Univerzitet u Beogradu – Elektrotehnički fakultet

Katedra za elektroniku

Projekat iz računarske elektronike
Repeat Pattern Game

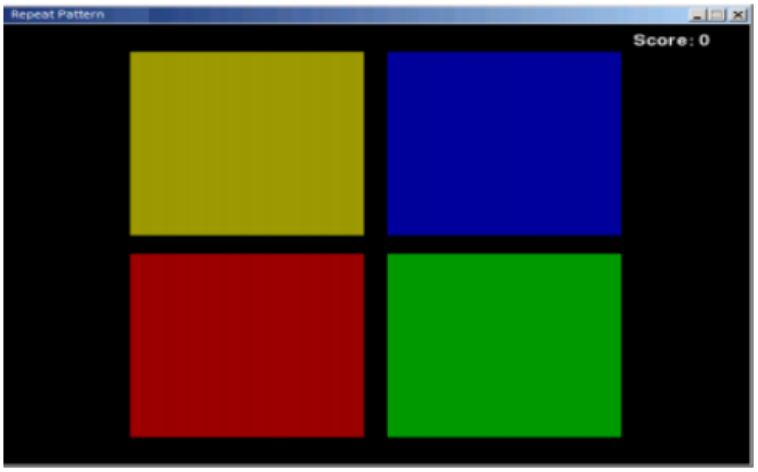
Studenti :
Ćirić Andrea 2016/0202
Cvejić Miloš 2016/0145

Beograd,
septembar 2020.

Sadržaj:

Projektni zadatak 3
Uputstvo 4
Programski kod 6
main.asm 6
procedure.inc 8
pages.asm 10
random_array.asm 15
draw_squares.asm 18
answer.asm 21
Upotrebljene gotove funkcije iz biblioteke Irvine32.inc 24

