

**UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET**

Katedra za elektroniku

Predmet: Racunarska elektronika



Projekat: Zmijica

Projekat radili:

Ime	Prezime	broj indeksa
Dragan	Bozinovic	211/2015
Kristijan	Mitrovic	214/2015

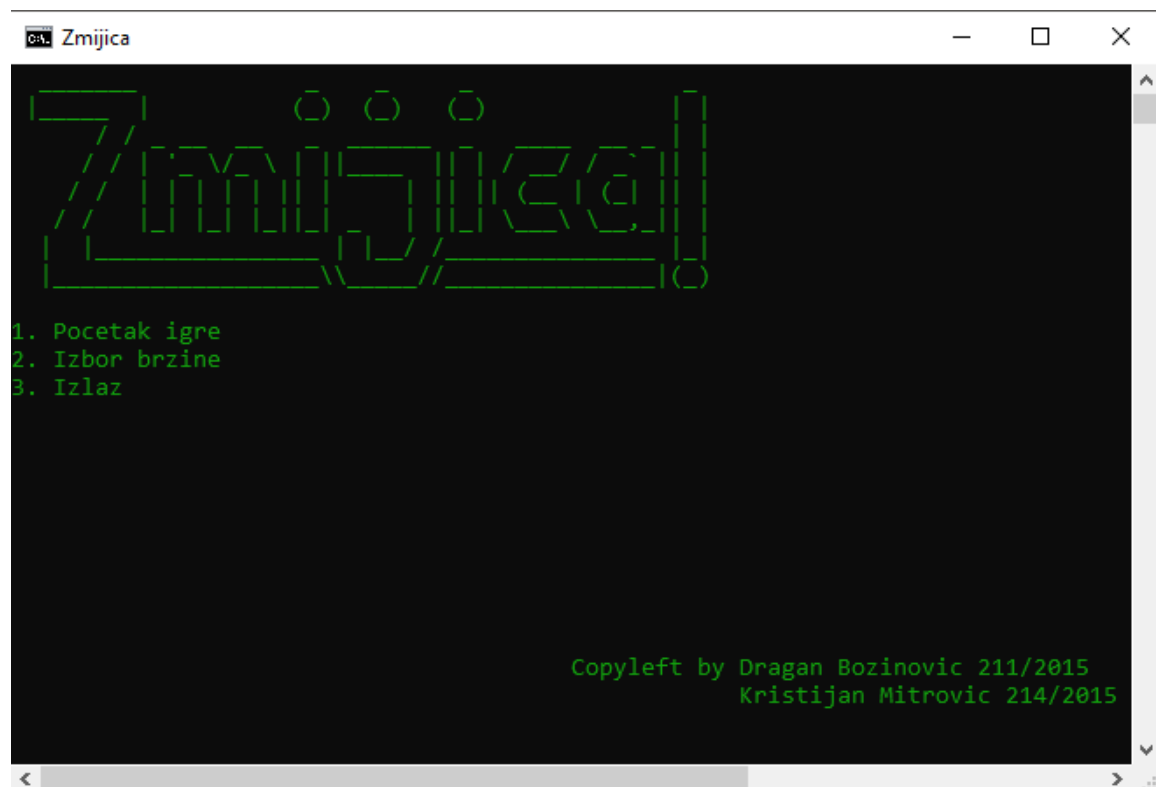
Tekst zadatka:

Potrebno je realizovati igru zmijica (Snake). Koristeci kursor tastere na tastaturi pomerati zmijicu u cetiri pravca. Blokove zmijice realizovati u vidu praznog prostora (spacebar ' ' FFFFh), tako da je prazan prostor kojim se iscrtava zmijica obojen u boju drugaciju od pozadine. Hrana se pojavljuje na nasumicno generisanim lokacijama (koriscenjem random funkcije) i takodje se realizuje koriscenjem praznog prostora obojenog u boju drugaciju od zmijice i pozadine. Pocetna pozicija zmijice je na sredini igralista i pocetna duzina zmijice je cetiri polja. Igrica se napusta pritiskom na taster ESC, prilikom kolizije sa preprekom ili prilikom kolizije zmijice sa samom sobom. Nakon zavrsetka igre, ispisuje se broj osvojenih poena. Prilikom pokretanja programa, i nakon napustanja igre, program treba da udje u glavni meni, u kojem je moguće odabrati pocetak igre, napustanje programa i brzinu kretanja zmijice.

Objasnjenje koda:

Program je napisan koristeci *Irvine32.inc* biblioteku I Irvinove preporuke za modele koji se koriste I velicinu steka za rad u *VS2017*.

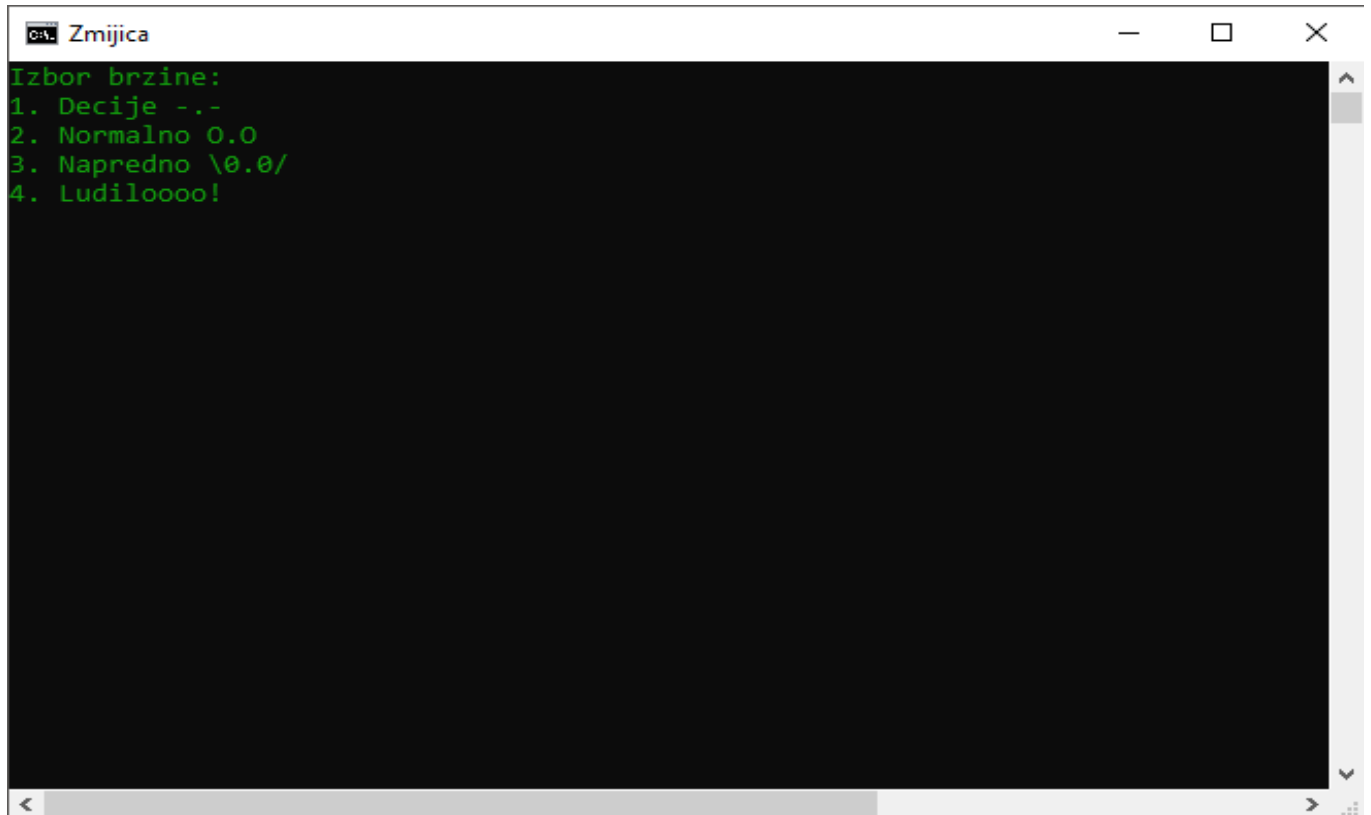
Na samom pocetku korisnika docekuje interaktivni *WELCOME* meni u kom se vrši izbor brzine kretanja zmijice, prilika da se pokrene igra I opcija za napustanje igrice.



