EEET2490 – Embedded System: OS and Interfacing, Semester 2025-1

Group Assignment Report

CLI, SCREEN DISPLAY, AND APPLICATION DEVELOPMENT FOR A BARE METAL OPERATING SYSTEM

Lecturer: Dr Linh Tran – linh.tranduc@rmit.edu.vn,

Team Number: 6

Team Members:

Huynh Ngoc Tai (s3978680)

Tran Quang Minh (s3988776)

Kim Nhat Anh (s3978831)

Vu Thien Minh Hao (s3988776)

Date : 26/05/2025

TABLE OF CONTENT

I. INTRODUCTION

This report documents the development of a bare-metal operating system running on Raspberry Pi. The assignment focuses on three major tasks: implementation of a welcome message and command line interface (CLI), display of images/videos/text on screen, and the development of an embedded game application. Our objective was to demonstrate understanding of OS-level features, low-level interfacing, framebuffer manipulation, and user interaction design.

II. WELCOME MESSAGE AND COMMAND LINE INTERPRETER (CLI)

Requirement: Implement a CLI system with features such as welcome message, command parsing, history, and execution for basic system commands like help, clear, showinfo, etc. Features like auto-completion and command history were to be added using special key handling.

Implementation:

- **Welcome Message:**

  An ASCII art banner is printed to UART upon boot, giving the OS a recognizable identity.

- **CLI Prompt:**

  The system consistently shows the prompt "FixingGoodOS>" to signal readiness.

- **Command Handling:**

  Input is collected using a character buffer commandBuffer, and processed when Enter is pressed.

  Autocompletion is triggered when the Tab key (\t) is pressed. The implementation checks for prefix matches among available commands:

  History is stored in a ring buffer. Pressing “-” and “+” to navigates back, while “=” navigates forward. The buffer is managed to overwrite the oldest command if full. (We don’t use “-” and “+” explicitly because it would require an extra Shift on keyboard for it)

  Backspace (127 or \b) removes the last character and redraws the line.

- **Commands Implemented:**

  -help: Lists all commands or detailed info if followed by a command name.

  -clear: Clears terminal screen using ANSI escape codes.

  -showinfo: Reads MAC and board revision from specific memory locations.

  -baudrate:

  Accepts integer input and sets UART baudrate via predesigned divider calculation.

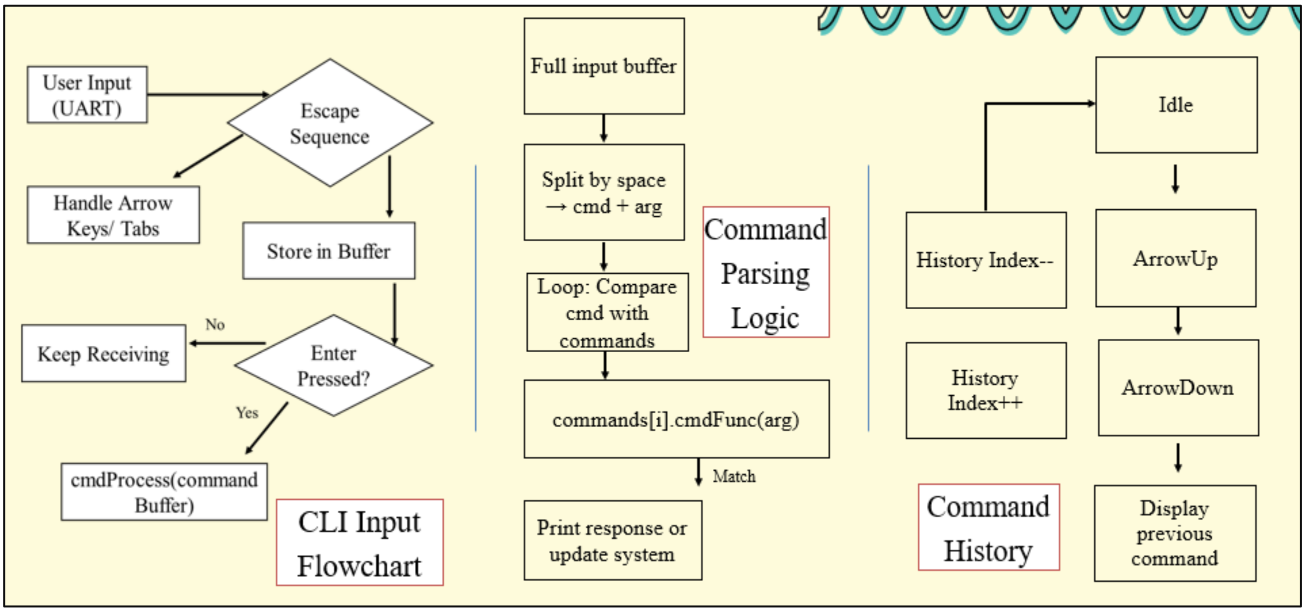
  -handshake:

