UEFI-game with bootloader functions

Ivan Shevchenko
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine
ivan.shevchenko@ucu.edu.ua

Viktoriia Kocherkevych
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine
v.kocherkevych@ucu.edu.ua

Yaroslav Klym
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine
yaroslav.klym@ucu.edu.ua

Volodymyr Kuchynskiy
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine
vokuchynskiy@gmail.com

***Abstract*—This report demonstrates and describes the implementation of three different games with boot managing capabilities for the UEFI environment. The project's objective was to explore the feasibility of game development based on the EDK-II framework for UEFI. Three games were implemented, namely Wordle, 2D Maze and 3D Maze.**

For code details, visit GitHub Page [1].

***Index Terms*—UEFI, BIOS, boot manager, boot loader, EDK-II**

I. Introduction

In this project, we aim to investigate and describe the aspects of Unified Extensible Firmware Interface (UEFI) development and the usage of low-level programming concepts for developing complicated software.

UEFI [2] – is a specification that defines the architecture used for booting the operating system and the interface for interaction with it.

It must be pointed out that UEFI is not a Basic Input/Output System (BIOS). Though it performs the same functions, it overcomes many of the original BIOS's limitations. The biggest problem with BIOS is that its boot sector has a memory limitation of 440 bytes, which is insufficient to boot a modern operational system (OS). UEFI, on the contrary, has no such restriction, and it also provides network booting, security booting, and other advanced functionality.

To be able to develop for UEFI, we use EFI Development Kit II (EDK-II) [3], [4] – an open-source modern firmware development environment that provides necessary tools for writing UEFI packages, modules, applications, and drivers [5].

In this project, our goal was to explore the capabilities of EDK-II. We have created three UEFI applications: three games, each implementing boot manager functionality. Specifically, they are Text-based User Interface (TUI) Wordle and 2D and 3D Graphical User Interface (GUI) Mazes.

II. Implementation details

A. EDK-II

As was stated above, we use EDK-II as our development environment. The first vital abstractions of this environment are module and package. The module is the smallest separately compilable part of the code, while the package groups zero or more modules. Modules and packages are configured using special module metadata (.inf), package metadata (.dec), and platform description (.dsc) files.

EDK-II is an open-source implementation of the conventions specified in the standard (UEFI). That includes all services, protocols, calling conventions, etc. However, alongside the specifications, an environment also implements a handful of libraries designed to encapsulate low-level protocols and provide a high-level abstraction, often mimicking interfaces present in the standard C library. For example, *PrintLib* provides the functionality of the EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL by wrapping it with the interface of the *printf* function from the standard C library.

B. Boot manager vs boot loader overview

It is important to note that we are implementing the boot manager and not the boot loader in this project.

The boot loader is the program responsible for loading the computer's operating system. On the other hand, the boot manager is a program responsible for providing an interface for choosing between different boot loaders on the computer if there is more than one.

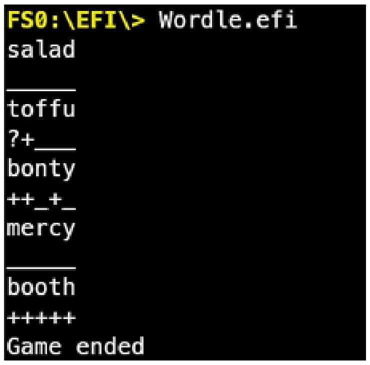
The main tasks of a boot loader are to prepare an environment for the operating system kernel and to load the kernel into the memory, while the boot manager's tasks are to find boot entries, provide an interface for choosing between them, and launch the respective one. It is worth noting that each boot entry can be both a boot loader and a UEFI executable that returns control to the parent process upon exiting, contrary to the boot loader that launches the OS and, therefore, never exits.

C. Boot managing

In our games, we explored two different ways of boot manager implementation.

