Univerzitet u Beogradu
Elektrotehnički fakultet
Katedra za elektroniku

Računarska elektronika

Projekat - 3D lavirint

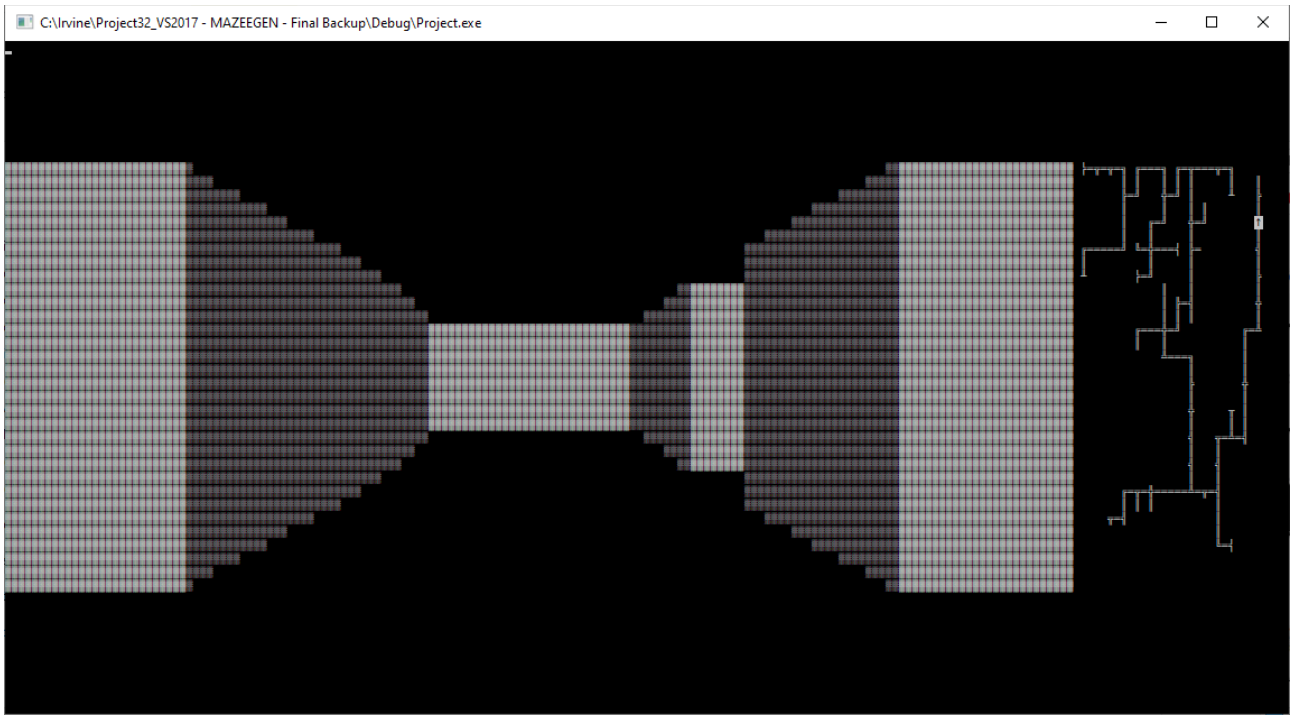
Mentor:
Turkmanović Haris

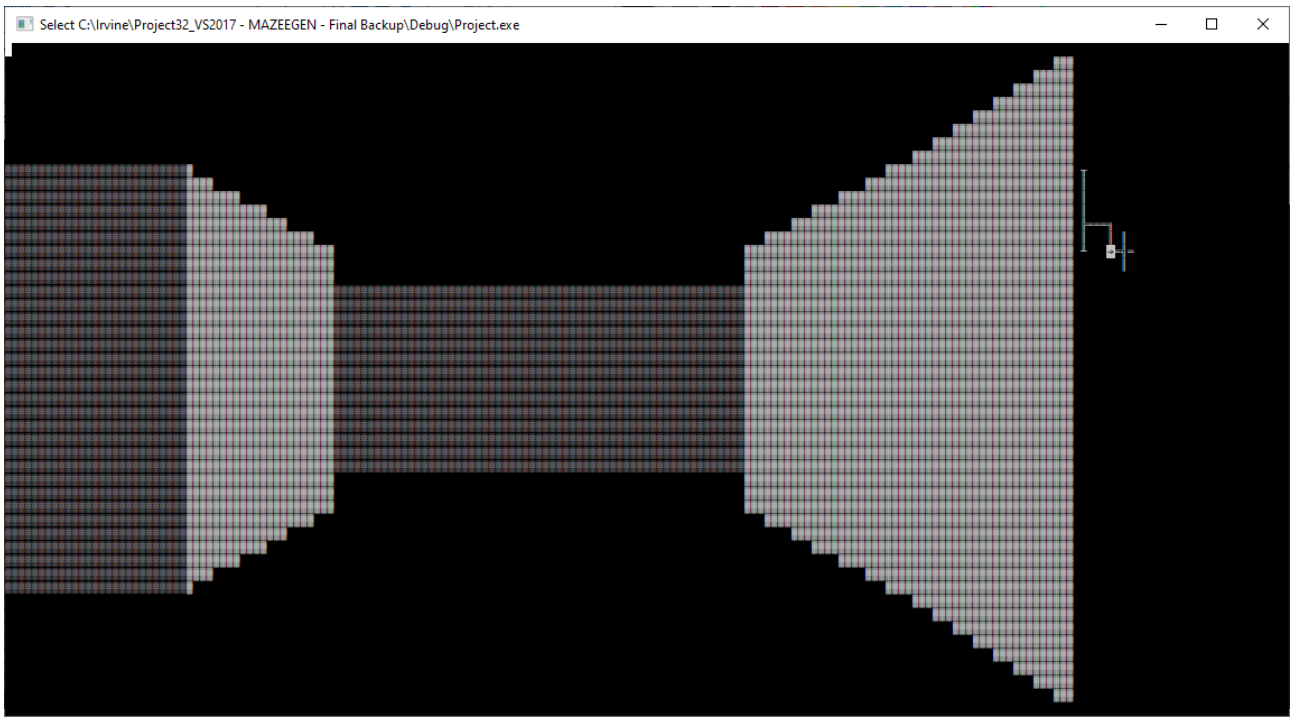
Studenti:
Janicijevic Lazar 2013/0088
Živković Radovan 2013/0629

Zadatak:

Cilj projekta je bio da se u MASM jeziku napiše igrica 3D lavirint.
U ovoj igri korisnik rešava lavirint i ima 3D prikaz iz "First Person" perspektive. Može se kretati po lavirintu pomoću tastera strelica na tastaturi.