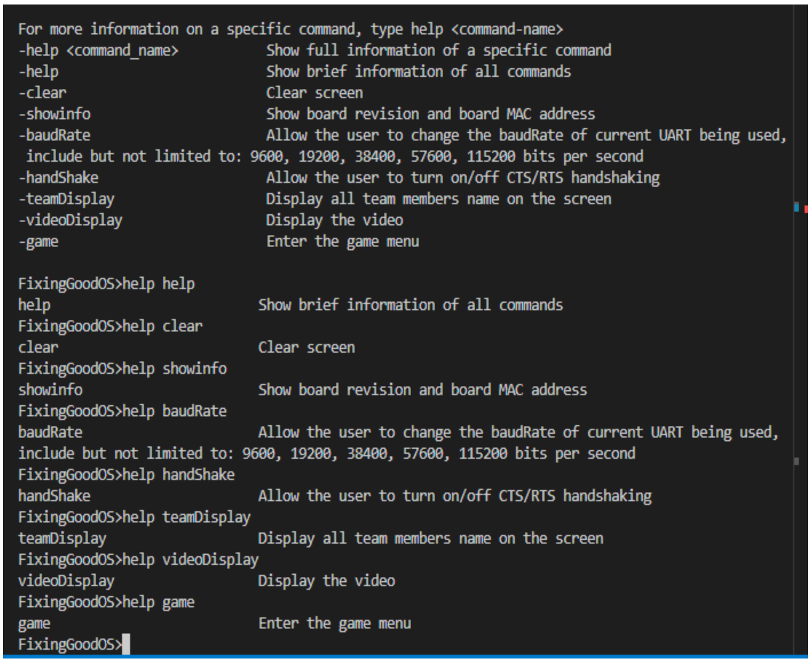
  Enables/disables CTS/RTS using GPIO 16/17 setup.

  -teamDisplay and videoDisplay are integration commands used to demonstrate Task 2.

Code Result Discussion: All CLI features function correctly on UART0. The auto-completion significantly improves usability. Command history is intuitive. Limitations include no fuzzy-matching or command arguments parsing.

Summarized program Flowchart/Diagrams:





III. IMAGE, VIDEO, AND TEXT DISPLAY

Requirement:

a) Show names of team members with colored text over a background image. Custom font must be used.

b) Display a short video as sequence of images.

Implementation:

- **Font:**

  We implemented a custom font system where each character glyph is represented as an 8x15 bitmap. These bitmaps were extracted from the Tamzen8x15.bdf file in the Tamzen font repository - https://github.com/sunaku/tamzen-font/tree/master/bdf.

  The BDF file was parsed and converted into a C-style header file containing a two-dimensional array, where each glyph is stored as a sequence of bytes. During rendering, each bit in the glyph data is read and drawn to the screen pixel-by-pixel using drawPixelARGB32().