1) Built-in boot manager: This method uses capabilities of *UefiBootManagerLib* EDK-II library. The program extracts boot entries from Non-Volatile Random Access Memory (NVRAM) and chooses the hard-coded options out of them. This approach has several disadvantages. Firstly, we must rewrite and recompile our program whenever a new boot entry is added. Secondly, frequent rewriting of NVRAM can damage it.

2) EFI system partition (ESP) boot management: This approach uses configuration files to access boot entries. These configuration files provide information about the Globally



```
FS0:\EFI\> Wordle.efi
salad
_____
toffu
?+___
bonty
++_+_
mercy
_____
booth
+++++
Game ended
```

Fig. 1. Wordle Game.

Unique Identifier (GUID) of the partition and the path to the location of the targeted executable that should be launched alongside the parameters that should be passed. In our implementation, one can also specify the path to the icon for each boot entry that will appear in the game. This approach is much better, as it is entirely user-customizable, and there is no need for NVRAM rewriting.

III. TUI GAME

The first stage of our project was focused on getting familiar with the UEFI environment. After setting up the environment, we implemented our first application – a simple text game analog to a popular Wordle game [6]. Upon the start of the game, the random 5-letter word is chosen, and the player gets six attempts to guess it. After each guess, the game responds to the player, specifying letters placed in the right and wrong places and letters not present in the word.

In our implementation, all possible words are read from the configuration file **words.txt** located in the file system's root directory. After entering the word, each letter is marked with "**+**" sign if it is in the right place, with "**?**" if it is in the wrong place, and with "_" if the letter is not in the word. The figure 1 shows an example of the game.

IV. 2D GUI GAME

The next step was introducing graphics to our game. For that purpose, we use EFI_GRAPHIC_OUTPUT_PROTOCOL. The idea behind this protocol is to represent a monitor as a 2D matrix of pixels with the ability to interact simultaneously with whole sub-matrices of pixels.

Our maze is divided into square cells, each of which is a path, a wall, or an exit, as shown in figure 2. In one of the path cells, there is a sprite that can be controlled by the user. Each cell is displayed in one go using the protocol described above.

After displaying the whole labyrinth at the beginning, until the sprite is not on the exit cell, the program listens for keys the user presses. If *Escape* was pressed, the program exits, returning the control to the parent process, and if one of the direction keys was pressed (*UP*, *RIGHT*, *DOWN*, or *LEFT*) the actor attempts to move in the specified direction. If the new cell has a path or exit, the attempt is considered successful, and the program redraws exactly two cells: the old cell is filled with black pixels, and the new cell is filled with the sprite image. Otherwise, the keypress is ignored.

