Univerzitet u Beogradu
Elektrotehnički fakultet

Igra memorije

Projekat 11

Arijan Amigh 2013/0087
Filip Marković 2013/0400

August 21, 2017

Sadržaj

1 Zadatak

Potrebno je realizovati igru memorije. U igri je moguće okrenuti maksimalno 2 karte u jednom potezu. Ukoliko su slike na kartama identične, karte ostaju okrenute i dobija se **K** poena. Ukoliko nisu, gubi se **K/4** poena. Nije moguće ponovo okretati okrenute karte. Igra se završava pritiskom na taster ESC ili pogadjanjem svih parova karata.

2 Realizacija

2.1 Makroi

Realizovana je igra memorije. Podešavanjem makroa **M** i **N** na vrhu fajla moguće je menjati širini i visinu table. **M** predstavlja broj redova dok **N** predstavlja broj kolona. Moguće je takodje makroima **X_SIZE, Y_SIZE, X_PADDING** i **Y_PADDING** podešavati veličinu pojedinačnog polja (u "pikselima"), ali je za to potrebno naknadno napraviti bazu binarnih slika iste veličine. Takodje je moguće podesiti količinu poena tačnog uparivanja karti preko makroa POINTS.

2.2 Procedure

2.2.1 linearIndex

Ova procedura linearizuje indekse - pretvara matricu u niz. Matrica se rastavlja na nizove po vrstama. Prima 3 argumenta - vrstu, kolonu i broj kolona. Računa linearni indeks po jednostavnoj formuli $i_{LIN} = i \cdot N + j - 1$. Gde je i vrsta, j kolona a N broj kolona matrice. Oduzima se 1 jer indeksiranje počinje od 0. Rezultat - linearni indeks - procedura smešta u registar eax za upotrebu u pozivajućoj proceduri/glavnom programu.

2.2.2 escCheck

Ova procedura proverava da li je pritisnut taster "ESC". Ukoliko jeste izlazi se iz programa. Procedura ima 2 režima rada. Prima 2 argumenta. Ukoliko je arg2 jednak 0, procedura ne poziva "readKey", već proverava da li se arg1 poklapa sa kodom "ESC" tastera. Ukoliko je arg2 jednak 1, procedura poziva "readKey" i proverava da li je rezultat u **dx** registru jednak kodu "ESC" tastera.

2.2.3 gameOver

Ova procedura se poziva kada se sve karte upare. Ne prihvata nikakve argumente. Ispisuje pobednički tekst sa krajnjim rezulatom i ciklično prolazi menja boje ispisanog teksta.

2.2.4 updateScore

Ova procedura ispisuje trenutni rezultat tokom igre. Prvo proverava da li se rezultat promenio nakon prošlog ispisa, i ako jeste - ispisuje novi rezultat. Procedura ne prima argumente.

2.2.5 refreshSquare

Ova procedura se koristi za iscrtavanje table i otkrivenih i neotkrivenih karata. Prima 3 argumenta - vrstu, kolonu i režim rada.

Preko prosledjene vrste i kolone računa koordinate te vrste i kolone na tabeli uz pomoć **X_SIZE, X_PADDING, Y_SIZE** i **Y_PADDING** makroa. Režim rada definiše šta će procedura da iscrta na izračunatoj poziciji u konzoli. Prosledjeni režim može biti:

- mode=0 - iscrtavanje neaktivnog pokrivene karte
- mode=1 - iscrtavanje aktivne (selektovane) pokrivene karte
- mode=2 - iscrtavanje neaktivnog otkrivene karte
- mode=3 - iscrtavanje aktivne (selektovane) otkrivene karte

2.2.6 generateRandom

Ova procedura generiše niz parova brojeva veličine $M \times N$, koji su nasumično rasporedjeni po nizu. Razbacani parovi brojeva predstavljaju sliku ispod svake karte. Procedura ne prima argumente.

2.2.7 main

Ovde se nalazi glavni deo logike programa, zajedno sa definisanjem binarnih slika, glavne *game loop* petlje u kojoj se čita pritisak tastera, ažurira lista otkrivenih i neotkrivenih karata, povećava ili smanjuje rezultat, kontroliše iscrtavanje na tabli blokova i tako dalje.

