Projekt

Assembler-Programmierung

.....

TIC-TAC-TOE

Unterlagen für die Abgabe:

- Deckblatt für das Projekt (Format: doc, docx)
- Kurzanleitung zur Bedienung (mögliche Formate: doc, docx, pdf)
- Entwurfsskizze (wenige Seiten) (mögliche Formate doc, docx, pdf)
- Testbeispiele bzw. Testprozeduren (mögliche Formate doc, docx, pdf)
- Projekt-Ordner **Zip-Komprimiert**

Mit Ihrer Unterschrift bestätigen Sie, dass Sie die Datei „AllgemeineHinweise(unbedingt Lesen).docx“ gelesen und verstanden haben.

Bearbeiter					Note
Name	Vorname	Sem. Gr.	Mat. Nr.	Unterschrift	
Kresse	Elmar	19-INB1	75610		

Hinweis zur Unterschrift: Scannen Sie Ihre Unterschrift ein und setzen Sie diese ein.
Bem.: Um Betrug vorzubeugen, sollten Sie in einem Zeichenprogramm einige Pixel gezielt verändern, so dass die gescannte Version von einer Originalunterschrift unterscheidbar ist.

..... Protokoll (wird im Kolloquium ausgefüllt) .....

....

Kurzanleitung:

Das Spiel kann normal in Form einer *.exe aufgeführt werden. Als erstes gelangt man so ins Hauptmenü, wo einem die Optionen Spiel spielen und Verlassen gegeben werden. Das Hauptmenü dient auch gleichzeitig zur Anzeige des Spielverlaufs (wer gewonnen hat oder ob es unentschieden steht). Nachdem man dann mit der Taste 1 auf der Tastatur in ein neues Spiel startet kann man abwechseln mit der Maustaste dann in eines der 9 Felder klicken wobei das Programm immer überprüft ob nach dem Klick jemand gewonnen hat und ob das gewählte Feld frei ist.
Mit ESC kann man ein laufendes Spiel abbrechen und in Hauptmenü gelangen.

Screenshots:

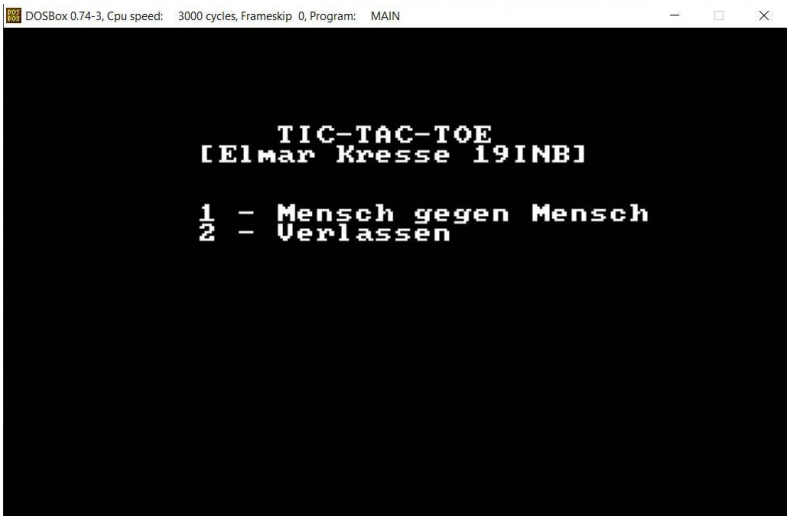


Abbildung 1 Hauptmenü

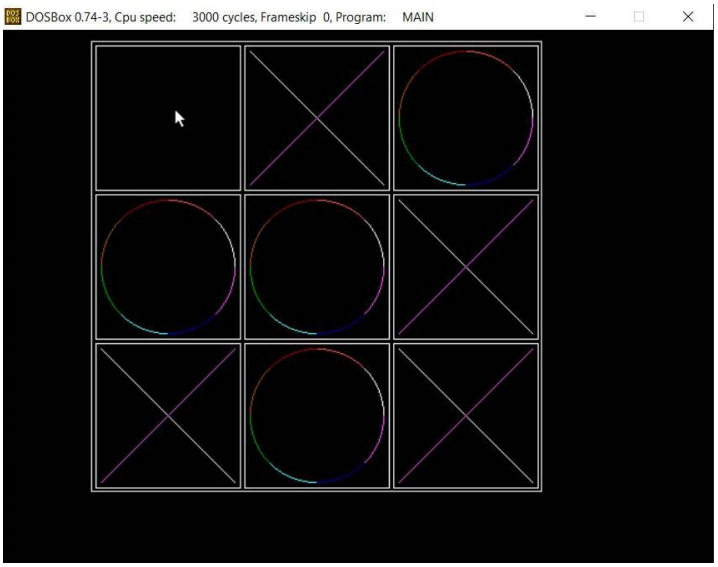


Abbildung 2 Spielfeld

Entwurfsskizzen:

