SCARAB: Design Document

For submission to SETU

Author: Stuart Rossiter

Student Number: C00284845

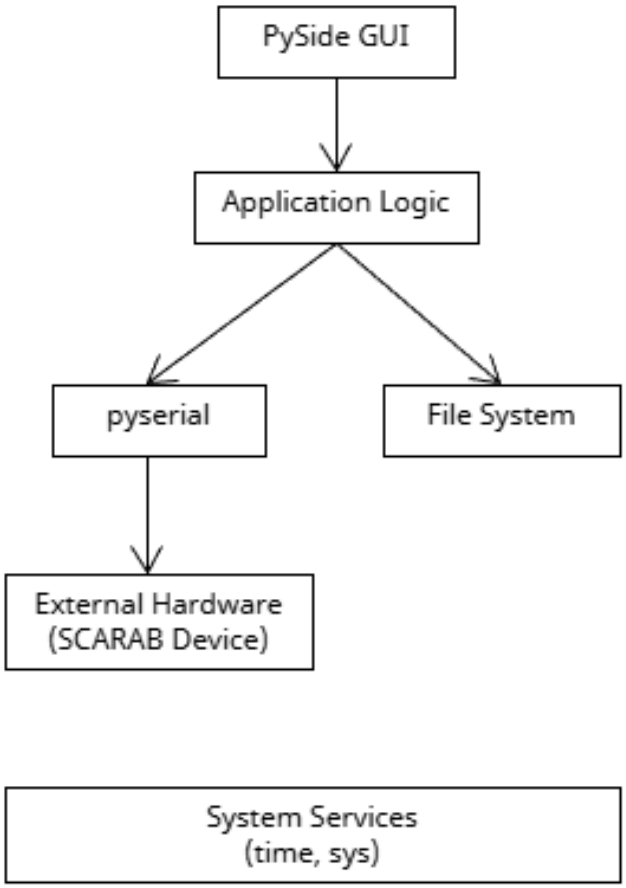
Course: BSc (Hons.) Software Development

Supervisor: Joseph Kehoe

Contents

1. SCARAB Program

1.1 Software Architecture



1.1.1 PySide

For GUI development, I am using Python's own binding of the GUI toolkit Qt, developed by The Qt Company themselves. The binding, known as PySide, is cross-platform, much like Qt. The PySide binding is open source under the GPLv3 and Qt commercial license (pyside, 2021).

1.1.2 Python

The application logic will be handled using Python. Python, created by Guido van Rossum in 1991, is a general-purpose language with a simple syntax, similar to the English language (W3Schools, n.d.). Given its simplicity, it allows for rapid development, aided by its abundance of external libraries.

1.1.3 pySerial

The pySerial library encapsulates serial port access, providing cross-platform backends for Python (pyserial, 2022). This will be used to interface with the SCARAB Device.

1.1.4 File System

The SCARAB program will need to store backed up save files. As such, interaction with the PC's file system is necessary.

1.1.5 External Hardware (SCARAB Device)

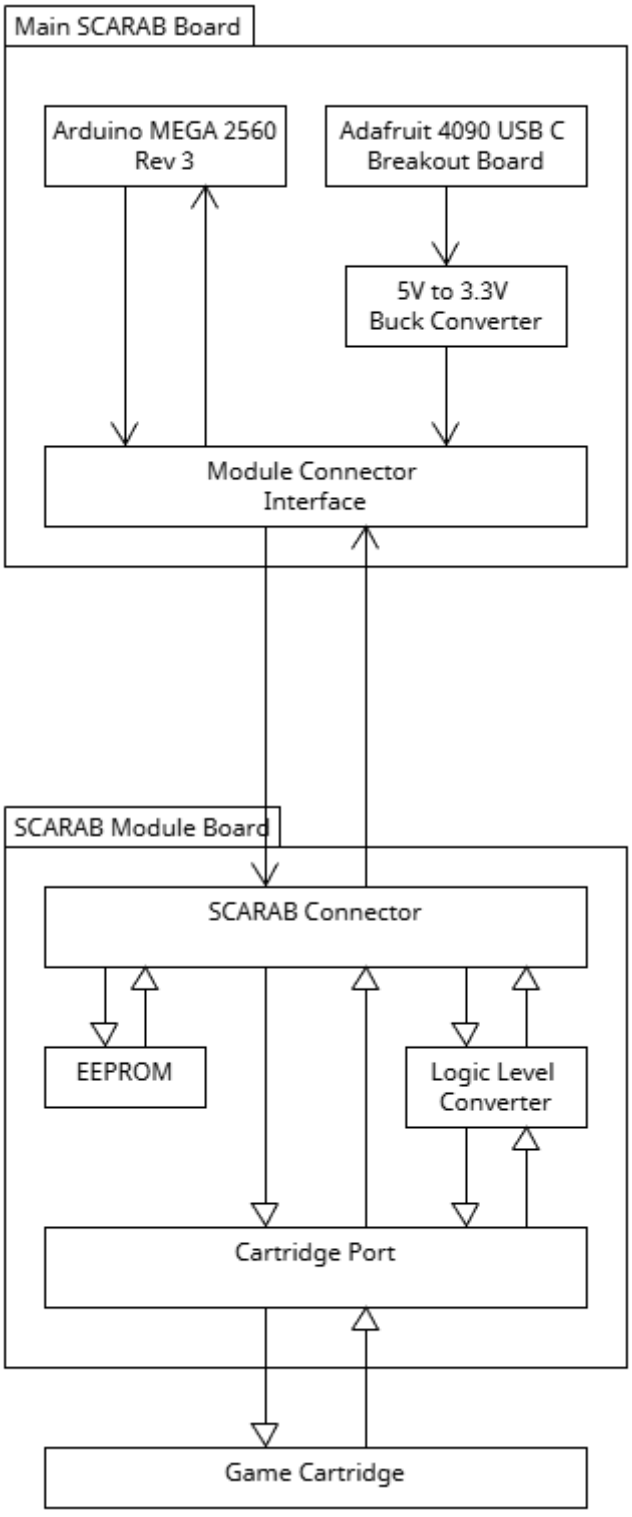
pySerial is being used to communicate with external hardware. In this case, it is communicating with the SCARAB Device itself. Specifically, it is communicating with the Arduino which powers the SCARAB.