3 Izvorni kod

1: projekat11.asm

```
include irvine32.inc
INCLUDE Macros.inc


;//---------------------------------------------------------------
;//BEGIN MACROS
N EQU 2;// 15 is max if in full screen
M EQU 2;// unlimited

;//BLOCK SIZES AND PADDINGS
X_PADDING EQU 2
Y_PADDING EQU 1
X_SIZE    EQU 20
Y_SIZE    EQU 10

;//M*N have to be even, and < 100
P EQU M*N

;//POINTS FOR CORRECT MATCH
POINTS EQU 20
;//END MACROS
;//---------------------------------------------------------------


;// ---------------- Structures ----------------------------------
_INPUT_RECORD STRUCT
; // original INPUT_RECORD struct is not working,
;//this one is made by INPUT_RECORD definition in
;// Irvine library documentation
EventType   WORD ?
WORD ? ; //For alignment
UNION
KeyEvent              KEY_EVENT_RECORD          <>
MouseEvent            MOUSE_EVENT_RECORD        <>
WindowBufferSizeEvent WINDOW_BUFFER_SIZE_RECORD <>
MenuEvent             MENU_EVENT_RECORD         <>
FocusEvent            FOCUS_EVENT_RECORD        <>
ENDS
_INPUT_RECORD ENDS

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

2: projekat11.asm

```
;//-------------------------------------------------------------
;//BRIEF:
;//This procedure takes linearizes matrix indexes
;//It takes 3 arguments - row, column and number of columns
;//Linear indexing is done in rows

;//BEGIN linearIndex PROCEDURE
.code
linearIndex proc c,
mrow: dword, mcol: dword, maxCol:dword

mov eax, mrow
dec eax
mov ebx, maxCol
mul ebx
add eax, mcol
dec eax

ret
linearIndex endp
;//END linearIndex PROCEDURE
;//-------------------------------------------------------------

;//-------------------------------------------------------------
;//BRIEF:
;//This procedure is called when all the cards have been turned
;//It takes no arguments
;//It displays a winning message and cycles
;// through colors for the text of that message
;//BEGIN gameOver PROCEDURE
.data
gameOverString byte
"␣WELL␣DONE!␣YOU␣HAVE␣MATCHED␣ALL␣THE␣CARDS.", 0dh, 0ah,
"␣YOU␣SHOULD␣BE␣PROUD␣OF␣YOURSELF!!!11!", 0

.code
gameOver proc c uses eax

mov  eax, 300; sleep, to allow OS to time slice
call Delay;
call Clrscr

call GetMaxXY
shr dl, 1
shr dh, 1
call Gotoxy
```

3: projekat11.asm

```
xor ebx, ebx
mov bl, 8
mov bh, 0
winner:
mov  eax, 100; sleep, to allow OS to time slice
call Delay;
xor eax, eax
mov dl, 16
mov al, bl
mul dl
add al, bh
;//postavljamo boju kvadrata koji treba iscrtati
call SetTextColor

mov dl, 0
mov dh, 20
call Gotoxy

mov  edx, offset gameOverString
;// Greeting message
call WriteString

call Crlf

mov edx, offset scoreString
call WriteString
mov eax, score
call WriteInt

inc bl
inc bh
jmp winner

ret
gameOver endp
;//END gameOver PROCEDURE
;//--------------------------------------------------------------
```

4: projekat11.asm

```
;//-------------------------------------------------------------
;//BRIEF:
;//This procedure updates the score on the screen
;//It takes no arguments
;//If the score is the same as it was the
;// last time the procedure was
;//called - then nothing is written to the console
;//BEGIN updateScore PROCEDURE
.data
lastScore sdword 1

.code
updateScore proc c uses eax

mov eax, score
cmp lastScore, eax
je dontUpdateScore

mov lastScore, eax

mov al, 16 * (4) + 7
;//postavljamo boju kvadrata koji treba iscrtati
call SetTextColor

mov dl, X_PADDING
mov eax, M
mov ebx, Y_SIZE + Y_PADDING
mul ebx
add eax, 6 * Y_PADDING
mov dh, al
call Gotoxy
mov edx, offset scorePadding
call WriteString
mov dl, X_PADDING
mov dh, al
call Gotoxy
mov edx, offset scoreString
call WriteString
mov eax, score
call WriteInt

dontUpdateScore:
ret
updateScore endp
;//END updateScore PROCEDURE
;//-------------------------------------------------------------
```

5: projekat11.asm

