VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING (SCOPE)

TITLE: THE MAZE RUNNER

Kartik Khanna

19BCE0531

Anushka Dixit

19BCE0577

Darshan Chaurasia

19BCE0509

Project Report

of

CSE2006 – MICROPROCESSOR & INTERFACING

Fall Semester 2021-22

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING (SCOPE)

The Maze Runner

Kartik Khanna
19BCE0531

Anushka Dixit
19BCE0577

Darshan Chaurasia
19BCE0509

Project Report
of
CSE2006 – Microprocessor & Interfacing

Fall Semester 2021-22

Submitted to
Faculty: Dr. Rajesh N
Signature:
Date: 30th November 2021
Slot : E1 + TE1
Contents

Abstract

The Maze Runner, created using EMU8086, is a virtual maze game. The objective of the game is to guide the virtual player through the maze, avoiding false turns and guiding them to escape the maze. A virtual emulator with a maze of walls is displayed in the game. The arrow keys on the keyboard are used to move the character in four directions – up, down, left and right. If the player moves in the direction where the path is blocked by a wall, the system creates a beeping sound – alerting the player about the lack of free path to move. The player must employ a variety of moves to progress through the maze. The goal of this game is to put your wits to the test by finding the shortest and the most efficient path to evade the maze using the arrow keys only.

1. Introduction

The Maze Runner is an arcade game where the player has to begin from one endpoint of the maze and exit the maze by reaching the other end point using arrow keys. It used arithmetic logic programming to achieve this goal.

A maze is a collection of paths, typically from an entrance to exit. The exit is the required destination for the player. All paths do not lead to the required destination. From the starting point, the paths diverge in different directions, facilitating the player to navigate through the maze in an attempt to reach the final goal. There can be more than one paths that finally lead to the exit. In the program, the maze is defined using an array of 0s and 1s.

The player will be able to navigate through arrow keys. In the program, the ASCII values of the arrow keys are defined as variables in the program and are used to tell the code what is the value of the input. If the value of the input is anything other than the pre-defined values, the system makes a beep sound. When the player moves using arrow keys, the character on the screen moves into the blank spaces present on the screen. If the character moves somewhere where there is a wall, the system makes a beeping sound, alerting the user that the move made is invalid and waits for further input. Total number of moves of the player are counted. For each key press moves count will increase and the number moves would act as a performance metric of the user.

The user input is taken using interrupts (INT 21H), hence the code stops functioning until the user gives an input. That is, the character cannot move unless the user instructs it to do so. Once the maze is completed and the character escapes it, another interrupt is used to break the play and print the winning message as the final output. The game will continue for as long as the character remains inside the maze, irrespective of the time and the number of wrong paths taken.

2. Literature Review

[1] A Guideline for Game Development-Based Learning: A Literature Review
Authors: Bian Wu and Alf Inge Wang
Journal/Conference: International Journal of Computer Games Technology
Year: 2012

The goal of this study is to review the published scientific literature on the topics of a game development-based learning (GDBL) method using game development frameworks (GDFs) with the goal of (a) summarising a guideline for using GDBL in a curriculum, (b) identifying relevant features of GDFs, and (c) presenting a synthesis of impact factors with empirical evidence on the educational effectiveness of the GDBL method.

[2] Using Assembly Language for Creating Games
Authors: Haris Turkmanovic David Vukoje Aleksandra Lekić Milan Prokin
Journal/Conference: IcEtran, 2018
Year: 2018

The purpose of this paper is to show several interesting and useful techniques to building assembly language programmes. A project dubbed "Arkanoid" was built to demonstrate the capabilities of the assembly language. This project is developed in assembly code and contains a handful of intriguing algorithms. The game is designed in the x86 Assembly language, which generates object code for processors in the x86 class. Visual Studio 2015 was chosen as the working environment since it provides useful tools for debugging and testing the developed software (game). When the software is run, it creates a "Arkanoid" game in the Windows OS Console.

[3] Model based design introduction: modeling game controllers to microprocessor architectures
Authors: Jungwirth Patrick Abdel-Hameed Badawy
Journal/Conference: SPIE Defense + Security, 2018, Anaheim
Year: 2018

The paper starts with a notion for a video game controller and then iterates the design using model-based design to get it to a working system.

[4] Evolving Assembly Programs: How Games Help Microprocessor Validation
Authors: F. Corno; E. Sanchez; G. Squillero
Journal/Conference: IEEE Xplore
Year: 2005

This paper has two objectives. They aim to show why the microprocessor validation and test problem and the chosen game are related, and to illustrate why an optimizer designed to win a game can be successful in a different, more practical area.

[5] Microprocessor-Based Television Games, Exercises, and Evaluation Procedures for the Physically and Mentally Handicapped
Authors: W M Honig, R H Eikelboom
Journal/Conference: IEEE Engineering in Medicine and Biology
Year: 1985

Techniques for assisting both physically and intellectually challenged people are described below. Patients helped to design and test television games and workouts for the disabled. The system began with a central gaming box containing a microprocessor, interchangeable memory cards, and body joint angle transducers that could be used with a conventional colour television set. Additional transducers and software have also been developed. Although the emphasis has been on therapeutic and diagnostic applications, all games and exercises developed have varied degrees of therapeutic, diagnostic, entertaining, and educational value.

[6] Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game
Authors: Philipp Rohlfshagen; Jialin Liu; Diego Perez-Liebana; Simon M. Lucas
Journal/Conference: IEEE Xplore
Year: 2017

Pac-Man and its equally popular successor Ms Pac-Man are frequently credited with being the forerunners of the arcade video game golden age. Both games have been featured in various academic study projects over the last two decades, and their significance extends well beyond the commercial realm of video games. This publication summarises peer-reviewed research on each game (or close versions thereof), with a focus on computational intelligence. The paper also discusses the possibilities for games like Pac-Man to be used in higher education, and it finishes with a discussion of future research opportunities.

[7] Development of the Game Hangman in Assembly Programming Language
Authors: Tešanović, Stefan & Mitrović, Predrag
Journal/Conference: Telfor Journal
Year: 2018

The authors discuss a Hangman implementation in this publication. Despite the fact that the game is well-known and has been developed countless times, the implementation provides an intriguing approach to a programming assignment written entirely in assembly. This discovery is particularly educational because it gives a fun approach to expose kids to assembly programming and the architecture of x86 processors.

[8] Analysis of microprocessor system interface with memory
Authors: Lim Chun Keat; Asral Bahari Jambek; Uda Hashim
Journal/Conference: IEEE International Circuits and Systems Symposium
Year: 2015

The experiment was carried out by loading the microprocessor system architecture into Altera Cyclone II field programmable gate array (FPGA).

[9] Multiplayer Support for the Arcade Learning Environment
Authors: Justin K. Terry, Benjamin Black, Luis Santos
Journal/Conference: arxiv
Year: 2021

ALE mainly developed during Atari 2600 games, Using technologies Reinforcement Learning, Atari, Multi-Agent Reinforcement Learning

[10] Structured Assembly Language Programming
Authors: Robert N. Cook.
Journal/Conference: SIGCSE
Year: 1982

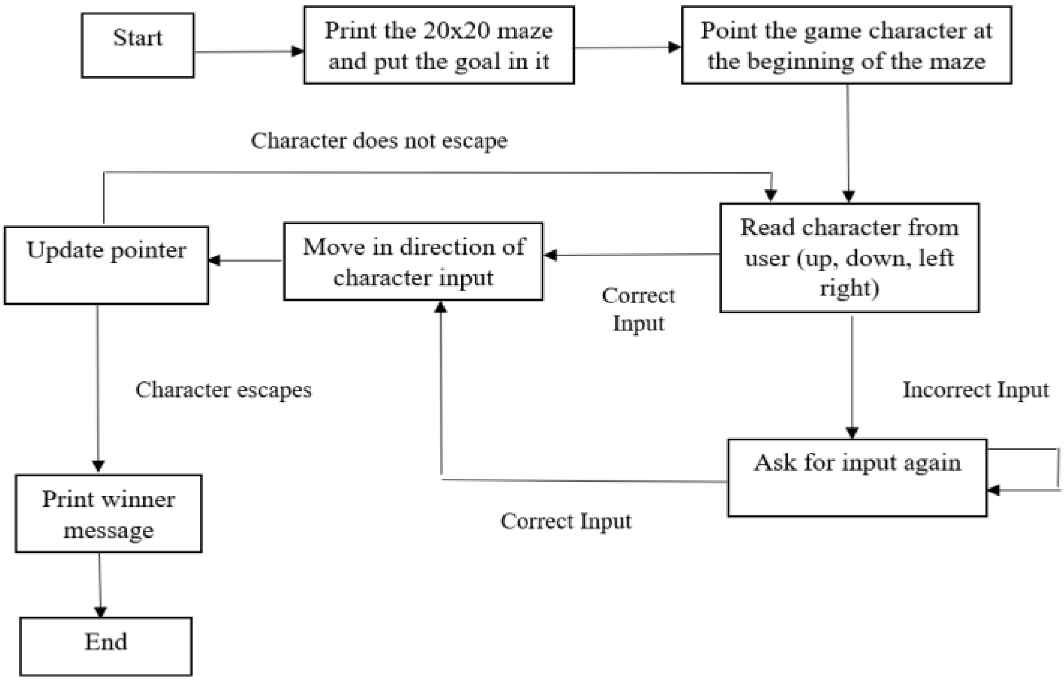
By solving the problem first using structured pseudocode and them "compiling" the pseudocode into assembly language, students can write readable self-documenting programs beginning with their first simple assignments. As the problems become more complex, the advantages of the approach described here become even more obvious.