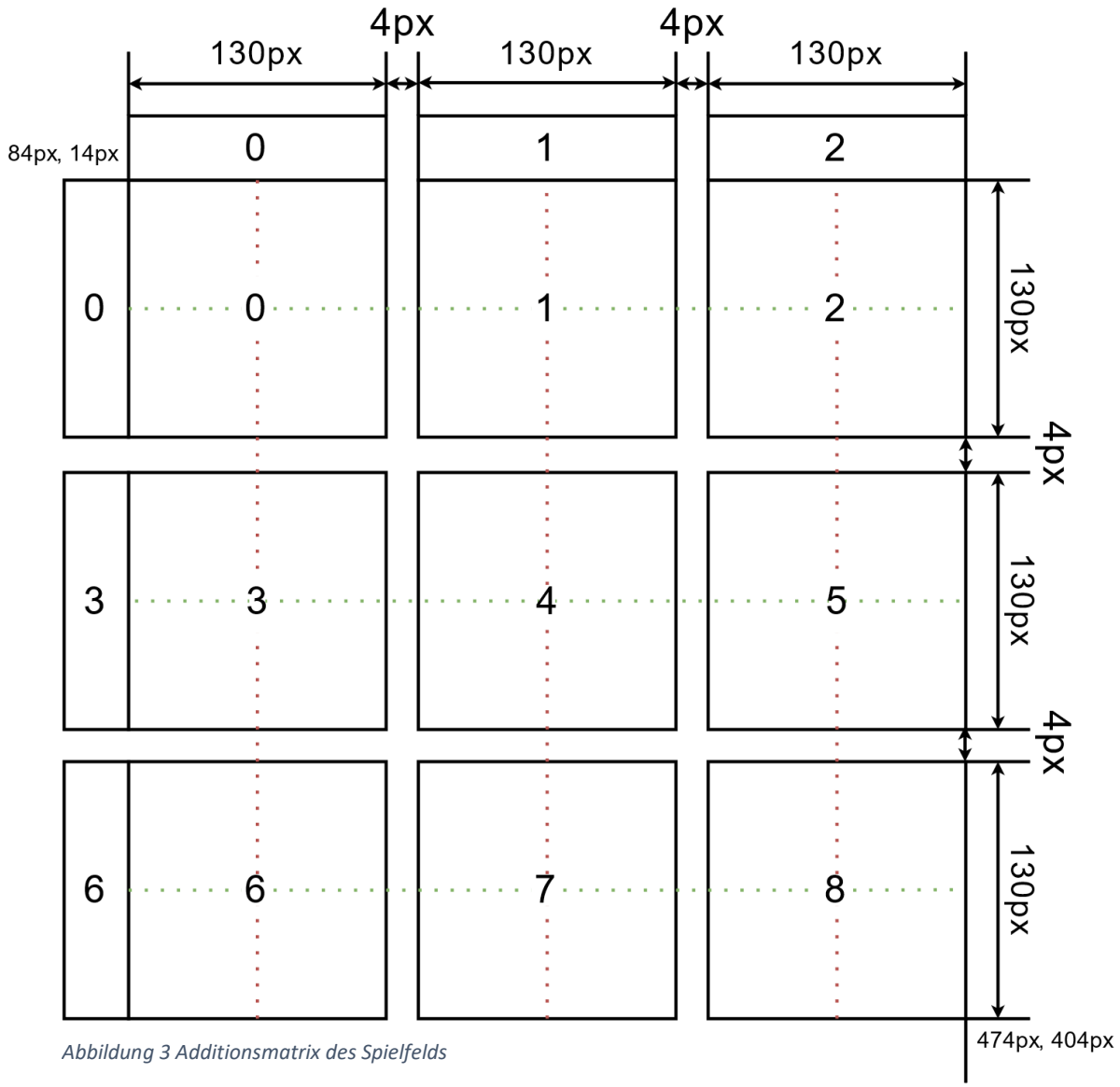


Abbildung 3 Additionsmatrix des Spielfelds

Daraus resultieren 8 Testfälle welche eine Kombination zum Gewinnen ergeben.

Vertikal	Horizontal	Quer
0,3,6 1,4,7 2,5,8	0,1,2 3,4,5 6,7,8	0,4,8 6,4,2

In der folgenden Abbildung habe ich mir zu erst einen groben Plan erstellt welche Schritte das Programm abarbeitet und was speziell zu beachten ist. Wichtig ist hierbei das diese nur eine Skizze / Vorlage ist und nicht alle Elemente im fertigen Projekt vorhanden / umgesetzt sein könnten.