1.1.6 System Services

The SCARAB PC Program will need to make use of some system libraries such as time for any necessary delays while the SCARAB Device connects, and sys for exiting.

2. SCARAB Device

2.1 Hardware Architecture



2.1.1 Arduino MEGA 2560 Rev 3

Based on the ATmega2560 microcontroller, the Arduino Mega 2560 Rev3 contains everything needed to support the ATMega2560, including 54 digital I/O pins, 16 analog inputs, 4 hardware serial ports, a 16MHz crystal oscillator, USB connection, and a power jack (Arduino, 2025). The Arduino serves as the intermediary device which translates between what the PC wants, and what the cartridge expects.

2.1.2 Adafruit 4090 USB C Breakout Board

The Adafruit USB C board will serve as the main power source for the cartridges and the EEPROM. While a microcontroller has enough power to interface with the cartridge, it doesn't support the Amps necessary to also power it. Using a USB-C cable, the SCARAB could draw the power necessary from a USB port on the PC. While this will require 2 USB cables to be connected between the SCARAB and the PC, it's preferable to the alternative of batteries.

2.1.3 5V to 3.3V Buck Converter

Not only do some cartridges such as the GBA or N64 use 3.3V logic, but they utilise 3.3V VCC too. The logic level converters would not work for the level of amps required, so a new solution was needed. This is where buck converters are being used. Buck converters convert a DC voltage to a lower DC voltage, such as 5V to 3.3V in this case (Yates, 2024).

2.1.4 Module Connector Interface / SCARAB Connector

A series of pins and pin headers, the Module Connector and SCARAB Connector serve as a link between the SCARAB Device and its various cartridge port modules.

2.1.5 EEPROM

The Electronically Erasable Programmable Read Only Memory (EEPROM) serves as the identification chip for the individual modules. The SCARAB will need to adjust itself based on the type of cartridge it is reading from, so the EEPROM will store the necessary data to assist in this.

2.1.6 Logic Level Converter

As mentioned above, some cartridges use 3.3V logic, where the Arduino MEGA 2560 uses 5V logic. These logic level converters will translate between the Arduino and the cartridge in these cases. They will not be present on every module, only the ones that require 3.3V.