Izgled WELCOME menija

Hendlovanje korisnikovog izbora na ovom i na svim ostalim mestima gde je to potrebno radjen je preko **call ReadChar** naredbe, koja korisnikov izbor smesta u **AL** registar, i potom se poredjenjem sa rednim brojem izbora skace na labelu ili proceduru koja treba dalje da nastavi izvršavanje saglasno izabranoj opciji.

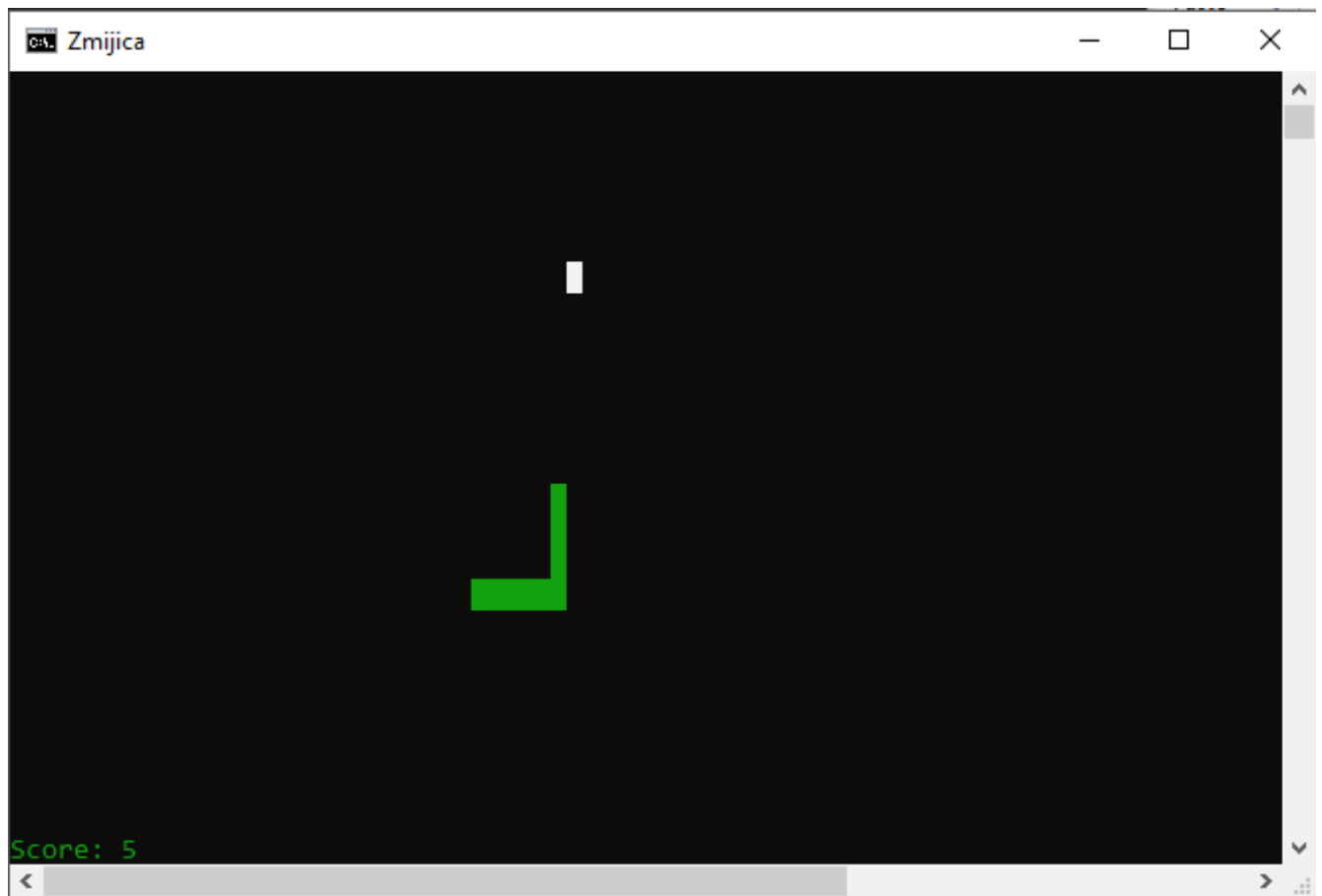
U meniju za izbor brzine postoje cetiri opcije od kojih svaka obezbedjuje kretanje zmijice brze od prethodne.



```
Izbor brzine:
1. Deciije -.-
2. Normalno 0.0
3. Napredno \0.0/
4. Ludiloooo!
```