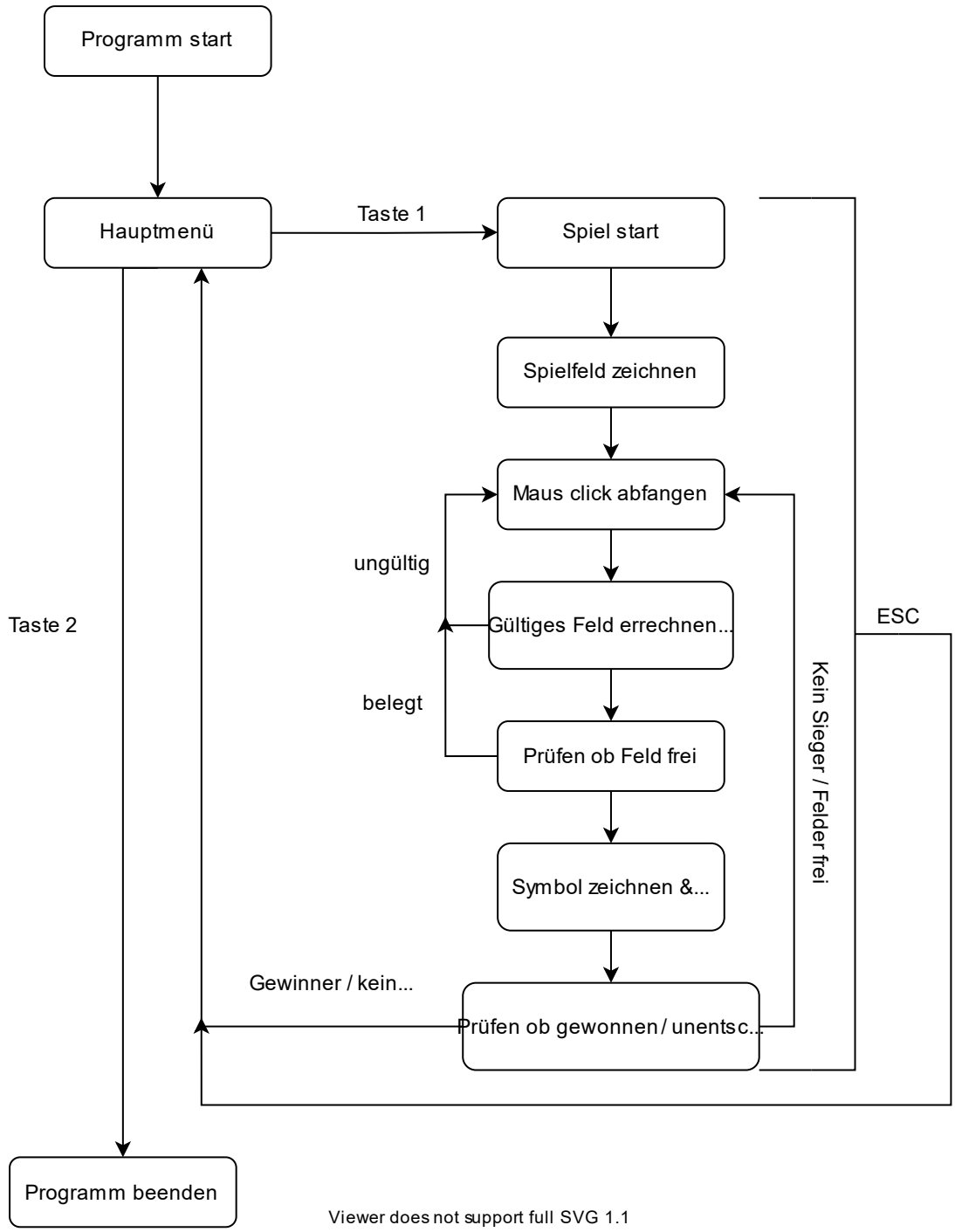


Abbildung 4 Programmablaufplan

Testbeispiele:

DDA Circle Test:
Grundlage: (https://de.wikipedia.org/wiki/Rasterung_von_Kreisen)

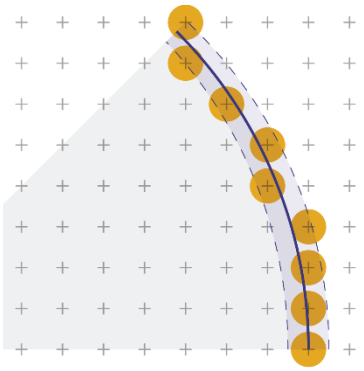
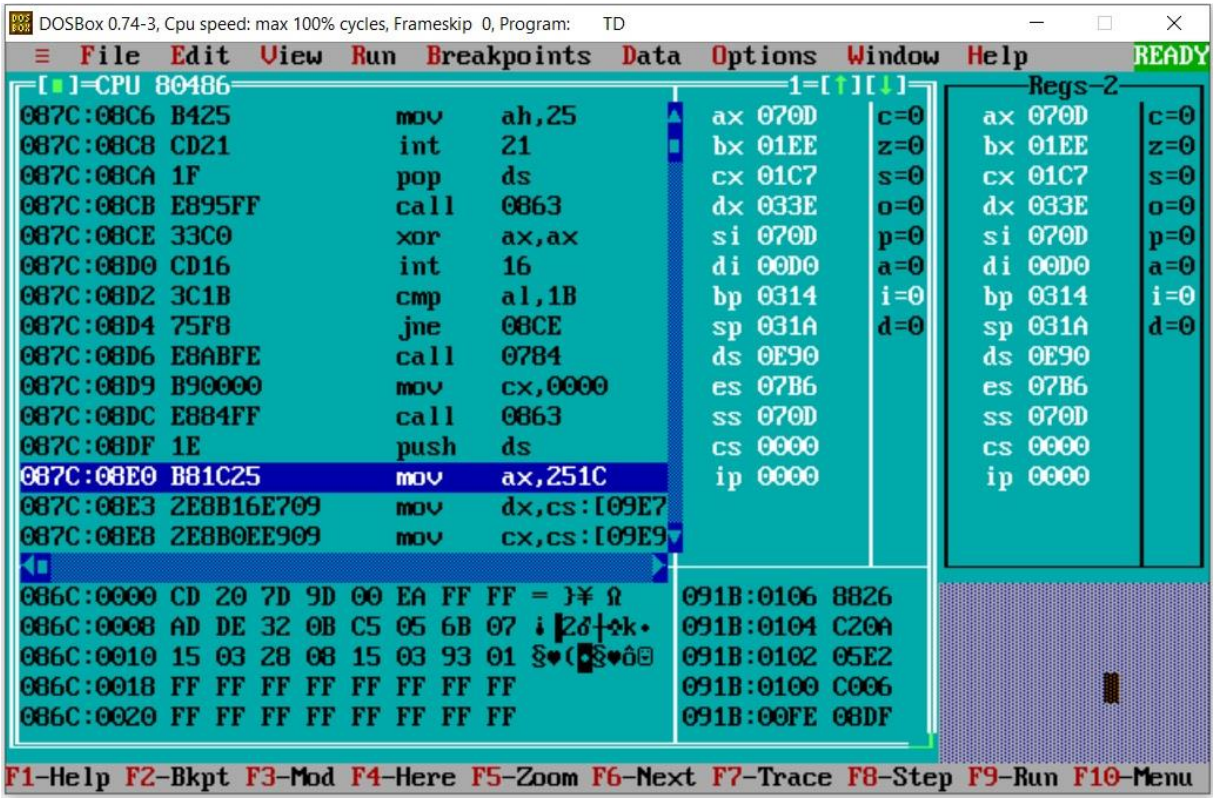


Abbildung 5 Rasterung von Kreisen

Für die Grundlage der Prüfung ist der Kreis in 8 Abschnitte unterteilt und es wurden während der Programmierung diese einzeln im Bild aufgerufen und erstellt. Sodass Schrittweise die Berechnung und Ausgabe kontrolliert wurde. Außerdem wird durch eine funktionale Einteilung in PROCs und MACROs auch die Schrittweise Funktionalität kontrolliert und garantiert.

