```
        [org 0x0100]

jmp menu

; 0
ZERO:
    db  00111100b
    db  01100110b
    db  01101110b
    db  01110110b
    db  01100110b
    db  01100110b
    db  00111100b
    db  00000000b

; 1
ONE:
    db  00011000b
    db  00111000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  01111110b
    db  00000000b

; 2
TWO:
    db  00111100b
    db  01100110b
    db  00000110b
    db  00001100b
    db  00110000b
    db  01100000b
    db  01111110b
    db  00000000b

; 3
THREE:
    db  00111100b
    db  01100110b
    db  00000110b
    db  00111100b
    db  00000110b
    db  01100110b
    db  00111100b
    db  00000000b

; 4
FOUR:
    db  00001100b
    db  00011100b
    db  00111100b
    db  01101100b
    db  01111110b
    db  00001100b
    db  00001100b
    db  00000000b

; 5
FIVE:
    db  01111110b
    db  01100000b
    db  01111100b
    db  00000110b
    db  00000110b
    db  01100110b
    db  00111100b
    db  00000000b

; 6
SIX:
    db  00111100b
    db  01100000b
    db  01111100b
    db  01100110b
    db  01100110b
    db  01100110b
    db  00111100b
    db  00000000b

; 7
SEVEN:
    db  01111110b
    db  01100110b
    db  00000110b
    db  00001100b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00000000b

; 8
EIGHT:
    db  00111100b
    db  01100110b
    db  01100110b
    db  00111100b
    db  01100110b
    db  01100110b
    db  00111100b
    db  00000000b

; 9
NINE:
    db  00111100b
    db  01100110b
    db  01100110b
    db  00111110b
    db  00000110b
    db  01100110b
    db  00111100b
    db  00000000b

; Font data (only for letter M)
A:
    db  00111100b   ; A
    db  01100110b
    db  01100110b
    db  01111110b
    db  01100110b
    db  01100110b
    db  01100110b
    db  00000000b

B:
    db  11111100b
    db  11000110b
    db  11000110b
    db  11111100b
    db  11000110b
    db  11000110b
    db  11111100b
    db  00000000b

C:
    db  00111100b
    db  01100110b
    db  11000000b
    db  11000000b
    db  11000000b
    db  01100110b
    db  00111100b
    db  00000000b

D:
    db  11111000b
    db  11001100b
    db  11000110b
    db  11000110b
    db  11000110b
    db  11001100b
    db  11111000b
    db  00000000b

E:
    db  11111110b
    db  11000000b
    db  11000000b
    db  11111100b
    db  11000000b
    db  11000000b
    db  11111110b
    db  00000000b

F:
    db  11111110b
    db  11000000b
    db  11000000b
    db  11111100b
    db  11000000b
    db  11000000b
    db  11000000b
    db  00000000b

G:
    db  00111100b
    db  01100110b
    db  11000000b
    db  11001110b
    db  11000110b
    db  01100110b
    db  00111110b
    db  00000000b

H:
    db  11000110b
    db  11000110b
    db  11000110b
    db  11111110b
    db  11000110b
    db  11000110b
    db  11000110b
    db  00000000b

I:
    db  00111100b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00111100b
    db  00000000b

J:
    db  00011110b
    db  00001100b
    db  00001100b
    db  00001100b
    db  11001100b
    db  11001100b
    db  01111000b
    db  00000000b

K:
    db  11000110b
    db  11001100b
    db  11011000b
    db  11110000b
    db  11011000b
    db  11001100b
    db  11000110b
    db  00000000b
```

```
L:
    db  11000000b
    db  11000000b
    db  11000000b
    db  11000000b
    db  11000000b
    db  11000000b
    db  11111110b
    db  00000000b

M:
    db  11000011b
    db  11100111b
    db  11111111b
    db  11011011b
    db  11000011b
    db  11000011b
    db  11000011b
    db  00000000b

N:
    db  11000011b
    db  11100011b
    db  11110011b
    db  11011011b
    db  11001111b
    db  11000111b
    db  11000011b
    db  00000000b

O:
    db  00111100b
    db  01100110b
    db  11000011b
    db  11000011b
    db  11000011b
    db  01100110b
    db  00111100b
    db  00000000b

P:
    db  11111100b
    db  11000110b
    db  11000110b
    db  11111100b
    db  11000000b
    db  11000000b
    db  11000000b
    db  00000000b

Q:
    db  00111100b
    db  01100110b
    db  11000011b
    db  11000011b
    db  11001011b
    db  01100110b
    db  00111110b
    db  00000000b

R:
    db  11111100b
    db  11000110b
    db  11000110b
    db  11111100b
    db  11011000b
    db  11001100b
    db  11000110b
    db  00000000b

S:
    db  00111100b
    db  01100110b
    db  01100000b
    db  00111100b
    db  00000110b
    db  01100110b
    db  00111100b
    db  00000000b

T:
    db  11111111b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00011000b
    db  00000000b

U:
    db  11000011b
    db  11000011b
    db  11000011b
    db  11000011b
    db 11000011b
    db 11000011b
    db 01111110b
    db 00000000b

V:
    db 11000011b
    db 11000011b
    db 11000011b
    db 11000011b
    db 01100110b
    db 00111100b
    db 00011000b
    db 00000000b

W:
    db 11000011b
    db 11000011b
    db 11000011b
    db 11000011b
    db 11011011b
    db 11111111b
    db 01100110b
    db 00000000b

X:
    db 11000011b
    db 01100110b
    db 00111100b
    db 00011000b
    db 00111100b
    db 01100110b
```

```
   db  11000011b
   db  00000000b

Y:
   db  11000110b
   db  01100110b
   db  00111100b
   db  00011000b
   db  00011000b
   db  00011000b
   db  00011000b
   db  00000000b

Z:
   db  11111111b
   db  00000110b
   db  00001100b
   db  00011000b
   db  00110000b
   db  01100000b
   db  11111111b
   db  00000000b

Arrow_left:
            db          00010000b
            db  00011000b
            db  11100100b
            db          11100010b
            db  11100100b
            db  00011000b
            db  00010000b
   db  00000000b
```

```
Player_shape:
            db          00011000b
            db  00011000b
            db  00111100b
            db          01011010b
            db  00011000b
            db  00100100b
            db  01000010b
   db  00000000b

declare_pie:
      db 0,0,0,0,0,0,0,0,0,0
               db 0,0,0,0,0,0,0,0,0,0
               db 0,0,43,43,43,43,43,0,0,0
               db 0,0,0,43,0,43,0,0,0,0
               db 0,0,0,43,0,43,0,0,0,0
               db 0,0,0,43,0,43,0,0,0,0
               db 0,0,0,43,0,43,0,0,0,0
               db 0,0,0,43,0,43,43,0,0,0
               db 0,0,0,0,0,0,0,0,0,0
               db 0,0,0,0,0,0,0,0,0,0

declare_enemy:
      db 0,0,0,0,0,0,0,0,0,0
               db 0,0,0,40,40,40,0,0,0,0
               db 0,0,0,40,0,0,40,0,0,0
               db 0,0,0,40,40,40,0,0,0,0
               db 0,0,0,0, 40,0,0,0,0,0
               db 0,0,0, 40,0,40,0,0,0,0
               db 0,0,0,40,0,0,40,0,0,0
               db 0,0,0,0,0,40,0,0,0,0
```

```
                db 0,0,0,0,40,0,40,0,0,0
                db 0,0,0,40,0,0,0,40,0,0

declare_reward:
    db 0,0,0,0,43,43,0,0,0,0
                db 0,0,0,43,43,43,43,0,0,0
                db 0,0,43,0,0,43,43,43,0,0
                db 0,43,43,43,43,43,43,43,43,0
                db 0,43,43,43,43,43,43,43,43,0
                db 0,0,43,43,43,43,43,43,0,0
                db 0,0,0,43,43,43,43,0,0,0
                db 0,0,0,0,43,43,0,0,0,0
                db 0,0,0,0,0,0,0,0,0,0
                db 0,0,0,0,0,0,0,0,0,0

declare_gate:
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0
                db 0,0,026,0,0,026,0,0,026,0


Arrow_left_pos: dw 0

;Game Initialization
```

```
maze1 :