```
;//-------------------------------------------------------------
;//BRIEF:
;//This procedure takes draws squares
;//It takes 3 arguments - row, column and mode (of operation)
;//It calculates the X and Y position of passed
;//row and column arguments. It does this by
;// using the SIZE and PADDING macros.
;//Modes of operation:
;//mode==0 -> draw inactive covered square in given postion
;//mode==1 -> draw active (highlighted) covered
;// square in given postion
;//mode==2 -> draw inactive uncovered shape in
;// given postion
;//mode==3 -> draw active uncovered (highlighted)
;// shape in given postion
;//BEGIN refreshSquare PROCEDURE
.data
x_0 byte 0
x_1 byte 0
y_0 byte 0
y_1 byte 0

shape dword 0

.code
refreshSquare proc c uses eax ebx edx ecx,
row:dword, column : dword, mode : dword

;//BEGIN calculating the coordinates based
;// on column and row number and block sizes and padding
xor eax, eax
xor ebx, ebx
mov eax, row
dec eax
mov ebx, Y_PADDING+Y_SIZE+1
mul ebx
add eax, Y_PADDING
mov y_0, al
mov y_1, Y_SIZE
add y_1, al

xor eax, eax
xor ebx, ebx
mov eax, column
dec eax
mov ebx, X_PADDING+X_SIZE+1
mul ebx
add eax, X_PADDING
mov x_0, al
mov x_1, X_SIZE
;//END calculating the coordinates based
;// on column and row number and block sizes and padding
```

6: projekat11.asm

```
;//Clearing all registers
xor eax, eax
xor edx, edx
xor ebx, ebx
xor ecx, ecx


mov al, 16 * (8) + 0
;//postavljamo boju kvadrata koji treba iscrtati
call SetTextColor
cmp mode, 2
je uncovered
;//change color here
mov al, 16 * (4) + 7
;//postavljamo boju kvadrata koji treba iscrtati
call SetTextColor
cmp mode, 3
je uncovered



mov al, 16 * (8) + 7
;//postavljamo boju kvadrata koji treba iscrtati
cmp mode, 0
je greyBlock
mov al, 16 * (8) + 4
;//postavljamo boju kvadrata koji treba iscrtati
greyBlock:
call SetTextColor



mov dh, y_0;// u dh stavljamo y cursor position

drawY:;// iscrtavanje po vertikali
xor eax, eax
mov dl, x_0;// u dl stavljamo x cursor position
mov al, y_1
sub al, y_0
add al, 1
mov cl, x_1;//u dl stavljamo x cursor position
mov al, 0DBh; //solid - block
```

7: projekat11.asm

```
drawX:; //iscrtavamo po x osi ecx puta
call Gotoxy
call WriteChar
inc dl
loop drawX
cmp dh, y_1
jz doneBlock

inc dh
jmp drawY

doneBlock :
add x_0, X_PADDING + X_SIZE + 1
;//doneOneRow:
add y_0, Y_SIZE + Y_PADDING + 1
add y_1, Y_SIZE + Y_PADDING + 1
mov x_0, X_PADDING
ret


;//DRAWING SHAPES
uncovered:

dec y_1;//padding for shape inside block

;//Calculating linear index
push N
push column
push row
call linearIndex
mov ebx, 4
mul ebx


mov ebx, object_array[eax]
;//mapping with random pair array
dec ebx;//indexing starts from 0
mov ebx, shapeArray[ebx * 4]
;//shapeArray contains adresses of
;// shapes in memory (dword => *4)
mov shape, ebx

xor eax, eax
xor ebx, ebx

mov dh, y_0;// u dh stavljamo y cursor position
inc dh;//padding for shape inside block
```

8: projekat11.asm

```
drawY2:;// iscrtavanje po vertikali
mov dl, x_0;// u dl stavljamo x cursor position
add dl, 2;//padding for shape inside block
mov al, y_1
dec al;//padding for shape inside block
sub al, y_0
add al, 1
mov cl, x_1;//u dl stavljamo x cursor position
sub cl, 4;//padding for shape inside block