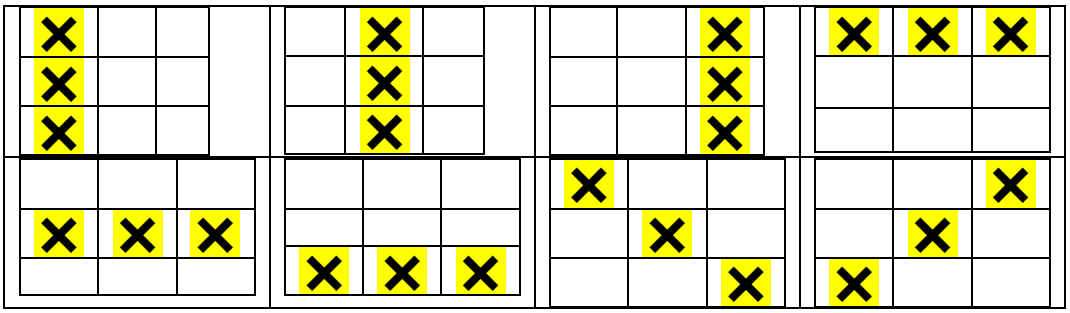
Gamelogic Test:

Die Spiellogik wurde insofern getestet und überprüft das mit dem Turbo Debugger Breakpoints gesetzt wurden und während der Laufzeit immer wieder nach Eingaben die Register und Variablen überprüft wurden auf erwartete Werte. Wie 0,1 in den Spieler Arrays und korrekte Werte für die Vergleiche.



Interface Test:
Beim Start des Programms wurden die Menü Tasten (1,2) und die Endloop für das Spiel getestet welche mit int16 auf eine Tastatureingabe wartet und mit ESC dann zum Anfang zurückspringt.

Spiele Tests:
Für das klicken der Reihenfolge pro 3 in einer Reihe gibt es 6 Möglichkeiten und um eine Runde TICTACTOE zu gewinnen muss man eine der 8 Varianten belegen.