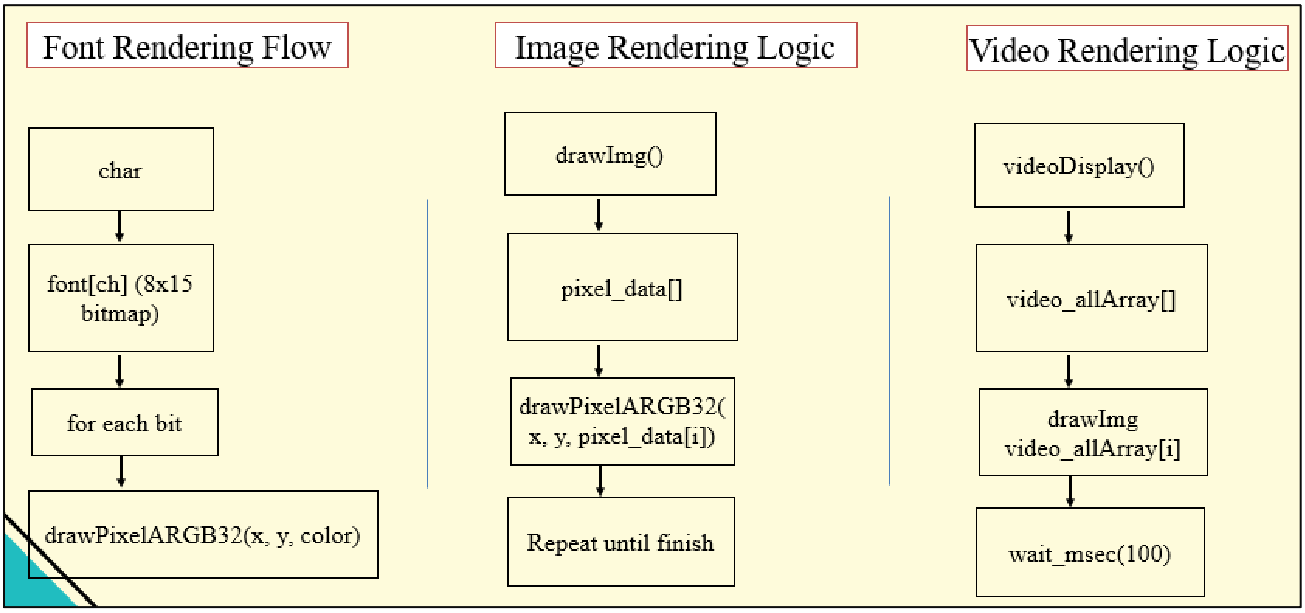
- **Background Image:** An image was converted to ARGB32 using https://javl.github.io/image2cpp/ and displayed full-screen on qemu.

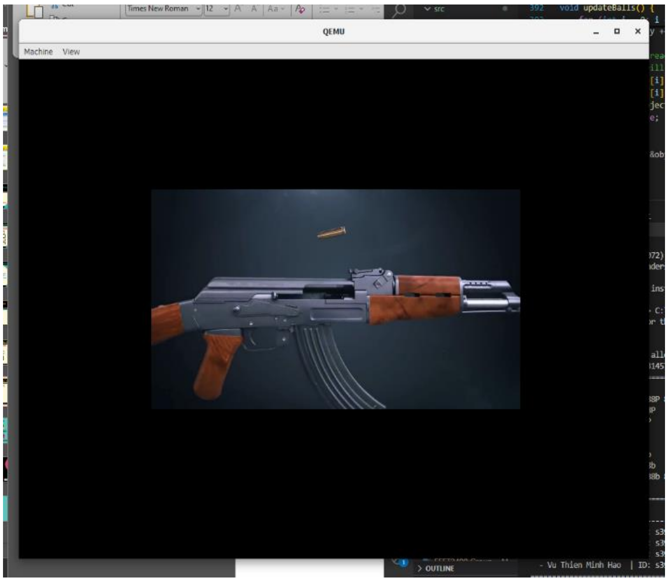
- **Text Overlay:** Each team member's name is drawn in a different color at fixed positions with the background image that's draw beforehand using drawPixelARGB32() function.