            db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
            db 1, 2, 0, 0, 1, 4, 0, 1, 0, 1, 0, 1, 3, 0, 0, 1, 0, 4, 0, 1
            db 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1
            db 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
            db 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1
            db 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1
            db 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1
            db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 3, 1
            db 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
            db 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
            db 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
            db 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1
            db 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1
            db 1, 0, 0, 4, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
            db 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1
            db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
            db 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
            db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 1
            db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
            db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1

maze2 :
  db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
  db 1, 2, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1
  db 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1
  db 1, 0, 0, 0, 1, 0, 0, 4, 1, 0, 0, 0, 1, 0, 0, 3, 0, 1, 0, 1
  db 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1
  db 1, 0, 0, 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1
  db 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1
  db 1, 0, 0, 3, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1
  db 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1
```

```
    db 1, 4, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1
    db 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1
    db 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1
    db 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1
    db 1, 0, 0, 0, 1, 0, 0, 4, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1
    db 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1
    db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1
    db 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1
    db 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1
    db 1, 1, 0, 1, 0, 0, 0, 6, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    db 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

maze3:
    db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    db 1, 2, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 3, 0, 1
    db 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 4, 1, 1, 1, 0, 1, 0, 1
    db 1, 0, 1, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1
    db 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1
    db 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1
    db 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1
    db 1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
    db 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    db 1, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
    db 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
    db 1, 0, 0, 0, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
    db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1
    db 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 4, 1
    db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1
    db 1, 0, 0, 0, 0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
    db 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 1
    db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1
```

```
  db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1

maze4:
        db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
        db 1, 2, 0, 0, 1, 3, 1, 1, 1, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 1
        db 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1
        db 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1
        db 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1
        db 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1
        db 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1
        db 1, 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 4, 1, 0, 1, 0, 1, 0, 1
        db 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1
        db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 3, 1
        db 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1
        db 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
        db 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
        db 1, 0, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1
        db 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1
        db 1, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1
        db 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
        db 1, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1
        db 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1
        db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1

maze5:
        db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
        db 1, 2, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 4, 1, 1, 1
        db 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1
        db 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1
        db 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 3, 0, 0, 1
        db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1
        db 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1
        db 1, 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 1
        db 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1
        db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1
        db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1
        db 1, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1
        db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
        db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
        db 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1
        db 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1, 0, 0, 0, 1, 0, 1
        db 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1
        db 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1
        db 1, 1, 1, 0, 0, 6, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1
        db 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1



timer_msg db 'TIME: ', 0
length9: dw 6
tickcount : dw 0
seconds : dw 0
time_exceeded db 'Time Exceeded!', 0
time_buffer: db '00', 0
length10 : dw 14
original_timer_offset : dw 0
original_timer_segment : dw 0
time_limit : dw 60

player_row : dw 0
player_col : dw 0
prev_row : dw 0
prev_col : dw 0
```

```
current_row : dw 0
current_col : dw 0
width : dw 20
size : dw 400
score_text db 'SCORE: '
score : dw 0
score_buffer : db '000', 0

lost_text : db 'You have lost!'