;mov al, 0DBh; //solid - block
mov ebx, shape
mov bx, [ebx]

drawX2:; //iscrtavamo po x osi ecx puta
mov al, 32;//space ascii
test bx, 8000h
jz drawSpace
mov al, 0DBh

drawSpace:
call Gotoxy
call WriteChar
inc dl
shl bx, 1
loop drawX2
cmp dh, y_1
jz doneBlock

inc dh
add shape, 2
jmp drawY2

doneBlock2 :
add x_0, X_PADDING + X_SIZE + 1
;//doneOneRow:
add y_0, Y_SIZE + Y_PADDING + 1
add y_1, Y_SIZE + Y_PADDING + 1
mov x_0, X_PADDING


ret
refreshSquare endp
;//END refreshSquare PROCEDURE
;//--------------------------------------------------------------
```

9: projekat11.asm

```
;//-------------------------------------------------------------
;//BRIEF:
;//This procedure generates an array
;//of randomly scattered pairs of numbers
;//The array represents the matrix of
;// cards - each pair of numbers equates to the same shapes
;//BEGIN generateRandom procedure
.data
rand_array DWORD P DUP(? )
ranCnt DWORD P
isSecond BYTE 1

.code
generateRandom proc c uses eax

;//Generating random array of size P
mov edi, OFFSET object_array
xor ecx, ecx;// clear counter
xor eax, eax
inc eax

G1 :
.IF isSecond == 2
mov dword ptr[edi + ecx * 4], eax
mov isSecond, 1
inc eax

.ELSE
mov dword ptr[edi + ecx * 4], eax
inc isSecond
.ENDIF

inc ecx;//increment counter
cmp ecx, LENGTHOF object_array
jne G1

call Crlf
```

10: projekat11.asm

```
;//FOR DEBUG PURPOSES ONLY
;//BEGIN WRITING OUT RANDOM ARRAY
;;mov edi, OFFSET object_array
;;xor ecx, ecx; clear counter
;;L1:
;;mov ax, [edi + ecx * 4]
;//get number from object_array(*2 ili * 4, zavisi)
;;call WriteDec
;;
;;mov eax, '␣'
;;call WriteChar
;;
;;inc ecx;//increment counter
;;cmp ecx, LENGTHOF object_array
;;jne L1
;;
;;call Crlf
;//END WRITING OUT RANDOM ARRAY
;//FOR DEBUG PURPOSES ONLY


;//random niz

mov edi, OFFSET rand_array
xor ecx, ecx; clear counter

call Randomize
R1 :
mov eax, ranCnt
call RandomRange
dec     ranCnt
mov[edi + ecx * 4], eax
;call WriteDec
;mov eax, '␣'
;call WriteChar

inc ecx; increment counter
cmp ecx, LENGTHOF object_array
jne R1

call Crlf

;//
xor ecx, ecx; clear counter

J1 :
mov edi, OFFSET rand_array
mov ebx, [edi + ecx * 4]
;//get number from rand_array(*2 ili * 4, zavisi)
mov eax, ecx
add eax, ebx
```

11: projekat11.asm

```
mov edi, OFFSET object_array
mov ebx, [edi + (eax) * 4]
;//clan sa kojim menjamo trenutni clan
mov edx, [edi + ecx * 4]
;//trenutni clan


mov dword ptr[edi + ecx * 4], ebx

mov dword ptr[edi + (eax) * 4], edx

inc ecx;//increment counter
cmp ecx, LENGTHOF object_array
jne J1

call Crlf

;//FOR DEBUG PURPOSES ONLY
;//BEGIN WRITING OUT RANDOM ARRAY
;;mov edi, OFFSET object_array
;;xor ecx, ecx; clear counter
;;
;;L7:
;;mov ax, [edi + ecx * 4]
;//get number from object_array(*2 ili * 4, zavisi)
;;call WriteDec
;;
;;mov eax, '␣'
;;call WriteChar
;;
;;inc ecx;//increment counter
;;cmp ecx, LENGTHOF object_array
;;jne L7
;;
;;call Crlf
;//END WRITING OUT RANDOM ARRAY
;//FOR DEBUG PURPOSES ONLY

ret
generateRandom endp
;//END generateRandom PROCEDURE
;//------------------------------------------------------------
```

12: projekat11.asm