3. Proposed Work

The proposed work uses ALP programming to emulate a maze-based game using the 8086 microprocessor. The work develops an arcade game, beginning with the concept of a character escaping a maze and further exploring methods to build a maze, take continuous user input, make use of arrow keys input and finally completing the arcade game with a winner message.

The game also makes use of system hardware sound whenever it makes a beeping sound, thus alerting the user of a wrong move made. If the user continues to make wrong moves, then the system will continue making the same sound. While running, the game utilises very low system RAM and space, thus allowing it to function even in low memory (16-bit) systems. This greatly lowers down the hardware requirements of the game. Since the code is locally stored in the system, no network requirements are needed.

4. Flowchart for algorithm


Start
Print the 20x20 maze and put the goal in it
Point the game character at the beginning of the maze
Character does not escape
Read character from user (up, down, left right)
Move in direction of character input
Update pointer
Correct Input
Incorrect Input
Character escapes
Ask for input again
Correct Input
Print winner message
End

5. Implementation

5.1 Approach

- In Maze Runner game, 20 rows will be used to create a maze, that is 20 arrays of 0's and 1's. It will be a 2-D matrix of 20 rows and 20 columns. At 1's place we will print block using PUTC 219 and blank space at 0's using PUTC 32. A loop will run to print all rows of maze from row 1 to row 20 sequentially. CX register will maintain the count of 20 to print all 20 arrays from maze 1 to maze 20 .

- When maze is created a character will be created which will perform 4 types of operations: move left, move right, move up and move down. These will be done using arrow keys, whose ASCII values are stored in the variables left, right, up and down – defined in the data segment. From Game loop label, game will start and ask for moves from keyboard by using up, down, right and left arrow keys, by making an interrupt every time user input is required.

- Character can move only in blank spaces that is 0's in the matrix. This loop for moving and asking input as interrupt will be continued.

- If the character tackles any obstacle, that is, when it tackles any 1 in memory location of arrays, it will not move and make a beep sound using:

```
MOV AH,
02 MOV DL,
07H INT 21H
```

informing of the obstacle ahead and then ask the user to enter the move again. This will be achieved using CMP instruction. That is, the next value of array is compared to 0. If the values are equal, the character moves ahead. If they are unequal, the system produces a beep.

- Character moves will be counted and stored at a memory location. When the character escapes the maze at row 18, that is, at maze with last array data as 0, the loop will stop.

- This will follow in the screen being cleared of the maze and the winner message being displayed, which will be initialised as string.

5.2 Algorithm

- On running the code, the screen would display rules of the game and ask the user to press any key.
- On getting a user input, the program starts to print a 20x20 maze line by line, which is pre-defined in the data segment as 20 arrays of 0s and 1s.
- The program then prints a Character, which the user can navigate through the maze using arrow keys.

- If the user makes a move in the direction where a wall is blocking it, a beep sound will be produced.
- Total number of moves used are counted.
- Once the user navigates through the maze, the program prints the number of moves used, a winner message and halts.

5.3 Working

The program will be written in assembly language and run on an emu8086 emulator. The game greets the user with a welcome message and instructs them to start the game by pressing any key.

- When the player first starts the game, a maze will appear on the screen thanks to a loop in the code in which command PUTC219 places white areas on the screen and command PUTC32 places blank spaces. The maze's walls will be white blocks, while the black areas will be a blank area for the player to walk through.

- To successfully leave the maze, the player must avoid stepping into the maze walls and travel via the open sections. Once the maze has been established on the player screen, a character representing the player appears in the start spot. The player may now walk around the maze by pressing the up, down, left, and right arrow buttons on their keyboard.

- A new loop will be established in the code for each movement, which will update the player's position after each movement. When a player pushes a key to proceed through the maze, a function is invoked that first determines the key the player used as input and whether or not the position is valid. The position is legitimate and the method returns true if the player wants to move to a clear area. Then, depending on whatever key was used as input, a function is called.

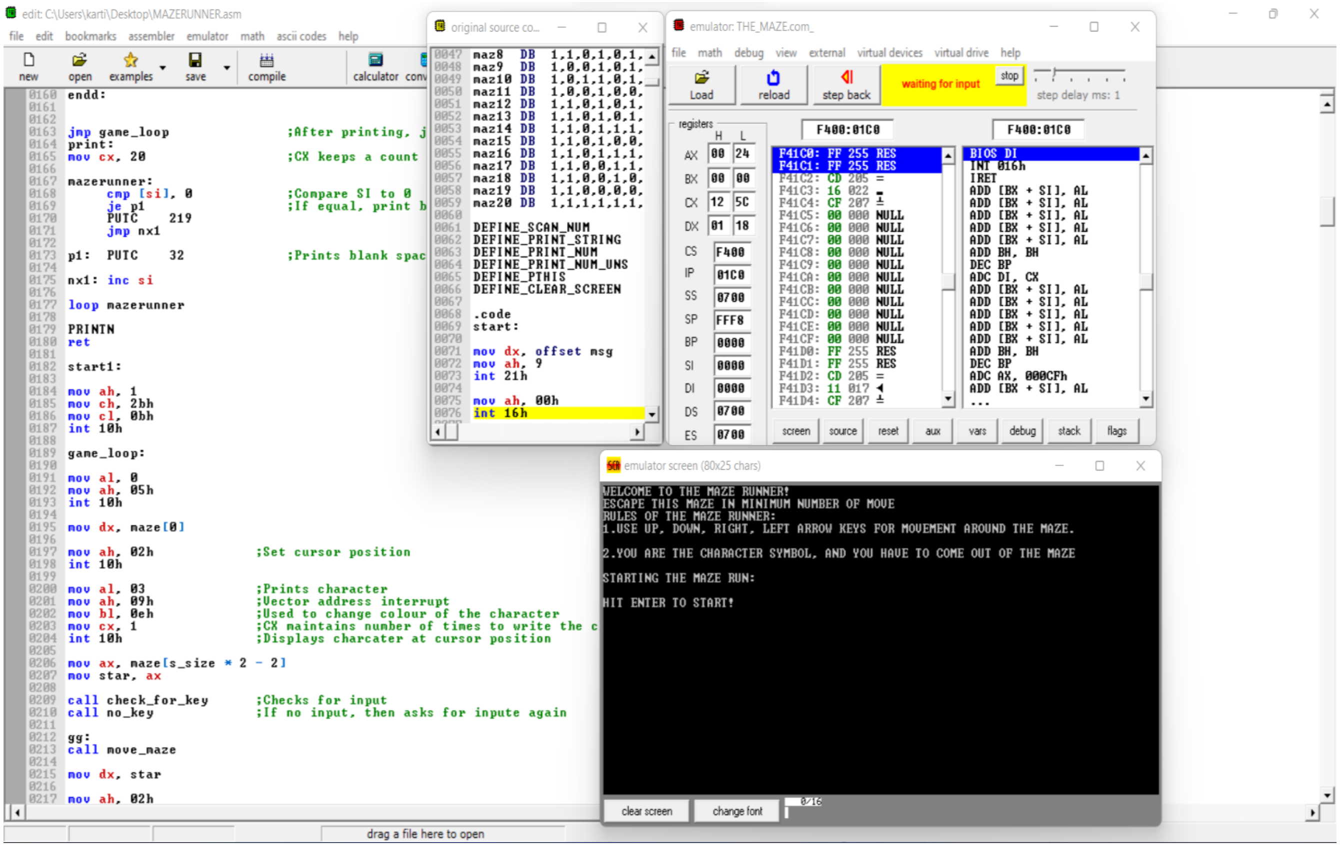
- When the down key is pushed, the x coordinate is reset to the previous position, while the y coordinate is increased by one. If the right key is pushed, the player's y coordinate is set to the same as the previous position, but the x coordinate is increased by one to move the player to the new place.

- As a result, the player's position is changed to the new location based on the key input by the player. If a white block is present, however, the function that validates the correctness of a position returns false, and the player's position is reset to his prior position while the system waits for the next input.

- While the player is travelling through the maze, another loop continues to run, counting each keypress to calculate the total number of moves the player has made during the game.

- The loop stops running after the player has successfully exited the maze, and a congrats message appears on the screen. In addition, the screen displays the player's total number of moves. The player must try to make the fewest number of movements possible.