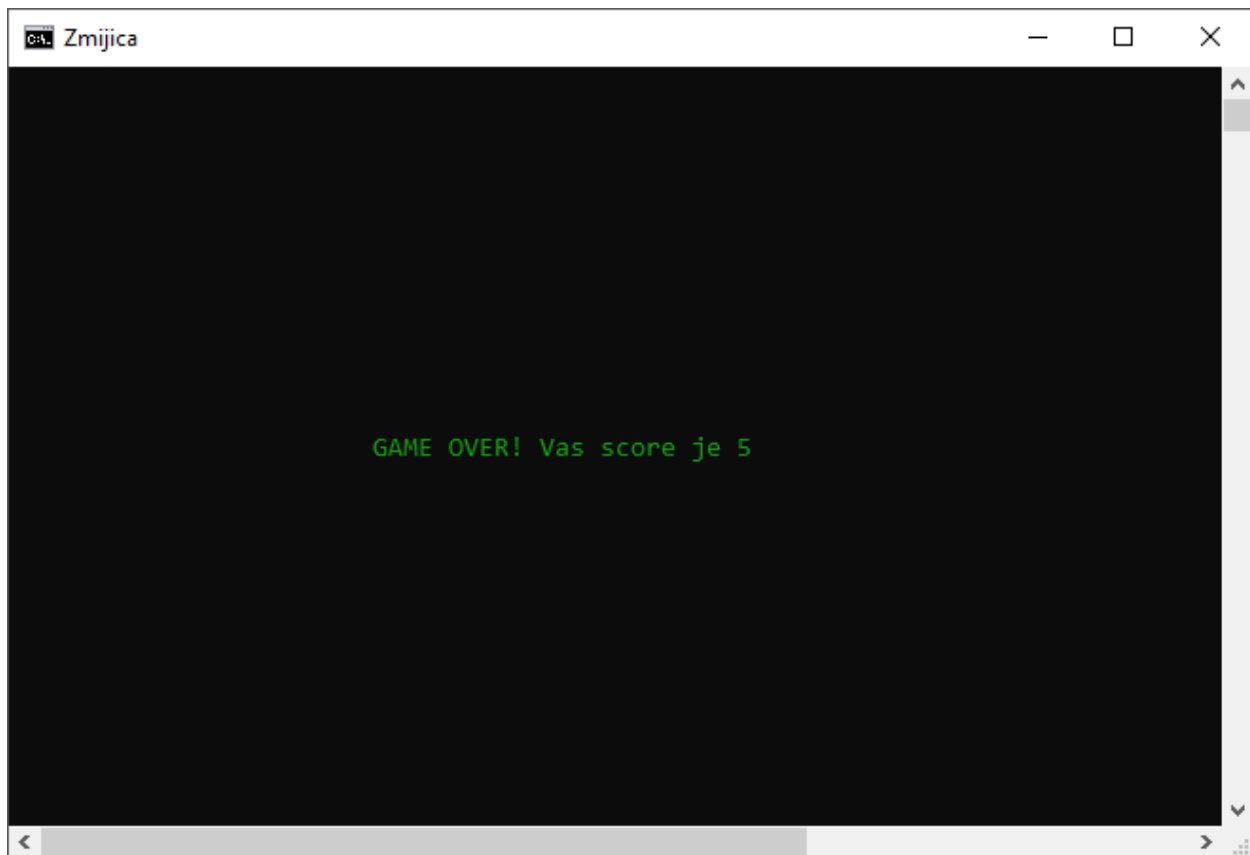
Izgled menija za izbor bzine

Po izboru brzine, korisnik se vraća na početni ekran odakle konačno može početi sa igrom.



Izgled prozora tokom trajanja igrice

Po pritiskanju *ESC* dugmeta ili prilikom sudara zmijice sa samom sobom, korisnika dočekuje obavestenje o postignutom skor i on biva vraćen na *WELCOME* meni.



Obavestenje o postignutom rezultatu

Kod i logika igrice su isparcani na najsitnije moguće celine grupisane u label e i procedure, kako bi krajnje upravljanje bilo sto jednostavnije. Svaka procedura odredjuje pipav deo posla koji joj je prepusten, tako da je guzva u **main** proceduri minimalna.

U **.const** sekciji definisane su neke konstante, poput dimenzija prozora, *ASCII* vrednosti odredjenih dugmadi na tastaturi koje se koriste prilikom hendlovanja korisnikove interakcije i default pozicija i smer kretanja zmije.

U **.data** sekciji definisani su svi stringovi, sve poruke koje obuhvataju i *WELCOME* meni, meni za izbor brzine, string koji obavestava korisnika o zavrsetku igre i postignutom skor, itd.

Zatim, tu je i promenljiva *frameBuffer*. Kretanje zmije bilo je moguće odraditi i bez ove promenljive, tako sto se novo polje iscrta na polju susednom od trenutnog položaja glave zmije, zavisno od smera kretanja zmije, od toga da li je korisnik promenio smer,

itd. Medjutim, na ovaj nacin nikako nije bilo moguće stvoriti privid kretanja zmije, jer bi rep uvek ostajao na prethodnoj poziciji, cak i ako smo pamtili njegove koordinate u *tailX* i *tailY* promenljivim, jer nije moguće odrediti na koje mesto treba pomeriti rep s obzirom na to da se nista ne zna o segmentu koji ide iznad repa, ka glavi. Ne znamo njegovu poziciju, da li je levo, desno, gore ili dole. Takodje je bilo potrebno na neki nacin detektovati sudar zmijice sa samom sobom, sto je opet zahtevalo poznavanje polja susednih trenutnom poloaju glave zmije, tj. da li je se nova glava zmije slucajno nasla na nekom polju koje je vec zauzeto nekim drugim segmentom zmije.

Zbog svega navedenog, prirodno se javila potreba da se sva polja igralista pamte u nekoj vrsti matrice, i da se elementi te matrice na pogodan nacin obeleze, tako da se razlikuju polja koja su slobodna od polja koja su zauzeta segmentima zmijice.

Za te potrebe napravljen je *frameBuffer*, niz od 24*80 (sto je visina*sirina igralista, polja po kom se zmija kreće, kao i dimenzije prozora) dvobajtnih memorijskih lokacija, u koji su inicijalno smestene nule koje predstavljaju prazan prostor.

Kako program napreduje, pocetnim postavljanjem zmijice na ekran u proceduri *initSnake*, neka od polja tog bafera bivaju popunjena jedinicama koje predstavljaju zmijicu. (Hrana se ne pamti u frejmbaferu, vec se pamte koordinate hrane foodX i foodY, pa se vrsi poredjenje sa njima pri ispitivanju da li je hrana pojedena.)

Dalje su generisane promenljive za pamcenje koordinata glave, repa i hrane, trenutnog i zeljenog smera kretanja zmije, brzina zmije, neki flegovi koji sluze kao indikatori da li je doslo do sudara, da li rep treba da bude obrisan (ako hrana nije pojedena) i jos neke pomocne promenljive.

U *.data?* sekciji su formirane promenljive koje sluze kontrolu standardnog ulaza i izlaza, neki bafer za smestanje dogadjaja koje generise operativni sistem prilikom pritiska dugmeta na tastaturi, itd.

main procedura pocinje regulacijom vidljivosti kursora (jer nije potrebno da kursor blinka dok igrate) i velicine prozora. Prozor biva smanjen i ostaje zadatih dimenzija dogod nije maksimiziran ili pomeren tako da je snap-ovan na levu ili desnu polovinu ekrana, kada menja svoj oblik, sto nije bilo moguće zaustaviti, jer je to pod kontrolom operativnog sistema. Svakako, dimenzije igralista ostaju iste i zmija se kreće po polju koje je predvidjeno, kako god prozor bio dimenzionisan.

Nadalje se vrsi iscrtavanje *WELCOME* menija i hendlovanje korisnikovog izbora.

Kada je utvrdjena brzina zmijice (koja je po default-u *2. Normalna*), i konacno izabrana opcija za pokretanje igrice, vrsi se ciscenje konzole, zatim pozivanje *initSnake* procedure koja, kako je pomenuto, inicijalizuje zmijicu na pocetnu poziciju upisujuci vrednosti u frejmbafer, zatim *Paint* procedure koja je zaduzena za iscrtavanje trenutnog stanja igre na konzolu, tako sto cita vrednost polja iz frejmbafera i jedno po jedno iscrtava, zatim se poziva procedura *createFood* koja generise hranu na nasumicnoj poziciji unutar dozvoljenih granica igralista i iscrtava je, vodeci racuna da se iscrtavanje ne desi na nekom polju koje je vec zauzeto segmentom zmijice, i konacno poziva *startGame* procedura koja je zapravo zaduzena za interakciju sa korisnikom, usmeravanje zmije u zeljenom smeru, proveru da li je doslo do sudara, itd.

startGame procedura takodje vrši kontrolu vremenskog razmaka između iscrtavanja segmenata zmijice, čime se dobija utisak prividnog kretanja.