2.1.7 Cartridge Port

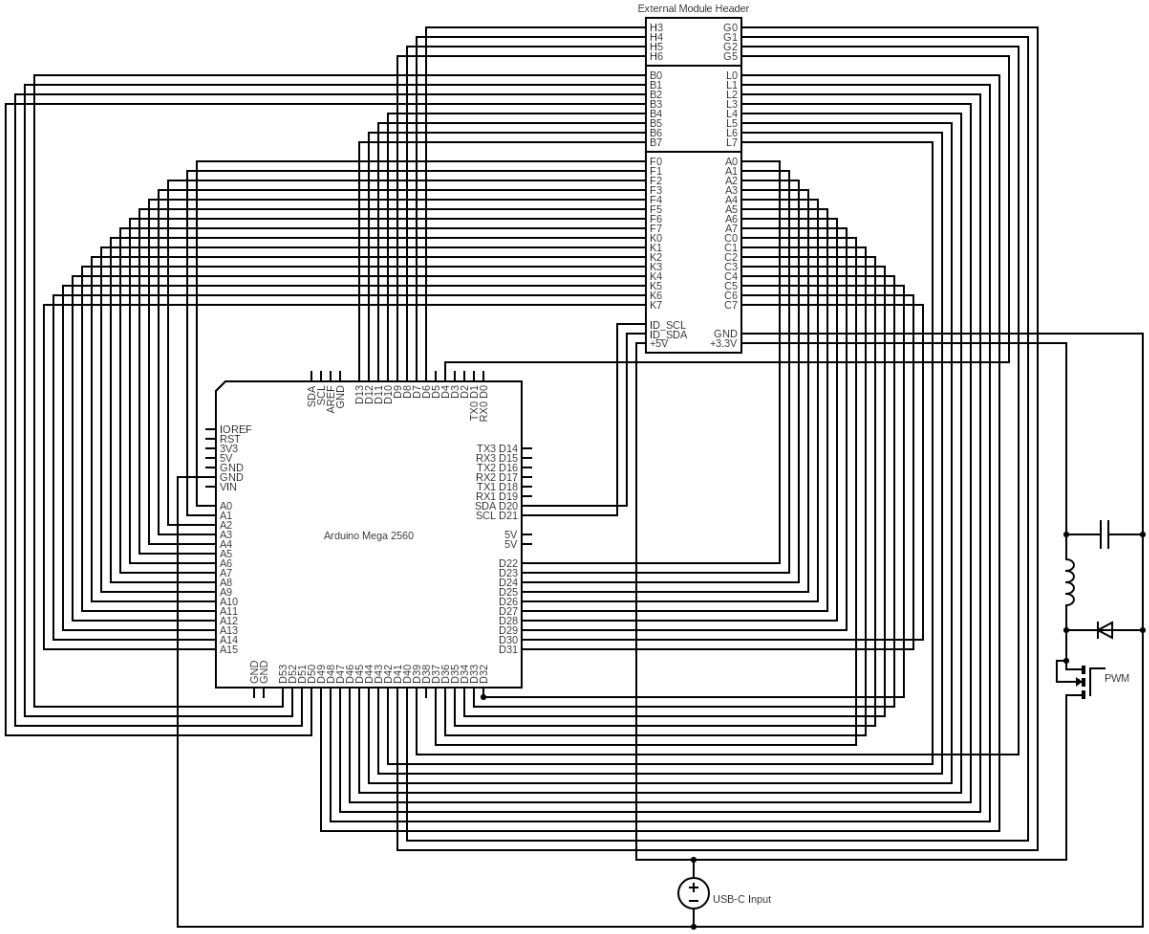
The cartridge ports serve as an interface between the Arduino and the cartridges themselves. They are fit for purpose, and mimic the port contained within the respective consoles that the cartridges are designed for.

2.1.8 Game Cartridge

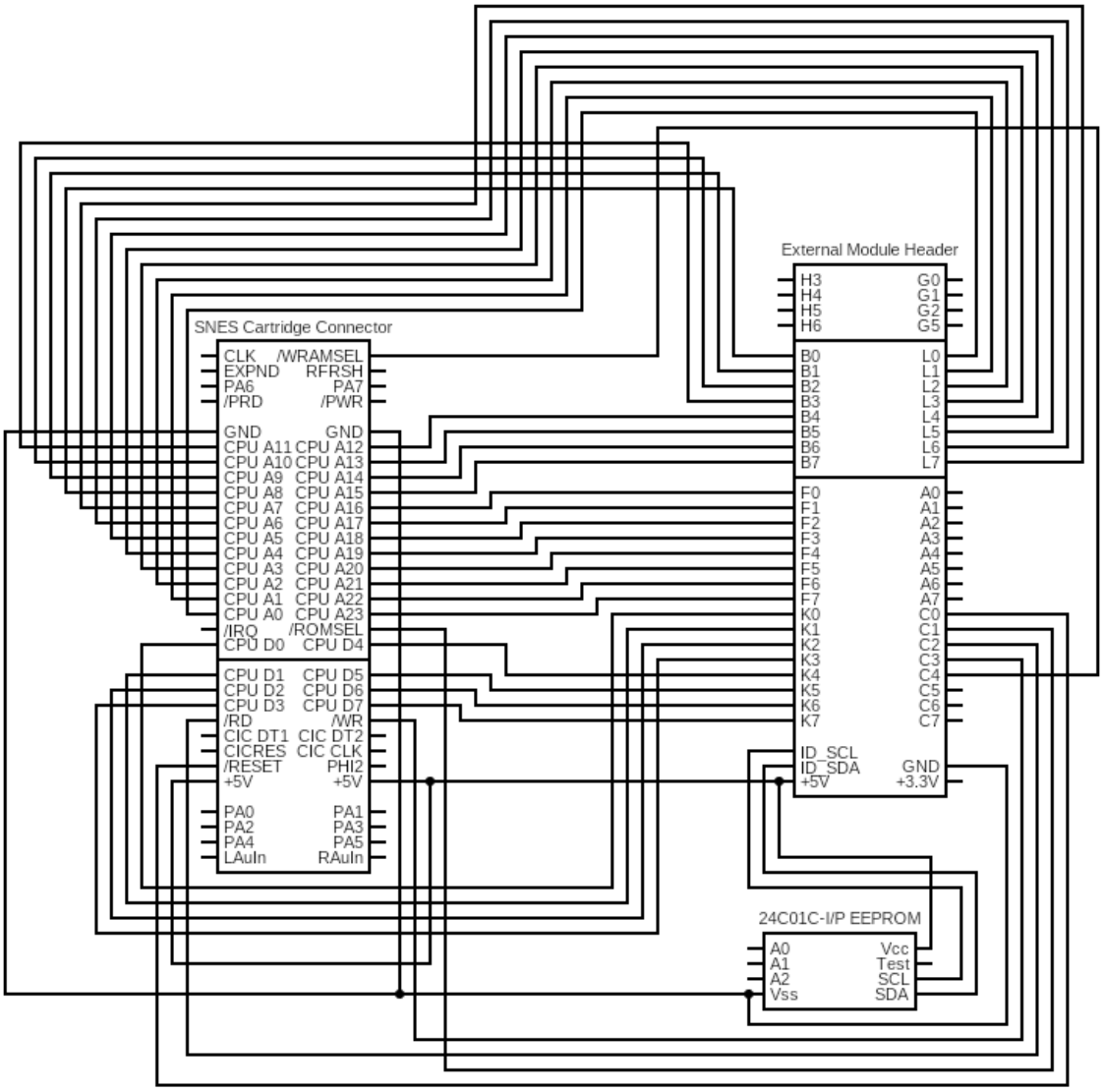
The game cartridges themselves are the main purpose of the SCARAB. They are being tested and checked for any issues, and having their save data backed up.