6. Results and Screenshots


edit: C:\Users\karti\Desktop\MAZERUNNER.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile calculator conv
original source co...
emulator: THE_MAZE.com_
file math debug view external virtual devices virtual drive help
Load reload step back waiting for input stop step delay ms: 1
registers H L
screen source reset aux vars debug stack flags
emulator screen (80x25 chars)
clear screen change font 0/16
drag a file here to open

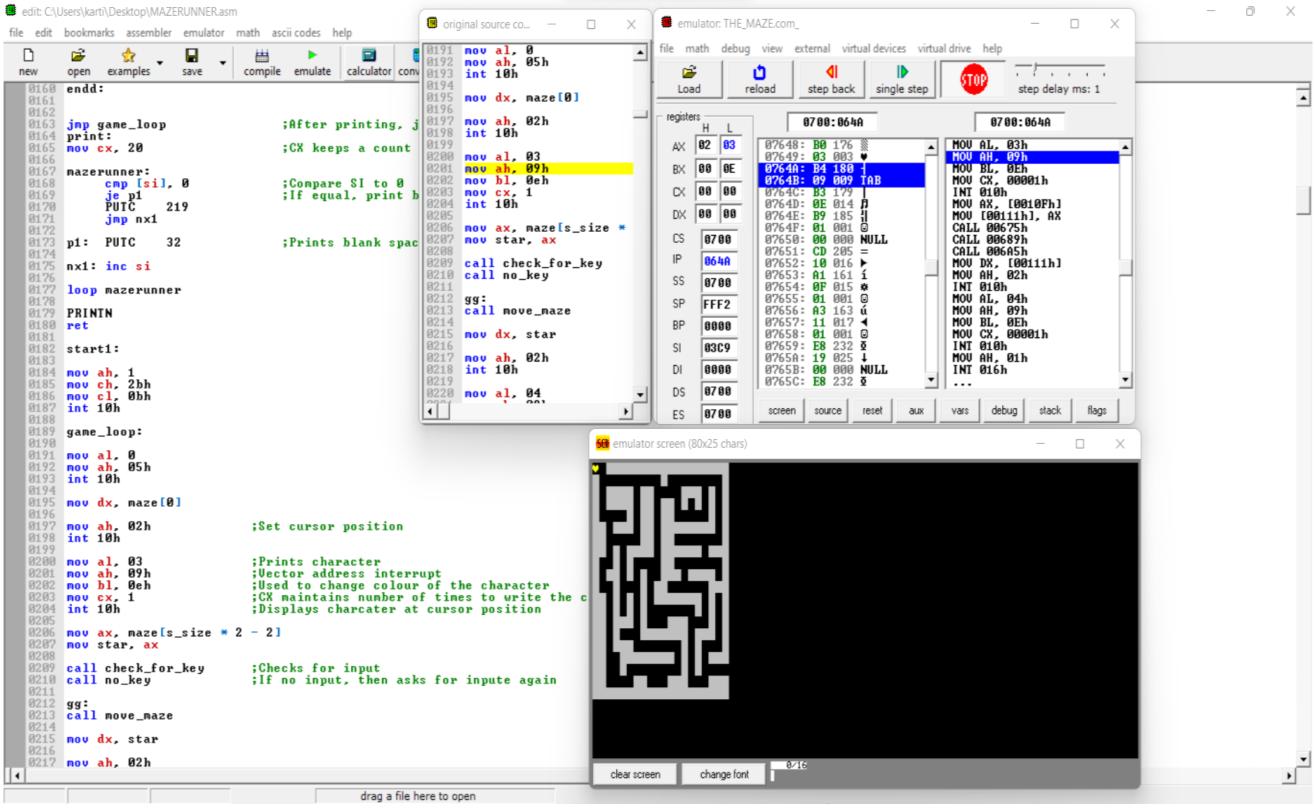
```
endd:


jmp game_loop                       ;After printing, j
print:
mov cx, 20                          ;CX keeps a count

mazerunner:
     cmp [si], 0                    ;Compare SI to 0
     je p1                          ;If equal, print b
     PUTC    219
     jmp nx1

p1:  PUTC    32                     ;Prints blank spac

nx1: inc si

loop mazerunner

PRINTN
ret

start1:

mov ah, 1
mov ch, 2bh
mov cl, 0bh
int 10h

game_loop:

mov al, 0
mov ah, 05h
int 10h

mov dx, maze[0]

mov ah, 02h                 ;Set cursor position
int 10h

mov al, 03                  ;Prints character
mov ah, 09h                 ;Vector address interrupt
mov bl, 0eh                 ;Used to change colour of the character
mov cx, 1                   ;CX maintains number of times to write the c
int 10h                     ;Displays charcater at cursor position

mov ax, maze[s_size * 2 - 2]
mov star, ax

call check_for_key          ;Checks for input
call no_key                 ;If no input, then asks for inpute again

gg:
call move_maze

mov dx, star

mov ah, 02h
```

```
maz8  DB  1,1,0,1,0,1,
maz9  DB  1,0,0,1,0,1,
maz10 DB  1,0,1,1,0,1,
maz11 DB  1,0,0,1,0,0,
maz12 DB  1,1,0,1,0,1,
maz13 DB  1,1,0,1,0,1,
maz14 DB  1,1,0,1,1,1,
maz15 DB  1,1,0,1,0,0,
maz16 DB  1,1,0,1,1,1,
maz17 DB  1,1,0,0,1,1,
maz18 DB  1,1,0,0,1,0,
maz19 DB  1,1,0,0,0,0,
maz20 DB  1,1,1,1,1,1,

DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
DEFINE_PTHIS
DEFINE_CLEAR_SCREEN

.code
start:

mov dx, offset msg
mov ah, 9
int 21h

mov ah, 00h
int 16h
```

```
     H  L
AX  00 24
BX  00 00
CX  12 5C
DX  01 18
CS  F400
IP  01C0
SS  0700
SP  FFF8
BP  0000
SI  0000
DI  0000
DS  0700
ES  0700
```

```
F400:01C0                  F400:01C0
F41C0: FF 255 RES          BIOS DI
F41C1: FF 255 RES          INT 016h
F41C2: CD 205 =            IRET
F41C3: 16 022 ▬            ADD [BX + SI], AL
F41C4: CF 207 ╧            ADD [BX + SI], AL
F41C5: 00 000 NULL         ADD [BX + SI], AL
F41C6: 00 000 NULL         ADD [BX + SI], AL
F41C7: 00 000 NULL         ADD [BX + SI], AL
F41C8: 00 000 NULL         ADD BH, BH
F41C9: 00 000 NULL         DEC BP
F41CA: 00 000 NULL         ADC DI, CX
F41CB: 00 000 NULL         ADD [BX + SI], AL
F41CC: 00 000 NULL         ADD [BX + SI], AL
F41CD: 00 000 NULL         ADD [BX + SI], AL
F41CE: 00 000 NULL         ADD [BX + SI], AL
F41CF: 00 000 NULL         ADD [BX + SI], AL
F41D0: FF 255 RES          ADD BH, BH
F41D1: FF 255 RES          DEC BP
F41D2: CD 205 =            ADC AX, 000CFh
F41D3: 11 017 ◄            ADD [BX + SI], AL
F41D4: CF 207 ╧            ...
```

```
WELCOME TO THE MAZE RUNNER!
ESCAPE THIS MAZE IN MINIMUM NUMBER OF MOVE
RULES OF THE MAZE RUNNER:
1.USE UP, DOWN, RIGHT, LEFT ARROW KEYS FOR MOVEMENT AROUND THE MAZE.

2.YOU ARE THE CHARACTER SYMBOL, AND YOU HAVE TO COME OUT OF THE MAZE

STARTING THE MAZE RUN:

HIT ENTER TO START!
```