lost_length : dw 14

win_text : db 'Victory!!!'
win_length : dw 10

reward_count: dw 0
super_count: dw 0



menu:
  mov ax, 0013h
  int 10h

        call clearscr_menu

;left side
        push word 5
        push word 5
        push word 8
        push word 30
        call draw_block

        push word 5
        push word 35
        push word 8
        push word 30
        call draw_block

        push word 5
        push word 65
        push word 8
        push word 60
        call draw_block

;Right side
        push word 150
        push word 5
        push word 8
        push word 55
        call draw_block

        push word 140
        push word 30
        push word 20
        push word 5
        call draw_block

        push word 140
        push word 52
        push word 20
        push word 5
        call draw_block

        push word 150
        push word 38
        push word 20
        push word 5
        call draw_block

        push word 0
        push word 25
        push word 60
        push word 10
        call draw_block

        push word 15
        push word 13
        push word 20
        push word 3
        call draw_block

        push word 23        ;working
        push word 60
        push word 9
        push word 14
        call draw_block

        push word 23
        push word 66
        push word 20
        push word 5
        call draw_block

        push word 25
        push word 18
        push word 10
        push word 15
```

```
        call draw_block

        push word 0
        push word 55
        push word 55
        push word 10
        call draw_block

        push word 150
        push word 40
        push word 8
        push word 30
        call draw_block

        push word 150
        push word 65
        push word 8
        push word 60
        call draw_block

;up side
        push word 25
        push word 0
        push word 10
        push word 10
        call draw_block

        push word 110
        push word 0
        push word 10
        push word 10
        call draw_block
        push word 90
        push word 0
        push word 10
        push word 20
        call draw_block

        push word 5
        push word 5
        push word 50
        push word 8
        call draw_block

        push word 70
        push word 0
        push word 10
        push word 10
        call draw_block

        push word 55
        push word 5
        push word 40
        push word 8
        call draw_block

        push word 110
        push word 5
        push word 80
        push word 8
        call draw_block

;down side
```

```
push word 18
push word 90
push word 50
push word 8
call draw_block

push word 70
push word 90
push word 30
push word 8
call draw_block

push word 100
push word 90
push word 60
push word 8
call draw_block

push word 60
push word 90
push word 10
push word 20
call draw_block

push word 140
push word 70
push word 5
push word 30
call draw_block

push word 120
push word 83
```

```
            push word 45
            push word 5
            call draw_block

;Draw characters

            mov cx, 116       ; X position
    mov dx, 100       ; Y position
    mov bl, 92      ; Color
    mov si, Arrow_left
    call plot_char
            call delay

    mov cx, 100       ; X position
    mov dx, 80       ; Y position
    mov bl, 92      ; Color
    mov si, M
    call plot_char
            call delay

            mov cx, 109       ; X position
    mov dx, 80       ; Y position
    mov bl, 92      ; Color
    mov si, A
    call plot_char
            call delay

            mov cx, 118       ; X position
    mov dx, 80       ; Y position
    mov bl, 92      ; Color
    mov si, Z
    call plot_char
```

```
        call delay

mov cx, 127     ; X position
mov dx, 80      ; Y position
mov bl, 92      ; Color
mov si, E
call plot_char
        call delay

mov cx, 141     ; X position
mov dx, 80      ; Y position
mov bl, 92     ; Color
mov si, R
call plot_char
        call delay

        mov cx, 150     ; X position
mov dx, 80      ; Y position
mov bl, 92      ; Color
mov si, U
call plot_char
        call delay

        mov cx, 159     ; X position
mov dx, 80      ; Y position
mov bl, 92      ; Color
mov si, N
call plot_char
        call delay

        mov cx, 168     ; X position
mov dx, 80      ; Y position
```

```
    mov bl, 92      ; Color
    mov si, N
    call plot_char
            call delay

            mov cx, 177       ; X position
    mov dx, 80       ; Y position
    mov bl, 92      ; Color
    mov si, E
    call plot_char
            call delay

            mov cx, 186       ; X position
    mov dx, 80       ; Y position
    mov bl, 92     ; Color
    mov si, R
    call plot_char
            call delay

;Draw Play button
            mov cx, 125       ; X position
    mov dx, 100       ; Y position
    mov bl, 92      ; Color
    mov si, P
    call plot_char
            call delay

            mov cx, 134       ; X position
    mov dx, 100       ; Y position
    mov bl, 92      ; Color
    mov si, L
    call plot_char
```

```
            call delay

            mov cx, 143       ; X position
    mov dx, 100       ; Y position
    mov bl, 92      ; Color
    mov si, A
    call plot_char
            call delay