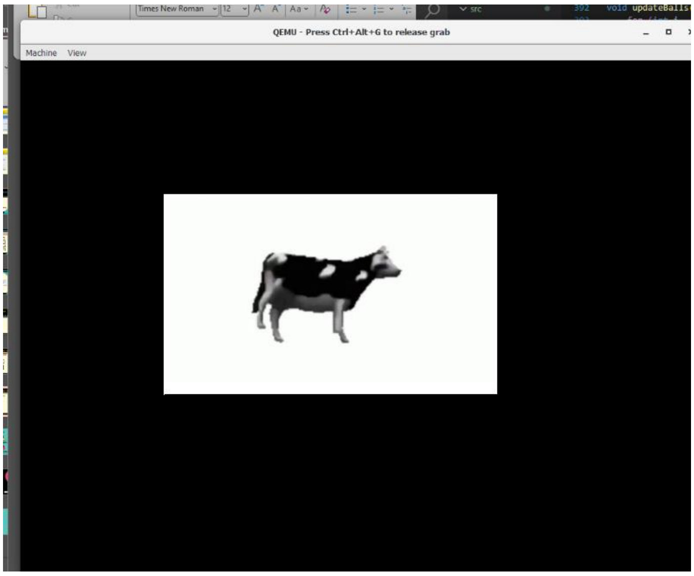
- **Video Display:**

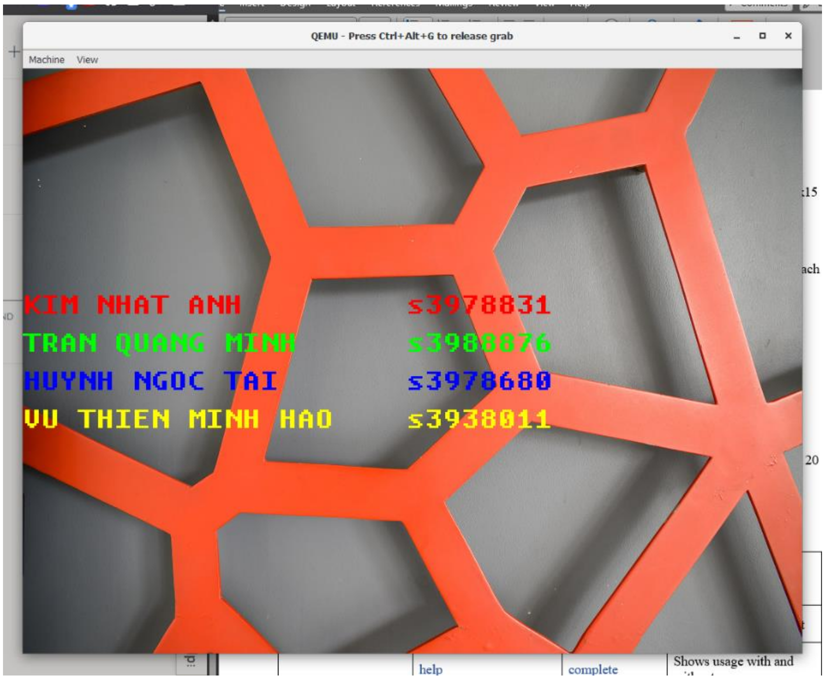
  - Each frame of the video is converted to a C array and stored in video_allArray[].
  - The videoDisplay() function loops through the frames and renders them with wait_msec(100) delay, producing ~10 FPS.
  - Video playback can be interrupted with Enter key via UART.

Summarized program Flowchart/Diagrams:









Code Result Discussion:

- For the teamDisplay, our team sucessfully display the team name + sid with the custom on the RMIT background, which was rendered beforehand.

- For the videoDisplay, our team decided to display 2 videos, which is the cow dancing video and the gun reloading, also addition to that, we managed to put the video on loop until you press Enter.

Overal, text and images render as expected. Video playback is smooth but limited to 30 frame for the cow video and 80 frames (10fps) due to processing overhead.

Summary of features implemented in both Tasks 1 & 2

Feature Group	Command/ Feature	Implementation	Testing (any issues/limitations)
CLI Basic Features	welcome screen	complete	Displays on UART boot
	help	complete	Shows usage with and without args
	clear	complete	Clears screen using ANSI codes
	showinfo	complete	Displays MAC and board revision
	baudrate	complete	Changes UART0 baudrate live
	handshake	complete	Enable the handshake mode which enable to use the CTS, RTS mode to communicate with the board.
CLI Enhancement	OS name in CLI	complete	Static prompt FixingGoodOS>
	Auto-completion in CLI	complete	Matches prefix and fills input
	Command history in CLI	complete	Works with "_" (going back) and "+" (going forward)
Image, Video, and Text Display	Background image and text display	complete	Shows names of 4 members in color with the RMIT background.
	Video display	complete	Plays 2 videos (cow video and gun reloading video) based on input argument. Videodisplay <cow/ak>

=> All of the function works on both the qemu and the real board, ensure the code flexibility on cross platform.

IV. APPLICATION DEVELOPMENT

Game: Basketball Star

Gameplay:

- Player controls a basketball hoop at the bottom of the screen.
- Balls fall from the top. Player must move left/right to catch them.
- Types of balls:
    - NORMAL → +10 points
    - SPECIAL → +30 points
    - BOMB → -100 points
    - ENLARGE → enlarges hoop temporarily
    - MULTIPLY → doubles score temporarily

Game Structure:

- Game loop continuously updates player position and ball states.
- UART input ('a', 'd') moves player.
- Collision is checked using AABB (axis-aligned bounding box).
- Menus support ESC to pause, with options: Continue, Restart, Exit.
- Game ends when stage score is achieved or lives run out.