```
;//------------------------------------------------------
;//BEGIN main PROCEDURE
.data
titleStr BYTE "Racunarska␣eletronika␣-␣PROJEKAT:␣Memory␣Game", 0
greeting BYTE "Memory␣Game", 0dh, 0ah, "Close␣the␣window,␣or␣press␣""Es

object_array DWORD P DUP(? );//array that holds the sequence of random

;//Defining images for cards
shapeArray dword 10 DUP(0)
square word 0000h, 7 DUP(3FFCh), 0000h
line word 9 DUP(0180h)
triline word 9 DUP(0C183h)
hline word 4 DUP(0000h), 0FFFFh, 4 DUP(0000h)
htriline word 0FFFFh, 3 DUP(0000h), 0FFFFh, 3 DUP(0000h), 0FFFFh
grid word 0F3CFh, 0F3CFh, 0000h, 0F3CFh, 0F3CFh, 0F3CFh, 0000h, 0F3CFh,
plus word 4 DUP(0180h), 0FFFFh, 4 DUP(0180h)
dtriangle word 4 DUP(0000h), 0180h, 07E0h, 1FF8h, 7FFEh, 0FFFFh
utriangle word 0FFFFh, 7FFEh, 1FF8h, 07E0h, 0180h, 4 DUP(0000h)
iks word 0000h, 6006h, 381Ch, 0E70h, 03C0h, 0E70h, 381Ch, 6006h, 0000h


guessed word P DUP(0);//arrray of matched cards
turned word P DUP(0);//array of turned cards - guessed + guessing
guess1 word 0;//first card to uncover
guess2 word 0;//second card to uncover
guess1ind word 0;//linear index of guess1
guess2ind word 0;//linear index of guess2
guess1coord_x dword 0;//coordinates of guesses
guess1coord_y dword 0
guess2coord_x dword 0
guess2coord_y dword 0

;//Score global variables
score sdword 0
scoreString byte "SCORE:", 0
scorePadding byte "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣", 0
```

13: projekat11.asm

```
.code
main PROC
;// -------------- Intro ------------------
INVOKE SetConsoleTitle, ADDR titleStr;// Set title

mov  edx, offset greeting;// Greeting message
call WriteString


;//BEGIN link shapeArray with shapes in memory
mov shapeArray[0], offset square
mov shapeArray[4], offset line
mov shapeArray[8], offset triline
mov shapeArray[12], offset hline
mov shapeArray[16], offset htriline
mov shapeArray[20], offset grid
mov shapeArray[24], offset plus
mov shapeArray[28], offset dtriangle
mov shapeArray[32], offset utriangle
mov shapeArray[36], offset iks
;//END link shapeArray with shapes in memory


;//FOR DEBUG PURPOSES ONLY
;//BEGIN TESTING CODE
;;mov eax, shapeArray[24]
;;mov ecx, 8
;;abc:
;;mov bx, [eax]
;;add eax, 2
;;loop abc
;//END TESTING CODE
;//FOR DEBUG PURPOSES ONLY

call generateRandom

call WaitMsg
call Clrscr
```

14: projekat11.asm

```
;// PROGRAM STARTS HERE
;// ------------------------------------------------------------

;// ----------------- Hides the cursor ----------------------
.data
stdInHandle HANDLE ?
cursorInfo CONSOLE_CURSOR_INFO <>

.code
INVOKE GetStdHandle, STD_OUTPUT_HANDLE
mov  stdInHandle, eax

INVOKE GetConsoleCursorInfo, stdInHandle, ADDR cursorInfo
mov  cursorInfo.bVisible, 0
INVOKE SetConsoleCursorInfo, stdInHandle, ADDR cursorInfo
;// ------------------------------------------------------------

;// -------------------- Draw blocks -------------------------
.data
y0_coord BYTE Y_PADDING
x0_coord BYTE X_PADDING
x1_coord BYTE X_SIZE
y1_coord BYTE Y_SIZE


.code
;// ----------------- Background color ----------------------------
xor eax, eax
mov al, 16 * (8);//postavljamo boju kvadrata koji treba iscrtati
call SetTextColor
call Clrscr