            mov cx, 152       ; X position
    mov dx, 100       ; Y position
    mov bl, 92      ; Color
    mov si, Y
    call plot_char
            call delay

;Draw Exit button
            mov cx, 125       ; X position
    mov dx, 115       ; Y position
    mov bl, 92      ; Color
    mov si, E
    call plot_char
            call delay

            mov cx, 134       ; X position
    mov dx, 115       ; Y position
    mov bl, 92      ; Color
    mov si, X
    call plot_char
            call delay

            mov cx, 143       ; X position
    mov dx, 115       ; Y position
    mov bl, 92     ; Color
    mov si, I
    call plot_char
            call delay

            mov cx, 152       ; X position
    mov dx, 115       ; Y position
    mov bl, 92      ; Color
    mov si, T
    call plot_char
            call delay

menu_loop:
            mov ah, 0x01; BIOS check for keystroke
    int 0x16
    jz menu_loop; Loop until a key is pressed

; Read the key and update position
            mov ah, 0x00; BIOS wait for key press
            int 0x16
            cmp al, 'w'; Check for up
            je move_up_menu
            cmp al, 's'; Check for down
            je move_down_menu
            jmp menu_loop; Ignore other keys

move_down_menu:
    mov cx, 1
            cmp word cx, [Arrow_left_pos]
            jz menu_loop
            mov cx, 116       ; X position
    mov dx, 100       ; Y position
    mov bl, 19      ; Color
    mov si, Arrow_left
            call plot_char

            mov cx, 116       ; X position
    mov dx, 115       ; Y position
    mov bl, 92      ; Color
    mov si, Arrow_left
            call plot_char

            mov word [Arrow_left_pos], 1

            mov ah, 0x00; BIOS wait for key press
            int 0x16

            cmp al, 0x0D
            jz start

            jmp menu_loop

move_up_menu:
    mov cx, 0
            cmp word cx, [Arrow_left_pos]
            jz menu_loop

            mov cx, 116       ; X position
    mov dx, 115       ; Y position
    mov bl, 19      ; Color
    mov si, Arrow_left
            call plot_char
```

```
        mov cx, 116      ; X position
    mov dx, 100      ; Y position
    mov bl, 92    ; Color
    mov si, Arrow_left
        call plot_char

        mov word [Arrow_left_pos], 0

        mov ah, 0x00; BIOS wait for key press
        int 0x16

        cmp al, 0x0D
        jz start
        jmp menu_loop




start :
    cmp word [Arrow_left_pos], 1
        jz near Print_credits
        ;Random maze
        push ax
        call Random
        pop ax

        cmp ax, 0
        jz near draw_maze1
        cmp ax, 1
        jz near draw_maze2
        cmp ax, 2
```

```
            jz near draw_maze3
            cmp ax, 3
            jz near draw_maze4
    cmp ax, 4
            jz near draw_maze5

draw_maze1:
    call clearscr
    mov ax, maze1
    push maze1
            mov bx, maze1           ; Player position in maze
            jmp generate

draw_maze2:
            call clearscr
            mov ax, maze2
            push maze2
            mov bx, maze2
            jmp generate

draw_maze3:
            call clearscr
            mov ax, maze3
            push maze3
            mov bx, maze3
            jmp generate

draw_maze4:
            call clearscr
            mov ax, maze4
            push maze4
            mov bx, maze4
            jmp generate

draw_maze5:
            call clearscr
            mov ax, maze5
            push maze5
            mov bx, maze5
            jmp generate

generate:
   call gen_maze
   call display_score_text
   call display_score

   ; Initialize the player position
   mov word[player_row], 5
   mov word[player_col], 5
   mov word[prev_row], 5
   mov word[prev_col], 5
   push word 0x03
   call draw_player

   call hook

            mov si, 21        ;player position

   jmp game_loop

Random:
   push bp
   mov bp,sp;
   push cx
```

```
    push ax
    push dx;
    mov ah, 00h ; interrupts to get system time
    int 1ah ; CX:DX now hold number of clock ticks since midnight
    mov ax, dx
    xor dx, dx
    mov cx, 5 ; Dividing the remaing number by number of mazes
    div cx
    mov [bp + 4], dx; ; Passing it into the output variable
    pop cx;
    pop ax;
    pop dx;
        pop bp
        ret

print_time_msg:
        pusha
        push es
        push ds
        mov ah, 0x13; BIOS interrupt to write string
        mov al, 1
        mov bh, 0
        mov bl, 0x07; (white)
        mov dx, 320 * 5 + 236 + 240; Position
        mov cx, [length9] ; Length
        push cs
        pop es
        mov bp, timer_msg
        int 0x10
        pop ds
        pop es
        popa
        ret

print_time_score:
        pusha
        push es
        push ds
        mov ah, 0x13; BIOS interrupt to write string
        mov al, 1
        mov bh, 0
        mov bl, 0x07; (white)
        mov dx, 320 * 5 + 236 + 245; Position
        mov cx, 2 ; Length
        push cs
        pop es
        mov bp, time_buffer
        int 0x10
        pop ds
        pop es
        popa
        ret
        ; Timer debounce subroutine
        debounce :
        push cx
        mov cx, 6000H; Delay loop count
        L1 :
        loop L1
        pop cx
        ret

; Print number to screen
printnum :
        push bp
        mov bp, sp