Rezultat:





Princip rada:

Inicijalizacija:

Pokreće se procedura koja postavlja veličinu konzole, da bi se mogao nacrtati ceo ekran.
Očisti se bafer ekrana a zatim se generiše i iscrta inicijalni 3D prikaz.

Glavna petlja:

Glavni program periodično proverava za ulaz sa tastature.

-Ako se detektuje strelica unapred: Proveri se da li je nova pozicija slobodna, i ako jeste, pozicija se promeni. Takođe se u bafer ekrana u mapu lavirinta upiše karakter koji odgovara novoj poziciji u lavirintu. Ovi karakteri izgarđuju mapu sa strane lavirinta kako ga korisnik rešava.
-Ako se detektuju stelice u levo ili desno: Pokreće se rotacija vektora smera.

U slucaju da je došlo do promene pozicije ili smera, pokreće se osvežavanje ekrana.
Pokreće se procedura koja proverava zidove u okolini igrača i u slučaju postojanja istih pokreće funkciju za crtanje 3d prikaza zida. Nakon što se ovo završi, bafer ekrana se ispiše na izlaz.
Skoči se na početak petlje.

Kretanje kroz lavirint:

Lavirint se u memoriji čuva kao matrica *BYTE*. Zid je definisan konstantom *FILE_WALL_CHAR* koje je u kodu postavljena na "#". Pozicija lavirinta se čuva u podatku *position* veličine *WORD.* Gornji bajt ovog podatka prestavlja Y a donji X poziciju u lavirintu. Ovaj sistem skladištenja vektora se koristi i u ostatku programa. Smer kretanja se čuva o podatku *direction* veličine *WORD*, gde se čuva jedinični vektor pomeraja. Rotacija ovog vektora se može vrtiti yamenom višeg i nižeg *BYTE* i negiranjem jednog od njih. Izbor *BYTE* za negiranje određuje smer roatcije.

xchg lowByte,highByte *neg lowByte*		*xchg lowByte,highByte* *neg highByte*	
highByte	lowByte	highByte	lowByte
0	1	0	1
1	0	-1	0
0	-1	0	-1
-1	0	1	0

Kretanje unapred se realizuje dodavanjem smera na poziciju, *BYTE* po *BYTE*.
Pristup lavirintu u memoriji se vrši pomoću procedura:

maze_write	Pozicija se prosleđuje preko *DX*, karakter za upis preko *AL*
maze_read	Pozicija se prosleđuje preko *DX*, karakter za čitanje vraća preko *AL*
wall_there	Pozicija se prosleđuje preko *DX*, postojanje zida vraća preko EQ flag
wall_there_rel_to_pos	Pozicija relativno u odnosu na trenutnu poziciju u trenutnom smeru se prosleđuje preko *DX*, a postojanje zida vraća preko EQ flag

Procedure *wall_there* i *wall_there_rel_to_pos* imaju provere opsega ulaznog podatka, i u slučaju podatka van opsega vraćaju EQ flag tj prijavljuju postojanje zida.

Generisanje lavirinta:

Lavirint koji se koristi u igrici se automatski generiše po svakom pokretanju. Ovo se pokreće preko procedure *generate_maze*. Uvek se generiše lavirint jednakih stranica, a dimenzija lavirinta se zadaje konstantom *MAZE_HEIGHT*. Neophodno je da ova konstanta veličina bude neparna zbog pravilnog rada algoritma za generaciju, u kodu je 31.
Za generaciju lavirinta korišćen je sledeći algoritam

1) Matrica lavirinta se inicijalizuje nulama

maze_initialize

2) Postave se zidovi tamo gde će se sigurno pojaviti

maze_initialize

3) Postave se slobodni prostori tamo gde će se sigurno pojaviti

maze_initialize

4) Lavirint se uokviri zidovima sa svih strana

maze_initialize

5) Pokupe se koordinate svih pozicija u lavirintu koje mogu postati zidovi i ubace u niz(*wildcards*), sačuva se broj elemenata u podatku *no_wildcards*
Ovo su pozicije u lavirintu u kojima je ostala upisana 0

get_wildcards

6) Pokupi se jedna koordinata slučajnim odabirom iz niza, a na njeno mesto se premešta poslednja koordinata iz niza, smanjuje se broj elemenata

test_candidate

7) Na pokupljenu koordinatu se postavi zid, ova lokacija u lavirintu sigurno ima 2 susedna zida i 2 susedna prazna prostora, pošto su oni fiksirani od početka

test_candidate

8) Pokreće se traverzija lavirinta rešavanjem pomoću pravila desne ruke na zidu. Cilj je da se dođe od jedne strane novog zida do druge.

find_pair

9) Ukoliko ne postoji put između te dve poziije, zid se izbacuje iz lavirinta

test_candidate

10)Ako niz pozicija nije prazan skoči se na korak 6

Ovaj algoritam garantuje da ne postoji nijedna petlja u lavirintu i da su bilo koje dve tacke povezane. Pre nego što je implementirana auto generacija, lavirint se učitavao iz tekst fajla, ovo se i dalje moze pokrenuti pozivanjem procedure *read_maze_from_file*, fajl se učitava sa lokacije zadate u *maze_file_name* koja je u programu postavljena na 'maze.txt'.