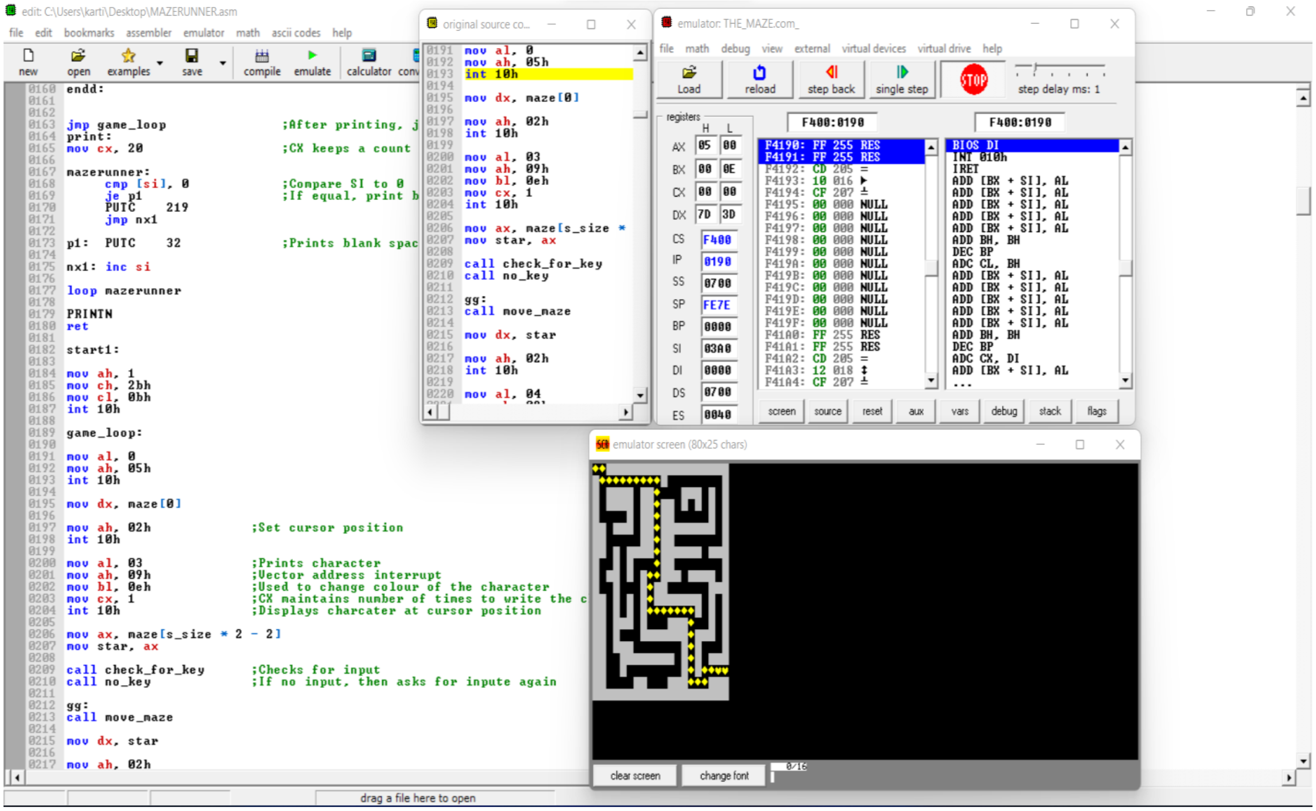

edit: C:\Users\karti\Desktop\MAZERUNNER.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator conv
original source co...
emulator: THE_MAZE.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step STOP step delay ms: 1
screen source reset aux vars debug stack flags
emulator screen (80x25 chars)
clear screen change font
drag a file here to open

```
endd:


jmp game_loop                       ;After printing, j
print:
mov cx, 20                          ;CX keeps a count

mazerunner:
      cmp [si], 0                   ;Compare SI to 0
      je p1                         ;If equal, print b
      PUTC    219
      jmp nx1

p1:   PUTC    32                    ;Prints blank spac

nx1: inc si

loop mazerunner

PRINTN
ret

start1:

mov ah, 1
mov ch, 2bh
mov cl, 0bh
int 10h

game_loop:

mov al, 0
mov ah, 05h
int 10h

mov dx, maze[0]

mov ah, 02h                     ;Set cursor position
int 10h

mov al, 03                      ;Prints character
mov ah, 09h                     ;Vector address interrupt
mov bl, 0eh                     ;Used to change colour of the character
mov cx, 1                       ;CX maintains number of times to write the c
int 10h                         ;Displays charcater at cursor position

mov ax, maze[s_size * 2 - 2]
mov star, ax

call check_for_key              ;Checks for input
call no_key                     ;If no input, then asks for inpute again

gg:
call move_maze

mov dx, star

mov ah, 02h
```

```
registers
    H  L
AX  02 03
BX  00 0E
CX  00 00
DX  00 00
CS  0700
IP  064A
SS  0700
SP  FFF2
BP  0000
SI  03C9
DI  0000
DS  0700
ES  0700
```

```
0700:064A

07648: B0 176 ░
07649: 03 003 ♥
0764A: B4 180 ┤
0764B: 09 009 TAB
0764C: B3 179 │
0764D: 0E 014 ♫
0764E: B9 185 ╣
0764F: 01 001 ☺
07650: 00 000 NULL
07651: CD 205 ═
07652: 10 016 ►
07653: A1 161 í
07654: 0F 015 ☼
07655: 01 001 ☺
07656: A3 163 ú
07657: 11 017 ◄
07658: 01 001 ☺
07659: E8 232 Φ
0765A: 19 025 ↓
0765B: 00 000 NULL
0765C: E8 232 Φ
```

```
0700:064A

MOV AL, 03h
MOV AH, 09h
MOV BL, 0Eh
MOV CX, 00001h
INT 010h
MOV AX, [0010Fh]
MOV [00111h], AX
CALL 00675h
CALL 00689h
CALL 006A5h
MOV DX, [00111h]
MOV AH, 02h
INT 010h
MOV AL, 04h
MOV AH, 09h
MOV BL, 0Eh
MOV CX, 00001h
INT 010h
MOV AH, 01h
INT 016h
...
```