;// -----------------  Drawing ---------------------------
;//BEGIN Initial drawing of screen
mov ebx, 0
mov ecx, N

drawRow:
inc ebx
mov eax, 0
mov ecx, N
drawCol:
inc eax
push 0;//mode - 0 means gray square
push eax;//column
push ebx;//row
call refreshSquare
loop drawCol
cmp ebx, M
jnz drawRow
;//END Initial drawing of screen
```

15: projekat11.asm

```
;// ------------------  Game loop and logic ----------------------------
;//MAIN GAME LOOP SECTION OF THE CODE
.data
nRead dd 0
InputRecord _INPUT_RECORD <>
ConsoleMode dd 0

Msg db "␣␣", 0
Msg2 db "Esc␣", 0
MsgX db "X:", 0
MsgY db "Y:", 0

arow dword 1;//active row
acol dword 1;//active column
amode dword 1;//active mode

prow dword 1;//previous row
pcol dword 1;//previous column
pmode dword 0;//previous mode

refreshFlag byte 0;//flag indicating arrow button press
guessCounter byte 0

left_code EQU 37
up_code EQU 38
right_code EQU 39
down_code EQU 40
enter_code EQU 0Dh

.code
inc refreshFlag

forever :
invoke GetStdHandle, STD_INPUT_HANDLE;// Get handle to standard input
mov stdInHandle, eax

invoke GetConsoleMode, stdInHandle, ADDR ConsoleMode
mov eax, 0090h;// ENABLE_MOUSE_INPUT | DISABLE_QUICK_EDIT_MODE | ENABLE
invoke SetConsoleMode, stdInHandle, eax
```

16: projekat11.asm

```
;//Sleep, to allow OS to time slice and check for keyboard input
mov  eax, 50
call Delay
call ReadKey

;//BEGIN Checking for arrow keys pressed
;//check if left arrow was pressed
cmp dx, left_code
jne test_up
mov amode, 1
inc refreshFlag
dec acol
cmp acol, 0
jne test_up
mov acol, N

;//check if up arrow was pressed
test_up:
cmp dx, up_code
jne test_right
mov amode, 1
inc refreshFlag
dec arow
cmp arow, 0
jne test_right
mov arow, M

;//check if right arrow was pressed
test_right:
cmp dx, right_code
jne test_down
mov amode, 1
inc refreshFlag
inc acol
cmp acol, N+1
jne test_down
mov acol, 1
```

17: projekat11.asm

```
;//check if down arrow was pressed
test_down:
cmp dx, down_code
jne testEnter
mov amode, 1
inc refreshFlag
inc arow
cmp arow, M + 1
jne testEnter
mov arow, 1

;//check if enter was pressed
testEnter:
cmp dx, enter_code
jne keyPressed

;//Pressing enter uncovers the active card
mov amode, 3
push amode
push acol
push arow
call refreshSquare

;//Calculating linear index
push N
push acol
push arow
call linearIndex

;//First or second guess?
cmp guessCounter, 0
jne secondGuess
mov guess1ind, ax
```

18: projekat11.asm

```
mov edi, offset guessed
mov ax, guess1ind
mov bx, word ptr[edi + eax * 2]
cmp bx, 1
;//If first guess already uncovered -
;//dont allow first guess - skip to next keypress
je keyPressed

inc guessCounter

mov ebx, arow
mov guess1coord_y, ebx
mov ebx, acol
mov guess1coord_x, ebx
mov ebx, object_array[eax * 4]
dec ebx
mov guess1, bx

;//BEGIN Marking turned card
mov edi, offset turned
mov ax, guess1ind
mov word ptr[edi + eax * 2], 1
;//END Marking turned card

jmp keyPressed

secondGuess:
cmp guessCounter, 1
jne keyPressed
mov guess2ind, ax

mov edi, offset guessed
mov ax, guess2ind
mov bx, word ptr[edi + eax * 2]
cmp bx, 1
;//if already uncovered - dont accept second guess
je keyPressed
;//if second guess same as first
;// guess - dont accept as second guess
cmp ax, guess1ind
je keyPressed
```

19: projekat11.asm

```
inc guessCounter

mov ebx, arow
mov guess2coord_y, ebx
mov ebx, acol
mov guess2coord_x, ebx
mov ebx, object_array[eax*4]
dec ebx
mov guess2, bx