Projektni zadatak

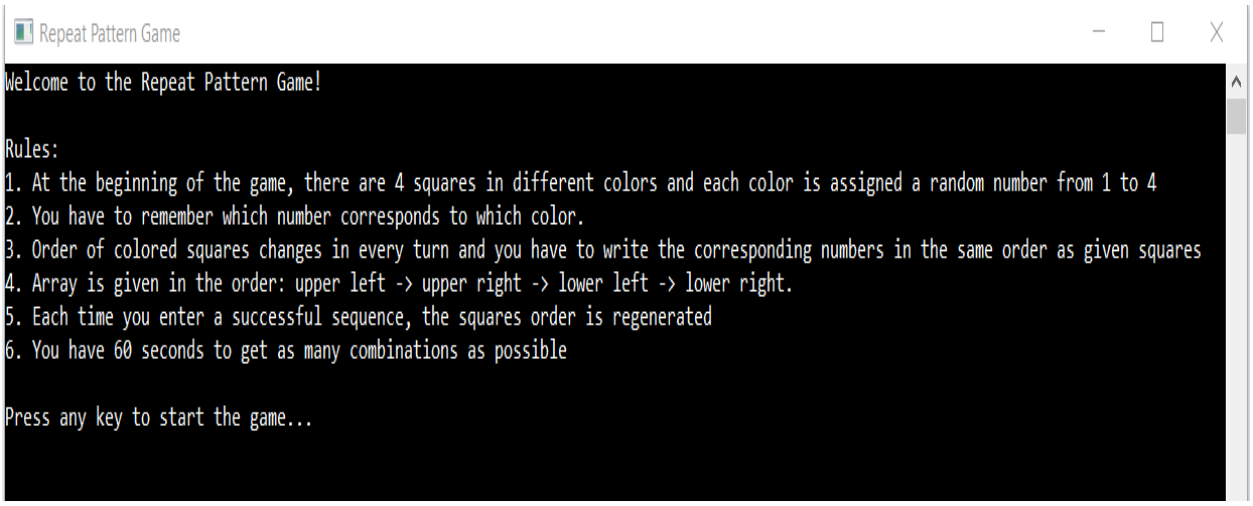


Slika 1 - slika uz projekat

Data je tebela 2x2 polja (polja veličine 8x8) sa različitim bojama. Cilj igrice je da se svaki put pogodi odgovarajuća kombinacija boja koja se prikazuje u tabeli pritiskom na odgovarajuće tastere u odgovarajućem redosledu gde svakoj od boja koje se mogu pojaviti u tabeli odgovara jedan taster. Boje koje se mogu pojaviti su: plava, crvena, žuta i zelena. Bojama se na svakom početku igrice po random šablonu dodeljuju brojevi od 1 do 4 koji zapravo predstavljaju taster koji je neposredno dodeljen toj boji. Nakon ove dodele parovi boja-taster se prikazuju igraču sve dok on ne pritisne taster “space” na tastaturi kako bi zapamtio kombinaciju, odnosno kojoj boji odgovara koji taster(broj), a zatim se kreće u izvršavanje igrice. Boje se pojavljuju u blokovima po slučajnom redosledu i poenta igrice je da se pogodi odgovarajući redosled boja pritiskom na odgovarajće tastere koji pripadaju datim bojama. Sve dok igrač ispravno pogađa boje, tabela se iznova generiše. Ukoliko igrač pogreši igra se prekida uz odgovarajuću poruku i mogućnost da se krene ispočetka.

Uputstvo

Na početku svake igre se prvo pristupa ekranu (slika 2) koji prikazuje pravila po kojima igra funkcioniše čime se korisniku jasno prikazuju smernice neophodne za razumevanje igre.