Design:

- **Background:** Static image displayed depending on which level you're on.
- **Sprites:** Hoops and balls are drawn using pixel data.
- **Physics:** Balls fall vertically; collision is checked against the hoop.
- **Controls:** Arrow keys are mapped through UART for player movement.
- **Game Logic:**
    - Score increments or decrements based on ball type.
    - The player will lose when the score is -100 points or the time limit was reached.
    - Player will win the game when he/she passed all 3 levels implemented.
- **Communication:**
    - UART logs each command received.
    - Acknowledges valid commands with ACK.
    - Logs current score and number of balls caught.

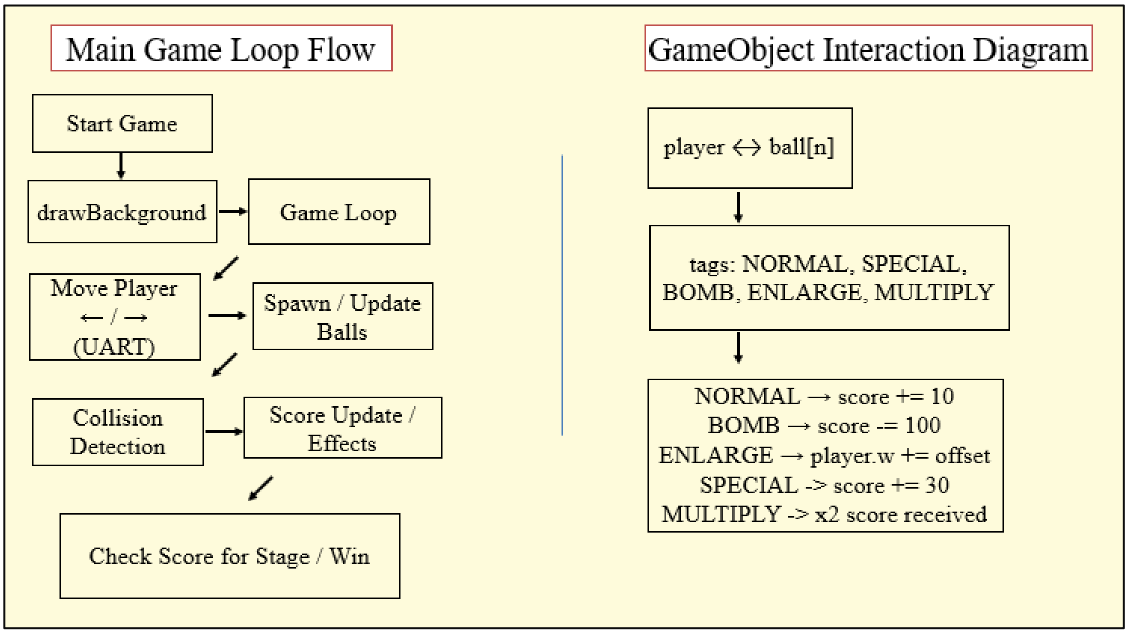
Code Result Discussion:

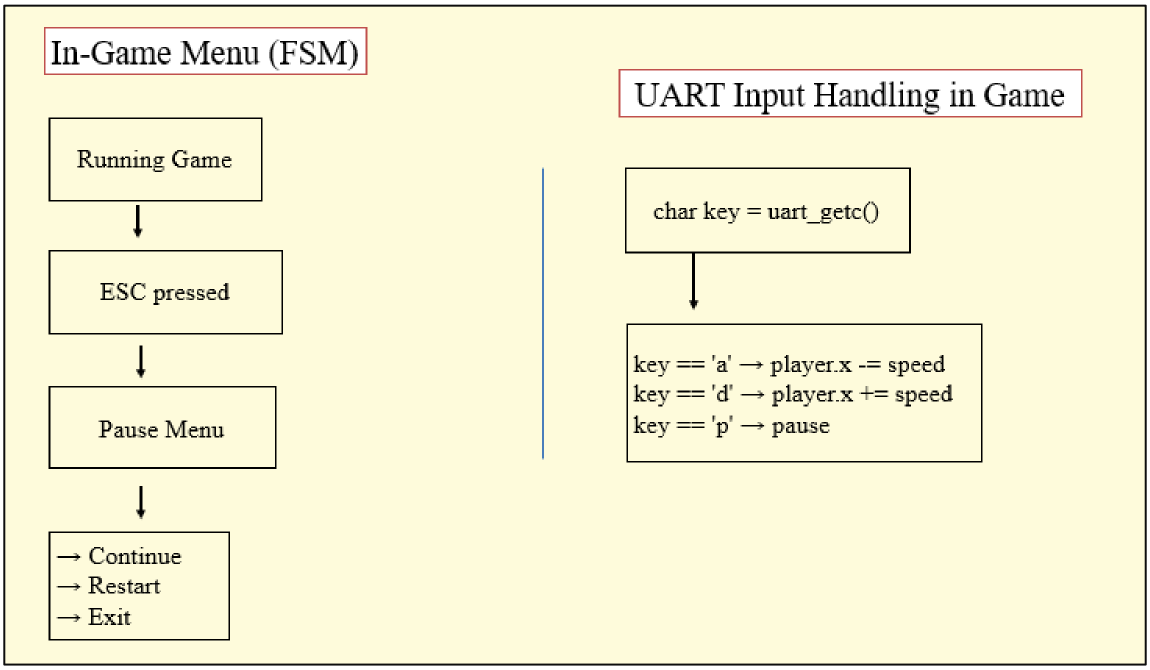
The game was developed and divided into stages, which incremented with the topic of the game revolving around a player who aimed to be a professional player (three stages: from the streetball players, to the high-school players and to the NBA players), also with the stage change, the droprate of the basketball will be tuned to be more difficult.