        push es
        push ax
        push bx
        push cx
        push dx
        push di
        push si
        mov ax,0xA000
        mov es,ax
        mov ax, [bp + 4]
        mov di, [bp + 4]
        mov bx, 10
        mov cx, 0

nextdigit1:
        mov dx, 0
        div bx
        add dl, 0x30
        push dx
        inc cx
        cmp ax, 0
        jnz nextdigit1
        mov si, time_buffer

nextpos1:
        cmp di, 10
        jl lesser
        pop dx
        mov [si], dl
        add si, 1
        loop nextpos1
        jmp exit_print_num
        lesser:
        pop dx
        mov byte [si], 0x30
        mov [si + 1], dl
        exit_print_num:
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        pop es
        pop bp
        ret 2

; Timer handler
timer :
        push ax
        push cx
        push dx

        call debounce

        inc word[cs:tickcount]
        cmp word[cs:tickcount], 18; Assuming ~18 ticks = 1 second(based on ~18.2 Hz timer)
        jb skip_increment

        mov word[cs:tickcount], 0
        inc word[cs:seconds]
        call print_time_score
        cmp word[cs:seconds], 61; Check if seconds reached 60
        jb print_time
        je skip_increment           ;un
        jmp timer


        print_time :
        ; Print seconds
        push word[cs:seconds]
        call printnum

        skip_increment :
        mov al, 0x20
        out 0x20, al
        cmp word[cs:seconds], 61
        je call_time_exceeded_message
        jnz exit_timer

        call_time_exceeded_message :
        call time_exceeded_message

        exit_timer :
        pop dx
        pop cx
        pop ax
        iret




; Hook timer interrupt to custom handler
hook :

        pusha
```

```
            push es
            call print_time_msg
            xor ax, ax
            mov es, ax
            mov ax, [es:8 * 4]
            mov[original_timer_offset], ax
            mov ax, [es:8 * 4 + 2]
            mov[original_timer_segment], ax
            cli
            mov word[es:8 * 4], timer
            mov[es:8 * 4 + 2], cs
            sti

            pop es
            popa
            ret

; Time exceeded message
time_exceeded_message :
            push ax
            push bx
            push cx
            push dx

            mov ah, 0x13
            mov bh, 0
            mov bl, 7; Color for message
            mov dx, 320 * 7 + 225 + 240; Display position for exceeded message
            mov cx, [length10]; Length of message
            push cs
            pop es
            mov bp, time_exceeded
```

```
        int 0x10; Display message

        pop dx
        pop cx
        pop bx
        pop ax


        ret


game_loop :
        ; Check for keyboard input to control movement
        mov ah, 0x01; BIOS check for keystroke
        int 0x16
        jz game_loop; Loop until a key is pressed

        ; Store previous position
        mov ax, [player_row]
        mov[prev_row], ax
        mov ax, [player_col]
        mov[prev_col], ax
        ; Read the key and update position
        mov ah, 0x00; BIOS wait for key press
        int 0x16
        cmp al, 'w'; Check for up
        je move_up
        cmp al, 's'; Check for down
        je move_down
        cmp al, 'a'; Check for left
        je move_left
        cmp al, 'd'; Check for right
        je move_right
        jmp game_loop; Ignore other keys

move_up :
        mov cl, [bx + si - 20]
        cmp cl, 1
        jnz check_for3up
        jmp game_loop

move_down :
        mov cl, [bx + si + 20]
        cmp cl, 1
        jnz check_for3down
        jmp game_loop

move_left :
        mov cl, [bx + si - 1]
        cmp cl, 1
        jnz near check_for3left
        jmp game_loop

move_right :
        mov cl, [bx + si + 1]
        cmp cl, 1
        jnz near check_for3right
        jmp game_loop


check_for3up :
        cmp cl, 3; Reward found
        jnz check_for4up
        inc word [reward_count]
        inc word [super_count]


        add word[score], 20
        mov byte[bx + si - 20], 0


        cmp word [super_count], 4       ;superman mode
        jz near timer_inc


        call display_score
        jmp validate_up


check_for4up :
        cmp cl, 4; Enemy found score decreases
        jnz check_for5up
        sub word[score], 15
        mov ax, [score]
        cmp ax, 0
        jl near Print_Lost; We should clearscr and print "Game Lost!"
        call display_score
        jmp game_loop


check_for5up :
        cmp cl, 5
        jz near Print_win; We need to add A string "Player Won"


validate_up:
        sub word[player_row], 5; Move player up
        sub si, 20
        call update_position; Update only player position


        jmp game_loop
```

```
check_for3down :
        cmp cl, 3; Reward found
        jnz check_for4down
        inc word [reward_count]
        inc word [super_count]

        add word[score], 20
        mov byte[bx + si + 20], 0

        cmp word [super_count], 4       ;superman mode
        jz near timer_inc

        call display_score
        jmp validate_down

check_for4down :
        cmp cl, 4; Enemy found score decreases
        jnz check_for5down
        sub word[score], 15
        mov ax, [score]
        cmp ax, 0
        jl near Print_Lost; We should clearscr and print "Game Lost!"