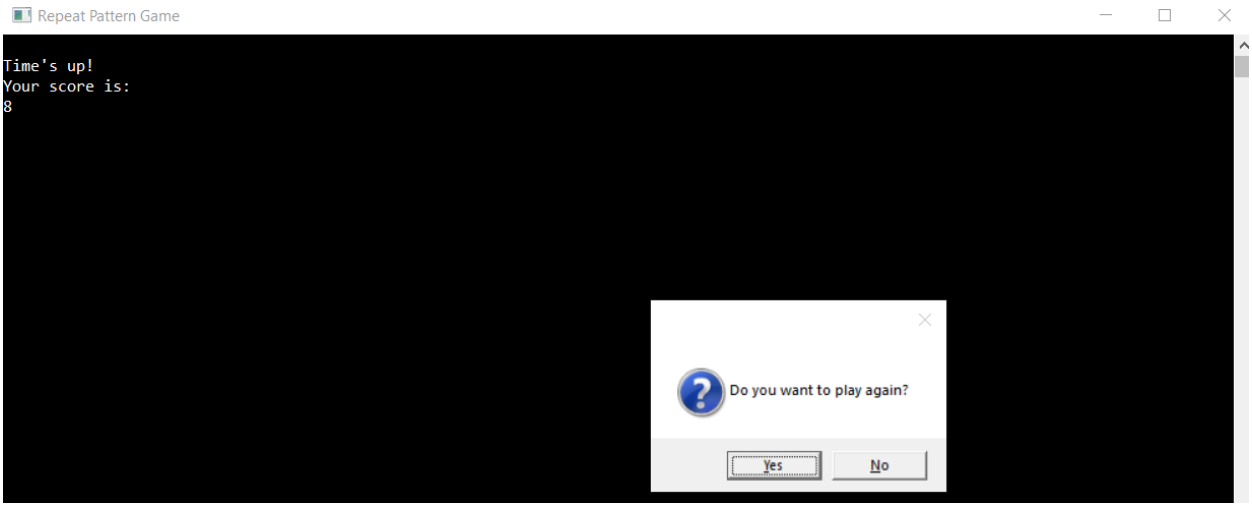


Slika 2 - Pravila igre

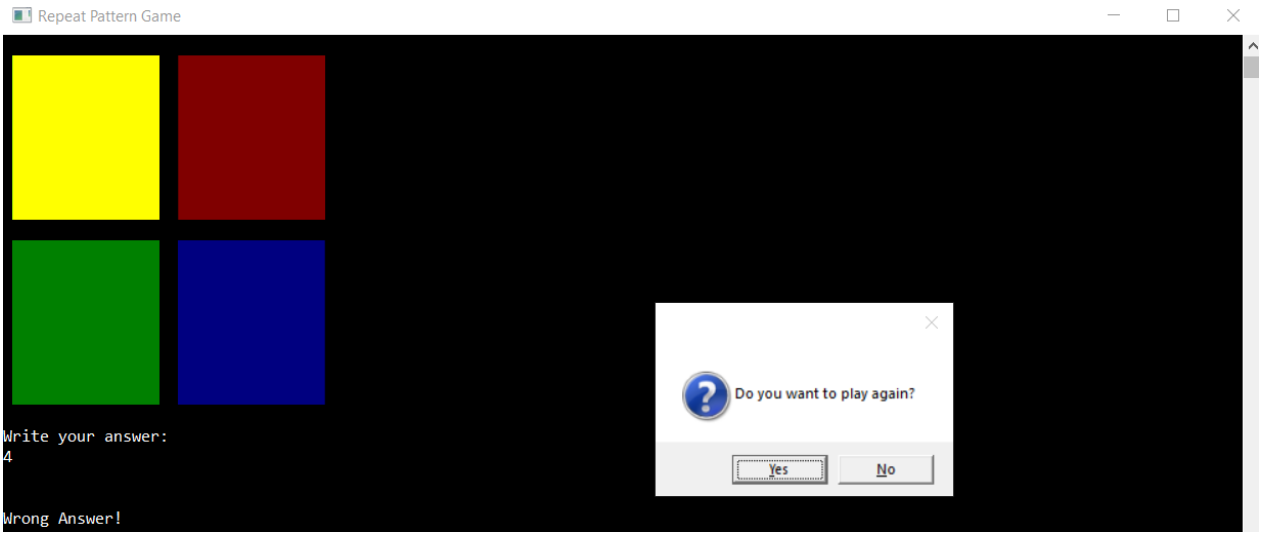
Nakon toga sledi test runda (slika 3) u kojoj svaka od boja dobija svoj ekvivalent na tastaturi. Pristupa se test primeru koji služi za proveru ispravnosti dodele boja.

Slika 3 - test runda

Nakon uspešnog test primera, pritiskom na taster otpočinje igra u kojoj za 60 sekundi treba pogoditi što više kombinacija, od kojih svaka tačna nosi 1 poen. Na kraju svake igre korisnik dobija informaciju o svom rezultatu sa odgovarajućom porukom i mogućnost da ponovo igra (slika 4). Ukoliko dođe do pogrešnog unosa igra se prekida (slika 5).



Slika 4 - kraj igre kada istekne vreme



Slika 5 - kraj igre nakon pogrešnog odgovora

Programski kod

main.asm

U main delu programa se pozivaju procedure ***start_screen*** u kojoj se definiše izgled prozora pri pokretanju igrice, ***example_screen*** u kojoj se definiše izgled prozora u toku test primer i ***game_screen*** u toku igranja igrice. Takođe i prikaz prozora za upit o ponovnom igranju na kraju igrice. Pomenute procedure nalaze se u fajlu ***pages.asm***. Unutar ovih procedura za izgled prozora pozivaju se ostale procedure za obradu podataka. Sve procedure su definisane u ***procedure.inc*** fajlu prikazanom u sledećem poglavlju.

```
INCLUDE Irvine32.inc
INCLUDE procedure.inc

.386
.model flat, stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD

.const
     n = 4      ; //number of elements in array

.data

     assign_array_indicator   BYTE      0
     score                    DWORD     0
     PlayAgain_message        BYTE      "Do you want to play again?", 0


.data?

     arrayGame      BYTE n dup(?)        ;//array of numbers generated during
the game
     arraySetup     BYTE n dup(?)        ;//numbers assigned to colors
     arrayOutColors BYTE n dup(?)        ;//order of colors on screen
     arrayOut       BYTE n dup(?)        ;//true answer array

.code
    main PROC

          INVOKE start_screen
     again:
          INVOKE example_screen, OFFSET arrayGame, OFFSET arrayOutColors,
OFFSET arraySetup, assign_array_indicator
          INVOKE game_screen, OFFSET arrayGame, OFFSET arrayOutColors, OFFSET
arraySetup, score
```

```
          ;// play again?
          mov ebx, 0
          mov edx, OFFSET PlayAgain_message
          mov assign_array_indicator, 0
          mov score, 0
          call MsgBoxAsk
          cmp eax, IDNO
          jne again

          INVOKE ExitProcess, 0

     main ENDP
END main
```