Game is functional with responsive controls and decent visuals, we also try to add in some plots to provide the game with a deeper meaning. Stop/Resume mode was also added to make the game user-friendly.

(More image of game asset is in the appendix)

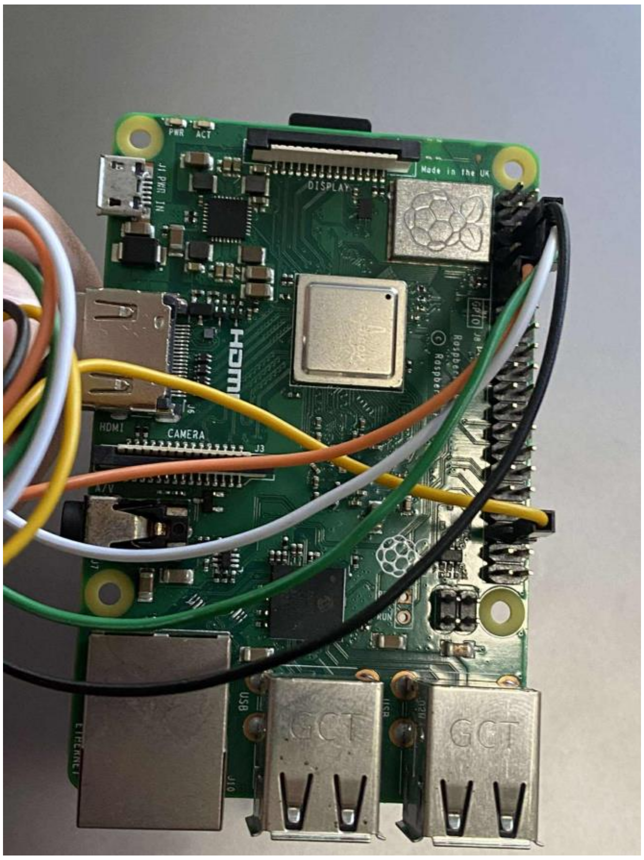
Summarized program Flowchart/Diagrams:





V. TESTING ON REAL BOARD

In order to use the real board, teraterm and cable connections to the Raspberry Pi board is needed. For Initialization, we use GPIO 14/15 for uart0.



On the real Raspberry Pi board, all features function properly except for the baudRate and handShake commands, which rely on hardware-specific UART configurations that may behave differently from QEMU. Despite this, the rest of the system demonstrates excellent portability across platforms.

To display output on the board, we connect it via HDMI to a computer and use OBS to capture the video feed. This setup allows us to view framebuffer output live — including videoDisplay, teamDisplay, and the game application. The game renders correctly, but may appear slightly laggy due to HDMI mirroring latency introduced by OBS. Overall, this demonstrates the system's compatibility and effectiveness on physical hardware.

VI. CONCLUSION AND LIMITATION

Despite the overall success of the project, several limitations were encountered during development and testing. Additionally, the framebuffer rendering system is entirely software-based and lacks hardware acceleration, which restricts video playback performance to approximately 10 FPS. When testing output on the physical board via HDMI and capturing it using OBS, we observed noticeable latency, particularly affecting the smoothness of the game. The system also does not support audio or external peripherals such as a mouse or keyboard. In terms of gameplay, the difficulty is static and lacks features like dynamic scaling or score tracking. These limitations provide valuable insight for future improvements, including potential real-time scheduling, input abstraction, and richer user interaction.