Unutar ove procedure postoji beskonacna **mainGameLoop** petlja, koja se prekida prilikom sudara ili pritiska *ESC* dugmeta. U okviru te petlje, prati se input korisnika. Ukoliko inputa nije bilo, zmija nastavlja kretanje u smeru u kom se već kretala. Ukoliko jeste, nizom podprocedura, tj. labela se vrši hendlovanje konkretnog slučaja, što će reći, ukoliko se zmijica kretala navise, nije moguće prihvatiti novi smer kretanja ka nanize, već samo levo ili desno, i slične nebrojene zackoljice koje se neizbežno javljaju i koje su brute force pristupom iskorenjene mucnim satima i satima prolaska kroz debugger.

Tokom te beskonacne petlje, poziva se još jedna bitna procedura **moveSnake**, koja osvežava stanje frejmbafera, čime se efektivno vrši pomeranje zmije, i još samo ostaje da se novo stanje frejmbafera iscrta na ekran kako bi se dobio utisak promene.

Ova procedura efektivno vrši pomeranje zmijice tako što, počevši od repa, ide segment po segment sve do glave, i pomera ga (upisom u frejmbafer, a preko funkcije **saveIndex**) na potrebnu poziciju. Ova procedura takodje ima bezbroj granicnih slučajeva koji mogu da se dese, naročito zato što se ovde mora vršiti provera da li je došlo do sudara i da li je pojedena hrana.

Kako je napomenuto, vodi se računa o svim susednim segmentima glave ili repa, i vrši se provera da li eventualno pomeranje glave na novu poziciju dovodi do preklapanja sa nekim postojećim segmentom zmije, tj. da li se na toj novoj željenoj poziciji u frejmbaferu već ne nalazi upisana jedinica koja predstavlja zmijicu. Takodje pri pomeranju repa mora se znati koji mu je susedan segment, kako bismo znali gde da ga pomerimo.

Procedure **accessIndex** i **saveIndex** koriscene su na svim mestima gde je potrebno pristupiti nekom polju frejmbafera i sa njega očitati vrednost tj. u njega upisati vrednost.

Kod igrice dat je na narednim slajdovima.

`comment &`

`/*****`

`***** IGRICA ZMIJICE *****`

Napisana koristeci Irvine32.inc biblioteku i Irvine-ove preporuke za modele koji se koriste i velicinu steka za rad u VS2017.

Na samom pocetku korisnika docekuye interaktivni WELCOME meni u kom se vrši izbor brzine kretanja zmijice, prilika da se pokrene igra sa default vrednoscu brzine i opcija za napustanje igrice.

U bilo kom trenutku tokom igranja moguće je napustiti igricu pritiskom na ESC dugme, cime se korisnik vraca na WELCOME meni gde se može izabrati opcija za konacan izlazak iz igre.

Kod i logika igrice su isparcani na najsitnije logicke celine grupisane u labele i procedure, kako bi krajnje upravljanje bilo jednostavno. Svaka procedura odradjuje pipav deo posla koji joj je prepusten, tako da je detaljisanje u main proceduri minimalno, bez guzve i necitkosti.

`*****/&`

`include Irvine32.inc`

`include macros.inc`

`.386`

`.model flat, stdcall`

`.stack 4096`

`ExitProcess proto, dwExitCode:dword`

.const

///
// Definisanje velicine prozora

xmin = 0 ///
// leva ivica

xmax = 79 ///
// desna ivica

ymin = 0 ///
// gornja ivica

ymax = 24 ///
// donja ivica

///
// Oznake za levo, desno, gore, dole, ESC, ASCII

LEFT_KEY = 025h

UP_KEY = 026h

RIGHT_KEY = 027h

DOWN_KEY = 028h

ESC_KEY = 01Bh

///
// Definisanje pocetnih koordinata zmijice i pocetnog smera kretanja

///
// Valja blago prepraviti kod u initSnake kako bi se zmija postavila

///
// vertikalno, umesto horizontalno, kao sto je trenutno.

headX_default = 40d

headY_default = 12d

tailX_default = 37d

tailY_default = 12d

direction_default = 'R' ///
// R-right, U-up, D-down, L-left

.data

///
// Stringovi za ispis WELCOME screena i izbor brzine zmijice

T1 byte " _____ _ _ _ _ ", 0dh, 0ah, 0

T2 byte " |_____| () () () | | ", 0dh, 0ah, 0

T3 byte " // _ _ _ _ _ _ _ _ _ _ | | ", 0dh, 0ah, 0

T4 byte " // | ' V _ \ | | | | _ _ | | / _ / _ ' | | | ", 0dh, 0ah, 0

T5 byte " // | | | | | | | | | | | | | | (_ | (_ | | | | ", 0dh, 0ah, 0

```

T6 byte " // | | | | | | _ | | | \ \ \ \ \ \ _ | | | ", 0dh, 0ah, 0
T7 byte " | | _____ | | / / _____ | | ", 0dh, 0ah, 0
T8 byte " | _____ \ \ \ \ // _____ | ( ) ", 0dh, 0ah, 0

```

```

welcomeString byte " ", 0dh, 0ah,
"1. Pocetak igre" , 0dh, 0ah,
"2. Izbor brzine" , 0dh, 0ah,
"3. Izlaz" , 0dh, 0ah, 0

```

```

copyright byte " " , 0dh, 0ah, " " , 0dh, 0ah, " " , 0dh, 0ah, " " , 0dh, 0ah,
" " , 0dh, 0ah, " " , 0dh, 0ah, " " , 0dh, 0ah, " " , 0dh, 0ah,
" " , 0dh, 0ah,
"
                                Copyleft by Dragan Bozinovic 211/2015" , 0dh, 0ah,
"
                                Kristijan Mitrovic 214/2015" , 0dh, 0ah, 0

```

```

speedString byte "Izbor brzine:" , 0dh, 0ah,
"1. Decije -.-" , 0dh, 0ah,
"2. Normalno O.O" , 0dh, 0ah,
"3. Napredno \0.0/" , 0dh, 0ah,
"4. Ludiloooo!" , 0dh, 0ah, 0

```

```

gameOverString byte "Game Over!" , 0

```

```

scoreString byte "Score: 0" , 0

```

```

yourScoreString byte "GAME OVER! Vas score je " , 0

```

```

; // Promenljive koriscene u programu
frameBuffer word 1920 dup(0) ; // Frejmbafer u kom se pamte stanja svakog od
; // 24*80 polja u konzoli, medju kojima su prazna

```

///
// polja predstavljena nulama i sama zmijica

headY byte headY_default ///
// Y koordinata glave zmije
headX byte headX_default ///
// X koordinata glave zmije
tailY byte tailY_default ///
// Y koordinata repa zmije
tailX byte tailX_default ///
// X koordinata repa zmije
foodY byte ? ///
// Y koordinata hrane
foodX byte ? ///
// X koordinata hrane

currDirection byte direction_default ///
// Trenutni smer kretanja zmije
newDirection byte direction_default ///
// Zeljeni smer kretanja koji je uneo igrac
snakeSpeed dword 100 ///
// Brzina zmijice koja je zapravo period sa
// kojim se osvezava iscrtavanje na ekranu

tempY byte 0 ///
// Pomocna promenljiva za smestanje Y koordinate
tempX byte 0 ///
// Pomocna promenljiva za smestanje X koordinate