procedure.inc

U procedure.inc fajlu nalaze se dekleracije procedura koje se pozivaju u projektu. Procedure su odvojene u .asm fajlove po celinama.

```
.386
.model flat, stdcall

;//-----------------------------------------------------------------------
----------------------------------------
;// This procedure generates a random four element array with different
values [1,4]

random_array PROTO,
                    Arr: PTR BYTE,
                    ArrSetup: PTR BYTE,
                    indicator: BYTE

;//-----------------------------------------------------------------------
----------------------------------------
;// This procedure draws a square in given color at given starting position

draw_square PROTO,
                    xposition: BYTE,
                    yposition: BYTE,
                    color: PTR BYTE,
                    k: PTR DWORD

;//-----------------------------------------------------------------------
----------------------------------------
;//This procedure sets parameters for all four squares  and calls draw_square
procedure for each square

draw_squares PROTO,
                    Arr: PTR BYTE,
                    OutArr: PTR BYTE

;//-----------------------------------------------------------------------
----------------------------------------
;//This procedure generates a true answer (AnswerArr)

true_answer PROTO,
                        Arr: PTR BYTE,
                        OutArrColors: PTR BYTE,
                        AnswerArr: PTR BYTE

;//-----------------------------------------------------------------------
----------------------------------------
;//This procedure reads input and compares it with real (true) answers

get_answer PROTO,
                        Arr: PTR BYTE,
                        indicator: PTR BYTE
```

```
;//-------------------------------------------------------------------------
-----------------------------------------
;//This function sets startup screen with game instructions

start_screen PROTO

;//-------------------------------------------------------------------------
-----------------------------------------
;//This function sets example screen

example_screen PROTO,
                    Arr: PTR BYTE,
                    OutArrColors: PTR BYTE,
                    ArrSetup : PTR BYTE,
                    assign_array_indicator: BYTE

;//-------------------------------------------------------------------------
-----------------------------------------
;//This function sets game screen

game_screen PROTO,
                         Arr: PTR BYTE,
                         OutArrColors: PTR BYTE,
                         ArrSetup : PTR BYTE,
                         score: DWORD

;//-------------------------------------------------------------------------
-----------------------------------------
```