This assignment provided valuable experience in low-level OS development, framebuffer rendering, and interactive application design. We deepened our understanding of UART, CLI parsing, framebuffer structure, and embedded graphics handling. Working with the Raspberry Pi in a bare-metal environment helped bridge theory and practical implementation. Looking forward, we aim to explore real-time scheduling and multi-threading for more advanced systems.

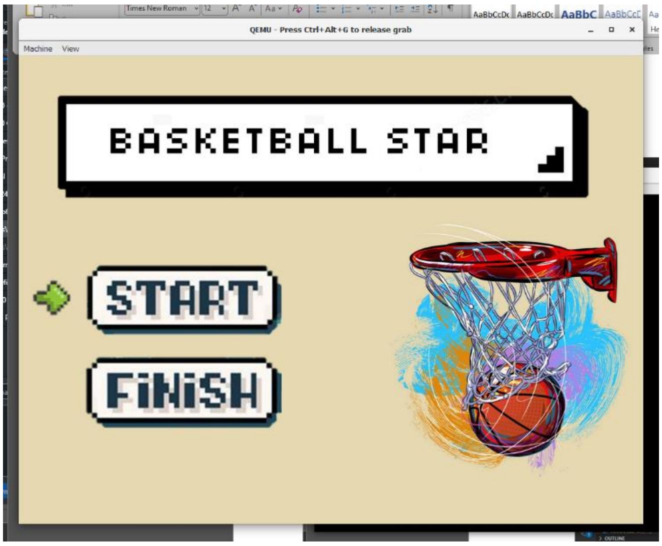
VII. REFERENCES (USE IEEE STYLES)

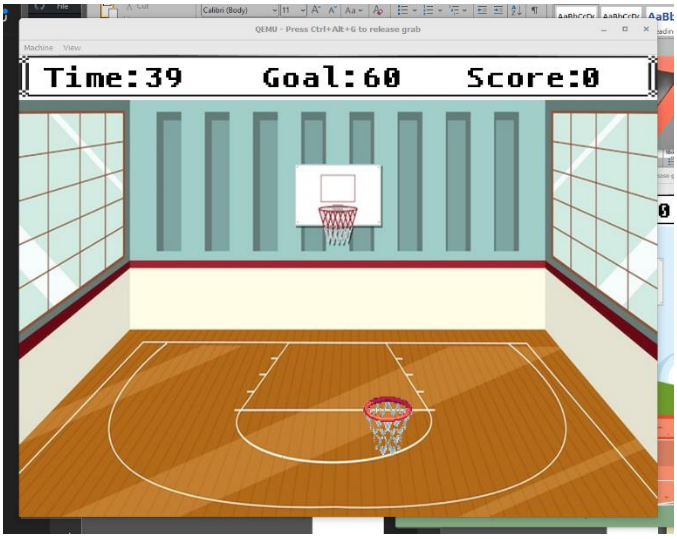
[1] Raspberry Pi Board Revision Info, https://www.raspberrypi-spy.co.uk/2012/09/checking-your-raspberry-pi-board-version/
[2] MAC Address Info, https://www.javatpoint.com/what-is-mac-address
[3] ASCII Art Generator, https://onlineasciitools.com/convert-text-to-ascii-art
[4] Image to C Array Converter, https://javl.github.io/image2cpp/
[5] Scratch Game Tutorials, www.youtube.com/watch?v=jFVJdRLZoQ4, www.youtube.com/watch?v=QXru0rSV2ZQ
[6] PCS Font Tutorial, https://github.com/bztsrc/raspi3-tutorial/tree/master/0A_pcscreenfont
[7] Framebuffer Tutorial, https://github.com/babbleberry/rpi4-osdev/tree/master/part5-framebuffer