        call display_score
        jmp game_loop

check_for5down :
        cmp cl, 5
        jz near Print_win; We need to add A string "Player Won"

validate_down:
        add word[player_row], 5; Move player down
        add si, 20
        call update_position; Update only player position

        jmp game_loop

check_for3left :
        cmp cl, 3; Reward found
        jnz check_for4left
        inc word [reward_count]
        inc word [super_count]

        add word[score], 20
        mov byte[bx + si - 1], 0

        cmp word [super_count], 4        ;superman mode
        jz near timer_inc

        call display_score
        jmp validate_left

check_for4left :
        cmp cl, 4; Enemy found score decreases
        jnz check_for5left
        sub word[score], 15
        mov ax, [score]
        cmp ax, 0
        jl near Print_Lost; We should clearscr and print "Game Lost!"
        call display_score
        jmp game_loop

check_for5left :
        cmp cl, 5
        jz near Print_win; We need to add A string "Player Won"
```

```
validate_left:
        sub word[player_col], 5; Move player up
        sub si, 1
        call update_position; Update only player position

        jmp game_loop

check_for3right :
        cmp cl, 3; Reward found
        jnz check_for4right
        inc word [reward_count]
        inc word [super_count]

        add word[score], 20
        mov byte[bx + si + 1], 0

        cmp word [super_count], 4       ;superman mode
        jz near timer_inc

        call display_score
        jmp validate_right

check_for4right :
        cmp cl, 4; Enemy found score decreases
        jnz check_for5right
        sub word[score], 15
        mov ax, [score]
        cmp ax, 0
        jl Print_Lost; We should clearscr and print "Game Lost!"
        call display_score
        jmp game_loop
```

```
check_for5right :
        cmp cl, 5
        jz Print_win; We need to add A string "Player Won"

check_for6right:
        cmp cl, 6
        jz open_gate
        jnz validate_right

open_gate:
        cmp word [reward_count], 2
        jb near game_loop

validate_right:
        add word[player_col], 5; Move player up
        add si, 1
        call update_position; Update only player position

        jmp game_loop

timer_inc:
        sub word [seconds], 10
        mov word [super_count], 0
        call display_score
        jmp near game_loop

Print_win :
        mov ah, 0x13
        mov al, 1
        mov bh, 0
        mov bl, 7
        mov dx, 0xA562; 160th column and 100th row
        mov cx, [win_length]
        push cs
        pop es
        mov bp, win_text
        int 0x10

        mov ax, 60
        sub word ax, [seconds]
        add word [score], ax            ;Inc score to the time remaining

        call display_score
        jmp end_game

        ; Player has lost the game
Print_Lost :
        mov ah, 0x13
        mov al, 1
        mov bh, 0
        mov bl, 7
        mov dx, 0xA562; 160th column and 100th row
        mov cx, [lost_length]
        push cs
        pop es
        mov bp, lost_text
        int 0x10

        jmp end_game

Print_credits:
        call clearscr_menu
```

```
          mov cx, 130       ; X position
mov dx, 60       ; Y position
mov bl, 64      ; Color
mov si, A
call plot_char
          call delay

          mov cx, 139       ; X position
mov dx, 60       ; Y position
mov bl, 64      ; Color
mov si, B
call plot_char
          call delay

          mov cx, 148       ; X position
mov dx, 60       ; Y position
mov bl, 64      ; Color
mov si, D
call plot_char
          call delay

          mov cx, 157       ; X position
mov dx, 60       ; Y position
mov bl, 64      ; Color
mov si, U
call plot_char
          call delay

          mov cx, 166       ; X position
mov dx, 60       ; Y position
mov bl, 64      ; Color
mov si, L
```

```
call plot_char
        call delay

        mov cx, 175      ; X position
mov dx, 60      ; Y position
mov bl, 64      ; Color
mov si, L
call plot_char
        call delay

        mov cx, 184      ; X position
mov dx, 60      ; Y position
mov bl, 64      ; Color
mov si, A
call plot_char
        call delay

        mov cx, 193      ; X position
mov dx, 60      ; Y position
mov bl, 64      ; Color
mov si, H
call plot_char
        call delay

        mov cx, 130      ; X position
mov dx, 70      ; Y position
mov bl, 64      ; Color
mov si, TWO
call plot_char
        call delay

        mov cx, 139      ; X position
mov dx, 70      ; Y position
mov bl, 64      ; Color
mov si, THREE
call plot_char
        call delay

        mov cx, 148     ; X position
mov dx, 70      ; Y position
mov bl, 64      ; Color
mov si, L
call plot_char
        call delay



        mov cx, 166     ; X position
mov dx, 70      ; Y position
mov bl, 64      ; Color
mov si, ZERO
call plot_char
        call delay

        mov cx, 175     ; X position
mov dx, 70      ; Y position
mov bl, 64      ; Color
mov si, FIVE
call plot_char
        call delay

        mov cx, 184     ; X position
mov dx, 70      ; Y position
mov bl, 64      ; Color
mov si, ZERO
```

```
    call plot_char
            call delay

            mov cx, 193       ; X position
    mov dx, 70       ; Y position
    mov bl, 64      ; Color
    mov si, FIVE
    call plot_char
            call delay

;Rubab
            mov cx, 130       ; X position
    mov dx, 90       ; Y position
    mov bl, 64      ; Color
    mov si, U
    call plot_char
            call delay

            mov cx, 139       ; X position
    mov dx, 90       ; Y position
    mov bl, 64      ; Color
    mov si, M
    call plot_char
            call delay

            mov cx, 148       ; X position
    mov dx, 90       ; Y position