pages.asm

Pages.asm predstavlja deo koda koji služi za definisanje izgleda konzole koji podrazumeva poruke koje se ispisuju na startu svake igre, prilikom pokretanja test runde i same igre, kao i iscrtavanje samih kvadrata na unapred definisane koordinate na konzoli po zadatom rasporedu boja koji se dobijaju korišćenjem procedura ***random_array*** i ***draw_squares***.

```
INCLUDE Irvine32.inc
INCLUDE procedure.inc

.data


     indicator      BYTE 0
     arr_indicator  BYTE 0

     winTitle            BYTE      "Repeat Pattern Game", 0

     Welcome_message1    BYTE      "Welcome to the Repeat Pattern Game!",
0dh, 0ah,
                                   "  ", 0dh, 0ah,
                                   "Rules: ", 0dh, 0ah,
                                   "1. At the beginning of the game, there
are 4 squares in different colors and each color is assigned a random number
from 1 to 4", 0dh, 0ah, 0

     Welcome_message2    BYTE      "2. You have to remember which number
corresponds to which color.", 0dh, 0ah,
                                   "3. Order of colored squares changes in
every turn and you have to write the corresponding numbers in the same order
as given squares", 0dh, 0ah,
                                   "4. Array is given in the order: upper
left -> upper right -> lower left -> lower right.", 0dh, 0ah, 0

     Welcome_message3    BYTE      "5. Each time you enter a successful
sequence, the squares order is regenerated", 0dh, 0ah,
                                   "6. You have 60 seconds to get as many
combinations as possible", 0dh, 0ah,
                                   "  ", 0dh, 0ah,
                                   "Press any key to start the game...", 0dh,
0ah, 0

     Color_message       BYTE      " ", 0dh, 0ah,
                                   " ", 0dh, 0ah,
                                   "The order of colors in sequence is: BLUE
GREEN RED YELLOW -> ", 0

     Blank_message       BYTE      " ", 0dh, 0ah,
                                   " ", 0dh, 0ah,0

     Test_message        BYTE      "THIS IS THE TEST ROUND!", 0dh, 0ah,
                                   " ", 0dh, 0ah, 0
```

```
     Answer_message      BYTE      "Write your answer: ", 0dh, 0ah, 0

     TryAgain_message    BYTE      "Try again!", 0dh, 0ah, 0

     Start_message       BYTE      " ", 0dh, 0ah,
                                   "WELL DONE!", 0dh, 0ah,
                                   "Press any key to start the game...", 0dh,
0ah, 0

     EndGame_message     BYTE      " ", 0dh, 0ah,
                                   "Wrong Answer!", 0dh, 0ah, 0

     TimeOut_message     BYTE      " ", 0dh, 0ah,
                                   "Time's up! ", 0dh, 0ah,
                                   "Your score is: ", 0dh, 0ah, 0

.data?
     arrayOut        BYTE      4     dup (?)
     time            DWORD     ?
     arraySetup      BYTE      4     dup (?)

.code
;//-------------------------------------------------------------------------
-----------------------------------------
;//This function sets startup screen with game instructions

     start_screen PROC

          push edx
          push eax

          INVOKE SetConsoleTitle, ADDR WinTitle

          call clrscr
          mov eax, 15
          call SetTextColor
          mov edx, OFFSET Welcome_message1
          call WriteString
          mov edx, OFFSET Welcome_message2
          call WriteString
          mov edx, OFFSET Welcome_message3
          call WriteString
          call readchar
          call clrscr

          pop eax
          pop edx
          ret

     start_screen ENDP

;//-------------------------------------------------------------------------
-----------------------------------------
;//This function sets example screen
```

```
    example_screen PROC,
                        Arr: PTR BYTE,
                        OutArrColors: PTR BYTE,
                        ArrSetup : PTR BYTE,
                        assign_array_indicator: BYTE
;//-------------------------------------------------------------------------
----------------------------------------

                push edx
                push eax
                push ecx
                push ebx

                call clrscr

                INVOKE random_array, Arr, ArrSetup, assign_array_indicator
                mov assign_array_indicator, 1
                INVOKE draw_squares, Arr, OutArrColors

                mov eax, 15
                call SetTextColor

                mov edx, OFFSET Color_message
                call WriteString

                mov ecx, 4
                mov edx, Arr
             L:
                mov ebx, [edx]
                movzx eax, bl
                call WriteDec
                mov al, 20h
                call WriteChar
                inc edx
                loop L

                mov edx, OFFSET Blank_message
                call WriteString

                mov edx, OFFSET Test_message
                call WriteString

                mov edx, OFFSET Answer_message
                call WriteString

                INVOKE true_answer, ArrSetup, OutArrColors, OFFSET arrayOut

           again:
                INVOKE get_answer, OFFSET arrayOut, OFFSET indicator
                movzx edi, indicator
                .if ( edi == 0)
                     mov edx, OFFSET TryAgain_message
                     call WriteString
                     mov edx, OFFSET Answer_message
                     call WriteString
                     jmp again
```

```
                .endif

                mov edx, OFFSET Start_message
                call WriteString
                call ReadChar
                call clrscr

                pop ebx
                pop ecx
                pop eax
                pop edx
                ret

     example_screen ENDP

;//-------------------------------------------------------------------------------
-----------------------------------------
;//This function sets game screen

     game_screen PROC,
                          Arr: PTR BYTE,
                          OutArrColors: PTR BYTE,
                          ArrSetup : PTR BYTE,
                          score: DWORD
;//-------------------------------------------------------------------------------
-----------------------------------------

          push edx
          push eax
          push ebx

          call GetMseconds
          mov time, eax

 again:
          call clrscr
          INVOKE random_array, Arr, ArrSetup, 1
          INVOKE draw_squares, Arr, OutArrColors

          mov eax, 15
          call SetTextColor

          mov edx, OFFSET Blank_message
          call WriteString

          mov edx, OFFSET Answer_message
          call WriteString

          INVOKE true_answer, ArrSetup, OutArrColors, OFFSET arrayOut

          INVOKE get_answer, OFFSET arrayOut, OFFSET indicator
          movzx edi, indicator
          .if (edi == 0)
               mov edx, OFFSET EndGame_message
               call WriteString
```

```
                jmp the_end
            .endif

            call GetMseconds
            sub eax, time
            .if (eax < 60000)
                inc score
                call clrscr
                jmp again
            .else
                call clrscr
                mov edx, OFFSET TimeOut_message
                call WriteString
                mov eax, score
                call WriteDec

            .endif


        the_end:
            pop ebx
            pop eax
            pop edx
            ret

         game_screen ENDP

END
```

