hitrost
`OBJ_crosshair_set_center_location_with_joystick()`	funkcija nastavi lokacijo namerka upoštevajoč pozicijo "joysticka"
`OBJ_crosshair_update_distance_to_target()`	funkcija izračuna razdaljo med namerkom in tarčo in shrani to informacijo med parametre namerka
`OBJ_is_crosshair_on_target()`	funkcija preveri, če se namerek nahaja znotraj tarče; funkcija vrne vrednost 1, če namerek v tarči, sicer pa vrednost 0
`OBJ_set_timeout_bar_value()`	funkcija nastavi vrednost "progress bar" indikatorja; vhodni argument je vrednost podana v procentih
`OBJ_set_score_text_value()`	funkcija nastavi vrednost besedila za izpis točk; vhodni argument je vrednost točk

Funkcije modula `MATH`

`MATH_init_random_generator()`	funkcija inicializira generator naključnih števil (tj. nastavi "seme" (angl. seed) za generacijo naključnih števil)

Funkcije modula `Game`

`GamePlay_Update_Aiming_LED_Indicator()`	funkcija posodobi stanje LED indikatorja točnostni namerka: bližje kot je namerek tarči, več LEDic sveti
`GamePlay_UpdateChanges()`	funkcija poskrbi, da se objekti, ki jih algoritem igre spreminja, posodobijo: tako njihovi parametri kot njihov prikaz na zaslonu

Priprava – preučitev podatkovnih struktur za implementacijo algoritma `GamePlay()`

Zgoraj ste se seznanili s ključnimi funkcije za implementacijo algoritma. Sedaj pa poglejte še nekaj ključnih podatkovnih struktur znotraj modula OBJ, ki pomembno vplivajo na delovanje aplikacije. Parametre nekaterih teh struktur boste nastavili vi tekom implementacije.

Znotraj modula `OBJ` so definirani sledeči objekti:

```
// ---- Object for defining application settings and status ------

settings_t      settings;
game_status_t   game_status;

canvas_t        canvas;


// --- Graphical objects ----

graphic_object_t    splash_screen;
graphic_object_t    background;
graphic_object_t    game_over_sprite;
graphic_object_t    press_any_key_sprite;

graphic_object_t    countdown_digit_1_sprite;
graphic_object_t    countdown_digit_2_sprite;
graphic_object_t    countdown_digit_3_sprite;

progress_bar_t  timeout_bar;


// ------- Compound objects -------

arduino_object_t    arduino[NUM_OF_ARDUINO_TYPES];

crosshair_object_t  crosshair;
arduino_object_t    target;


// ----- Text objects ------

text_object_t   score_box_title;
text_object_t   score_text;

text_object_t   time_box_title;
```

objekti, ki hranijo nastavitve in status aplikacije

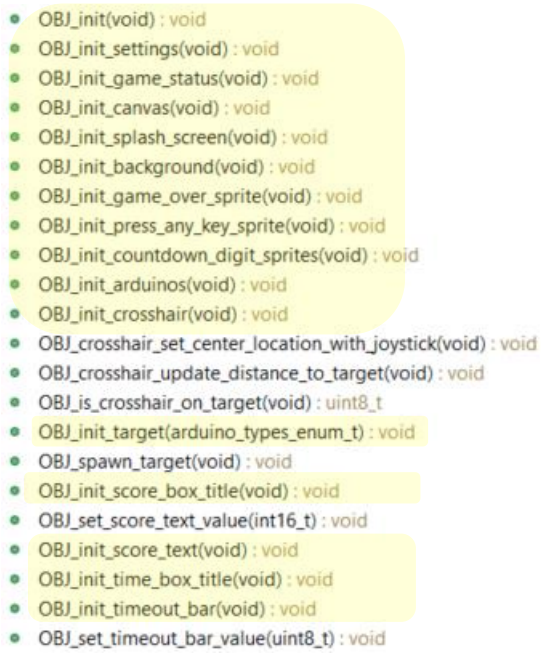
grafični objekti aplikacije

"progress bar" indikator preostalega časa za igranje

sestavljeni objekti aplikacije, ki so izpeljani iz tipa *grafičnega* objekta

tekstovni objekti za izrisovanje besedila

Začetne vrednosti parametrov teh objektov nastavljamo z `OBJ_init` funkcijami, ki imajo vlogo nekakšnega "konstruktorja" objektov.