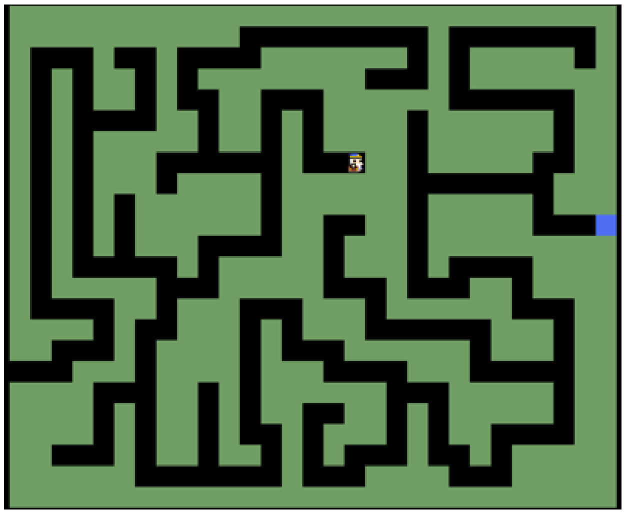


Fig. 2. 2D Labyrinth Game

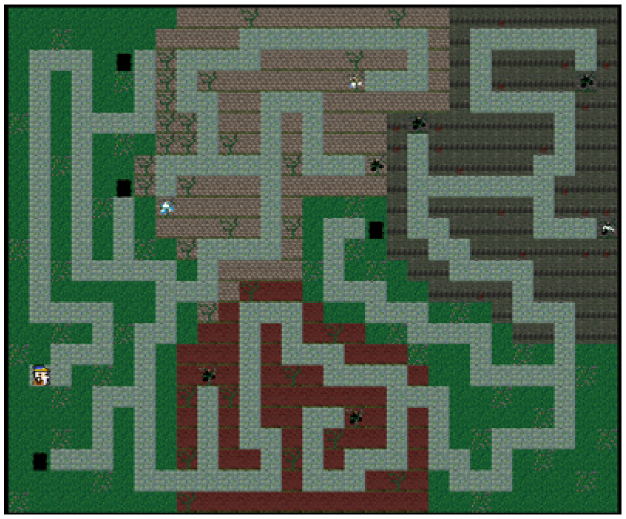


Fig. 3. 2D Labyrinth Game Updated

Upon exiting the labyrinth, the built-in EDK-II boot manager (II-C1) is used.

By the end of the project, we had updated this game with two main features: updated graphic with textures (displayed in the figure 3) and switched to ESP boot managing (II-C2) as it is more configurable.

V. 3D GUI GAME

This game is the 3D representation of the previous 2D labyrinth with the same ability to boot the operating system the user finds in the maze. In the figure 4, there is an example of an exit that boots Alpine [7] – a lightweight OS.

To create 3D graphics for this game, we used ray casting [8]. Ray casting creates a 3D perspective of the 2D map. The main

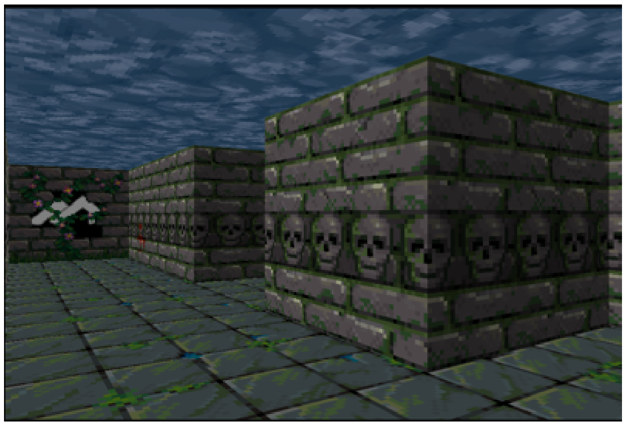


Fig. 4. 3D Labyrinth Game

idea behind the algorithm is to go through every vertical stripe of the screen, calculate the distance to the nearest wall, and, depending on the distance and direction of the view, draw a vertical line representing the wall. As we also used textures for our game, during drawing, we colored them respectively to the scaled columns of texture.

The maze itself implements the functionality of an ESP boot manager II-C2 and, upon escape, launches the respective executable specified in the configuration file.

VI. Conclusions

In conclusion, we dived into the specifics of the development for UEFI based on the EDK-II firmware. We managed to create TUI and GUI games for UEFI, which perform the functionality of a boot manager and can be launched not only in Quick Emulator (QEMU) but also directly in the hardware.

References

[1] Y. Klym, V. Kocherkevych, and I. Shevchenko, "UEFI-game with bootloader functions," Jan. 2024. [Online]. Available: https://github.com/ishevche/UEFI-Game.git

[2] *Unified Extensible Firmware Interface (UEFI) Specification*, Aug. 2022.

[3] "EDK-II Source Code." [Online]. Available: https://github.com/tianocore/edk2

[4] "EDK-II Wiki." [Online]. Available: https://github.com/tianocore/tianocore.github.io/wiki

[5] TianoCore, "EDK-II Module Writer's Guide." [Online]. Available: https://tianocore-docs.github.io/edk2-ModuleWriteGuide/draft/

[6] "Wordle Game." [Online]. Available: https://wordlegame.org/

[7] "Alpine linux." [Online]. Available: https://alpinelinux.org/

[8] "Raycasting Graphics Tutorial." [Online]. Available: https://lodev.org/cgtutor/raycasting.html