Jede dieser 8 Varianten wurden manuell für Kreuz und Kreis getestet.
Zudem wurde das Spiel durch andere Testpersonen (Familie) getestet sodass dieser Bereich größtmöglich abgedeckt wurde.
Zusätzlich wurde das Programm mit einem 32Bit Windows XP Rechner getestet und auf einem Linux Rechner mit einer DOSBox. Wobei unter beiden Systemen keine Änderungen festgestellt werden konnten die dem Spielfluss und der Programmausführung Probleme bereiten sollten.

```
;**************************************
;   Elmar Kresse 19-INB-1 (2021)
;   main.asm
;   file contains source code for the
;   main operations
;
;**************************************

.model small
.STACK 100h
.486
esc_code = 1Bh              ;esc key
video_seg = 0B800h

.data
INCLUDE data.asm


.code
INCLUDE draw.asm
INCLUDE game.asmn
INCLUDE sound.asm
INCLUDE logic.asm

ISR1Ch: push ds
        push ax
        mov ax, @DATA
        mov ds, ax
        pop ax
        pop ds
        iret

;start of the main program
;-----------------------------------------------------------------------------------------
Beginn:

        mov ax, @DATA
        mov ds, ax
        mov ax, 3
        int 10h
        ;read old VT entry
        mov al, 1Ch
        mov ah, 35h
        int 21h
        ;es:bx address old ISR
        ;save
        mov oldIOFF, bx
        mov oldISeg, es
        ;store new ISR
        push ds ; store ds
        push cs
        pop ds  ; ds <- cs
        mov dx, OFFSET ISR1Ch ; address in ds:dx
        mov al, 1Ch ; <vn>
        mov ah, 25h
        int 21h
        pop ds  ; restore ds


        call start_programm

;-----------------------------------------------------------------------------------------

;Endlosschleife
;-----------------------------------------------------------------------------------------
endlloop:
        xor ax, ax          ;delete ah
        int 16h             ;Keyboard query, waits for any key is pressed
```

```
69          cmp al, esc_code    ;If esc is not pressed, the program remains in the
            endless loop
70          jne short endlloop  ;jmp not equal
71          call reset_timerisr
72          mov cx, 0
73          call start_programm
74
75  ;-------------------------------------------------------------------------------
    ------------------
76
77  ;End of the program and return to DOS
78  ;-------------------------------------------------------------------------------
    ----------------
79  ende:
80
81      PUSH DS
82      MOV AX, 251Ch               ;Restore old user timer interrupt vector
83      MOV DX, [CS:old_timer]
84      MOV CX, [CS:old_segment]
85      MOV DS, CX
86      INT 21h
87      POP DS
88
89      mov ax, 0                   ;reset mouse programm
90      int 33h                     ;mouse interrupt
91
92      MOV ax, 3                   ;Reset screen mode
93      INT 10h
94
95      mov dx, oldIOFF             ;order is important
96      mov ax, oldISeg
97      mov ds, ax
98      mov al, 1Ch
99      mov ah, 25h
100     int 21h
101
102     mov ah, 4Ch                 ;Backflow to DOS
103     int 21h
104     .stack 100h
105     end Beginn
106 ;-------------------------------------------------------------------------------
    --------------
107
108
109
110
111
112
113 ; Convert X: Y coordinates to fields
114 ; Check whether the field is free
115 ; if free field
116 ; test whether the player has won
117
118 ;   1  2  3      0  1  2
119 ; 1|2  3  4    0|0  1  2
120 ; 4|5  6  7    3|3  4  5
121 ; 7|8  9  10   6|6  7  8
122
123 ;8 test cases
124 ; 0,3,6
125 ; 1,4,7
126 ; 2,5,8
127
128 ; 0,1,2
129 ; 3,4,5
130 ; 6,7,8
131
132 ; 0,4,8
133 ; 6,4,2
```

```
;**************************************
;   Elmar Kresse 19-INB-1 (2021)
;   data.asm
;   file contains variables and constant
;   for music, drawing, text and game
;
;**************************************


;Sounds
placeSound DD 4070, 100000 ; Note with pause
placeSoundLength = $ - placeSound

gameOverSound DD 4560, 100000, 4304, 50000, 7239, 100000, 9121, 200000
gameOverSoundLength = $ - gameOverSound


;coordinates and size
x_wert DW ?                 ;16 Bit variable
y_wert DW ?                 ;16 Bit variable
d_wert DW ?                 ;16 Bit variable


spieler DW ?    ;current player (0,1)
field DW ?      ;selected field (0-8)
balance dw 0    ;balance for check won (3 = 3 in a row)


;Player Arrays
player1 DW 0,0,0,0,0,0,0,0,0
player2 DW 0,0,0,0,0,0,0,0,0

;main menue strings
title_str DB "TIC-TAC-TOE$"
signature_str DB "[Elmar Kresse 19INB]$"
choose1_str DB "1 - Mensch gegen Mensch$"
choose2_str DB "2 - Verlassen$"
cross_win DB  "Kreuz hat gewonnen$"
circle_win DB "Kreis hat gewonnen$"
draw DB "Unentschieden$"

;reset ISR var
oldIOFF DW ?
oldISeg DW ?

;timer vector vars
old_timer DW 0          ;original int 1c vector
old_segment DW 0        ;original int 1c segment
timer_ticks DW 0        ;timer_ticks
```

```
;**************************************
;   Elmar Kresse 19-INB-1 (2021)
;   logic.asm
;   file contains source code for the
;   logic (mouse, timer, programm)
;
;       - handle_timer_tick PROC
;       - prepare_timer_tick PROC
;       - prepare_mouse PROC
;       - mouseProc PROC FAR
;       - start_programm PROC
;**************************************

handle_timer_tick PROC
                                    ; Call  the original interrupt vector
                                    ; by pushing flags register and doing a FAR
                                    CALL to old vector
    PUSHF
    CALL DWORD ptr [CS:old_timer]
    INC WORD ptr [CS:timer_ticks]        ; Increase timer counter

    IRET
handle_timer_tick ENDP

prepare_timer_tick PROC

                PUSH DS
                MOV AX, 351Ch           ; Get user timer interrupt vector
                INT 21h
                                        ; ES:BX now has the address of the interrupt
                                        vector
                MOV [CS:old_timer], BX
                MOV AX, ES
                MOV [CS:old_segment], AX

                MOV AX, 251ch           ; Set user timer interrupt vector
                PUSH CS
                POP DS
                MOV DX, OFFSET handle_timer_tick
                INT 21h

                POP ds

    ret
prepare_timer_tick ENDP

prepare_mouse PROC

            mov dx, OFFSET mouseProc     ;jump to click_event_handler
            mov ax, 0Ch
            mov cx, 00000010b           ; on left click
            push cs                     ;es:dx
            pop es
            int 33h

            mov ax, 1                   ;show cursor
            int 33h

    ret
prepare_mouse ENDP

mouseProc PROC FAR                  ;other doc segment therefore FAR
            enter 0, 0
            push ax
            push bx
            push cx
            push dx
            push es
            push ds

            mov ax, 2                   ;hide cursor
```

```
            int 33h

                                        ;dx contains coordinates of "rows"
            mov y_wert, dx              ;save y coordinate in variable
                                        ;cx contains coordinates of "columns"
            mov x_wert, cx              ;save y coordinate in variable

            call game_logic

            xor bx, bx                  ;button pressed clear result

            mov ax, 1                   ;show cursor
            int 33h

            pop ds
            pop es
            pop dx
            pop cx
            pop bx
            pop ax
            leave
            ret
mouseProc    ENDP

start_programm PROC

            mov AX, video_seg
            mov es, AX


            mov ax, @DATA
            mov ds, ax


            mov ax, 12h
            int 10h

            mov dx, 03C4h           ;address of the index port (share)
            mov al, 2               ;index of control register
            out dx, al              ;Set index: output out :: dx to port al(2)
            inc dx                  ;index of dataport
            mov al, 0Fh             ;value (1 = bit plane enabled, here all) fi
            out dx, al


        call prepare_timer_tick

        mov ax,0Dh
        int 10h

        call paint_menu

        check_mode_loop:
        xor ax, ax                  ;delete ah
        int 16h                     ;keyboard query

        cmp al,'1'
        je short playgame

        cmp al,'2'
        je ende

        jmp check_mode_loop

    playgame:
        call start_game


ret
start_programm ENDP