Yabove byte 0d ///
// Red iznad trenutnog
Xleft byte 0d ///
// Kolona levo od trenutne
Ybelow byte 0d ///
// Red ispod trenutnog
Xright byte 0d ///
// Kolona desno od trenutne

flag_tail byte 1d ///
// Fleg koji oznacava da li rep treba da bude obrisani ili ne
search word 0d ///
// Vrednost sledeceg segmenta zmijice koji se iscrtava
flag_endTheGame byte 0d ///
// Fleg koji oznacava da li igra treba da se prekine
playerScore dword 0d ///
// Total score
welcomeDelay dword 100

windowRect SMALL_RECT <xmin, ymin, xmax, ymax> ///
// Velicina prozora
winTitle byte "Zmijica" , 0 ///
// Naslov programa
cursorInfo CONSOLE_CURSOR_INFO <> ///
// Informacije o kursoru

.data?

```
;// Promenljive koje su potrebne za hendlovanje podataka unetih u konzolu tj.  
interakciju sa korisnikom  
stdOutHandle handle ?  
stdInHandle handle ?           ;// Promenljiva za kontrolu inputa u konzolu  
numInp dword ?                 ;// Broj bajtova u ulaznom baferu  
temp byte 16 dup(?)            ;// Promenljiva koja sadrzi podatke tipa  
INPUT_RECORD  
bRead dword ?                  ;// Broj procitanih ulaznih bajtova
```

.code

```
;// -----  
;// main procedura koja postavlja WELCOME screen, hendluje izbor brzine zmijice  
;// i prosledjuje podprocedurama inicijalizaciju ekrana za pocetak igre, postavljanje  
;// pocetne zmijice na sredinu ekrana, generisanje hrane na nasumicnom mestu i poziva  
;// najvazniju proceduru startGame koja prati komande koje zadaje korisnik i kontrolise  
;// kretanje zmijice.  
;// Zbog pozivanja silnih procedura koje svaka za sebe obavljaju deo posla, interfejs  
;// u main proceduri je prilicno jednostavan, na stranu to sto je potreban veliki  
;// broj komandi za obavljanje nekih jednostavnih funkcija, sto nije slucaj sa nekim  
;// visim programskim jezikom.  
;// -----
```

main PROC

```
invoke GetStdHandle, STD_OUTPUT_HANDLE  
;// Postavlja handle za ispis podataka  
mov stdOutHandle, eax  
  
invoke GetConsoleCursorInfo, stdOutHandle, addr cursorInfo ;// Cita trenutno  
stanje kursora
```

```
mov cursorInfo.bVisible, 0
;/// Postavlja vidljivost kursora na nevidljiv
invoke SetConsoleCursorInfo, stdOutHandle, addr cursorInfo ;/// Postavlja novo
stanje kursora
```

```
invoke SetConsoleTitle, addr winTitle ;/// Postavlja title prozora
invoke SetConsoleWindowInfo, stdOutHandle, TRUE, addr windowRect ;/// Dimenzije
prozora
mov eax, green + (black * 16)
;/// Boja interfejsa i prozora. Upisuju se u al i ah registre, zato je zapis ovakav
call SetTextColor
```

menu:

```
call Randomize ;/// Postavlja seme za
randomizaciju, slicno C-ovskoj logici
call clrscr ;/// Brise ekran konzole
call welcomeZmijica ;/// Ispis velikog stilizovanog ZMIJICA
mov edx, offset welcomeString
call WriteString ;/// Ispis menija
mov edx, offset copyright
call WriteString
```

```
welcomeLoop: ;/// Loopovanje kroz WELCOME meni dok se ne unese
;/// pravilan izbor
call ReadChar
```

```
cmp al, '1' ;/// 1. Pocni igricu
je initializeGame
```

```
cmp al, '2' ;/// 2. Izbor brzine
je speed
```

```
cmp al, '3'           ;// 3. Izlazak iz programa
jne welcomeLoop      ;// Ili se vrtilo dok se ne unese ispravan izbor
```

EXIT

```
speed:                ;// Meni za izbor brzine
```

```
call clrscr
mov edx, offset speedString
call WriteString      ;// Ispis menija za izbor brzine
```

```
loopSpeed:           ;// Loopovanje kroz meni za izbor brzine
call ReadChar
```

```
cmp al, '1'          ;// Decije
je speed1
```

```
cmp al, '2'          ;// Normalno
je speed2
```

```
cmp al, '3'          ;// Napredno
je speed3
```

```
cmp al, '4'          ;// Ludilo
je speed4
```

```
jmp loopSpeed
```

```
speed1:              ;// Brzina zmijice odredjena je refresh rate-om
mov snakeSpeed, 150
jmp menu
```

speed2:

mov snakeSpeed, 100

jmp menu

speed3:

mov snakeSpeed, 50

jmp menu

speed4:

mov snakeSpeed, 35

jmp menu ;// Povratak na glavni meni po izboru brzine

initializeGame: ;// Postavlja flegove potrebne za generisanje

;// zmijice i hrane i poziva glavnu proceduru

;// startGame koja hendluje samu igru

mov eax, 0 ;// Ciscenje registara

mov edx, 0

call clrscr

call initSnake ;// Postavlja zmiju na pocetnu poziciju

call Paint ;// Iscrtava igraliste na kom se nalazi
zmija

call createFood ;// Postavlja hranu na nasumicno mesto
na ekranu

call startGame ;// Poziv glavne funkcije za pokretanje
igre

mov eax, green + (black * 16) ;// Ako je procedura startGame zavrсила posao, to
znaci

call SetTextColor ;// da je iz nekog razloga (sudar ili ESC) kraj
igre

jmp menu ;// i igrac se vracа na pocetni meni

main ENDP

;;; -----

welcomeZmijica PROC ;;; Iscrtava veliko stilizovano Zmijica na WELCOME screen

```
mov edx, offset T1
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T2
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T3
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T4
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T5
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T6
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T7
```



```
call WriteString
mov eax, welcomeDelay
call delay
mov edx, offset T8
call WriteString
mov eax, welcomeDelay
call delay
```

```
RET
```

```
welcomeZmijica ENDP
```

```
initSnake PROC
```

```
    ;// Postavlja zmijicu duzine 4 polja na koordinate definisane sa headX/Y_default
```

```
mov dh, headY_default    ;// Y pozicija glave
mov dl, headX_default    ;// X pozicija glave
mov bx, 1                ;// To je prvi segment zmijice (glava) koji se
upisuje u bx
call saveIndex           ;// a potom pamti u frejmbaferu preko saveIndex
procedure
```

```
mov dh, headY_default    ;// Y pozicija vratnog dela
mov dl, headX_default - 1 ;// X pozicija vratnog dela
mov bx, 2                ;// Drugi segment zmije (vrat)
call saveIndex
```

```
mov dh, headY_default    ;// Y pozicija kicmenog dela
mov dl, headX_default - 2 ;// X pozicija kicmenog dela
mov bx, 3                ;// Treci segment zmije (kicma)
call saveIndex
```

```
mov dh, headY_default      ;// Y pozicija repa
mov dl, headX_default - 3  ;// X pozicija repa
mov bx, 4                  ;// Cetvrti segment zmije (rep)
call saveIndex
```

```
RET
```

```
initSnake ENDP
```

```
clearMem PROC                ;// Brise frejmbafer, resetuje poziciju zmije i duzinu
                                ;// i postavlja sve flegove na njihovu default vrednost
```

```
mov dh, 0                    ;// Postavlja registar kojim se krece kroz Y koordinate na 0
mov bx, 0                    ;// Postavlja data registar na 0
```

```
rowLoop:                    ;// Obilazak matrice po redovima
cmp dh, 24                   ;// Broji dok ne dostigne 24 (poslednji red) i iskace
je endRowLoop
```

```
mov dl, 0                    ;// Postavlja registar kojim se krece kroz X koordinate na 0
```

```
columnLoop:                 ;// Obilazak kolona unutar trenutnog reda
cmp dl, 80                   ;// Kada dodje do 80 (poslednja kolona), iskace
je endColumnLoop
```

```
call saveIndex              ;// Poziva proceduru za upis u frejmbafer na osnovu dh i dl
```

```
inc dl                      ;// Petlja se nastavlja u sledecoj koloni
jmp columnLoop
```

```
endColumnLoop:              ;// Kraj unutrasnje petlje
```

```

inc dh                ;// Povecava broj reda i nastavlja petlju u njemu
jmp rowLoop

endRowLoop:           ;// Kraj spoljasnje petlje
mov tailY, tailY_default ;// Resetuje koordinate glave i repa na default
mov tailX, tailX_default
mov headY, headY_default
mov headX, headX_default

mov flag_endTheGame, 0 ;// Brise fleg koji oznacava kraj igre (dakle, nije kraj igre jos)
mov flag_tail, 1       ;// Postavlja fleg za brisanje repa (nikakva hrana nije
                        ;// pojedena, vraca zmijicu na 4 polja)

mov currDirection, direction_default ;// Trenutni i sledeci smer kretanja vraceni na
default

mov newDirection, direction_default
mov snakeSpeed, 100
mov playerScore, 0      ;// Resetuje score igraca

RET

clearMem ENDP