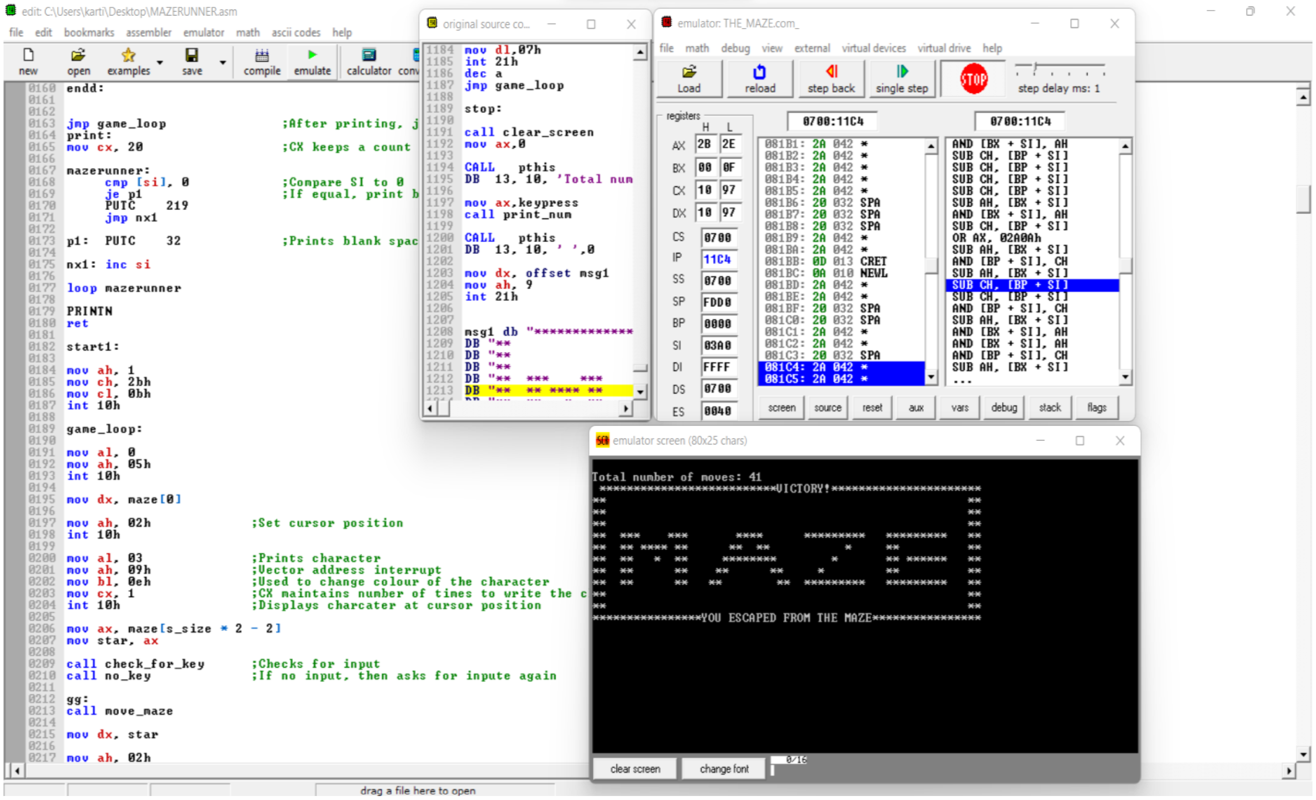

edit: C:\Users\karti\Desktop\MAZERUNNER.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator conv
original source co...
emulator: THE_MAZE.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step STOP step delay ms: 1
screen source reset aux vars debug stack flags
emulator screen (80x25 chars)
clear screen change font
drag a file here to open

```
endd:


jmp game_loop                       ;After printing, j
print:
mov cx, 20                          ;CX keeps a count

mazerunner:
      cmp [si], 0                   ;Compare SI to 0
      je p1                         ;If equal, print b
      PUTC    219
      jmp nx1

p1:   PUTC    32                    ;Prints blank spac

nx1: inc si

loop mazerunner

PRINTN
ret

start1:

mov ah, 1
mov ch, 2bh
mov cl, 0bh
int 10h

game_loop:

mov al, 0
mov ah, 05h
int 10h

mov dx, maze[0]

mov ah, 02h                     ;Set cursor position
int 10h

mov al, 03                      ;Prints character
mov ah, 09h                     ;Vector address interrupt
mov bl, 0eh                     ;Used to change colour of the character
mov cx, 1                       ;CX maintains number of times to write the c
int 10h                         ;Displays charcater at cursor position

mov ax, maze[s_size * 2 - 2]
mov star, ax

call check_for_key              ;Checks for input
call no_key                     ;If no input, then asks for inpute again

gg:
call move_maze

mov dx, star

mov ah, 02h
```

```
registers
    H  L
AX  05 00
BX  00 0E
CX  00 00
DX  7D 3D
CS  F400
IP  0190
SS  0700
SP  FE7E
BP  0000
SI  03A0
DI  0000
DS  0700
ES  0040
```

```
F400:0190

F4190: FF 255 RES
F4191: FF 255 RES
F4192: CD 205 ═
F4193: 10 016 ►
F4194: CF 207 ╧
F4195: 00 000 NULL
F4196: 00 000 NULL
F4197: 00 000 NULL
F4198: 00 000 NULL
F4199: 00 000 NULL
F419A: 00 000 NULL
F419B: 00 000 NULL
F419C: 00 000 NULL
F419D: 00 000 NULL
F419E: 00 000 NULL
F419F: 00 000 NULL
F41A0: FF 255 RES
F41A1: FF 255 RES
F41A2: CD 205 ═
F41A3: 12 018 ↕
F41A4: CF 207 ╧
```

```
F400:0190

BIOS DI
INT 010h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC CX, DI
ADD [BX + SI], AL
...
```



edit: C:\Users\karti\Desktop\MAZERUNNER.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator conv
drag a file here to open
original source co...
emulator: THE_MAZE.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step STOP step delay ms: 1
screen source reset aux vars debug stack flags
emulator screen (80x25 chars)
clear screen change font 0/16

```
endd:


jmp game_loop                   ;After printing, j
print:
mov cx, 20                      ;CX keeps a count

mazerunner:
     cmp [si], 0                ;Compare SI to 0
     je p1                      ;If equal, print b
     PUTC    219
     jmp nx1

p1:  PUTC    32                 ;Prints blank spac

nx1: inc si

loop mazerunner

PRINTN
ret

start1:

mov ah, 1
mov ch, 2bh
mov cl, 0bh
int 10h

game_loop:

mov al, 0
mov ah, 05h
int 10h

mov dx, maze[0]

mov ah, 02h                 ;Set cursor position
int 10h

mov al, 03                  ;Prints character
mov ah, 09h                 ;Vector address interrupt
mov bl, 0eh                 ;Used to change colour of the character
mov cx, 1                   ;CX maintains number of times to write the c
int 10h                     ;Displays charcater at cursor position

mov ax, maze[s_size * 2 - 2]
mov star, ax

call check_for_key          ;Checks for input
call no_key                 ;If no input, then asks for inpute again

gg:
call move_maze

mov dx, star

mov ah, 02h
```

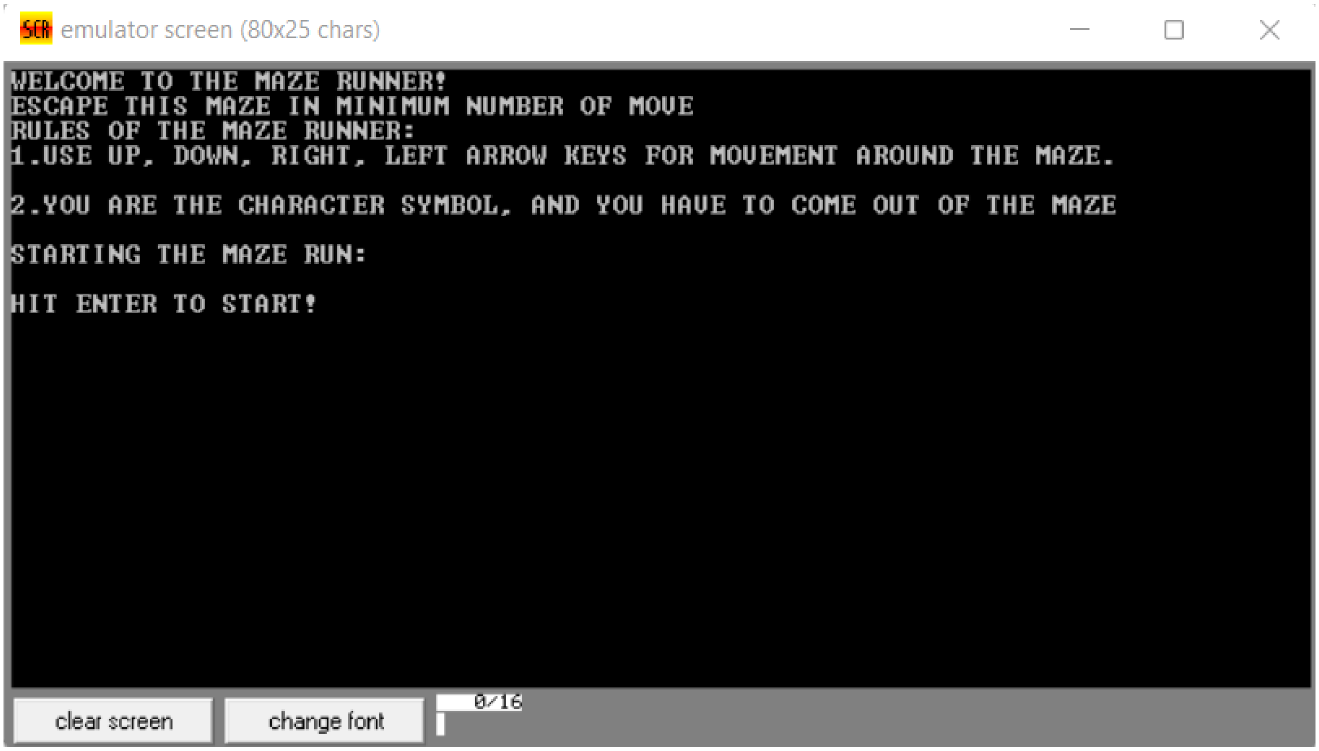
```
mov dl,07h
int 21h
dec a
jmp game_loop

stop:

call clear_screen
mov ax,0

CALL   pthis
DB  13, 10, 'Total num

mov ax,keypress
call print_num

CALL   pthis
DB  13, 10, ' ',0

mov dx, offset msg1
mov ah, 9
int 21h


msg1 db "*************
DB "**
DB "**
DB "**
DB "**  ***    ***
DB "**  ** **** **
```

```
registers
     H    L
AX   2B   2E
BX   00   0F
CX   10   97
DX   10   97
CS   0700
IP   11C4
SS   0700
SP   FDD0
BP   0000
SI   03A0
DI   FFFF
DS   0700
ES   0040
```

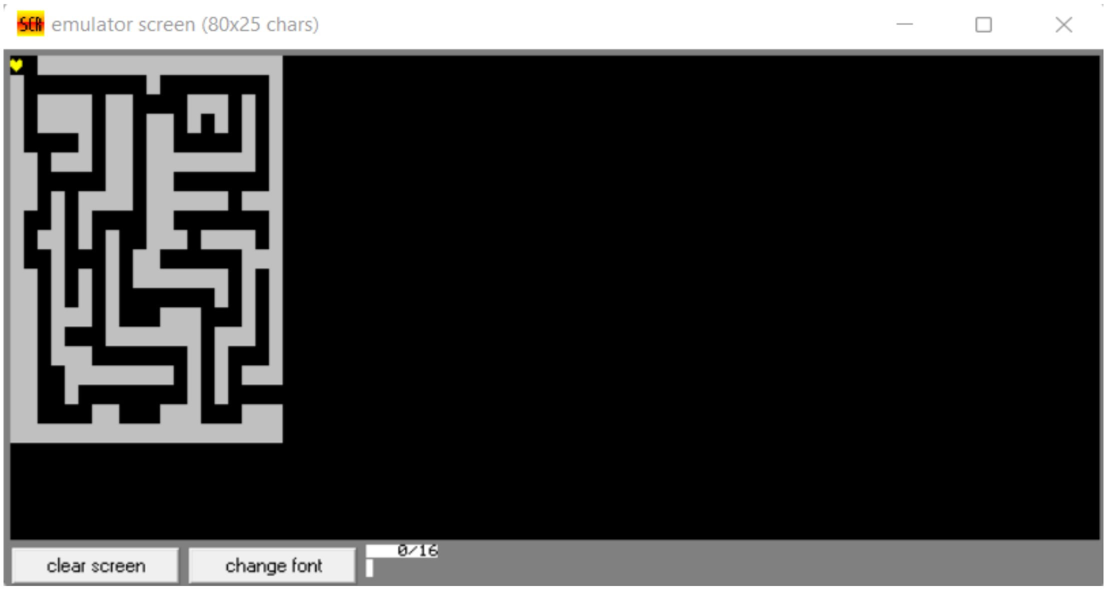
```
0700:11C4
081B1: 2A 042 *
081B2: 2A 042 *
081B3: 2A 042 *
081B4: 2A 042 *
081B5: 2A 042 *
081B6: 20 032 SPA
081B7: 20 032 SPA
081B8: 20 032 SPA
081B9: 2A 042 *
081BA: 2A 042 *
081BB: 0D 013 CRET
081BC: 0A 010 NEWL
081BD: 2A 042 *
081BE: 2A 042 *
081BF: 20 032 SPA
081C0: 20 032 SPA
081C1: 2A 042 *
081C2: 2A 042 *
081C3: 20 032 SPA
081C4: 2A 042 *
081C5: 2A 042 *
```