    mov bl, 64      ; Color
    mov si, E
    call plot_char
            call delay
```

```
        mov cx, 166      ; X position
mov dx, 90      ; Y position
mov bl, 64     ; Color
mov si, R
call plot_char
        call delay

        mov cx, 175      ; X position
mov dx, 90      ; Y position
mov bl, 64     ; Color
mov si, U
call plot_char
        call delay

        mov cx, 184      ; X position
mov dx, 90      ; Y position
mov bl, 64     ; Color
mov si, B
call plot_char
        call delay

        mov cx, 193      ; X position
mov dx, 90      ; Y position
mov bl, 64     ; Color
mov si, A
call plot_char
        call delay

        mov cx, 202      ; X position
mov dx, 90      ; Y position
mov bl, 64     ; Color
mov si, B
```

```
call plot_char
        call delay

        mov cx, 130       ; X position
mov dx, 100       ; Y position
mov bl, 64      ; Color
mov si, TWO
call plot_char
        call delay

        mov cx, 139       ; X position
mov dx, 100       ; Y position
mov bl, 64      ; Color
mov si, THREE
call plot_char
        call delay

        mov cx, 148       ; X position
mov dx, 100       ; Y position
mov bl, 64      ; Color
mov si, L
call plot_char
        call delay


        mov cx, 166       ; X position
mov dx, 100       ; Y position
mov bl, 64      ; Color
mov si, ZERO
call plot_char
        call delay
```

```
            mov cx, 175       ; X position
    mov dx, 100       ; Y position
    mov bl, 64      ; Color
    mov si, NINE
    call plot_char
            call delay

            mov cx, 184       ; X position
    mov dx, 100       ; Y position
    mov bl, 64      ; Color
    mov si, TWO
    call plot_char
            call delay

            mov cx, 193       ; X position
    mov dx, 100       ; Y position
    mov bl, 64      ; Color
    mov si, EIGHT
    call plot_char
            call delay

end_game:

            mov ax, 0x4c00
            int 0x21

update_position :
    pusha
    ; Clear the previous position
    push word 0x00
    call clear_previous_position
```

```
; Draw the player at the new position
push word 0x03
call draw_player
popa
ret


clear_previous_position :
                        push bp
                        mov bp, sp
                        push es
                        push ax
                        push bx
                        push cx
                        push si
                        push di

                        mov ax, 0xA000
                        mov es, ax
                        mov ax, 320
                        mul word[prev_row]
                        add ax, [prev_col]
                        shl ax, 1
                        mov di, ax
                        mov cx, 0
      outer2:
               inc cx
               cmp cx, 9
               jg end_block_player1
               mov si, di
               mov bx, 1
```

```
                    innerloop2 :
                            mov ax, [bp + 4]
                            mov[es:si], ax
                            inc bx
                            add si, 1
                            cmp bx, 9
                            jle innerloop2
                            add di, 320
                            jmp outer1
      end_block_player1 :
                    pop di
                    pop si
                    pop cx
                    pop bx
                    pop ax
                    pop es
                    pop bp
                    ret 2

    draw_player:                            ;Stoppppp here
            push bp
            mov bp, sp
            push es
            push ax
            push bx
            push cx
            push si
            push di

            mov ax, 0xA000
            mov es, ax
            mov ax, 320
```

```
mul word[player_row]
add ax, [player_col]
shl ax, 1
mov di, ax
mov cx, 0
outer1:
        inc cx
        cmp cx, 9
        jg end_block_player
        mov si, di
        mov bx, 1

        innerloop1 :
                mov ax, [bp + 4]
                mov[es:si], ax
                inc bx
                add si, 1
                cmp bx, 9
                jle innerloop1
                add di, 320
                jmp outer1
        end_block_player :
                pop di
                pop si
                pop cx
                pop bx
                pop ax
                pop es
                pop bp
                ret 2
```

```
clearscr :
    pusha
    mov ax, 0x0013
    int 0x10
    mov ax, 0xA000
    mov es, ax
    xor di, di
    mov al, 19
    mov cx, 64000
    rep stosb
    popa
    ret
draw_characters:
    push bp
            mov bp, sp
            push es
            pusha
            mov ax, 0xA000
            mov es, ax
            mov ax, 320
            mul word [current_row]
            add ax, [current_col]
            shl ax, 1
            mov di, ax
            mov dx, di

            mov bx, [bp + 4]
            mov cx, 0
            mov si, 0
            mov ax, 0

            outer_char:
```

```
            mov al, [bx + si]
            mov [es:di], al
            inc si
            inc cx
            cmp si, 100
            jz near end
            cmp cx, 10
            jnz iter_char
            add dx, 320
            mov di, dx
            mov cx, 0
            jmp outer_char
iter_char :
            inc di
            jmp outer_char

end_char :
            popa
            pop es
            pop bp
            ret 2

gen_maze :
            push bp
            mov bp, sp
            push es
            pusha

            mov ax, 0xA000
            mov es, ax
            xor di, di
```

```
        mov bx, [bp + 4]
        mov cx, 0
        mov si, 0
        mov ax, 0
Check:
        mov al, 1