```

startGame PROC

comment &/*

Ova procedura je zapravo glavna, zaduzena je za obavljanje glavnog posla, a to je upravljanje zmijicom, reagovanje na kontrole koje zadaje korisnik, i zavisno od trenutnog smera kretanja menja ili ne menja kretanje zmije.

Procedura takodje vrši kontrolu vremenskog razmaka izmedju iscrtavanja, cime se prividno kontrolise brzina zmije.

Izvršava se beskonacna petlja iz koje se iskace kada korisnik pritisne ESC ili dodje do sudara zmije sa samom sobom, i pri izlasku se resetuju flegovi i cisti frejmbafer.

`*/&`

`mov eax, green + (black * 16)`

`call SetTextColor`

`mov dh, 24` `/// Ispisivanje skora u donjem levom uglu`

`mov dl, 0`

`call GotoXY`

`mov edx, offset scoreString`

`call WriteString`

`/// Uzima input iz konzole i smesta u memoriju`

`invoke getStdHandle, STD_INPUT_HANDLE`

`mov stdInHandle, eax`

`mov ecx, 10`

`/// Cita dva dogadjaja iz bafera`

`invoke ReadConsoleInput, stdInHandle, addr temp, 1, addr bRead`

`invoke ReadConsoleInput, stdInHandle, addr temp, 1, addr bRead`

`mainGameLoop:` `/// Glavna beskonacna petlja`

`/// Broj dogadjaja u baferu`

`invoke GetNumberOfConsoleInputEvents, stdInHandle, addr numInp`

`mov ecx, numInp`

`cmp ecx, 0` `/// Provera da li je input bafer prazan`

`je done` `/// Ako jeste, to znaci da nije bilo interakcije korisnika`

`/// sa programom i preskace se hendlovanje bilo kakvog inputa`

`/// i nastavlja sa kretanjem zmijice u smeru u kom je zapocela kretanje`

`/// Cita jedan event iz bafera i smesta u temp`

```

invoke ReadConsoleInput, stdInHandle, addr temp, 1, addr bRead
mov dx, word PTR temp          ;// Samo u slucaju da je event tipa KEY_EVENT,
cmp dx, 1                      ;// sto su ugradjeni tipovi za dogadjaje koji se signaliziraju
jne mainGameLoop              ;// operativnom sistemu kada se desi input na
tastaturi, onda se procesira dalje taj dogadjaj

mov dl, byte PTR [temp+4]      ;// Posto se signal generise i u slucaju da se dugme
pritisne i otpusti
cmp dl, 0                      ;// biti koji znace otpustanje se preskacu
je mainGameLoop

mov dl, byte PTR [temp+10] ;// a pritisnuti koji vracaju hexa kod dugmeta koje je
pritisnuto se procesiraju

cmp dl, ESC_KEY               ;// Ako je ESC pritisnut, vraca se na pocetni meni
je quit

cmp currDirection, 'U'        ;// Samo u slucaju da se zmija krece navise ili nanize, moze
se izvorsiti
je handleHorMovement         ;// promena smera kretanja u levo ili desno, zavisno od
pritisnutog tastera
cmp currDirection, 'D'
je handleHorMovement

jmp handleVerMovement        ;// Ako se ne prodju gornje dve provere, to znaci da je
trenutno kretanje zmiije
;// horizontalno, pa se mogu procesirati samo promene smera u vertikalnom pravcu
handleHorMovement:
cmp dl, LEFT_KEY              ;// Ukoliko je pritisnuta strelica ulevo
je handleHorMovement1

cmp dl, RIGHT_KEY             ;// Ukoliko je pritisnuta strelica udesno
je handleHorMovement2

```

```
jmp mainGameLoop
```

```
handleHorMovement1:
```

```
mov newDirection, 'L'
```

```
jmp mainGameLoop
```

```
handleHorMovement2:
```

```
mov newDirection, 'R'
```

```
jmp mainGameLoop
```

```
handleVerMovement:
```

```
cmp dl, UP_KEY      ;// Pritisnuta strelica navise
```

```
je handleVerMovement1
```

```
cmp dl, DOWN_KEY    ;// Nanize
```

```
je handleVerMovement2
```

```
jmp mainGameLoop
```

```
handleVerMovement1:
```

```
mov newDirection, 'U'
```

```
jmp mainGameLoop
```

```
handleVerMovement2:
```

```
mov newDirection, 'D'
```

```
jmp mainGameLoop
```

```
done:
```

```
mov bl, newDirection      ;// Postavlja novi smer kretanja za smer kretanja zmije
```

```
mov currDirection, bl     ;// Mora ovako preko registra, jer nije dozvoljeno
```

///
dodeljivanje iz promenljive u promenljivu

call MoveSnake ///
mov eax, snakeSpeed ///
kretanja Uvodi se kasnjenje u iscrtavanju koje daje utisak
call Delay

mov bl, currDirection
mov newDirection, bl

cmp flag_endTheGame, 1 ///
je quit Ako je se desio sudar, fleg je to zapamtio
 pa se izlazi iz igre

jmp mainGameLoop ///
 U suprotnom se vraća na glavni loop

quit:

mov eax, green + (black * 16)
call SetTextColor
call clrscr
push eax
push edx
mov dh, 12
mov dl, 24
call GotoXY
mov edx, offset yourScoreString
call writeString
mov eax, playerScore
call writedec
mov eax, 2500
call delay
pop edx
pop eax

```
call clearMem           ;// Ispisuje postignut skor, vraca podesavanja na default i  
ide na glavni meni
```

```
RET
```

```
startGame ENDP
```

```
MoveSnake PROC
```

```
comment &/*
```

Ova procedura osvezava frejmbafer, cime se efektivno vrši pomeranje zmije. Pocevši od repa, ova procedura traži sledeći susedni segment. Svi segmenti bivaju osveženi premestanjem na nove pozicije, pri čemu se poslednji briše ukoliko hrana nije pojedena i novi segment se dodaje na početak i postaje glava, zavisno od toga u kom smeru se vrši kretanje zmije.