```

```
;*************************************
;   Elmar Kresse 19-INB-1 (2021)
;   game.asm
;   file contains source code for the
;   basic functions of the game logic
;
;       - conkf MACRO x_kord, y_kord
;       - conkf_prog PROC FAR
;       - game_logic PROC
;       - check_won_value MACRO a, b, c
;       - check_draw PROC
;       - reset_player_data PROC
;       - check_won PROC
;       - reset_timerisr PROC
;       - start_game PROC
;*************************************

conkf MACRO x_kord, y_kord ;MACRO for coordinates to field
        push dx
        push cx


        mov cx, x_kord
        mov dx, y_kord
        mov field, 0h
        call conkf_prog

        pop cx
        pop dx
    ENDM

conkf_prog PROC FAR     ;PROC for coordinates to field
push ax
push bx
push cx
push dx
        ;horizontal x
        cmp cx, 482
        jg out_of_range
        cmp cx, 352
        mov x_wert, 357
        jnl x_3
        cmp cx, 218
        mov x_wert, 223
        jnl x_2
        cmp cx, 84
        mov x_wert, 89
        jnl x_1
        jmp out_of_range

    x_3:
        mov ax, 2
        jmp y_kordi
    x_2:
        mov ax, 1
        jmp y_kordi
    x_1:
        mov ax, 0

y_kordi:
        ;vertical y
        cmp dx, 412
        jg out_of_range
        cmp dx, 282
        mov y_wert, 287
        jnl y_3
        cmp dx, 148
        mov y_wert, 153
        jnl y_2
        cmp dx, 14
        mov y_wert, 19
```

```
72          jnl y_1
73          jmp out_of_range
74
75      y_3:
76          mov bx, 6
77          jmp calc_field
78      y_2:
79          mov bx, 3
80          jmp calc_field
81      y_1:
82          mov bx, 0
83
84      calc_field:
85          add ax, bx          ;(A = ax) * (B = bx) = (goal = AX)
86          mov field, ax
87          jmp kord_final
88
89      out_of_range:
90          mov field, 10       ;not in range(0-8) 10 is set for later checks
91
92      kord_final:
93
94
95      pop dx
96      pop cx
97      pop bx
98      pop ax
99
100
101
102         ret
103 conkf_prog ENDP
104
105 game_logic PROC
106
107 conkf x_wert, y_wert
108
109 ;check valid field 0-8
110 cmp field, 0ah
111 je unknown
112
113 ;Feld frei
114     push ax
115     push si
116     mov si, field
117     shl si, 1
118     cmp player1[si], 1
119     pop si
120     pop ax
121     je unknown
122
123     push ax
124     push si
125     mov si, field
126     shl si, 1
127     cmp player2[si], 1
128     pop si
129     pop ax
130     je unknown
131
132
133
134 mov ax, spieler
135 cmp ax, 0
136 je  kreuz_spieler
137 jne kreis_spieler
138
139 kreuz_spieler:
140             cross_draw x_wert, y_wert, 120
141
142
```

```
143             ;Sound
144             PUSH AX BX CX DX
145             MOV BX, OFFSET placeSound
146             MOV CX, OFFSET placeSoundLength
147             CALL playbeep
148             POP DX CX BX AX
149
150             mov spieler, 1  ;Spieler umschalten
151
152                 ;Geklicketes Feld belegen
153                 push ax
154                 push si
155                 mov si, field
156                 shl si, 1
157                 mov player1[si], 1
158                 pop si
159                 pop ax
160
161             push cx
162             mov cx, offset player1
163             call check_won
164             pop cx
165             cmp ax, 1
166             jne unknown
167             mov cx, 2
168             call reset_timerisr
169             call start_programm
170
171 kreis_spieler:
172             ;Kreis zeichnen
173             circle_draw x_wert, y_wert, 60
174
175
176             ;Sound
177             PUSH AX BX CX DX
178             MOV BX, OFFSET placeSound
179             MOV CX, OFFSET placeSoundLength
180             CALL playbeep
181             POP DX CX BX AX
182
183             mov spieler, 0      ;Spieler umschalten
184
185                 ;Geklicketes Feld belegen
186                 push ax
187                 push si
188                 mov si, field
189                 shl si, 1
190                 mov player2[si], 1
191                 pop si
192                 pop ax
193
194             push cx
195             mov cx, offset player2
196             call check_won
197             pop cx
198
199             cmp ax, 1
200             jne unknown
201             mov cx, 3
202             call reset_timerisr
203             call start_programm
204
205 unknown:
206
207 call check_draw
208 ret
209 game_logic ENDP
210
211 check_won_value MACRO a, b, c
212 push ax
213 push bx
```

```
214                 mov bx, 0
215                 mov ax, a;
216                 shl ax, 1
217                 add bp, ax
218                 add bx, ds:[bp]
219                 mov bp, cx
220                 mov ax, b;
221                 shl ax, 1
222                 add bp, ax
223                 add bx, ds:[bp]
224                 mov bp, cx
225                 mov ax, c;
226                 shl ax, 1
227                 add bp, ax
228                 add bx, ds:[bp]
229                 mov bp, cx
230                 cmp bx, 3
231     pop bx
232     pop ax
233                 je got_winner
234     ENDM
235
236     check_draw PROC
237
238         ;Überprüfen ob Unentschieden
239                     push bx ;Counter ob alle Felder belegt sind
240                     push ax ;Zahlvariable für Shiften
241                     push cx ;Pointer für das Array des jeweiligen Spielers
242                     push dx ;Zählvariable für Schleifen
243                     push bp
244                     mov bx, 0
245                     mov ax, 0
246                     mov dx, 0
247
248                     mov cx, offset player1 ;pointer for player1
249
250                 player1_field_loop:
251                     mov ax, dx
252                     shl ax, 1
253                     mov bp, cx
254                     add bp, ax
255                     add bx, ds:[bp]
256                     inc dx
257                     cmp dx, 9
258                     jne player1_field_loop
259
260                     mov cx, offset player2  ;pointer for player2
261                     mov dx, 0
262                 player2_field_loop:
263                     mov ax, dx
264                     shl ax, 1
265                     mov bp, cx
266                     add bp, ax
267                     add bx, ds:[bp]