        cmp[bx + si], al
        jz wall
        mov al, 2
        cmp[bx + si], al
        jz player
        mov al, 3
        cmp[bx + si], al
        jz reward
        mov al, 4
        cmp[bx + si], al
        jz enemy
        mov al, 5
        cmp[bx + si], al
        jz exit_point
        mov al, 6
        cmp [bx+si], al
        jz gate


        push word 10; size
        push word 0x0000
        call block
        jmp next

wall :
        push word 9; size
```

```
            push word 0x6
            call block
            jmp next

player :
            push word 9; size
            push word 0x3
            call block
            jmp next

enemy :
            mov dx, declare_enemy
            push dx
            call draw_characters
            jmp next

reward :
            mov dx, declare_reward
            push dx
            call draw_characters
            jmp next

gate:
            mov dx, declare_gate
            push dx
            call draw_characters
            jmp next

exit_point :
            mov dx, declare_pie
            push dx
            call draw_characters
            jmp next
next :
            inc si
            inc cx
            cmp si, [size]
            jz end
            cmp cx, [width]
            jnz iter
            add word[current_row], 5
            mov word[current_col], 0
            mov cx, 0
            jmp Check
iter :
            add word[current_col], 5
            jmp Check

end :
            popa
            pop es
            pop bp
            ret 2

block :
   push bp
   mov bp, sp
   push es
   push ax
   push bx
   push cx
   push si
   push di
```

```
mov ax, 0xA000
mov es, ax
mov ax, 320
mul word[current_row]
add ax, [current_col]
shl ax, 1
mov di, ax
mov cx, 0
outer:
            inc cx
            cmp cx, [bp + 6]
            jg end_block
            mov si, di
            mov bx, 1

            innerloop :
                    mov ax, [bp + 4]
                    mov[es:si], ax
                    inc bx
                    add si, 1
                    cmp bx, [bp + 6]
                    jle innerloop
                    add di, 320
                    jmp outer
end_block :
            pop di
            pop si
            pop cx
            pop bx
            pop ax
            pop es
            pop bp
```

```
                ret 4

display_score_text:
        pusha
        push es
        push ds
        mov ah, 0x13; BIOS interrupt to write string
        mov al, 1
        mov bh, 0
        mov bl, 0x07; (white)
        mov dx, 320 * 4 + 228; Position
        mov cx, 7; Length
        push cs
        pop es
        mov bp, score_text
        int 0x10
        pop ds
        pop es
        popa
        ret
display_score :
        pusha
        push es
        push ds
        call convert_to_ascii
        mov ah, 0x13; BIOS interrupt to write string
        mov al, 1
        mov bh, 0
        mov bl, 0x07;  (white)
        mov dx, 320 * 4 + 235
        mov cx, 3
        push cs
        pop es
        mov bp, score_buffer
        int 0x10
        pop ds
        pop es
        popa
        ret

convert_to_ascii :
        pusha
        push ds
        push es
        mov ax, [score]; Load the score value
        mov cx, 100; Divisor for decimal conversion
        mov bx, score_buffer; Point to the score buffer

        ; Convert tens place
        xor dx, dx
        div cl
        add al, '0'; Convert quotient to ASCII
        mov[bx], al
        inc bx

        ; Convert units place
        mov al, ah
        mov ah, 0
        xor dx, dx
        mov cl, 10
        div cl
        add al, '0'
        mov[bx], al; Store ASCII  in buffer
        inc bx

        mov al, ah
        add al, '0'
        mov [bx], al

        pop es
        pop ds
        popa
        ret


        clearscr_menu :
   pusha
   mov ax, 0x0013
   int 0x10
   mov ax, 0xA000
   mov es, ax
   xor di, di
   mov al, 19
   mov cx, 64000
   rep stosb
   popa
   ret



plot_char:
    pusha
    mov di, 8
   row_loop:
    mov al, [si]
    push cx
    mov ah, 8

   pixel_loop:
    test al, 10000000b
    jz skip_pixel

    push ax
    mov ah, 0Ch        ; Function 0Ch - Write pixel
    mov al, bl         ; Color
    mov bh, 0          ; Page 0
    int 10h            ; Draw pixel
    pop ax

   skip_pixel:
    inc cx
    shl al, 1
    dec ah
    jnz pixel_loop

    pop cx
    inc dx
    inc si
    dec di
    jnz row_loop

    popa
    ret

delay:
        push cx

        mov cx, 3 ; change the values to increase delay time

        delay_loop1:

                push cx
                mov cx, 0xFFFF
        delay_loop2:
                loop delay_loop2
                pop cx
                loop delay_loop1
                pop cx

                ret

draw_block:
        push bp
        mov bp, sp
        push es
        push ax
        push bx
        push cx
        push si
        push di

        mov ax, 0xA000
        mov es, ax
        mov ax, 320
        mul word [bp+8]
        add ax, [bp+10]
```

```
shl ax, 1

mov di, ax

mov cx, 0

outer1block:

        inc cx

        cmp word cx, [bp+4]

        jg end_block_playerblock

        mov si, di

        mov bx, 1


        innerloop1block :

                mov ax, 6

                mov[es:si], ax

                inc bx

                add si, 1

                cmp word bx, [bp+6]

                jle innerloop1block

                add di, 320

                jmp outer1block

        end_block_playerblock :

                pop di

                pop si

                pop cx

                pop bx

                pop ax

                pop es

                pop bp

                ret
```