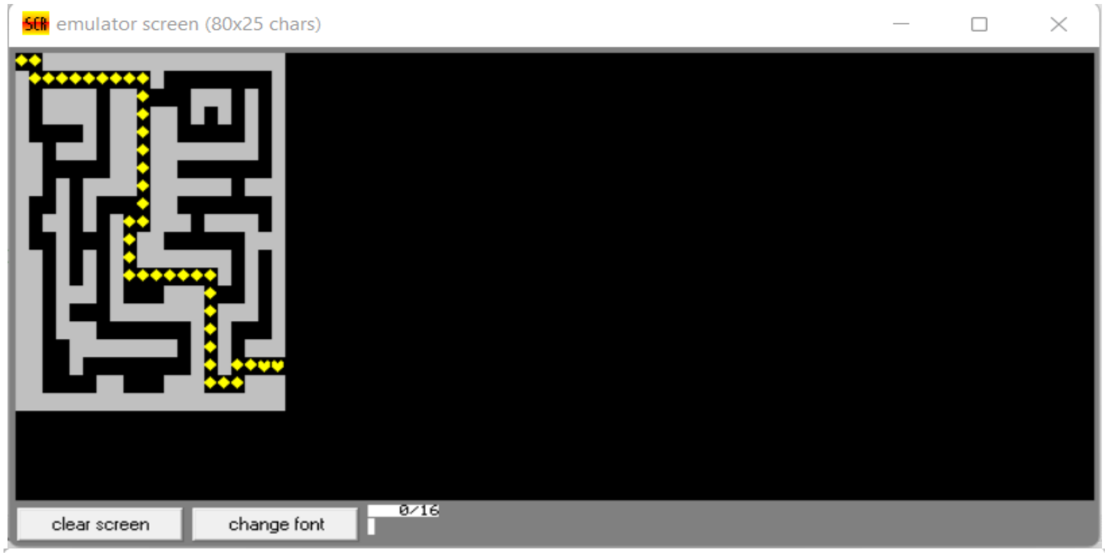
```
0700:11C4
AND [BX + SI], AH
SUB CH, [BP + SI]
SUB CH, [BP + SI]
SUB CH, [BP + SI]
SUB CH, [BP + SI]
SUB AH, [BX + SI]
AND [BX + SI], AH
SUB CH, [BP + SI]
OR AX, 02A0Ah
SUB AH, [BX + SI]
AND [BP + SI], CH
SUB AH, [BX + SI]
SUB CH, [BP + SI]
SUB CH, [BP + SI]
AND [BP + SI], CH
SUB AH, [BX + SI]
AND [BX + SI], AH
AND [BX + SI], AH
AND [BP + SI], CH
SUB AH, [BX + SI]
...
```

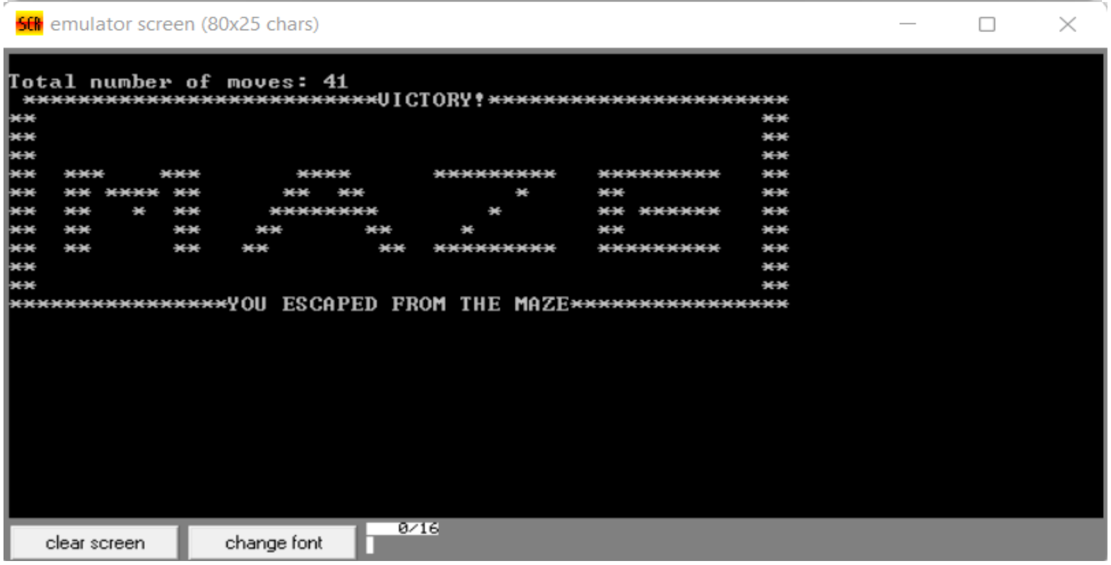
```
Total number of moves: 41
 ***************************VICTORY!**********************
**                                                      **
**                                                      **
**                                                      **
**  ***    ***      ****      *********   *********    **
**  ** **** **     **  **          *      **           **
**  **   *  **    ********        *       ** *******   **
**  **      **   **      **      *        **           **
**  **      **  **        **  *********   *********    **
**                                                      **
**                                                      **
****************YOU ESCAPED FROM THE MAZE****************
```



```
WELCOME TO THE MAZE RUNNER!
ESCAPE THIS MAZE IN MINIMUM NUMBER OF MOVE
RULES OF THE MAZE RUNNER:
1.USE UP, DOWN, RIGHT, LEFT ARROW KEYS FOR MOVEMENT AROUND THE MAZE.

2.YOU ARE THE CHARACTER SYMBOL, AND YOU HAVE TO COME OUT OF THE MAZE

STARTING THE MAZE RUN:

HIT ENTER TO START!
```







```
Total number of moves: 41
 **************************VICTORY!**********************
**                                                     **
**                                                     **
**                                                     **
**  ***    ***       ****      *********   *********   **
**  ** **** **      **  **           *     **          **
**  **   *  **     ********        *       ** ******   **
**  **      **    **      **     *         **          **
**  **      **   **        **  *********   *********   **
**                                                     **
**                                                     **
****************YOU ESCAPED FROM THE MAZE****************
```

7. Conclusion

The game ‘Maze Runner’ is an arcade game built using arithmetic logical programming on the platform EMU8086. It is a game with low hardware requirements, making it compatible with a large number of systems. The game begins with a character at the start of the maze. Objective is to use the arrow keys to navigate through the maze and escape it. Once escaped, user gets a message indicating victory. A well-loved arcade game, maze runner builds cognitive skills and helps develop quick and logical thinking.

8. References

[1] https://www.tutorialspoint.com/assembly_programming/index.htm, Sept 2020

[2]https://www.researchgate.net/publication/340570563_TEACHING_OF_808886_PROGRAMMING_WITH_8086_ASSEMBLY_EMULATOR, April 2020

[3]https://www.tutorialspoint.com/microprocessor/microprocessor_8086_interrupts.htm , Jan 2020

[4]https://www.researchgate.net/publication/330602360_Development_of_the_Game_Hangman_in_Assembly_Programming_Language , Nov 2018

[5]https://www.tutorialspoint.com/microprocessor/microprocessor_8086_interrupts.htm , Oct 2018

[6] Haris Turkmanovic David Vukoje Aleksandra Lekić Milan Prokin | Using Assembly Language for Creating Games | June 2018 | Conference: IcEtran

[7] https://www.javatpoint.com/instruction-set-of-8086, January 2018

[8] https://stackoverflow.com/questions/30546917/validate-maze-in-assembly-8086, Aug 2015

[9] https://www.dreamincode.net/forums/topic/302122-assembly-language-mouse-in-the-maze-program/, Nov 2012

[10] F. Corno; E. Sanchez; G. Squillero | Evolving assembly programs: how games help microprocessor validation (2005) | IEEE Journals & Magazine | IEEE Xplore

[11]http://www.gabrielececchetti.it/Teaching/CalcolatoriElettronici/Docs/i8086_instruction_set.pdf , Sept 2005