268                     inc dx
269                     cmp dx, 9
270                     jne player2_field_loop
271
272
273                     cmp bx, 9
274
275                     pop bp
276                     pop dx
277                     pop cx
278                     pop ax
279                     pop bx
280
281                     je draw_end
282                     jmp check_draw_end
283
284     draw_end:
```

```
285             mov cx, 4 ; 4 = Unentschieden
286             call reset_timerisr
287             call start_programm
288
289     check_draw_end:
290     ret
291     check_draw ENDP
292
293     reset_player_data PROC
294
295                     push ax
296                     push si
297                     push bx ;Schleifen Zähler
298                     mov bx, 0
299
300     resetloop_player1:
301                     mov si, bx
302                     shl si, 1
303                     mov player1[si], 0
304                     inc bx
305                     cmp bx, 9
306                     jne resetloop_player1
307
308                     mov bx, 0
309     resetloop_player2:
310                     mov si, bx
311                     shl si, 1
312                     mov player2[si], 0
313                     inc bx
314                     cmp bx, 9
315                     jne resetloop_player2
316
317                     pop bx
318                     pop si
319                     pop ax
320     ret
321     reset_player_data ENDP
322
323     check_won PROC
324     push bp
325     mov bp, cx
326
327     ;vertical
328             check_won_value 0, 3, 6
329             check_won_value 1, 4, 7
330             check_won_value 2, 5, 8
331
332     ;horizontal
333
334             check_won_value 2, 1, 0
335             check_won_value 3, 4, 5
336             check_won_value 6, 7, 8
337
338     ;cross
339             check_won_value 0, 4, 8
340             check_won_value 6, 4, 2
341             jmp ende_check_won
342
343     got_winner:
344     pop bp
345
346     ;play gameover sound
347     MOV BX, OFFSET gameOverSound
348     MOV CX, gameOverSoundLength
349     CALL playbeep
350
351     mov ax, 1 ;var for save winner exists
352     ret
353
354     ende_check_won:
355     pop bp
```

```
356   mov ax, 0
357   ret
358   check_won ENDP
359
360   reset_timerisr PROC
361               PUSH CX
362               PUSH DS
363               PUSH AX
364               PUSH DX
365               MOV AX, 251Ch           ; Restore old user timer interrupt vector
366               MOV DX, [CS:old_timer]
367               MOV CX, [CS:old_segment]
368               MOV DS, CX
369               INT 21h
370               POP DX
371               POP AX
372               POP DS
373               POP CX
374       ret
375   reset_timerisr ENDP
376
377   start_game PROC
378           CLI
379
380           mov ax,00h
381           int 33h
382           or ax,ax
383
384           call reset_player_data ;Spieler Array reset
385           mov ax,12h          ;Videomodus
386           int 10h
387
388           call prepare_display
389
390           mov ax, 1       ;Curser an
391           int 33h
392
393           ;Ersten Spieler setzten
394
395           call prepare_mouse
396           mov spieler, 0
397           call prepare_display
398
399           STI
400           ret
401   start_game endp
```

```
;**************************************
;   Elmar Kresse 19-INB-1 (2021)
;   draw.asm
;   file contains source code for
;   drawing and maths for circle
;
;       - draw_pixel MACRO x, y, c
;       - circle_draw macro x, y, radius
;           Reference Digital differential analyzer (graphics algorithm)
;
https://en.wikipedia.org/wiki/Digital_differential_analyzer_(graphics_algorithm)
;
;       - cross_draw MACRO x, y, d
;       - rectangle_draw MACRO x, y, d
;       - line_print_down PROC
;       - line_print_straight PROC
;       - paint_menu PROC FAR
;       - prepare_display PROC
;**************************************

draw_pixel MACRO x, y, c
            push ax
            push bx
            push cx
            push dx
            MOV AH, 0Ch     ;setzen der Konfig fürs schreiben
            MOV AL, c     ; Farbe
            MOV BH, 00h     ;Set Seitennummber
            MOV CX, x
            MOV DX, y
            INT 10h
            pop dx
            pop cx
            pop bx
            pop ax
    ENDM

circle_draw macro x, y, radius
    push dx
    push cx
    push bx
    push ax

    ;Werte zurücksetzen
    mov balance, 0
    mov ax, 0
    mov bx, 0

    ;Verschiebung Mitte
    add x, 60
    add y, 60

    mov bx, radius
    mov balance, radius
    neg balance


    zeichne_kreis_schleife:

    ;0 to 45
    mov cx, x
    add cx, bx
    mov dx, y
    sub dx, ax
    draw_pixel cx, dx, 0Fh
    ;90 to 45
    mov cx, x
    add cx, ax
    mov dx, y
    sub dx, bx
    draw_pixel cx, dx, 0Ch
```

```
71        ;90 to 135
72        mov cx, x
73        sub cx, ax
74        mov dx, y
75        sub dx, bx
76        draw_pixel cx, dx, 04h
77        ;180 to 135
78        mov cx, x
79        sub cx, bx
80        mov dx, y
81        sub dx, ax
82        draw_pixel cx, dx, 06h
83        ;180 to 225
84        mov cx, x
85        sub cx, bx
86        mov dx, y
87        add dx, ax
88        draw_pixel cx, dx, 12h
89        ;270 to 225
90        mov cx, x
91        sub cx, ax
92        mov dx, y
93        add dx, bx
94        draw_pixel cx, dx, 0Bh
95        ;270 to 315
96        mov cx, x
97        add cx, ax
98        mov dx, y
99        add dx, bx
100       draw_pixel cx, dx, 01h
101       ;360 to 315
102       mov cx, x
103       add cx, bx
104       mov dx, y
105       add dx, ax
106       draw_pixel cx, dx, 0Dh
107
108
109       push dx
110       mov dx, balance
111       add dx, ax
112       add dx, ax
113       mov balance, dx
114       pop dx
115
116
117       cmp balance, 0
118       jl balance_negative
119       ;balance_positive:
120       dec bx
121
122       push dx
123       mov dx, balance
124       sub dx, bx
125       sub dx, bx
126       mov balance, dx
127       pop dx
128
129
130       balance_negative:
131       inc ax
132
133       cmp ax, bx
134       jg end_drawing
135       jmp zeichne_kreis_schleife
136
137
138       end_drawing:
139       ; pause the screen for dos compatibility:
140
141       pop ax
```

```