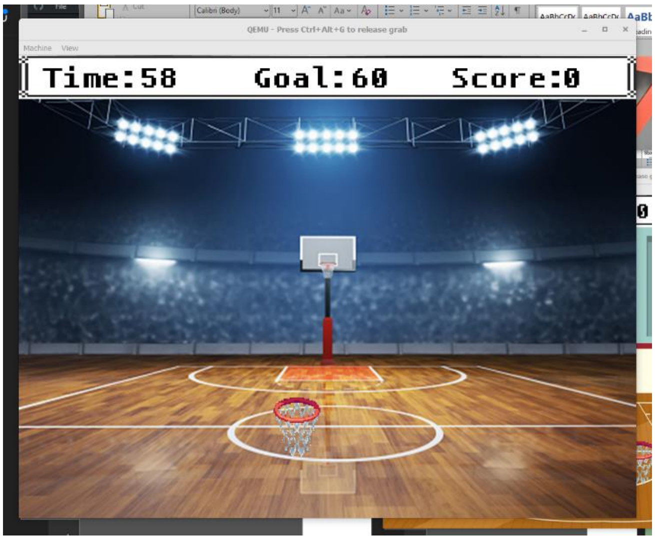
[8] Tamzen-font repository, https://github.com/sunaku/tamzen-font/tree/master/bdf

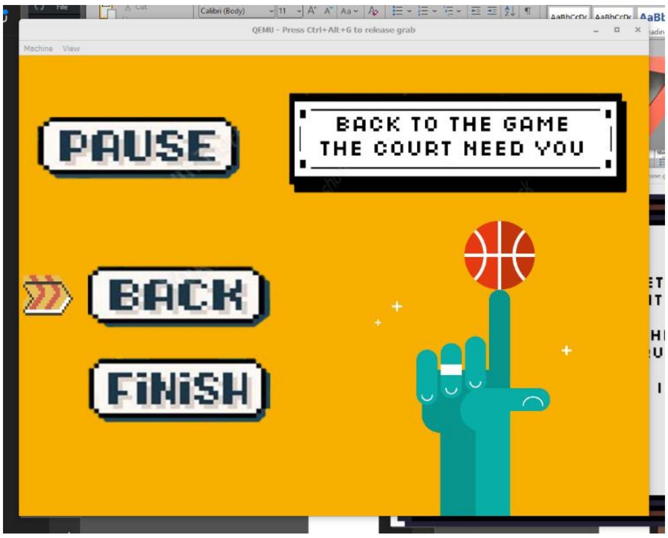
VIII. APPENDIX

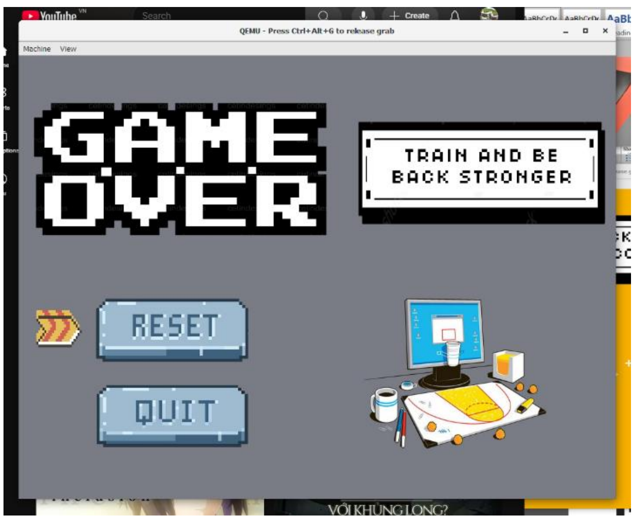
[Game Asset]












Slide4.JPG
Slide5.JPG
Slide6.JPG
Slide7.JPG
x2
Slide8.JPG
Slide9.JPG
BASKETBALL STAR
START
FINISH
Slide10.JPG
BASKETBALL STAR
START
FINISH
Slide11.JPG


STAGE 1:
THE COURT IS CRACKED. THE RIM IS RUSTY. BUT THIS IS WHERE IT STARTS.
NO COACH. NO CROWD. JUST A BALL, A HOOP, AND A DREAM.
OUT HERE, HE LEARNS TO LOVE THE GAME.
PRESS X TO CONTINUE


STAGE 2:
A NEW TEAM. A BIGGER STAGE.
NOW IT'S ABOUT WORKING TOGETHER, LISTENING, AND GROWING.
HE'S NOT JUST PLAYING—HE'S LEARNING WHO HE IS.
PRESS X TO CONTINUE


STAGE 3
BIG ARENAS. BIG MOMENTS.
THE CROWD IS LOUD, BUT HE STAYS CALM.
HE KNOWS THE GAME. HE TRUSTS HIMSELF.
PRESS X TO CONTINUE


x2