Takodje se ovde vrši provera da li je doslo do sudara i da li je eventualno pojedena hrana.

```
*/&
```

```
cmp flag_tail, 1
```

```
jne dontEraseTail      ;// Rep se ne briše ako fleg ne diktira tako
```

```
mov dh, tailY          ;// Ucitavaju se koordinate repa
```

```
mov dl, tailX
```

```
call accessIndex       ;// Pristupa se frejmbaferu na zadatim koordinatama i po  
povratku
```

```
dec bx                 ;// u bx imamo upisanu vrednost koju je vratio  
frejmbafer, cija
```

```
;// vrednost se dekrementira, cime efektivno dobijamo vrednost sledeceg segmenta
```

```
mov search, bx         ;// Vrednost sledeceg segmenta se stavlja u search
```



```
mov bx, 0                ;// Iz frejmbafera se vrednost koja odgovara repu
stavlja na 0, tj. brise
```

```
call saveIndex
```

```
call GotoXY              ;// kao i sa ekrana
```

```
mov eax, green + (black * 16)
```

```
call SetTextColor
```

```
mov al, ''
```

```
call WriteChar
```

```
push edx                ;// Kursor se postavlja u donji desni ugao
```

```
mov dl, 79
```

```
mov dh, 23
```

```
call GotoXY
```

```
pop edx
```

```
mov al, dh              ;// Y koordinata repa se smesta u al
```

```
dec al
```

```
mov Yabove, al         ;// Cuva se indeks reda iznad trenutnog
```

```
add al, 2
```

```
mov Ybelow, al         ;// Indeks reda ispod trenutnog
```

```
mov al, dl              ;// X koordinata repa se smesta u al
```

```
dec al
```

```
mov Xleft, al          ;// Cuva se indeks kolone levo od trenutne
```

```
add al, 2
```

```
mov Xright, al         ;// Cuva se indeks kolone desno od trenutne
```

```
cmp Ybelow, 24          ;// Ako indeks izlazi van okvira ekrana na donju stranu
```

```
jne next1
```

```
mov Ybelow, 0           ;// vraca se na 0, tj. na gornju
```

next1:

`cmp Xright, 80` `;``//` Ako indeks izlazi van okvira ekrana u desnu stranu

`jne next2`

`mov Xright, 0` `;``//` vraća se na 0, tj. na levu

next2:

`cmp Yabove, 0` `;``//` Ako indeks izlazi van okvira ekrana u gornju stranu

`JGE next3`

`mov Yabove, 23` `;``//` vraća se na 23, tj. na donju

next3:

`cmp Xleft, 0` `;``//` Ako indeks izlazi van okvira ekrana u levu stranu

`JGE next4`

`mov Xleft, 79` `;``//` vraća se na 79, tj. na desnu

next4:

`mov dh, Yabove` `;``//` Y koordinata piksela iznad repa

`mov dl, tailX` `;``//` X koordinata piksela iznad repa

`call accessIndex` `;``//` Pristupa se pikselu u frejmbaferu

`cmp bx, search` `;``//` Provera da li je piksel sledeci segment zmije

`jne melseif1`

`mov tailY, dh` `;``//` i pomera rep na novu lokaciju, ako jeste

`jmp mendif`

melseif1:

`mov dh, Ybelow` `;``//` Y koordinata piksela ispod repa

`call accessIndex` `;``//` Pristupa se pikselu u frejmbaferu

`cmp bx, search` `;``//` Provera da li je piksel sledeci segment zmije

`jne melseif2`

`mov tailY, dh` `;``//` i pomera rep na novu lokaciju, ako jeste

jmp mendif

melseif2:

```
mov dh, tailY      ;// Y koordinata piksela levo od repa
mov dl, Xleft      ;// X koordinata piksela levo od repa
call accessIndex   ;// Pristupa se pikselu u frejmbaferu
cmp bx, search      ;// Provera da li je piksel sledeci segment zmije
jne melse
mov tailX, dl       ;// i pomera rep na novu lokaciju, ako jeste
jmp mendif
```

melse:

```
mov dl, Xright      ;// Pomera rep na piksel desno od repa
mov tailX, dl
```

mendif:

dontEraseTail:

```
mov flag_tail, 1    ;// Postavlja se fleg za brisanje repa
mov dh, tailY
mov dl, tailX
mov tempY, dh        ;// Pamti se indeks reda u promenljivu
mov tempX, dl        ;// Pamti se indeks kolone u promenljivu
```

whileTrue: ;// Prolazak kroz sve segmente zmijice i podesavanje vrednosti svakog

```
mov dh, tempY
mov dl, tempX
call accessIndex     ;// Vrednost piksela izvadjena iz frejmbafera
dec bx               ;// U bx se smesta vrednost sledeceg segmenta
mov search, bx
```

```

push ebx                ;// Vrednost trenutnog segmenta refresuje se
vrednoscu prethodnog segmenta
add bx, 2               ;// (zbog kretanja zmije, segmenti se krecu)
call saveIndex
pop ebx

cmp bx, 0               ;// Provera da li je trenutni segment glava zmije
je break

mov al, dh              ;// Indeks reda trenutnog segmenta
dec al                  ;// Indeks reda iznad trenutnog
mov Yabove, al
add al, 2               ;// Indeks reda ispod trenutnog
mov Ybelow, al

mov al, dl              ;// Indeks kolone trenutnog segmenta
dec al                  ;// Indeks kolone levo od trenutne
mov Xleft, al
add al, 2               ;// Indeks kolone desno od trenutne
mov Xright, al

cmp Ybelow, 24          ;// Ako novi indeks izlazi van granica, vrati ga
jne next21
mov Ybelow, 0

next21:
cmp Xright, 80          ;// Ako novi indeks izlazi van granica, vrati ga
jne next22
mov Xright, 0

next22:
cmp Yabove, 0           ;// Ako novi indeks izlazi van granica, vrati ga

```

```
JGE next23
mov Yabove, 23
```

```
next23:
cmp Xleft, 0      ;// Ako novi indeks izlazi van granica, vrati ga
JGE next24
mov Xleft, 79
```

```
next24:
mov dh, Yabove    ;// Indeks reda piksela iznad trenutnog segmenta
mov dl, tempX      ;// Indeks kolone piksela iznad trenutnog segmenta
call accessIndex   ;// Pristup pikselu u frejmbaferu
cmp bx, search     ;// Provera da li je piksel sledeci segment zmije
jne elseif21
mov tempY, dh      ;// pomeri indeks na novu lokaciju, ako jeste
jmp endif2
```

```
elseif21:
mov dh, Ybelow    ;// Indeks reda piksela ispod trenutnog segmenta
call accessIndex   ;// Pristup pikselu u frejmbaferu
cmp bx, search     ;// Provera da li je piksel sledeci segment zmije
jne elseif22
mov tempY, dh      ;// pomeri indeks na novu lokaciju, ako jeste
jmp endif2
```

```
elseif22:
mov dh, tempY      ;// Indeks reda piksela levo od trenutnog segmenta
mov dl, Xleft      ;// Indeks kolone piksela levo od trenutnog segmenta
call accessIndex   ;// Pristup pikselu u frejmbaferu
cmp bx, search     ;// Provera da li je piksel sledeci segment zmije
jne else2
```

```
mov tempX, dl          ;// pomeri indeks na novu lokaciju, ako jeste
jmp endif2
```