Crtanje 3D zidova:

Za crtanje zida koristi se procedura *draw_wall(x_rel:BYTE,y_rel:BYTE)*.
Za zid na određenoj daljini od trenutne pozicije(*y_rel*) određuju se visina i širina prednje i zadnje strane zida na ekranu po formulama:

	Visina	Širina
Bliža strana	$y\_scale = \frac{HEIGHT}{0.5 + y\_rel}$	$x\_scale = \frac{WIDTH}{0.5 + y\_rel}$
Dalja strana	$y\_scale1 = \frac{HEIGHT}{1.5 + y\_rel}$	$x\_scale1 = \frac{WIDTH}{1.5 + y\_rel}$

WIDTH je širina 3D prikaza i u kodu se zove *SCR_3D_WIDTH*
HEIGHT je širina 3D prikaza i u kodu se zove *SCR_HEIGHT*
y_rel je daljina trenutne pozicije od zida duž smera kretanja.
x_rel je odstupanje zida od centralne linije duž smera kretanja.

Kada se crta prednja strana zida potrebno nacrtati pravougaonik širine *x_scale* i visine *y_scale* koji je horizontalno pomeren od centra za *x_rel* * *x_scale.*
Ako zid nije u centru(*x_rel* ≠ 0) potrebno je takođe nacrtati zidove sa strane.
Primer dat na zidu desno(*x_rel* > 0)

1) Pronađe se početna x kordinata *x1* = *x_rel* * *x_scale1*
2) Pronađe se krajnja koordinata *x2* = *x_rel* * *x_scale*
3) Pronađe se razlika te dve koordinate *xdif* = *x2* - *x1*
4) Pronađe se razlika pozicija gornjih ivica lavirinta *ydif* = (*y_scale* - *y_scale1*) / 2
6) Pronađu se početna i krajnja y pozicija *y1* = -*y_scale* / 2, *y2* = *y_scale* / 2
5) Odradi se korekcija y koordinata na osnovu pređenih kolona i oduzima/dodaje na *y1*, *y2*
   correction = *ydif* / *xdif* * x_count
6)Prođe se od *y1* do *y2* i upiše karakter zida u bafer ekrana na poyiciji *x1*, y_iterator
7) Inkrementira se *x1* i x_count
8) Ako je *x1* <= *x2* skače se na korak 5

Za crtanje svih relevantnih zidova pokreće se funkcija *build_3d_view* koja proverava sve zidove do daljine *DRAW_DEPTH* i širine *DRAW_WIDTH*.
Ako se paralelni i normalni zidovi crtaju razlicitim karakterima, takođe je bitno u kom redosledu se crtaju na ekranu. Očigledno je da dalji zidovi trebaju prvi da se crtaju, ali takođe treba paziti na raspored zidova sa strane jer moze doći do preklapanja i na istoj daljini, pravilno je kretanje *y_rel* od *DRAW_DEPTH* do 0, a *x_rel* od ± *DRAW_WIDTH*(3) do 0. Pokuša se crtanje i na trenutnoj poziciji, ali tu nikada ne postoji zid.

Irvine biblioteka i brz ispis ekrana:

Pre ispisa na izlaz, stanje ekrana se čuva u privremenom baferu ekrana *screen_buffer* koji je ralizovan kao matrica *BYTE*. Ovde se upisuju karakteri pre ispisa na ekran da bi se ubrzao ispis. Ispis ne ekran se vrši procedurama *Irvine* biblioteke. Kada se ispis vrši karakter po karakter vreme osvežavanja ekrana je veoma dugačko. Rešenje za ovo je da se redovi matrice *screen_buffer* terminiraju line break karakterima CR i LF i da se poslednji element postavi na 0. Ovo omogućava gotovo trenutno osvežavanje ekrana štampanjem celog bafera kao string jednim pozivom *WriteString*.
Bafer ekrana je podeljen na levi deo za 3d prikaz i desni deo za crtanje mape istraženih delova lavirinta. Pri generisanju 3d prikaza briše se samo taj deo ekrana, da bi mapa ostala neizmenjena. Na kraju ispisa ekrana se docrtava još jedan karakter koji prikazuje poziciju u lavirintu i smer kretanja, ovo se vrši procedurom *draw_arrow.*
Za pristup baferu ekrana koriste se funkcije

clear_buffer	Upisuje 20h u ceo bafer i postavlja line break karaktere
clear_3d	Upisuje 20h u deo bafera za 3d prikaz
write_to_buffer	Upisuje karakter iz AL na poziciju DX sa proverom opsega
write_to_3d(X:WORD,Y:WORD,char:WORD)	Upisuje karakter char na poziciju XY u deo bafera za 3d prikaz sa proverom opsega
print_screen	Ispisuje se ceo bafer na izlaz

Od procedure iz biblioteke Irvine korišćene su funkcije

ReadFile	Za učitavanje lavirinta iz tekst fajla
RandomRange	U algoritmu generisanja algoritma
Clrscr	Za inicijalno osveženje ekrana
Gotoxy	Za resetovanje mesta ispisa bafera i ispis strelice
MsgBox	Za prijave grešaka
WriteChar	Za ispis strelice
WriteString	Za ispis bafera na ekran

WinAPI:

Kako nam je rezolucija crtanja ograničena veličinom karaktera i ekrana, poželjno je da ekran bude što veći. Ako se pokuša ispis bafera koji je veći od ekrana, dolazi do besmislene slike jer svaki red ima više od jednog line break-a. Rešenje za ovo je da se veličina ekrana po uključenju postavi na dovoljno veliku preko WinAPI procedura. Procedure koje su korišćene u projektu su:

GetStdHandle	Procedura koja vraća hendl otvorenog otvorenog prozora
SetConsoleScreenBufferSize	Procedurea koja postavlja veličinu bafera ekrana prozora
SetConsoleWindowInfo	Procedura koja postavlja veličinu prozora

Ove procedure se pozivanju u funkciji *setup_console*.