[12] https://ieeexplore.ieee.org/document/1330848/references#references, June 2004

Appendix (Code)

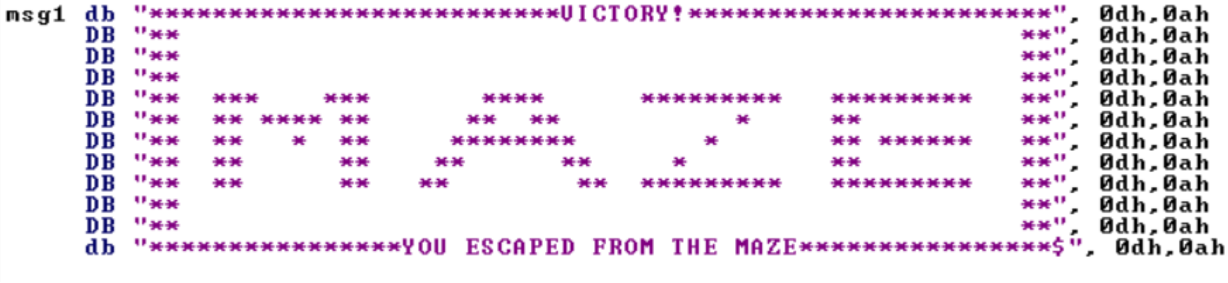
Link: https://bit.ly/Maze-Runner-Code

```
INCLUDE "emu8086.inc"
name "THE MAZE RUNNER"

org 100h                    ;Data Segment starts
.data
  a dw 0
  b dw 0
  aal dw 0
  c dw 0
  d dw 0

s_size  equ     2

maze dw s_size dup(0)

star    dw      ?           ;Position of the character, used to change character

left    equ     4bh          ;Left stores the ASCII value of LEFT Arrow Key
right   equ     4dh          ;Right stores the ASCII value of RIGHT Arrow Key
up      equ     48h          ;Up stores the ASCII value of UP Arrow Key
down    equ     50h          ;Down stores the ASCII value of DOWN Arrow Key

cur_dir db      right

wait_time dw    0
keypress dw 0

msg db "WELCOME TO THE MAZE RUNNER!", 0dh,0ah                           ;Printing messages
        db "ESCAPE THIS MAZE IN MINIMUM NUMBER OF MOVE", 0dh,0ah            ;0dh
: Carriage Return
        db "RULES OF THE MAZE RUNNER:", 0dh,0ah                         ;0ah : Line Feed
Character
        db "1.USE UP, DOWN, RIGHT, LEFT ARROW KEYS FOR MOVEMENT AROUND
THE MAZE.", 0dh,0ah, 0ah
        db "2.YOU ARE THE CHARACTER SYMBOL, AND YOU HAVE TO COME OUT
OF THE MAZE", 0dh,0ah, 0ah
        db "STARTING THE MAZE RUN:  ", 0dh,0ah,0ah                      ;Moves to the next
line and moves to start of next line
        db "HIT ENTER TO START!$"

   ;Defining the maze using array of 0 and 1
   maz1  DB  0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1   ;0 represents path where one can move
   maz2  DB  1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1   ;1 represents walls
   maz3  DB  1,0,1,1,1,1,0,1,1,0,0,0,1,1,1,0,1,0,1
   maz4  DB  1,0,1,1,1,1,0,1,1,0,1,1,0,1,0,1,0,1,0,1
   maz5  DB  1,0,0,0,0,1,0,1,1,0,1,1,0,0,0,0,0,1,0,1
   maz6  DB  1,1,0,1,1,1,0,1,1,0,1,1,1,1,1,1,1,0,1
   maz7  DB  1,1,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,1
   maz8  DB  1,1,0,1,0,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1
   maz9  DB  1,0,0,1,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,1
   maz10 DB  1,0,1,1,0,1,0,1,0,0,1,1,1,0,1,1,1,1,0,1
   maz11 DB  1,0,0,1,0,0,0,1,0,1,1,0,0,0,0,0,0,1,1,1
   maz12 DB  1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,0,1,0,1
   maz13 DB  1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,1
   maz14 DB  1,1,0,1,1,1,0,1,0,0,0,1,1,1,0,0,0,1,0,1
   maz15 DB  1,1,0,1,0,0,0,1,1,1,1,1,1,1,0,1,1,1,0,1
   maz16 DB  1,1,0,1,1,1,0,0,0,0,0,0,0,1,0,1,0,0,0,1
   maz17 DB  1,1,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1
   maz18 DB  1,1,0,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0   ;maze exit
   maz19 DB  1,1,0,0,0,0,1,1,0,0,0,1,1,1,0,0,0,1,3,1
   maz20 DB  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

  DEFINE_SCAN_NUM                  ;procedure that gets the multi-digit SIGNED number from
the keyboard, and stores the result in CX register.
  DEFINE_PRINT_STRING              ;macro with 1 parameter, prints out a string.
  DEFINE_PRINT_NUM                 ;procedure that prints a signed number in AX register.
  DEFINE_PRINT_NUM_UNS             ;procedure that prints out an unsigned number in AX
register.
  DEFINE_PTHIS                     ;procedure to print a null terminated string at current cursor
position
  DEFINE_CLEAR_SCREEN              ;procedure to clear the screen and set cursor position to
top of it

.code                              ;Code Segment begins
start:
mov dx, offset msg                 ;Prints message defined by 'msg' in the data segment
mov ah, 9                          ;Asks for user input to begin the game
int 21h
mov ah, 00h                        ;Screen chnage on input
int 16h
CALL CLEAR_SCREEN                  ;Clears screen once user input is obtained
pmaz1:                             ;Printing first row of the maze as defined in the data segment
lea si,maz1
call print

pmaz2:                             ;Printing second row of the maze as defined in the data segment
lea si,maz2
call print
```

```
pmaz3:                          ;Printing third row of the maze as defined in the data segment
lea si,maz3
call print

pmaz4:                          ;Printing fourth row of the maze as defined in the data segment
lea si,maz4
call print

pmaz5:                          ;Printing fifth row of the maze as defined in the data segment
lea si,maz5
call print

pmaz6:                          ;Printing sixth row of the maze as defined in the data segment
lea si,maz6
call print

pmaz7:                          ;Print 7th row of the maze
lea si,maz7
call print

pmaz8:                          ;Print 8th row of the maze
lea si,maz8
call print

pmaz9:
lea si,maz9
call print

pmaz10:
lea si,maz10
call print

pmaz11:
lea si,maz11
call print

pmaz12:
lea si,maz12
call print

pmaz13:
lea si,maz13
call print

pmaz14:
```

```
lea si,maz14
call print

pmaz15:
lea si,maz15
call print

pmaz16:
lea si,maz16
call print

pmaz17:
lea si,maz17
call print

pmaz18:
lea si,maz18
call print

pmaz19:
lea si,maz19
call print

pmaz20:
lea si,maz20
call print
endd:

jmp game_loop              ;After printing, jumps to game loop
print:
mov cx, 20                 ;CX keeps a count of 20 to print all rows
mazerunner:
    cmp [si], 0            ;Compare SI to 0
    je p1                   ;If equal, print blank space
    PUTC    219
    jmp nx1

p1:  PUTC    32            ;Prints blank space
nx1: inc si
loop mazerunner

PRINTN
ret

start1:
mov ah, 1
mov ch, 2bh
mov cl, 0bh
int 10h
game_loop:
mov al, 0
mov ah, 05h
int 10h
mov dx, maze[0]
mov ah, 02h             ;Set cursor position
int 10h
mov al, 03              ;Prints character
mov ah, 09h             ;Vector address interrupt
mov bl, 0eh             ;Used to change colour of the character
mov cx, 1               ;CX maintains number of times to write the character
int 10h                         ;Displays character at cursor position
mov ax, maze[s_size * 2 - 2]
mov star, ax
call check_for_key      ;Checks for input
call no_key             ;If no input, then asks for input again
gg:
call move_maze
mov dx, star
mov ah, 02h
int 10h
mov al, 04
mov ah, 09h
mov bl, 0eh
mov cx, 1
int 10h
check_for_key:
mov ah, 01h
int 16h                 ;Asks for keyboard input
jz no_key               ;Waits if no key
mov ah, 00h
int 16h
cmp al, 1bh
je stop_game
mov cur_dir, ah      ;Value of AH moves to Current direction, as input by the user
jmp gg
no_key:
mov ah, 00h
int 1ah
cmp dx, wait_time
jb  check_for_key
add  dx, 4
mov  wait_time, dx
```