else2:

```
mov dl, Xright          ;// Pomeri indeks na piksel desno od segmenta
mov tempX, dl
```

endif2:

```
jmp whileTrue          ;// Nastavlja se petlja dok se ne dodje do glave zmije
```

break:

```
mov al, headY           ;// Y koordinata glave
dec al
```

```
mov Yabove, al          ;// Indeks reda iznad
```

```
add al, 2
```

```
mov Ybelow, al          ;// Indeks reda ispod
```

```
mov al, headX
```

```
dec al
```

```
mov Xleft, al
```

```
add al, 2
```

```
mov Xright, al
```

```
cmp Ybelow, 24          ;// Ako prelazi granice, vrati ga
```

```
jne next31
```

```
mov Ybelow, 0
```

next31:

```
cmp Xright, 80
```

```
jne next32
```

```
mov Xright, 0
```

```
next32:
```

```
cmp Yabove, 0
```

```
JGE next33
```

```
mov Yabove, 23
```

```
next33:
```

```
cmp Xleft, 0
```

```
JGE next34
```

```
mov Xleft, 79
```

```
next34:
```

```
cmp currDirection, 'U'
```

```
;;; Ako je smer kretanja navise, Y koordinata glave
```

```
jne elseif3
```

```
;;; se penje u red iznad
```

```
mov al, Yabove
```

```
mov headY, al
```

```
jmp endif3
```

```
elseif3:
```

```
cmp currDirection, 'D'
```

```
;;; Ako je nanize, ispod
```

```
jne elseif32
```

```
mov al, Ybelow
```

```
mov headY, al
```

```
jmp endif3
```

```
elseif32:
```

```
cmp currDirection, 'L'
```

```
;;; Ako je levo, onda levo
```

```
jne else3
```

```
mov al, Xleft
```

```
mov headX, al
jmp endif3
```

```
else3:
```

```
mov al, Xright           ;// Desno, onda desno
mov headX, al
```

```
endif3:
```

```
mov dh, headY
mov dl, headX
```

```
call accessIndex        ;// Pristupa se poziciji gde bi trebalo da je nova glava
cmp bx, 0                ;// Ako je prazan piksel, onda nije doslo do sudara
je NoHit
```

```
mov eax, 2000           ;// Ako jeste, ispisuje se poruka i izlazi iz procedure
mov dh, 24
mov dl, 11
call GotoXY
mov edx, offset gameOverString
call WriteString
```

```
call Delay
mov flag_endTheGame, 1
```

```
RET
```

```
NoHit:                  ;// Ako nije bilo sudara
mov bx, 1
call saveIndex
```



```
mov cl, foodX
mov ch, foodY
```

```
cmp cl, dl      ;// Ako se koordinate glave i hrane ne poklapaju,
jne foodNotGobbled ;// hrana nije pojedena
cmp ch, dh
jne foodNotGobbled
```

```
call createFood ;// Ako je pojedena, pravi se nova
mov flag_tail, 0
```

```
mov eax, green + (black * 16)
call SetTextColor
```

```
push edx
```

```
mov dh, 24      ;// Apdejtuje se skor
mov dl, 7
call GotoXY
mov eax, playerScore
inc eax
call WriteDec
mov playerScore, eax
```

```
pop edx
```

```
foodNotGobbled:
call GotoXY
mov eax, blue + (green * 16)
```

```
call setTextColor
mov al, ''
call WriteChar
mov dh, 24
mov dl, 79
call GotoXY
```

```
RET
```

```
MoveSnake ENDP
```

```
createFood PROC
```

```
;; Na random poziciji se generise hrana, ukoliko je ta pozicija prazna, kako
;; ne bi dolazilo do preklapanja zmijice i hrane prilikom generisanja
```

```
redo:
```

```
mov eax, 24
call RandomRange
mov dh, al
```

```
mov eax, 80
call RandomRange
mov dl, al
```

```
call accessIndex      ;; Sadržaj lokacije se smesta u bx
```

```
cmp bx, 0              ;; I ako nije prazan, loopuje se dok se ne potrefi prazna pozicija
jne redo
```

```
mov foodY, dh
mov foodX, dl
```

```
mov eax, white + (white * 16);
call setTextColor
call GotoXY
mov al, ' '
call WriteChar
```

```
RET
```

```
createFood ENDP
```

```
accessIndex PROC
```

```
;; Procedura pristupa fremjbaferu i vraca vrednost na poziciji Y=dh, X=dl u registar bx
```

```
mov bl, dh    ;; Indeks reda u bl
mov al, 80
mul bl        ;; Indeks reda se mnozi sa 80 kako bi se dobio potreban segment
frejmbafera
push dx
mov dh, 0     ;; U dh ostaje samo indeks kolone
add ax, dx    ;; Dodaje se offset kolone na segment reda kako bi se dobila adresa
piksela
pop dx
mov esi, 0
mov si, ax    ;; Generisana adresa se kopira u indeksni registar
shl si, 1     ;; Koji se siftuje za jedan posto su elementi tipa word

mov bx, frameBuffer[si] ;; U bx se upisuje vrednost celije
```

RET

accessIndex ENDP

saveIndex PROC

;;; Procedura slicna accessIndex, samo sto se sad upisuje vrednost piksela na datoj poziciji

```
push ebx
mov bl, dh
mov al, 80
mul bl
push dx
mov dh, 0
add ax, dx
pop dx
mov esi, 0
mov si, ax
pop ebx
shl si, 1

mov frameBuffer[si], bx
```

RET

saveIndex ENDP

Paint PROC

///
Procedura cita vrednosti iz frejmbafera piksel po piksela i stampa ih u konzolu.

mov eax, blue + (green * 16)

call SetTextColor

mov dh, 0

loop1: ///
Loopuje se kroz redove

cmp dh, 24

JGE endLoop1

mov dl, 0

loop2: ///
Loopuje se kroz kolone

cmp dl, 80

JGE endLoop2

call GOTOXY

mov bl, dh ///
Naredne linije dohvataju vrednost piksela

mov al, 80 ///
iz frejmbafera na zadatim koordinatama (dl, dh)

mul bl

push dx

mov dh, 0

add ax, dx

pop dx

mov esi, 0

mov si, ax

shl si, 1

mov bx, frameBuffer[si]

```
cmp bx, 0          ;// Ako je piksel prazan, ne stampa se nista u konzolu
je NoPrint
```

```
cmp bx, 0FFFFh     ;// Ako je deo zida, skace se na labelu za iscrtavanje zida
je printWall
```

```
mov al, ' '        ;// Inace je deo zmije pa se stampa whitespace u beloј boji
call WriteChar
jmp noPrint
```

```
printWall:         ;// Iscrtava zidove
mov eax, blue + (gray * 16)
call SetTextColor
```

```
mov al, ' '
call WriteChar
mov eax, blue + (green * 16)
call SetTextColor
```

```
NoPrint:
inc dl
jmp loop2          ;// Nastavlja se dalje loopovanje kroz kolone
```

```
endLoop2:
inc dh
jmp loop1          ;// I redove
```

```
endLoop1:
RET
Paint ENDP
END main
```