# Cheat Engine for Emulator Project – Z5529461

## a) Background

An emulator allows researchers to analyse, test and research software code in safe and controlled environments. It allows us to execute and observe what a malicious piece of software do, enabling us to research it without risking any real machine or system. Because of the sandbox nature of emulator, this allow us to manipulate the system and observe the change to whole system. This give me idea to build cheat engine to a simple game emulator.

The main output of my project is a fully functional **cheat engine** embedded within an SDL2-based Game Boy emulator (LLD_gbemu). This system allows real-time memory scanning, refinement, editing, and freezing of in-game values such as score, health, or inventory.

The cheat engine UI was built entirely using SDL2 rendering and text rendering through SDL_ttf. It can be toggled with a hotkey (F1), and users can interact with it to perform memory searches, write new values to memory, and freeze/ unfreeze memory addresses to maintain fixed values

## b) What did I Do?

My process began with exploring the base code of the emulator from **rockytriton/LLD_gbemu** to understand its memory and rendering systems. Because this emulator uses language c and use SDL library to implements its UI, I build the cheat engine using the same programming language. I then designed and implemented a modular cheat engine (cheat_engine.c) that could:

- Search through memory by value
- Refine previous search results
- Read/write/freeze memory addresses

Next, I built the cheat_ui.c to handle rendering the UI. It displays mode selection, input buffer, and dynamic menus for each function. I handled input with SDL events and ensured state transitions (search → refine → edit) were smooth.

Key time investments:

- Week 1: Familiarized with emulator and SDL2 framework
- Week 2: Built memory scanning and value editing tools
- Week 3: Designed and debugged SDL-based cheat UI
- Week 4: Integrated all systems and tested on real ROMs

## c) What do I Learn?

This project pushed me to work across multiple layers of system programming, emulator development, and user interface design. Some of the key technical and personal lessons I gained include:

- **Understanding Low-Level Memory Manipulation:** I learned how Game Boy WRAM and HRAM function, how addresses are mapped, and how to safely modify values during

runtime. This gave me a stronger grasp of memory models and data formats like BCD (Binary-Coded Decimal), which are not common in modern systems.
- **Working with SDL2 and SDL_ttf:** I became comfortable with rendering graphics, handling fonts, processing keyboard input, and managing stateful UI in SDL. It taught me about practical rendering order, double buffering, and debugging SDL-specific issues like zero-width text or improper render clearing.
- **Debugging and Emulator Stability:** I learned how sensitive emulator loops are to unhandled input or unexpected render states. I also gained experience with segmentation faults caused by out-of-bound memory access and how to trace and fix them systematically.
- **Designing for Usability:** Creating a cheat UI from scratch forced me to consider clarity, simplicity, and feedback for the user. It taught me how even basic UI elements need careful thought to feel intuitive and responsive.
- **Discipline in Code Organization:** By separating cheat_engine.c and cheat_ui.c, I learned to think in terms of modular design. This helped keep the logic manageable and debug-friendly, especially when switching between backend memory operations and frontend rendering.

Overall, this project taught me how to build something real and interactive on top of complex systems. It deepened my interest in reverse engineering, emulation, and systems programming—and showed me that I can tackle ambitious, low-level challenges with enough iteration and persistence.

## d) Challenges and reflection

The biggest challenge was handling **SDL rendering and event handling** within an emulator loop. I had never previously worked with SDL2, so integrating a new UI system while keeping emulator performance smooth was difficult. I had to learn about rendering pipelines, font rendering limitations (like the TTF zero-width crash), and synchronizing user input with emulator states. During programming the project, I encountered bugs such as:

- SDL text rendering failure on empty strings
- UI flickering and state rendering overlaps
- Emulator crashes when toggling UI rapidly

Another major challenge was working with the **Game Boy memory model**, especially formats like **BCD** in Tetris. I learned to read memory maps, trace emulator memory, and convert value formats (e.g. decimal to BCD) correctly to manipulate the game. Through these challenges, I learned:

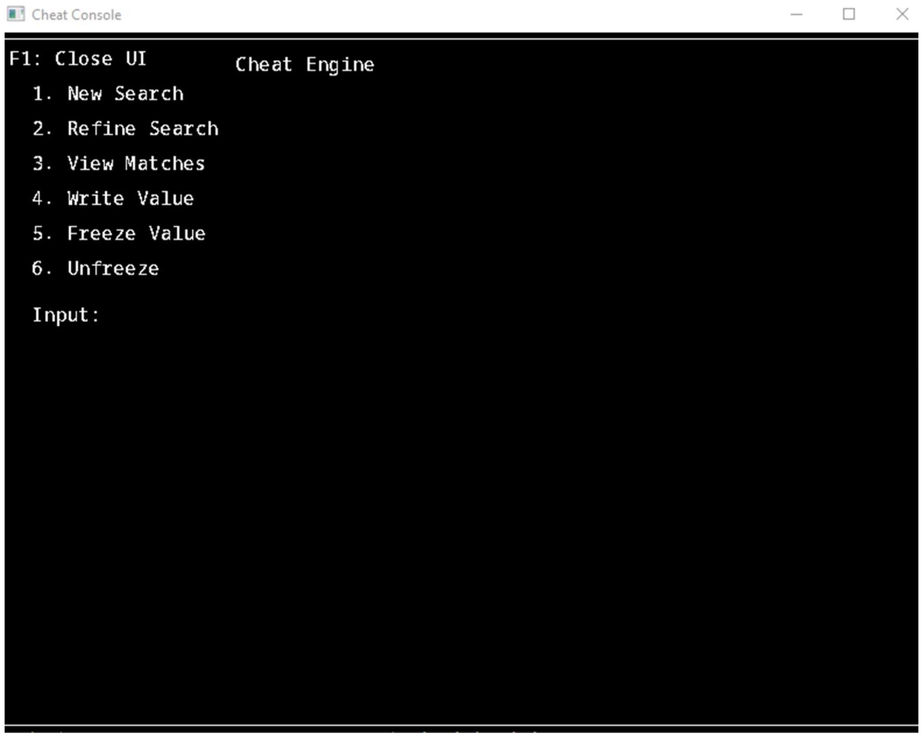
- How to manipulate emulator memory directly and safely
- How to build simple but functional UIs with SDL2
- That cheat systems must account for memory layout, input timing, and system rendering performance

If I could do it again, I would start the UI with a debug console and only move to graphical SDL after confirming logic. I also would have added automated memory testing to reduce crash risk from invalid addresses.

## Reference

1) https://www.youtube.com/playlist?list=PLVxiWMqQvhg_yk4qy2cSC3457wZJga_e5 - **Gameboy Emulator Development** - YouTube Series by Low level Devel -
2) https://github.com/rockytriton/LLD_gbemu - **rockytriton/LLD_gbemu GitHub Repository** - I use this as the base emulator for cpu logic and sdl rendering
3) https://gbdev.io/pandocs/ - To understand the architecture of Gameboy
4) https://lazyfoo.net/tutorials/SDL/ - L**azy Foo's SDL Tutorials -** Official documentation for rendering text in SDL using fonts, essential for cheat UI text.
5) https://wiki.libsdl.org/SDL2_ttf - **SDL2_ttf Library Documentation**
6) https://gamehacking.org/system/gb - GameHacking.org – Game Boy Cheat Database – understanding what kind of cheats apply