random_array.asm

U ***random_array.asm*** se generiše niz slučajnih brojeva od 1 do 4, pri čemu svi elementi moraju biti jedinstveni .

```
INCLUDE Irvine32.inc
INCLUDE procedure.inc

.data

     uno byte 0
     due byte 0
     tre byte 0
     quattro byte 0

.code

;//-------------------------------------------------------------------------
-------------------------------------
;// This procedure generates a random four element array with different
values [1,4]
random_array PROC,
                    Arr: PTR BYTE,
                    ArrSetup : PTR BYTE,
                    indicator : BYTE
;//-------------------------------------------------------------------------
-------------------------------------
          push ecx
          push esi
          push edi
          push eax
          push ebx

          call Randomize

     variable_reset :
          mov uno, 0
          mov due, 0
          mov tre, 0
          mov quattro, 0

          mov ecx, 4
          mov esi, 0
          mov edi, Arr
          mov ebx, ArrSetup

     assign:                         ;//generating random value
          mov eax, 4
          call RandomRange
          add eax, 1

          cmp al, uno
```

```
          je assign
          cmp al, due
          je assign
          cmp al, tre
          je assign
          cmp al, quattro
          je assign

          cmp esi, 0
          je assign_first
          cmp esi, 1
          je assign_second
          cmp esi, 2
          je assign_third
          cmp esi, 3
          je assign_fourth
          cmp esi, 4
          je assign_end


     ;// assign values to constants

     assign_first:
          mov uno, al
          jmp assign_array

     assign_second :
          mov due, al
          jmp assign_array

     assign_third :
          mov tre, al
          jmp assign_array

     assign_fourth :
          mov quattro, al
          jmp assign_array


     assign_end :
          mov ecx, 4
          mov esi, 0

          jmp the_end


     assign_array :                ;// writing elements into array

          mov [edi], al

          .if indicator == 0    ;// Generates arraySetup at the begining of
the game
               mov[ebx], al
               inc ebx
          .endif
```

```
            inc esi
            inc edi

            ; //loop assign
            dec ecx
            cmp ecx, 0
            jnz assign

      the_end:

            pop ebx
            pop eax
            pop edi
            pop esi
            pop ecx
            ret

random_array ENDP

END
```