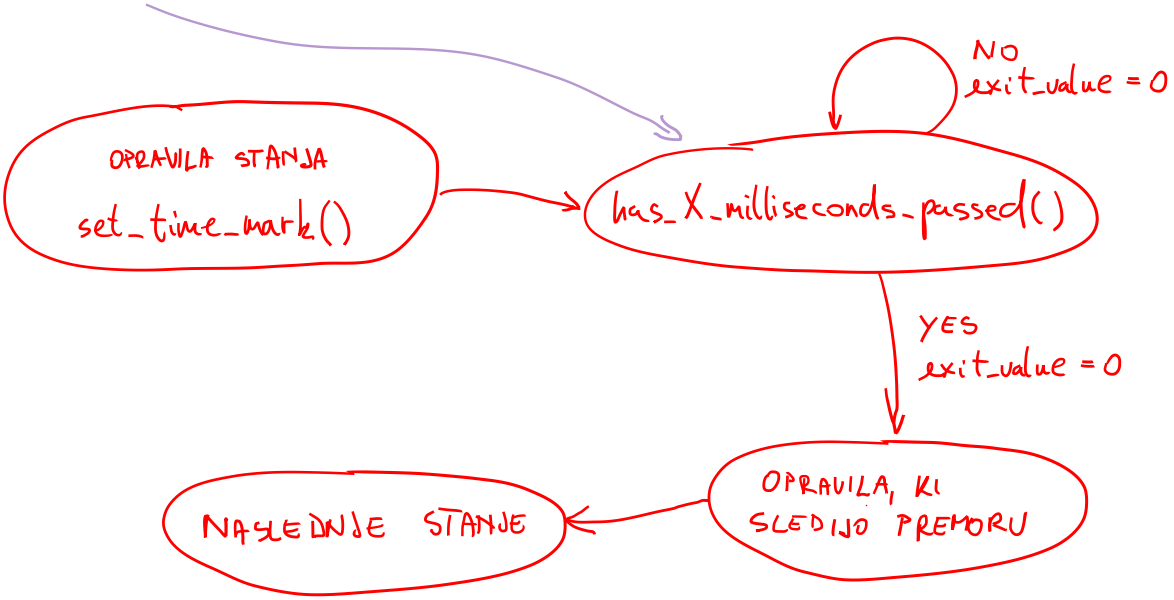


Če je torej potrebno spremeniti kako začetno (tj. privzeto, angl. "default") vrednost, to storite znotraj ustrezne `OBJ_init` funkcije.

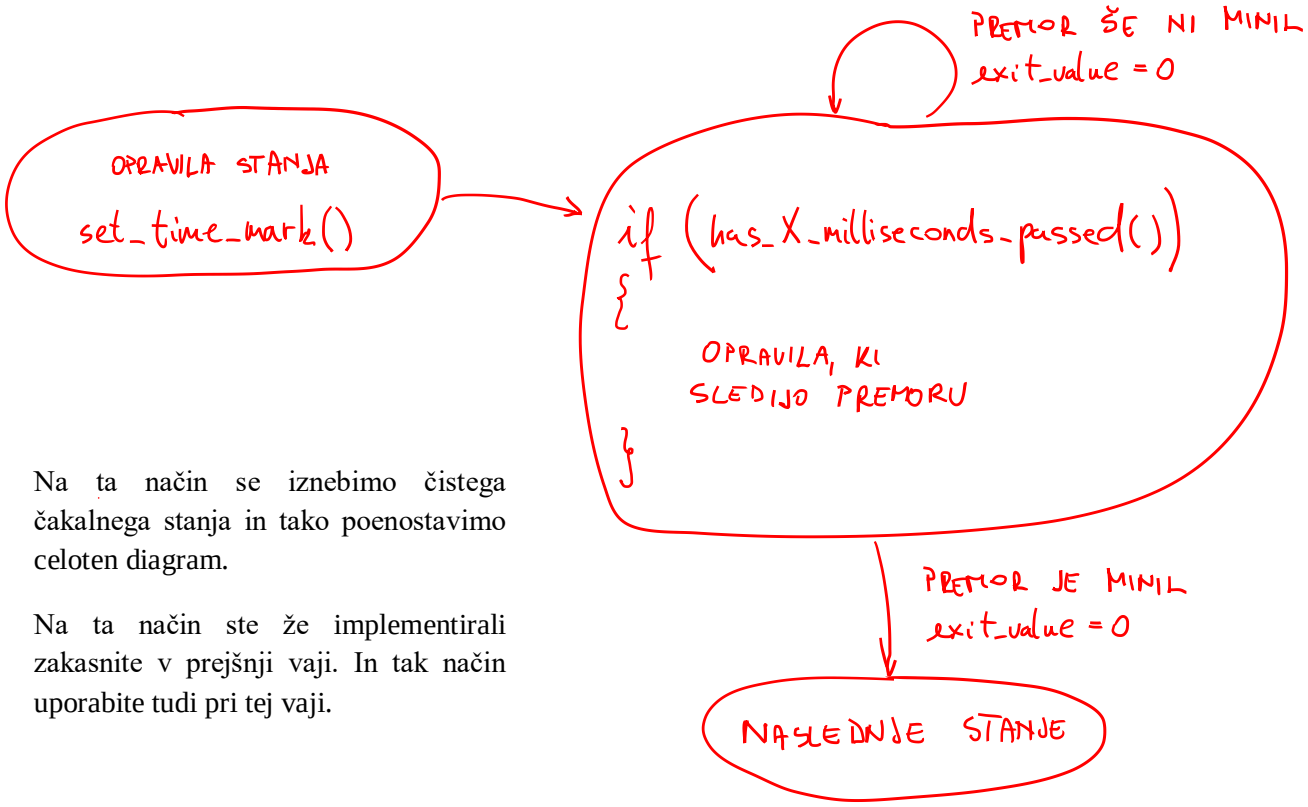
Priprava – premislek o implementaciji časovnih zakasnitev

Za konec le še kratek razmislek, kako se še lahko implementira časovna zakasnitev s pomočjo avtomata stanj.

Pri prejšnji vaji smo predstavili idejo stanja, ki je namenjen zgolj čakanju, ki smo ga poimenovali "čakalno stanje". Poglejte skico spodaj.



Ta diagram je mogoče malenkost poenostaviti tako, da se znebimo "čistega čakalnega stanja" in ga *združimo s stanjem, ki vsebuje opravila, ki sledijo takoj po premoru*. Poenostavljen diagram izgleda takole:



Na ta način se iznebimo čistega čakalnega stanja in tako poenostavimo celoten diagram.

Na ta način ste že implementirali zakasnite v prejšnji vaji. In tak način uporabite tudi pri tej vaji.