## Appendix

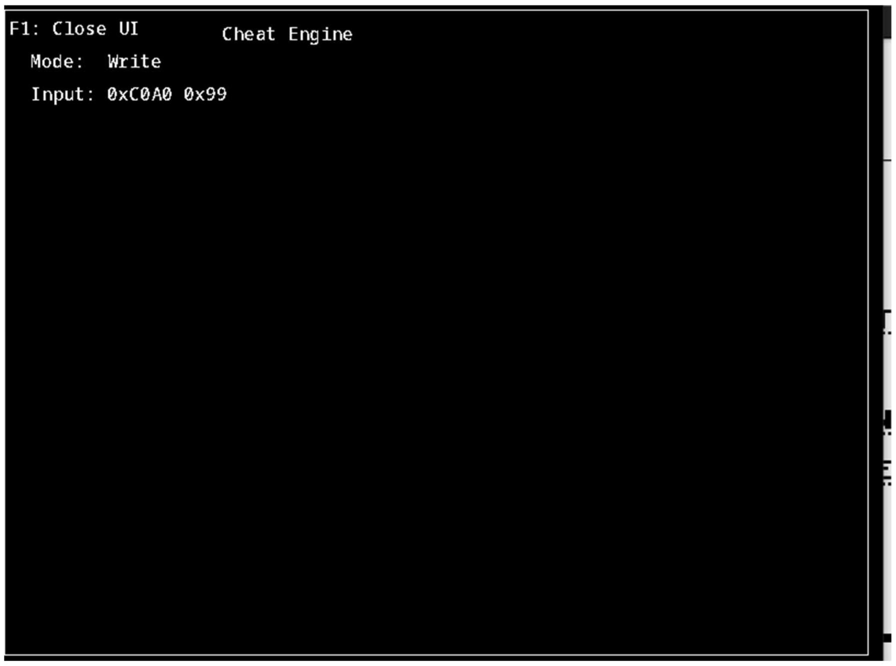


```
Cheat Console

F1: Close UI        Cheat Engine
  1. New Search
  2. Refine Search
  3. View Matches
  4. Write Value
  5. Freeze Value
  6. Unfreeze

  Input:
```

*Figure 1 Cheat Engine UI*



```
F1: Close UI        Cheat Engine
  Mode:  Write
  Input: 0xC0A0 0x99
```

*Figure 2 Writing to game memory*

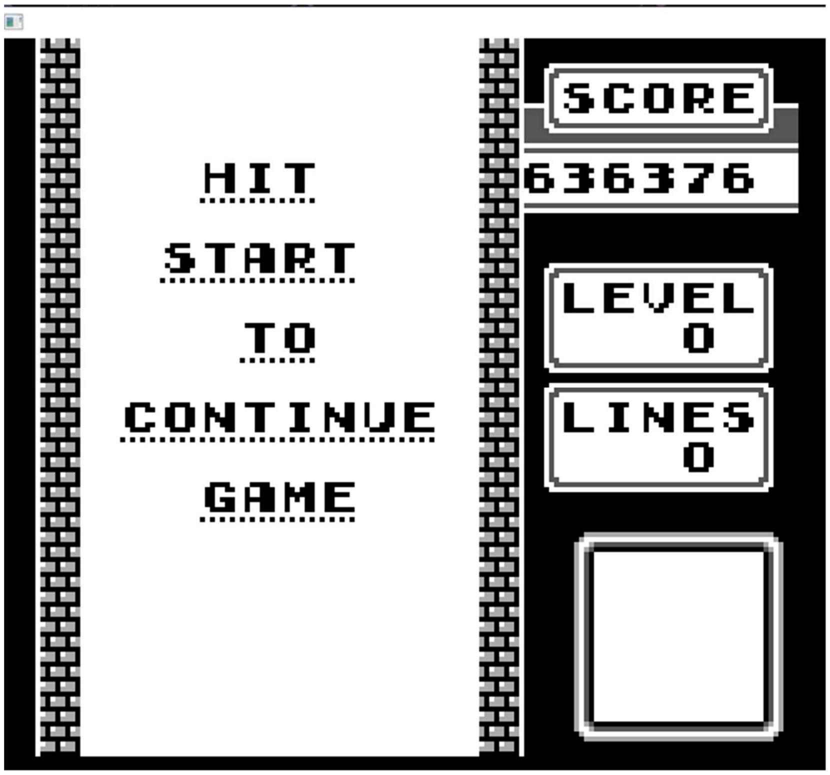


*Figure 3 Tetris Score after using cheat engine*