;//BEGIN Marking turned card
mov edi, offset turned
mov ax, guess1ind
mov word ptr[edi + eax * 2], 1
;//END Marking turned card

mov guessCounter, 0

mov ax, guess1
mov bx, guess2
cmp ax, bx
jne resetGuess
;//if match
mov edi, offset guessed
mov ax, guess1ind
mov word ptr[edi + eax * 2], 1
mov ax, guess2ind
mov word ptr[edi + eax * 2], 1

mov edi, offset turned
mov ax, guess1ind
mov word ptr[edi + eax * 2], 1
mov ax, guess2ind
mov word ptr[edi + eax * 2], 1
```

20: projekat11.asm

```
;//Cards matched - add 20 to score
add score, POINTS


;//FOR DEBUG PURPOSES ONLY
;//BEGIN TESTING ARRAY
;;xor ebx, ebx
;;mov ecx, P
;;
;;def:
;;mov ax, guessed[ebx * 2]
;;inc ebx
;;loop def
;//END TESTING ARRAY

;//Jump to avoid resetGuess
jmp keyPressed


;//A pair of guesses was made and blocks did not match
;//Reseting unmatched blocks into hidden state
resetGuess:
;//Wrong guess, subtract 5 from score
sub score, POINTS/4
mov edi, offset turned
mov ax, guess1ind
mov word ptr[edi + eax * 2], 0
mov ax, guess2ind
mov word ptr[edi + eax * 2], 0
;//pause before reseting blocks so player has time to remember
mov  eax, 800
call Delay
mov amode, 0
mov pmode, 1

;//hide active block
push amode
push guess1coord_x
push guess1coord_y
call refreshSquare

;//hide passive block
push pmode
push guess2coord_x
push guess2coord_y
call refreshSquare
```

21: projekat11.asm

```
keyPressed:
;//Procedure to write current score on screen
call updateScore

;//Check if game over - every card matched
mov ecx, P
mov eax, 0
mov edi, offset guessed
scoreCheck:
mov bx, word ptr[edi + eax * 2]
cmp bx, 0
je gameNotOver
inc eax
loop scoreCheck
call gameOver
gameNotOver:

;//Check if arrow keys pressed - change active and passive block
;//active block is the currectly highlighted block
;//passive block means that the block
;// was previously highlighted, but now needs
;//to be redrawn as inactive or non-highlited
cmp refreshFlag, 0
je forever;//if no arrow key pressed - go back to start
dec refreshFlag

;//Calculating linear index
push N
push pcol
push prow
call linearIndex
```

22: projekat11.asm

```
;//if passive block was turned - set pmode to 2 -
;//look at refreshSquare procedure documentation
;// for more info
mov pmode, 0
mov bx, turned[eax*2]
cmp bx, 0
je notTurned
mov pmode, 2
notTurned:
;//calling refreshSquare procedure
push pmode
push pcol
push prow
call refreshSquare

;//Calculating linear index
push N
push acol
push arow
call linearIndex


;//if active block is turned - set amode to 3 -
;// look at refreshSquare procedure documentation
;//for more info
mov amode, 1
mov bx, turned[eax * 2]
cmp bx, 0
je notTurned2
mov amode, 3
notTurned2:
;//calling refreshSquare procedure
push amode
push acol
push arow
call refreshSquare
```

23: projekat11.asm

```
;//updating passive block position to current active
;//block position for next iteration of the loop
mov eax, acol
mov pcol, eax
mov eax, arow
mov prow, eax

;//uncoditional jump to begining of game_loop
jmp forever
exit

main ENDP

end main

;//END main PROCEDURE
;//-------------------------------------------------------------

;//---------------------------------------------------------------
;//BRIEF:
;//This procedure checks if esc is pressed
;//It has 2 modes of operation
;//If arg2 is 0 then it doesn't poll for keys, it just checks dx
;//If arg 2 is 1 then it polls for key and checks if it is esc
;//BEGIN escCheck PROCEDURE
.code
escCheck proc uses eax ebx edx, arg1:dword, arg2:dword
cmp arg2, 1
jne noRead
mov  eax, 50
call Delay
call ReadKey
cmp dx, 1Bh
jne noEsc
exit
noRead:
mov edx, arg1
cmp dx, 1Bh
jne noEsc
exit
noEsc:
ret
escCheck endp
;//END escCheck PROCEDURE
;//---------------------------------------------------------------
```