KOD:

```
INCLUDE Irvine32.inc

.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExitCode:dword


SCR_WIDTH = 190; Target width of the screen
SCR_HEIGHT = 50; Target height of the screen
MAZE_HEIGHT = 31; Dimension of maze
FILE_WALL_CHAR = '#'; Character that is recognized as wall in maze file
FILE_SPACE_CHAR = ' ';Not critical, used for internal maze generation
; Maze file

POLLING_PERIOD = 20; Wait time before keyboard event check[ms]
CHR_FOG = 176; Character for potential walls out of view
CHR_NS_WALL = 177; Character that is drawn for north and south walls
CHR_EW_WALL = 178; Character that is drawn for east and west walls
COL_MAZE = lightGray + (black * 16); Color of maze map and 3d view, lower nibble
is foreground, higher nibble is background
COL_ARROW = black + (lightGray * 16); Color of maze map and 3d view, lower
nibble is foreground, higher nibble is background

SCR_3D_WIDTH = SCR_WIDTH - MAZE_HEIGHT ; Efective width of 3d view
MAZE_Y_OFFSET = (SCR_HEIGHT - MAZE_HEIGHT) / 2 - 1; Parameter for centering maze
DRAW_DEPTH = 6; How many walls are drawn from current position + current
position
DRAW_WIDTH = 3; How far to each side are walls drawn
MAZE_PATH EQU "maze.txt"

.DATA
screen_buffer BYTE SCR_HEIGHT*(SCR_WIDTH + 2) - 2 dup(0); Screen buffer for 3d
view and maze map
maze BYTE MAZE_HEIGHT*(MAZE_HEIGHT + 2) + 1 dup(0); Location for storing maze in
memory from file

.DATA
maze_file_name BYTE MAZE_PATH, 0
maze_file_error_msg BYTE "Encountered error while loading file ", MAZE_PATH, 0

.DATA
direction WORD 0001h; Direction of movement stored as a vector, lower byte is
dx, higher byte is dy
position WORD 0101h; Current position, lower byte is x, higher byte is y
char_facing_wall WORD CHR_NS_WALL
char_side_wall WORD CHR_EW_WALL

;Used for maze generation
.data
wildcards WORD MAZE_HEIGHT*(MAZE_HEIGHT+1)/2    dup(0)
no_wildcard WORD 0

.data
out_file_name BYTE "maze_out.txt",0
out_file_msg BYTE "Maze saved to file",0

.code
maze_print PROC uses edx eax
xor edx,edx
call gotoxy
```

```
print_loop:
call maze_read
call writeChar
inc dl
cmp dl,MAZE_HEIGHT
jb print_loop
xor dl,dl
inc dh
mov al,0Dh
call writeChar
mov al,0Ah
cmp dh,MAZE_HEIGHT
call writeChar
jb print_loop
ret
maze_print ENDP

maze_write PROC uses edx; pos passed throug dl, dh, char returned through al
push edx
push eax
mov ax, MAZE_HEIGHT + 2
mul dh
xor dh, dh
add ax, dx
movzx edx, ax
pop eax
mov BYTE PTR maze[edx], al
pop edx
ret
maze_write ENDP

maze_initialize PROC uses eax edx
;Sets all to zeros
xor edx,edx
xor eax,eax
blank_loop_y :
xor dl, dl
blank_loop_x :
call maze_write
inc dl
cmp dl, MAZE_HEIGHT
jb blank_loop_x
inc dh
cmp dh, MAZE_HEIGHT
jb blank_loop_y
;Sets permanent walls
xor dx,dx
mov al, FILE_WALL_CHAR
wall_loop_y :
xor dl, dl
wall_loop_x :
call maze_write
add dl, 2
cmp dl, MAZE_HEIGHT
jb wall_loop_x
add dh, 2
cmp dh, MAZE_HEIGHT
jb wall_loop_y
;Sets permanent spaces
mov dx,0101h
mov al, FILE_SPACE_CHAR
space_loop_y :
mov dl,01h
space_loop_x :
```

```
call maze_write
add dl, 2
cmp dl, MAZE_HEIGHT
jb space_loop_x
add dh, 2
cmp dh, MAZE_HEIGHT
jb space_loop_y
xor dx,dx
;Sets side and top walls
mov al,FILE_WALL_CHAR
loop_init:
xor dh,dh
call maze_write
xchg dh,dl
call maze_write
mov dl,MAZE_HEIGHT-1
call maze_write
xchg dl,dh
call maze_write
inc dl
cmp dl,MAZE_HEIGHT
jb loop_init
ret
maze_initialize ENDP

load_wildcards PROC uses edx eax
xor edx,edx
xor edi,edi
wildcard_loop:
call maze_read
cmp ax, 0
jne @F
mov WORD PTR wildcards[edi], dx
add edi,2
@@:
inc dl
cmp dl,MAZE_HEIGHT
jb wildcard_loop
xor dl,dl
inc dh
cmp dh,MAZE_HEIGHT
jb wildcard_loop
shr edi, 1
ret
load_wildcards ENDP

find_pair PROC uses edx ecx eax poz1:word, poz2:word
local   dir:word
mov dx,poz1
mov cx,poz2
sub cl,dl
sar cl,1
sub ch,dh
sar ch,1                ;U cx je sada vektor koji pokazuje ka zidu
mov ax,cx
xchg ah,al
neg al          ;U ax je sada vektor koji pokazuje napred, duz zida, ovo je
okretanje na levo, neg ah je desno
mov WORD PTR dir,ax
jmp search_loop_bypass_check
search_loop:

push eax
mov al, '0'
```

```
call maze_write
mov al, 20h
call maze_write
pop eax

cmp dx,poz1
jne bypass_dir_check
cmp ax,dir
je done_searching
bypass_dir_check:
search_loop_bypass_check:
cmp dx,poz2
je done_searching
add dh,ch
add dl,cl
call wall_there ;Ovo je desni zid(pravilo desne strane)
je @F;ako ga nema okrecem se desno i koracam napred(vec uradjeno)
xchg ch,cl
neg ch
xchg al,ah
neg ah
jmp search_loop
@@:
sub dh,ch
sub dl,cl
add dh,ah
add dl,al
call wall_there ;Ovo je zid ispred
je      @F;Ako ga nema mozemo koracati napred
jmp search_loop
@@:;Ako ga ima okrecemo se u levo
sub dh,ah
sub dl,al
xchg ch,cl
neg cl
xchg al,ah
neg al
jmp search_loop
done_searching:
cmp dx, poz2
ret
find_pair ENDP

test_candidate PROC uses eax ecx edx
xor eax, eax
mov ax, no_wildcard
call RandomRange
shl eax,1
add eax, OFFSET wildcards
xor ecx, ecx
mov cx, no_wildcard
dec cx
mov no_wildcard, cx
shl cx,1
add ecx, OFFSET wildcards; OFFSET posledje wildcard
mov cx, WORD PTR[ecx]; Vredost poslednjeg wildcard
mov dx, WORD PTR[eax]; Pozicija sa kojom radimo     dx
mov WORD PTR[eax], cx; Poslednja zauzima mesto koje je ostalo
mov al,FILE_WALL_CHAR
call maze_write
test dl,01h
jnz @F
        mov ax,dx
        mov cx,dx
```

```
        inc al