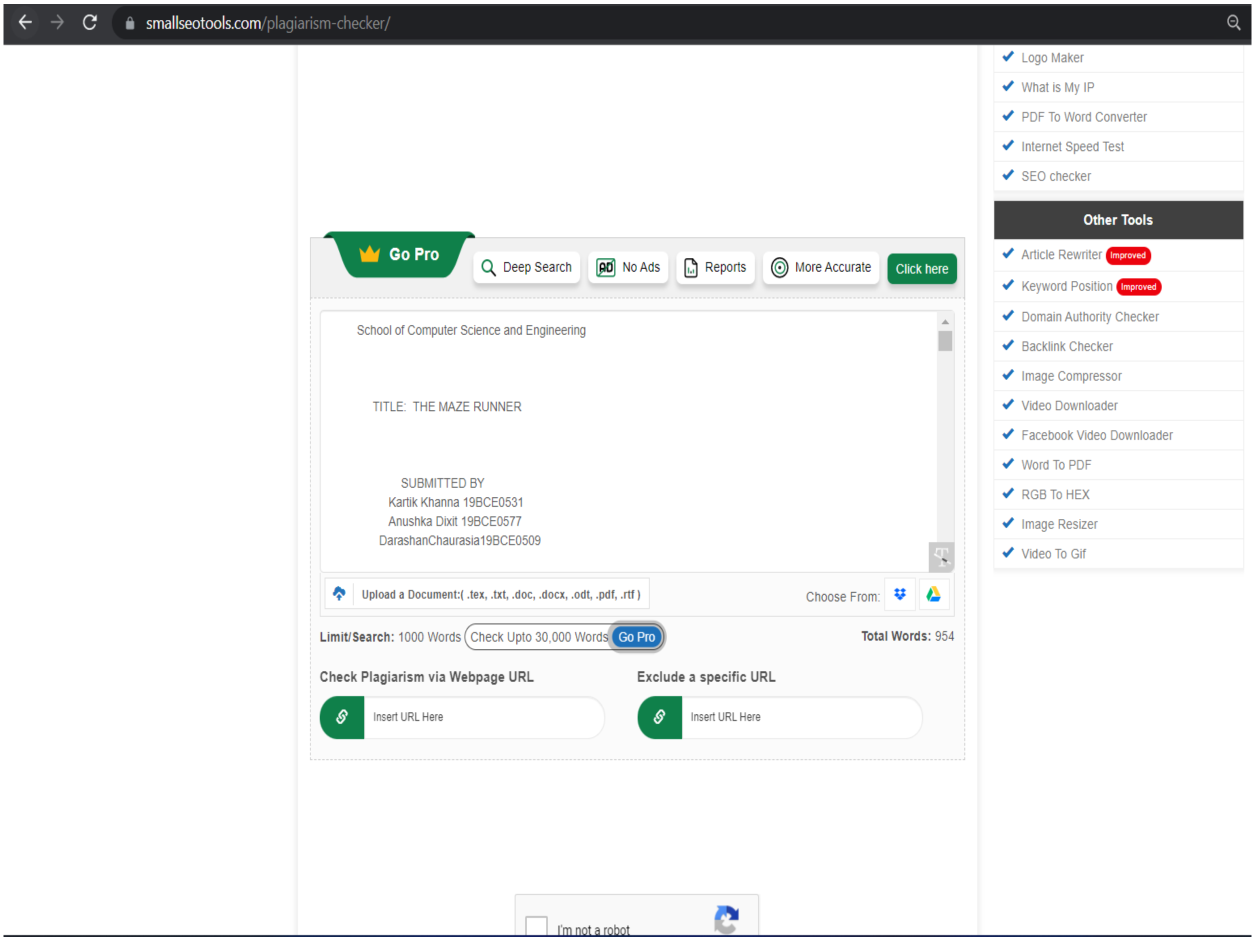
```
   jmp game_loop
stop_game:


stop:
call clear_screen                              ;Clears screen
mov ax,0
CALL   pthis
DB  13, 10, 'Total number of moves: ',0        ;Prints total number of moves
mov ax,keypress
call print_num
CALL   pthis
DB  13, 10, ' ',0
mov dx, offset msg1                                    ;Prints message 1
mov ah, 9
int 21h
```



```
msg1 db "**************************VICTORY!************************", 0dh,0ah
     DB "**                                                      **", 0dh,0ah
     DB "**                                                      **", 0dh,0ah
     DB "**                                                      **", 0dh,0ah
     DB "**  ***    ***       ****       *********   *********   **", 0dh,0ah
     DB "**  ** **** **      **  **            *     **          **", 0dh,0ah
     DB "**  **   *  **     ********         *       ** ******   **", 0dh,0ah
     DB "**  **      **    **      **      *         **          **", 0dh,0ah
     DB "**  **      **   **        **   *********   *********   **", 0dh,0ah
     DB "**                                                      **", 0dh,0ah
     DB "**                                                      **", 0dh,0ah
     db "****************YOU ESCAPED FROM THE MAZE****************$", 0dh,0ah
```

```
hlt                                  ;Halts code

move_maze endp            ;End program
```

Plagiarism Report



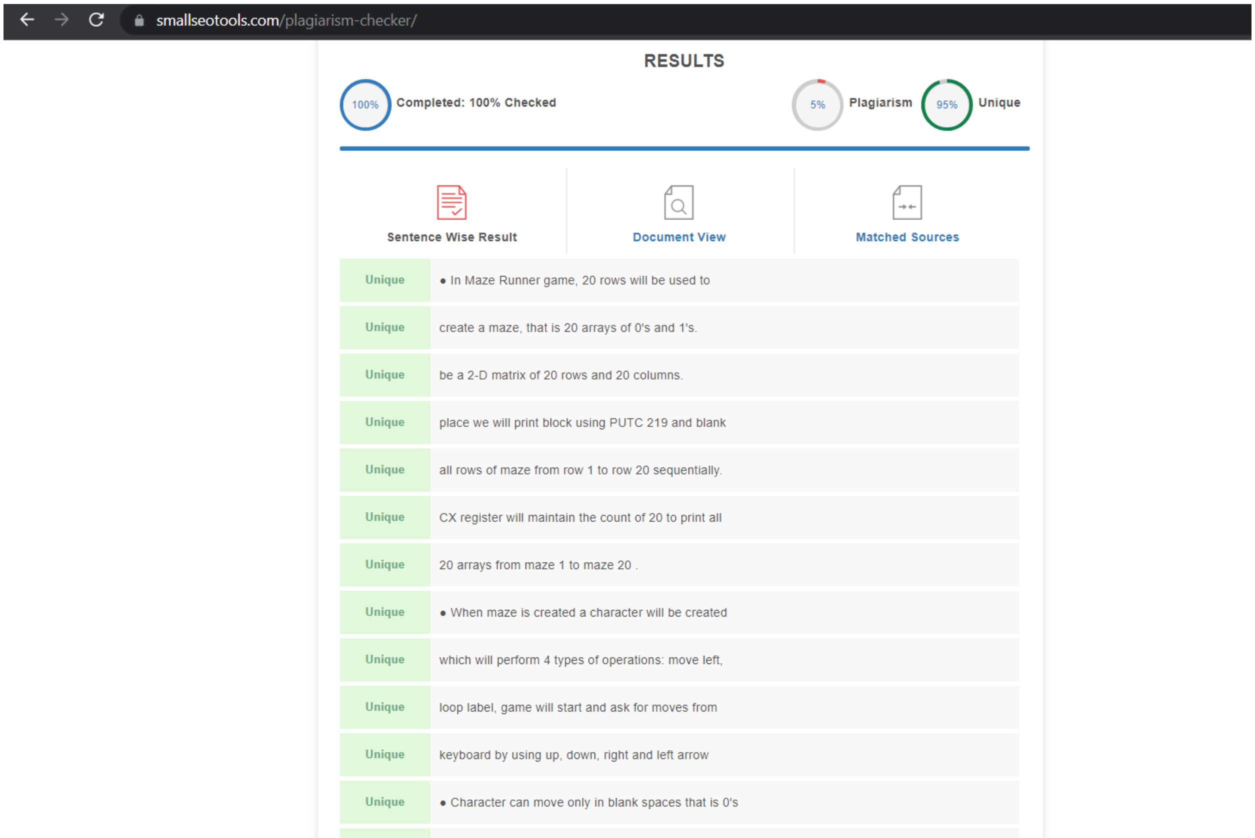
School of Computer Science and Engineering

TITLE: THE MAZE RUNNER

SUBMITTED BY
Kartik Khanna 19BCE0531
Anushka Dixit 19BCE0577
DarashanChaurasia19BCE0509

Limit/Search: 1000 Words

Total Words: 954



RESULTS

Completed: 100% Checked

Plagiarism 5%

Unique 95%

Unique	• In Maze Runner game, 20 rows will be used to
Unique	create a maze, that is 20 arrays of 0's and 1's.
Unique	be a 2-D matrix of 20 rows and 20 columns.
Unique	place we will print block using PUTC 219 and blank
Unique	all rows of maze from row 1 to row 20 sequentially.
Unique	CX register will maintain the count of 20 to print all
Unique	20 arrays from maze 1 to maze 20 .
Unique	• When maze is created a character will be created
Unique	which will perform 4 types of operations: move left,
Unique	loop label, game will start and ask for moves from
Unique	keyboard by using up, down, right and left arrow
Unique	• Character can move only in blank spaces that is 0's