draw_squares.asm

Draw_squares.asm služi za iscrtavanje kvadrata na konzoli tako što joj se kroz ulazne parametre prosleđuju koordinate na konzoli od kojih počinje da se crta kvadrat kao i boja kvadrata. Dimenzije kvadrata su unapred definisane na početku procedure.

```
INCLUDE Irvine32.inc
INCLUDE procedure.inc

.data
colors BYTE 0h, 1h, 2h, 4h, 0Eh
p dword 0

.data?
     xposition BYTE ?
     yposition BYTE ?

.code
;//-----------------------------------------------------------------------------
-
;// This procedure draws a square in given colour at given starting position
     draw_square PROC,
                          xpos: BYTE,
                          ypos: BYTE,
                          color: PTR BYTE,
                          k:  PTR DWORD

;//-----------------------------------------------------------------------------
-
          push edx
          push eax
          push ebx
          push edi
          push esi

          ;//dimensions of a square (how many 0DBh objects in a row and
column)
               mov esi, 16
               mov edi, 8

          ;//setting parameters for drawing functions
               mov dh, xpos
               mov dl, ypos
               mov ebx, color
               mov ecx, [ebx]
               movzx eax, cl
               mov ecx, k
               mov [ecx], eax
               call SetTextColor
               mov al, 0DBh

          ;//drawing 0DBh objects
          .WHILE(esi > 0)
```

```
                .WHILE(edi > 0)
                        call Gotoxy
                        call WriteChar
                        dec edi
                        inc dh
                .ENDW
                mov dh, xpos
                mov edi, 8
                dec esi
                inc dl
        .ENDW

                pop esi
                pop edi
                pop ebx
                pop eax
                pop edx
                ret
    draw_square ENDP

;//-----------------------------------------------------------------------------
-------------------------------
;//This procedure sets parameters for all four squares  and calls draw_square
procedure for each square

    draw_squares PROC,
                        Arr: PTR BYTE,
                        OutArrColors: PTR BYTE
;//-----------------------------------------------------------------------------
-------------------------------
                pushad

                mov edi, 0
                mov ebx, Arr
                mov edx, OutArrColors

        draw :
                mov ecx, [ebx]
                movzx eax, cl
                mov ecx, eax
                mov eax, OFFSET colors
                mov esi, 0h

                ;//sets a color for each square
                .while (esi != ecx)
                        inc esi
                        inc eax
                .endw

                CMP edi, 0
                JE first_square
                CMP edi, 1
                JE second_square
                CMP edi, 2
                JE third_square
```

```
            CMP edi, 3
            JE fourth_square

        ;//setting position of squares
        first_square :
            mov xposition, 1
            mov yposition, 1
            mov edi, 1
            JMP variables_set

        second_square :
            mov xposition, 1
            mov yposition, 19
            mov edi, 2
            JMP variables_set

        third_square :
            mov xposition, 10
            mov yposition, 1
            mov edi, 3
            JMP variables_set

        fourth_square :
            mov xposition, 10
            mov yposition, 19
            mov edi, 4

        variables_set :
            INVOKE draw_square, xposition, yposition, eax, OFFSET p
            mov eax, [p]                    ;// saves the order of colors on
the screen
            mov [edx], eax
            inc edx
            inc ebx
            cmp edi, 4
            jne draw

            popad
        ret
    draw_squares ENDP

END
```