        dec cl
jmp t_label
@@:
        mov ax,dx
        mov cx,dx
        inc ah
        dec ch
t_label:

INVOKE find_pair, ax, cx
je dont_remove_wall                     ;Equals means you're safe
mov al, FILE_SPACE_CHAR
call maze_write
dont_remove_wall:
ret
test_candidate ENDP


generate_maze PROC uses eax
call Randomize
call maze_initialize
call load_wildcards
mov eax, edi
mov WORD PTR no_wildcard, ax
generate_wall :
call test_candidate
cmp no_wildcard, 0
ja generate_wall
ret
generate_maze ENDP

setup_console PROC
LOCAL outHandle : DWORD, scrBuffSize : COORD ; SCREEN_BUFFER_INFO:
CONSOLE_SCREEN_BUFFER_INFO
LOCAL ConsoleRect : SMALL_RECT
INVOKE GetStdHandle, STD_OUTPUT_HANDLE
mov outHandle, eax
mov scrBuffSize.X, SCR_WIDTH + 1
mov scrBuffSize.Y, SCR_HEIGHT
mov ConsoleRect.Top, 0
mov ConsoleRect.Left, 0
mov ConsoleRect.Bottom, SCR_HEIGHT-1
mov ConsoleRect.Right,SCR_WIDTH
INVOKE SetConsoleScreenBufferSize, outHandle, scrBuffSize
INVOKE SetConsoleWindowInfo, outHandle, 1, ADDR ConsoleRect
ret
setup_console ENDP

draw_arrow PROC uses EDX EAX    ;Draws arrow on screen over maze map after
screen is updated, no args
;The maze map is not stored explicitly in memory, it is stored in the screen
buffer, so the arrow is writen
;seperately so that data is not lost
        mov eax,COL_ARROW
    call SetTextColor   ;Sets color used for the arrow
        ;Basically a case statement checking direction and putting the
appropriate arrow char in al
        mov dx,direction
        cmp dx,0001h
        jne @F
        mov al,1Ah      ;Right
        jmp finished_choosing_direction
@@:     cmp dx,00FFh
```

```
        jne @F
        mov al,1Bh      ;Left
        jmp finished_choosing_direction
@@:     cmp dx,0100h
        jne @F
        mov al,19h      ;Down
        jmp finished_choosing_direction
@@:     cmp dx,0FF00h
        jne finished_choosing_direction
        mov al,18h      ;Up
finished_choosing_direction:
        mov dx,position
        ;The edge of the 3d view, offseted by position in maze gives position in
3d map
        add dl,SCR_3D_WIDTH
        add dh,MAZE_Y_OFFSET
        call GotoXY
        call WriteChar
        xor dx,dx
        call GotoXY                     ;Goes to 0,0 to move cursor out of the
way
        mov eax,COL_MAZE
        call SetTextColor       ;Sets color for normal writing to screen
        ret
draw_arrow ENDP

maze_read PROC uses edx; pos passed throug dl, dh, char returned through al
mov ax, MAZE_HEIGHT +2
mul dh
xor dh, dh
add ax, dx
movzx edx, ax
mov al, BYTE PTR maze[edx]
xor ah, ah
ret
maze_read ENDP

print_screen PROC uses EDX      ;Writes out entire screen buffer to std out as
one string
;The matric the screen buffer is stored in ends rows with line breaks making it
easy to print to output
xor edx,edx
call gotoxy
mov edx, OFFSET screen_buffer
call WriteString
call draw_arrow
ret
print_screen ENDP

clear_buffer PROC uses ECX EDI EAX ;clears the entire screen buffer, and adds
line break characters to end of rows
mov ecx, LENGTH screen_buffer
mov edi, OFFSET screen_buffer
mov al, ' '
rep stosb               ;Writes ' ' into the entire screen_buffer
mov ecx, SCR_HEIGHT-1
mov ax, 0d0ah   ;Carriage return and line feed characters ready to be placed in
string
lea edi, [screen_buffer -2]
setup_line_breaks:
lea edi, [edi + SCR_WIDTH + 2]  ;Increments by 1 row length
mov BYTE PTR[edi], ah
mov BYTE PTR[edi + 1], al
loop setup_line_breaks
```

```
xor al, al
mov BYTE PTR[screen_buffer + LENGTH screen_buffer - 1], al
ret
clear_buffer ENDP

print_maze_to_file PROC uses ECX EDI EAX EBX; clears the entire screen buffer,
and adds line break characters to end of rows
mov ecx, MAZE_HEIGHT - 1
mov ax, 0d0ah; Carriage return and line feed characters ready to be placed in
string
lea edi, [maze - 2]
setup_line_breaks_maze:
lea edi, [edi + MAZE_HEIGHT + 2]; Increments by 1 row length
mov BYTE PTR[edi], ah
mov BYTE PTR[edi + 1], al
loop setup_line_breaks_maze
xor al, al
mov BYTE PTR[screen_buffer + LENGTH maze - 1], al
mov edx,OFFSET out_file_name
call createOutputFile
mov ecx,MAZE_HEIGHT*(MAZE_HEIGHT+2)
mov edx,OFFSET maze
push eax
call WriteToFile
pop eax
call CloseFile
mov edx,OFFSET out_file_msg
xor ebx,ebx
call MsgBox

ret
print_maze_to_file ENDP

clear_3d PROC uses ECX EAX      ;Clears only 3d view portion of screen buffer,
also adds "FOG" character
xor edx,edx
clear_loop_x:
xor dh,dh
clear_loop_y:
mov al,' '
cmp dh,SCR_HEIGHT/2+SCR_HEIGHT/(DRAW_DEPTH*2+3)-1
jge dont_draw_fog
cmp dh,SCR_HEIGHT/2-SCR_HEIGHT/(DRAW_DEPTH*2+3)
jle dont_draw_fog
mov al,CHR_FOG
dont_draw_fog:
call write_to_buffer    ;Procedure for writing to screen buffer dl -x dh -y al
-char
inc dh
cmp dh,SCR_HEIGHT
jb clear_loop_y
inc dl
cmp dl,SCR_3D_WIDTH
jb clear_loop_x
ret
clear_3d ENDP

.code
write_to_buffer PROC uses EDX ;Procedure for writing to screen buffer dl-x dh-y
al-char, accepts invalid adresses but does not write
cmp dl,SCR_WIDTH
jae skip_write_buf      ;Check for address out of range
cmp dh,SCR_HEIGHT
jae skip_write_buf      ;Check for address out of range
```

```
push eax                        ;EAX is needed for multiplication
mov eax, SCR_WIDTH+2
mul dh
xor dh,dh
add ax,dx
pop edx                         ;eax is exchanged for edx
xchg eax,edx