142         pop bx
143         pop cx
144         pop dx
145
146     endm
147
148     cross_draw MACRO x, y, d
149             push ax
150             push bx
151             push cx
152             push dx
153
154
155             ;Erste FOR Schleife nach rechts unten
156             mov cx, x
157             mov dx, y
158
159             xor cx,cx   ; cx-register is the counter, setn to 0
160         loop1:
161             push cx
162             mov cx, cx
163             mov dx, y
164             add dx, cx
165             add cx, x
166             draw_pixel cx, dx, 0Fh
167             pop cx
168             inc cx
169             cmp cx, d  ; cx vergleich mit der Größe (Dimension d)
170             jle loop1   ; Loop while less or equal
171
172
173             xor cx,cx   ; cx-register is the counter, setn to 0
174         loop2:
175             push cx
176             mov cx, cx
177             mov dx, y
178             sub dx, cx
179             add cx, x
180             add dx, d
181             draw_pixel cx, dx, 0Dh
182             pop cx
183             inc cx
184             cmp cx, d  ; cx vergleich mit der Größe (Dimension d)
185             jle loop2   ; Loop while less or equal
186
187             pop ax
188             pop bx
189             pop cx
190             pop dx
191
192         ENDM
193
194     rectangle_draw MACRO x, y, d
195             ;Verwaltet move und call
196             mov cx, x
197             mov dx, y
198             mov x_wert, cx
199             mov y_wert, dx
200             mov d_wert, d
201             call line_print_down
202             call line_print_straight
203         ENDM
204
205     line_print_down PROC
206         xor cx,cx   ; cx-register is the counter, setn to 0
207         loop3:
208             push cx
209             mov cx, cx
210             mov dx, y_wert
211             add cx, x_wert
212             draw_pixel cx, dx, 0Fh
```

```
213             add dx, d_wert
214             draw_pixel cx, dx, 0Fh
215             pop cx
216             inc cx
217             cmp cx, d_wert  ; cx vergleich mit der Größe (Dimension d)
218             jle loop3   ; Loop while less or equal
219                 ret
220     line_print_down endp
221
222     line_print_straight PROC
223             xor dx,dx   ; cx-register is the counter, setn to 0
224         loop4:
225             push dx
226             mov dx, dx
227             mov cx, x_wert
228             add dx, y_wert
229             draw_pixel cx, dx, 0Fh
230             add cx, d_wert
231             draw_pixel cx, dx, 0Fh
232             pop dx
233             inc dx
234             cmp dx, d_wert  ; cx vergleich mit der Größe (Dimension d)
235             jle loop4   ; Loop while less or equal
236             ret
237     line_print_straight endp
238
239     paint_menu PROC FAR
240
241             push ax
242             push bx
243             push dx
244             ;set pointer
245             mov ah,2
246             mov dh,5
247             mov dl,14
248             mov bh,0
249             int 10h
250             mov dx,offset title_str
251             mov ah,9
252             int 21h
253             ;set pointer
254             mov ah,2
255             mov dh,6
256             mov dl,10
257             mov bh,0
258             int 10h
259             mov dx,offset signature_str
260             mov ah,9
261             int 21h
262
263
264             ;set pointer
265             mov ah,2
266             mov dh,7
267             mov dl,11
268             mov bh,0
269             int 10h
270
271             cmp cx, 2
272             mov dx,offset cross_win
273             je zeige_gewinner
274
275
276             cmp cx, 3
277             mov dx,offset circle_win
278             je zeige_gewinner
279
280
281             cmp cx, 4
282             jne kein_gewinner
283
```

```
284         mov ah,2
285         mov dh,7
286         mov dl,13
287         mov bh,0
288         int 10h
289         mov dx,offset draw
290         je zeige_gewinner
291
292         zeige_gewinner:
293         mov ah,9
294         int 21h
295
296         kein_gewinner:
297
298
299         ;set pointer
300         mov ah,2
301         mov dh,9
302         mov dl,10
303         mov bh,0
304         int 10h
305         mov dx,offset choose1_str
306         mov ah,9
307         int 21h
308
309         ;set pointer
310         mov ah,2
311         mov dh,10
312         mov dl,10
313         mov bh,0
314         int 10h
315         mov dx,offset choose2_str
316         mov ah,9
317         int 21h
318
319         pop ax
320         pop bx
321         pop dx
322
323         ret
324 paint_menu endp
325
326 prepare_display PROC
327         ;Videomode clear alles entfernen
328         ;Spielfeld initialisieren
329         rectangle_draw 80, 10, 405 ; Kreuz MACRO
330         rectangle_draw 84, 14, 130 ; 1
331         rectangle_draw 218, 14, 130 ; 2
332         rectangle_draw 352, 14, 130 ; 3
333
334         rectangle_draw 84, 148, 130 ; 4
335         rectangle_draw 218, 148, 130 ; 5
336         rectangle_draw 352, 148, 130 ; 6
337
338         rectangle_draw 84, 282, 130 ; 7
339         rectangle_draw 218, 282, 130 ; 8
340         rectangle_draw 352, 282, 130 ; 9
341         ret
342 prepare_display ENDP
343
344 ;---------------------------------------- DRAW ENDE
    ----------------------------------------
```

```
;*************************************
;   Elmar Kresse 19-INB-1 (2021)
;   sound.asm
;   file contains source code for
;   only playing sounds
;
;       - playbeep PROC
;source:
http://muruganad.com/8086/8086-assembly-language-program-to-play-sound-using-pc-speaker.html
;*************************************


playbeep PROC
       MOV AL, 182
       OUT 43h, AL
   beep:

       ;set frequency
          MOV AX, [BX]
          OUT 42h, AL
          MOV AL, AH
          OUT 42h, AL
       ;sound on
          IN AL, 61h
          OR AL, 00000011b
          OUT 61h, AL

      ADD BX, 4
      MOV WORD ptr [CS:timer_ticks], 0

      ;break
          timer_loop:
          CMP WORD ptr [CS:timer_ticks], 3
          jne timer_loop
          MOV WORD ptr [CS:timer_ticks], 0

       ;sound off
          IN AL, 61h
          AND AL, 11111100b
          OUT 61h, AL

        ADD BX, 4
        SUB CX, 8
        JNE beep
        ret
playbeep ENDP

```