**EE2003 Information Security – Spring 2023** **Section L, M, N**

# Term Project **Weightage: 10%**

| Phase 1: Due 30th April | Phase 2: Due 14th May |
|---|---|

## Group Work

Project can be done individually or in a group of two students.

## Task

For this project, you will develop a simplified version of a classic DOS game Boulder Dash in assembly language. Read through the above link, watch a gameplay, and play it yourself to get an understanding of the rules and terminology. Our simplified version will only have a subset of game features though.

Basically, we have a character named 'rockford' that is navigating underground 'caves' to reach an exit door. Caves contain 'dirt' which rockford can easily dig through as it moves. There are several 'boulders' in the cave which he should be careful about. In case he walks to a location directly below a boulder, he will get crushed and die. There are 'diamonds' scattered throughout the cave which he can collect to gain some points before heading to exit.

To keep the code easy, we will adopt some simplified rules. In the original game, rockford can quickly move away before getting crushed by a boulder, i.e. boulders take time to fall down. In our game, as soon as rockford moves under a boulder, it will immediately die. Furthermore, unlike the original game, we will allow boulders to remain stacked on top of each other.

## Program requirements

WATCH THE DEMO VIDEO

### Phase 1

At the program starts, read the cave information from a text file. A sample file cave1.txt is provided on google classroom. The file would contain 20 lines of 78 columns each, defining the structure of cave as per the mapping below. Exactly one 'R' and one 'T' is expected in the file.

| | | |
|---|---|---|
| x = dirt | R = rockford initial location | T = target exit door |
| B = boulder | D = diamond | W = wall |

Note that as per MS-DOS conventions, each line in text files ends with two invisible characters: CR (ASCII code 13) and LF (ASCII code 10). Hence the file size would be exactly 1600 bytes: 20×78=1560 plus 20 CR plus 20 LF characters.

You should catch and report two file reading errors (1) non-existent input file (2) files with insufficient data (less than 1600 bytes). For basic program functionality, you do not need to validate the data any further (e.g. checking for unknown characters or checking correct structure).

Having successfully read the cave file, you will draw the game layout (title, headers, footers, boundary walls) and the cave internals as per the file contents. The cursor should be hidden before doing these drawings.

Finally, keep the program visible until the user quits it with the Esc key.

In addition to the sample cave file given to you, make sure you test the program with new cave files with different contents.

You are free to customize the game look and feel with different colors.

## Phase 2

In this phase, you will implement the actual gameplay. The program will accept user input in the form of arrow keys to move rockford. For each arrow key press, you will move rockford in the respective direction, if possible. Obviously, it can not move into a wall or boulder. After taking each movement step, the program should check the consequences of that move:

- Increment score if rockford collected a diamond, or
- Finish the game on reaching the target, or
- Game over by boulder crush

At any point user can press the Esc key to exit the program.

## Extra enhancements

Once you have completed the basic functionality implemented (as shown in demo video), proceed to enhance the game with extra features. Below are some suggestions with increasing levels of difficulty.

- Allow user to restart after game ends.
- Display an initial splash screen.
- Allow multiple lives for rockford.
- Stricter validation of cave file structure and contents.
- Multiple levels: user proceeds to next level after completing one.
- Keeping time: user gets extra score if they complete before time, or lose if they run out of time.

You can get more ideas from the detailed rules of the original game.

# Hints and Tips

## 1. Splitting code in multiple files

When your code gets long, splitting it to multiple files will make it more manageable. You can put different logical parts (e.g. file reading, game layout, keys handling) into separate files. Nasm supports the following syntax to include another file in current.

```
%include 'other_file.asm'
```

Upon assembling, the included file is effectively copy pasted into the current file. So, it is ok to declare variables in one file and use them in another.

## 2. Allocating a large block of memory

In NASM, we can use the "times" directive to allocate memory blocks.

```
buffer:   times 2000 db 0    ; allocate 2000 bytes
```

## 3. Useful BIOS and DOS Interrupts

Utilizing BIOS or DOS interrupts will ease several tasks. A shortlist is given below for your convenience. You can also use other interrupts not listed here.

| Task | int | Service | Documentation |
|---|---|---|---|
| Print string | 21h | AH=09h | Ralf Brown List |
| Get string input | 21h | 0Ah | RBL |
| Open file | 21h | 3Dh | RBL |
| Read an opened file | 21h | 3Fh | RBL |
| Close file | 21h | 3Eh | RBL |
| Change cursor shape or hide it | 10h | 01h | RBL and wiki |
| Set cursor position | 10h | 02h | RBL and wiki |
| Wait for keystroke | 16h | 0h | RBL and wiki. Also see this SO answer |

## 4. Extended ASCII chart

All printable chars can be seen in the chart on Wikipedia. Characters used in demo video are below.

| | |
|---|---|
| **Dirt** | B1h |
| **Rockford** | 02h |
| **Target (exit)** | 7Fh |
| **Boulder** | 09h |
| **Diamond** | 04h |
| **Wall** | DBh |

## 5. Making the bell sound

Just print a string containing the bell character (ASCII 7). Take care it should not overwrite any existing text — yes even though bell character is not visible, it takes space.

## Deliverables

For each phase, submit your complete assembly code. If you created multiple files, zip them before uploading. Name the starting file ‘main.asm’.

## Marks Breakdown (tentative)

You will be interviewed to make sure you haven’t plagiarized. Final mark will depend on how well you are able to explain your code.

Both members of a group are expected to understand the whole code.

**D1 (4 absolutes)**

10 file name input

10 opening file, reading data, then closing it

10 drawing static layout

10 drawing cave layout according to file

**D2 (6 absolutes)**

15 moving rockford in response to arrow keys, preventing impossible moves

15 implementing game won, game lost and diamond points

10 game extra enhancements - actual marks will depend on the effort involved

10 code style: variable naming, file organization, comments, subroutine structure