cmp edx,LENGTH screen_buffer-2
;e skip_write_buf       ;Ignore also the last character in screen buffer, it is
used for NULL
mov BYTE PTR screen_buffer[edx],al
skip_write_buf:
ret
write_to_buffer ENDP

write_to_3d PROC uses EDX EAX X:WORD, Y:WORD, char:WORD ;Writes char to
screen_buffer only in 3d view zone
;put char to WORD for compatibility, INVOKE had problems passing BYTE, procedure
uses lower byte
cmp X,SCR_3D_WIDTH      ;Check addressress out of range
jae skip_write_3d
cmp Y,SCR_HEIGHT        ;Check address out of range
jae skip_write_3d
mov dl,BYTE PTR X
mov dh,BYTE PTR Y
xor eax,eax
mov ax,char
call write_to_buffer
skip_write_3d:
ret
write_to_3d ENDP

wall_there PROC uses EDX EAX    ;Checks if there is a wall in maze position dl-x
dh-y, returns via equals flag
cmp dl,MAZE_HEIGHT      ;Check address out of range
jae skip_read_wall
cmp dh,MAZE_HEIGHT      ;Check address out of range
jae skip_read_wall
call maze_read
cmp al,FILE_WALL_CHAR
ret
skip_read_wall:
cmp eax,eax             ;sets flag on (wall on) in case of outside range
ret
wall_there ENDP

wall_there_rel_to_pos PROC uses ECX EDX ;Checks if there is wall relative to
current position, in the current direction, returns via equals flag
;dl - x_rel  dh - y_rel
mov cx, position
xchg cx,dx
cmp direction,0001h
jne @F
add dl,ch
add dh,cl
jmp spremio_poziciju
@@:
cmp direction,0100h
jne @F
sub dl,cl
add dh,ch
jmp spremio_poziciju
@@:
cmp direction,00FFh
```

```
jne @F
sub dl,ch
sub dh,cl
jmp spremio_poziciju
@@:
add dl,cl
sub dh,ch
spremio_poziciju:
call wall_there
ret
wall_there_rel_to_pos ENDP

read_maze_from_file PROC uses eax edx ecx  ;Reads maze in from memory to file
specified by maze_file_name
mov edx,OFFSET maze_file_name
call OpenInputFile
cmp eax,INVALID_HANDLE_VALUE
je error_reading_file
mov edx,OFFSET maze
mov ecx,LENGTH maze
push eax
call ReadFromFile
pop eax
jc error_reading_file
call CloseFile
ret
error_reading_file:
mov edx,OFFSET maze_file_error_msg
call MsgBox
ret
read_maze_from_file ENDP

get_maze_char PROC uses edx     ;Returns char used to represent passed position
in maze in map, dl-x dh-h , returns al-char
                xor eax,eax
                add dh,0FFh
                add dl,00h
                call wall_there
                je @F
                add al,01h
@@:             add dh,01h
                add dl,01h
                call wall_there
                je @F
                add al,02h
@@:             add dh,01h
                add dl,0FFh
                call wall_there
                je @F
                add al,04h
@@:             add dh,0FFh
                add dl,0FFh
                call wall_there
                je @F
                add al,08h
@@:
;Odredjivanje karaktera
                cmp al,00h
                jne @F
                mov al,20h
                jmp done_choosing_char
@@:             cmp al,01h
                jne @F
                mov al,0D0h
```

```
                jmp done_choosing_char
@@:             cmp al,02h
                jne @F
                mov al,0C6h
                jmp done_choosing_char
@@:             cmp al,03h
                jne @F
                mov al,0C8h
                jmp done_choosing_char
@@:             cmp al,04h
                jne @F
                mov al,0D2h
                jmp done_choosing_char
@@:             cmp al,05h
                jne @F
                mov al,0BAh
                jmp done_choosing_char
@@:             cmp al,06h
                jne @F
                mov al,0C9h
                jmp done_choosing_char
@@:             cmp al,07h
                jne @F
                mov al,0CCh
                jmp done_choosing_char
@@:             cmp al,08h
                jne @F
                mov al,0B5h
                jmp done_choosing_char
@@:             cmp al,09h
                jne @F
                mov al,0BCh
                jmp done_choosing_char
@@:             cmp al,0Ah
                jne @F
                mov al,0CDh
                jmp done_choosing_char
@@:             cmp al,0Bh
                jne @F
                mov al,0CAh
                jmp done_choosing_char
@@:             cmp al,0Ch
                jne @F
                mov al,0BBh
                jmp done_choosing_char
@@:             cmp al,0Dh
                jne @F
                mov al,0B9h
                jmp done_choosing_char
@@:             cmp al,0Eh
                jne @F
                mov al,0CBh
                jmp done_choosing_char
@@:             cmp al,0Fh
                jne done_choosing_char
                mov al,0CEh
                done_choosing_char:
                ret
get_maze_char ENDP

draw_wall PROC uses EAX EDX ECX x_rel:SBYTE, y_rel:SBYTE        ;For passed
relative position to player draws walls y-depth x- left to right
        LOCAL
x_scale:WORD,y_scale:WORD,x1_scale:WORD,y1_scale:WORD,xdif:WORD,ydif:WORD
```

```
        LOCAL   x1:WORD,x2:WORD,y1:WORD,y2:WORD
        movzx cx,y_rel
        shl cx,1
        inc cx
        mov ax,SCR_3D_WIDTH
        xor edx,edx
        div cx
        shl ax,1
        mov x_scale,ax          ;Generates width of front of wall
        mov ax,SCR_HEIGHT
        xor edx,edx
        div cx
        shl ax,1
        mov y_scale,ax          ;Generates height of front of wall