answer.asm

Procedura ***answer.asm*** proverava tačnost unesene sekvence tako što upoređuje unos sa tastature sa tačnom vrednošću iz generisanog niza.

```
INCLUDE Irvine32.inc
INCLUDE procedure.inc

.data
     temp BYTE 0

.code

;//-------------------------------------------------------------------------
------------------------
;//This procedure generates a true answer (AnswerArr)

    true_answer PROC,
                          Arr: PTR BYTE,
                          OutArrColors: PTR BYTE,
                          AnswerArr: PTR BYTE
;//-------------------------------------------------------------------------
------------------------
          pushad

          mov esi, 1
          mov eax, Arr
          mov ecx, OutArrColors
          mov ebx, AnswerArr

    again:
          mov edx, [ecx]
          movzx edi, dl

          ;//determines what color it is

          cmp edi, 1h
          jz b

          cmp edi, 2h
          jz g

          cmp edi, 4h
          jz r

          cmp edi, 0Eh
          jz y

          ;//determines which number is assigned to that color

     b:
          mov edx, [eax]
```

```
            mov [ebx], dl
            jmp next

      g:
            add eax, 1
            mov edx, [eax]
            mov [ebx], dl
            jmp next

      r:
            add eax, 2
            mov edx, [eax]
            mov [ebx], dl
            jmp next

      y:
            add eax, 3
            mov edx, [eax]
            mov[ebx], dl
            jmp next

      next:

            mov eax, Arr
            inc ebx
            inc ecx
            inc esi
            cmp esi, 5
            jnz again

            popad
            ret
      true_answer ENDP

;//-------------------------------------------------------------------------
------------------------
;//This procedure reads input and compares it with real (true) answers
      get_answer PROC,
                              Arr: PTR BYTE,
                              indicator: PTR BYTE
;//-------------------------------------------------------------------------
------------------------

            pushad

            mov esi, 1
            mov ebx, Arr

      again:
            mov ecx, [ebx]
            movzx edx, cl
            call ReadInt
            cmp eax, edx
            jne end_false
            mov ecx, indicator
            mov edi, 1
```

```
            mov [ecx], edi
            inc ebx
            inc esi
            cmp esi, 5
            jne again
            jmp the_end

      end_false:
            mov ecx, indicator
            mov edi, 0
            mov [ecx], edi

      the_end:

            popad
            ret
      get_answer ENDP
```

```
END
```

Upotrebljene gotove funkcije iz biblioteke Irvine32.inc

Clrscr - The *Clrscr* procedure clears the console window.

WriteString - The *WriteString* procedure writes a null-terminated string to the console window. Pass the string's offset in EDX.

WriteDec - The *WriteDec* procedure writes a 32-bit unsigned integer to the console window in decimal format with no leading zeros. Pass the integer in EAX.

GetMseconds - The *GetMseconds* procedure gets the number of milliseconds elapsed since midnight on the host computer, and returns the value in the EAX register. The procedure is a great tool for measuring the time between events. No input parameters are required.

SetTextColor - The *SetTextColor* procedure sets the foreground and background colors for text output. When calling *SetTextColor*, assign a color attribute to EAX.

ReadChar - The *ReadChar* procedure reads a single character from the keyboard and returns the character in the AL register.

WriteChar - The *WriteChar* procedure writes a single character to the console window. Pass the character (or its ASCII code) in AL.

SetConsoleTitle - The *SetConsoleTitle* function lets you change the console window's title.

Randomize - The *Randomize* procedure initializes the starting seed value of the Random32 and RandomRange procedures. The seed equals the time of day, accurate to 1/100 of a second. Each time you run a program that calls Random32 and RandomRange, the generated sequence of random numbers will be unique. You need only to call Randomize once at the beginning of a program.

RandomRange - The *RandomRange* procedure produces a random integer within the range of 0 to n-1, where n is an input parameter passed in the EAX register. The random integer is returned in EAX.

Gotoxy - The *Gotoxy* procedure locates the cursor at a given row and column in the console window. By default, the console window's X-coordinate range is 0 to 79 and the Y-coordinate range is 0 to 24. When you call *Gotoxy*, pass the Y-coordinate (row) in DH and the X-coordinate (column) in DL.