        add cx,2
        mov ax,SCR_3D_WIDTH
        xor edx,edx
        div cx
        shl ax,1
        mov x1_scale,ax         ;Generates width of back of wall
        mov ax,SCR_HEIGHT
        xor edx,edx
        div cx
        shl ax,1
        mov y1_scale,ax         ;Generates height of back of wall
        mov cx,y_scale
        sub cx,ax
        shr cx,1
        mov ydif,cx             ;Finds the distance between tops of back and
front wall
        .IF (x_rel > 0)         ;For blocks on the right which have a sidewall
on the left
                mov ax,x1_scale
                xor edx,edx
                movsx cx,x_rel
                imul cx
                mov dx,x1_scale
                shr dx,1
                sub ax,dx
                add ax, SCR_3D_WIDTH/2
                mov x1,ax       ;Leftmost position
                mov ax,x_scale
                xor edx,edx
                movsx cx,x_rel
                imul cx
                mov dx,x_scale
                shr dx,1
                sub ax, dx
                add ax, SCR_3D_WIDTH / 2
                mov x2,ax       ;Rightmost position
                sub ax,x1
                mov xdif,ax ;Difference between horizontal positions to
calculate slope of connecting line
                xor ecx,ecx
                inc cx          ;Itterator for x position (counter for slope
calculation)
                mov ax,x1       ;Itterator for x position (x coordinate)
                nop                     ;Complier ignores previous instruction
without nop
                x_loop_right:
                        nop             ;Compiler ignores next instruction
without nop
                        push ecx
                        push eax
```

```
                        xor eax,eax
                        mov ax,ydif
                        mul cx
                        xor dx,dx
                        div xdif        ;Correction for sloped line in ax, will
be added to y1_scale
                        mov dx,y1_scale
                        shr dx,1
                        add dx,SCR_HEIGHT/2
                        add dx,ax
                        push edx        ;Bottom y position
                        mov dx,y1_scale
                        shr dx,1
                        mov cx,SCR_HEIGHT/2
                        sub cx,ax
                        sub cx,dx       ;Top y position, also y itterator
                        pop edx         ;Retrieved bottom y position, needed for
cmp
                        pop eax         ;Retrieved itterator for x, needed for
write operation
                        y_loop_right:
                                INVOKE write_to_3d, ax, cx, char_side_wall
;Writes x and y itterator to 3d view of screen_buffer
                                inc cx
                                cmp cx,dx
                                jl y_loop_right
                        pop ecx
                        inc ax
                        inc cx
                        cmp ax,x2
                        jle x_loop_right
        .ELSEIF (x_rel < 0)             ;For blocks on the left which have a
sidewall on the right
                mov ax,x_scale
                xor edx,edx
                movsx cx,x_rel
                imul cx
                mov dx,x_scale
                shr dx,1
                add ax, dx
                add ax, SCR_3D_WIDTH / 2
                mov x2,ax ;Leftmost position
                mov ax,x1_scale
                xor edx,edx
                movsx cx,x_rel
                imul cx
                mov dx,x1_scale
                shr dx,1
                add ax,dx
                add ax, SCR_3D_WIDTH/2
                mov x1,ax       ;Rightmost position
                sub ax,x2
                mov xdif,ax     ;Difference between horizontal positions to
calculate slope of connecting line
                xor ecx,ecx
                inc cx          ;Itterator for x position (counter for slope
calculation)
                mov ax,x1   ;Itterator for x position (x coordinate)
                nop         ;Complier ignores previous instruction without nop
                x_loop_left:
                        nop     ;Compiler ignores next instruction without nop
                        push ecx
                        push eax
                        xor eax,eax
```

```
                        mov ax,ydif
                        mul cx
                        xor dx,dx
                        div xdif        ;Correction for sloped line in ax, will
be added to y1_scale
                        mov dx,y1_scale
                        shr dx,1
                        add dx,SCR_HEIGHT/2
                        add dx,ax
                        push edx        ;Bottom y position
                        mov dx,y1_scale
                        shr dx,1
                        mov cx,SCR_HEIGHT/2
                        sub cx,ax
                        sub cx,dx       ;Top y position, also y itterator
                        pop edx         ;Retrieved bottom y position, needed for
cmp
                        pop eax         ;Retrieved itterator for x, needed for
write operation
                        y_loop_left:
                                INVOKE write_to_3d, ax, cx, char_side_wall
;Writes x and y itterator to 3d view of screen_buffer
                                inc cx
                                cmp cx,dx
                                jl y_loop_left
                        pop ecx
                        dec ax
                        inc cx
                        cmp ax,x2
                        jge x_loop_left
        .ENDIF
        ;Drawing facing wall, acording to x_scale and y_scale
        xor eax,eax
        movsx ax,x_rel
        mul x_scale
        add ax, SCR_3D_WIDTH/2
        mov dx,x_scale
        shr dx,1
        add ax,dx
        mov x2,ax       ;Rightmost x position
        sub ax,x_scale
        mov x1,ax       ;Leftmost x position
        mov ax,y_scale
        shr ax,1
        add ax, SCR_HEIGHT/2
        mov y2,ax       ;Uppermost y position
        sub ax, y_scale
        mov y1,ax       ;Lowest y position
        mov ax,x1
        inc ax
        x_loop_center:
                mov cx,y1
                y_loop_center:
                        INVOKE write_to_3d, ax, cx, char_facing_wall    ;ax is x
itterator, cx is y itterator
                        inc cx
                        cmp cx,y2
                        jl y_loop_center
                inc ax
                cmp ax,x2
                jle x_loop_center
        ret
draw_wall ENDP
```

```
build_3d_view PROC uses edx     ;Procedure that checks neighboring walls and
calls draw_wall for relevant walls that are there
        call clear_3d           ;First clears the 3d view of screen_buffer
        mov dh, DRAW_DEPTH      ;Goes from farthest wall to closest to avoid
wrong character overlap y_rel = 4 3 2 1 0
        y_loop_build:           ;Alson has to toggle from positive to negative
x_rel to avoid overlap x_rel = -2 2 -1 1 0
        mov dl, -DRAW_WIDTH
        x_loop_build:
        call wall_there_rel_to_pos
        jne @F
        INVOKE draw_wall,dl, dh         ;dl and dh are used as x_rel and y_rel
itterators to draw a wall
@@:
        neg dl
        call wall_there_rel_to_pos
        jne @F
        INVOKE draw_wall,dl, dh         ;Done once more with a negated x_rel
@@:
        neg dl                                                  ;
        inc dl
        cmp dl,0
        jl x_loop_build
        call wall_there_rel_to_pos
        jne @F
        INVOKE draw_wall,dl, dh         ;At the end done once in the center
@@:
        dec dh
        cmp dh,00h
        jge y_loop_build
        ret
build_3d_view ENDP


;
;
;               MAIN
;
;
.code
main proc
.code

call setup_console

;Sets up text color and refreshes screen to apply
mov eax,COL_MAZE
call SetTextColor
call Clrscr
call generate_maze      ;Generates maze internally
call clear_buffer
call build_3d_view
call print_screen
main_loop:
        mov eax, POLLING_PERIOD ;Waits a little bit before checking for
characcter again
        call delay
        call ReadKey
        jz main_loop
        cmp dx,VK_LEFT
        je handle_left
        cmp dx,VK_RIGHT
        je handle_right
        cmp dx,VK_UP
```

```
        je handle_up
        cmp dx, VK_SPACE
        je handle_space
        cmp al, 'n'
        je handle_n
        cmp al, 'f'
        je handle_f
        cmp al, 'o'
        je handle_o
        jmp main_loop
        handle_left:    ;Rotates direction vector and changes characters the
walls are drawn with
        mov ax,char_facing_wall
        xchg char_side_wall,ax
        xchg char_facing_wall,ax
        mov ax, direction
        xchg al,ah
        neg ah
        mov direction, ax
        jmp update_display
        handle_right:   ;Rotates direction vector and changes characters the
walls are drawn with
        mov ax,char_facing_wall
        xchg char_side_wall,ax
        xchg char_facing_wall,ax
        mov ax, direction
        xchg al,ah
        neg al
        mov direction, ax
        jmp update_display
        handle_space:           ;Adds current position to maze map, adds
direction to current position and checks for collision, if none, updates
position
        mov dx, position
        mov ax,direction
        push edx
        add dl,al
        add dh,ah
        call wall_there
        mov al,FILE_SPACE_CHAR
        je @F
        mov al,FILE_WALL_CHAR
        @@:
        call maze_write
        pop edx
        jmp update_display
        handle_n:
        call generate_maze
        call clear_buffer
        jmp update_display
        handle_f:
        call read_maze_from_file
        call clear_buffer
        jmp update_display
        handle_o:
        call print_maze_to_file
        jmp update_display
        handle_up:              ;Adds current position to maze map, adds
direction to current position and checks for collision, if none, updates
position
        mov dx, position
        call get_maze_char
        add dl,SCR_3D_WIDTH
        add dh,MAZE_Y_OFFSET
```

```
        call write_to_buffer
        mov dx,position
        mov ax, direction
        add dl,al
        add dh,ah
        call wall_there
        je main_loop
        mov position, dx
        update_display:
        ;Screen updating after applied changes
        call build_3d_view
        call print_screen
        jmp main_loop
main endp
end main
```