2.2 Circuit Diagrams

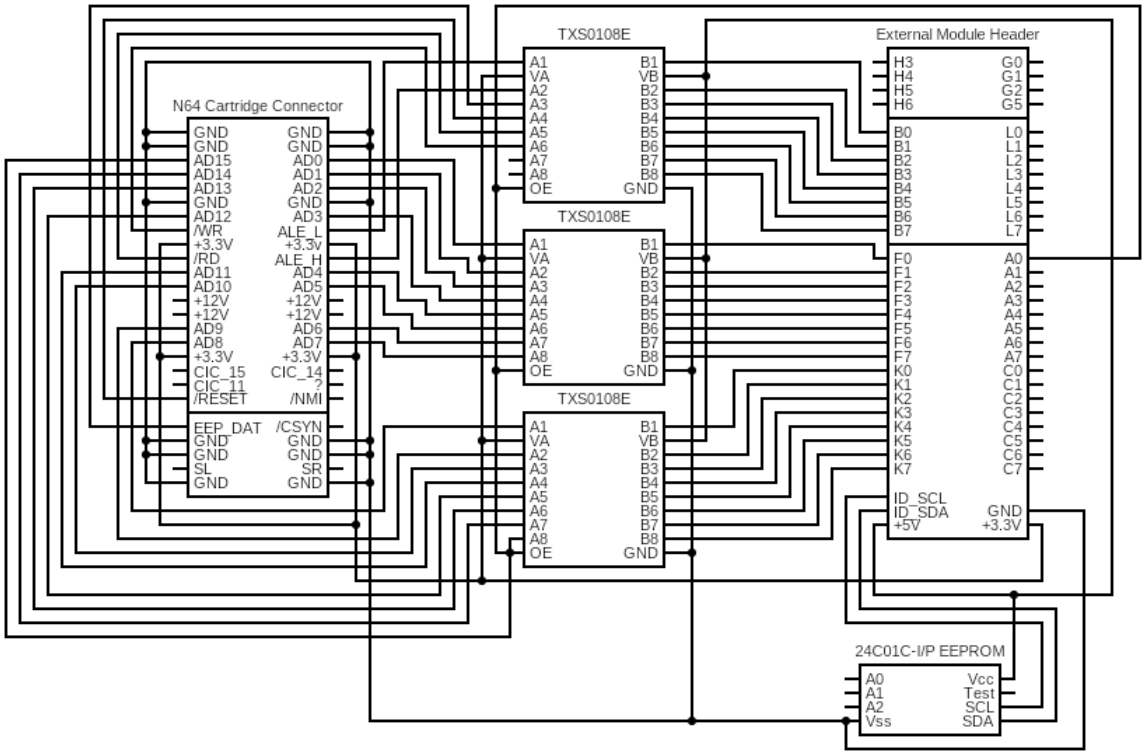
2.2.1 SCARAB Device



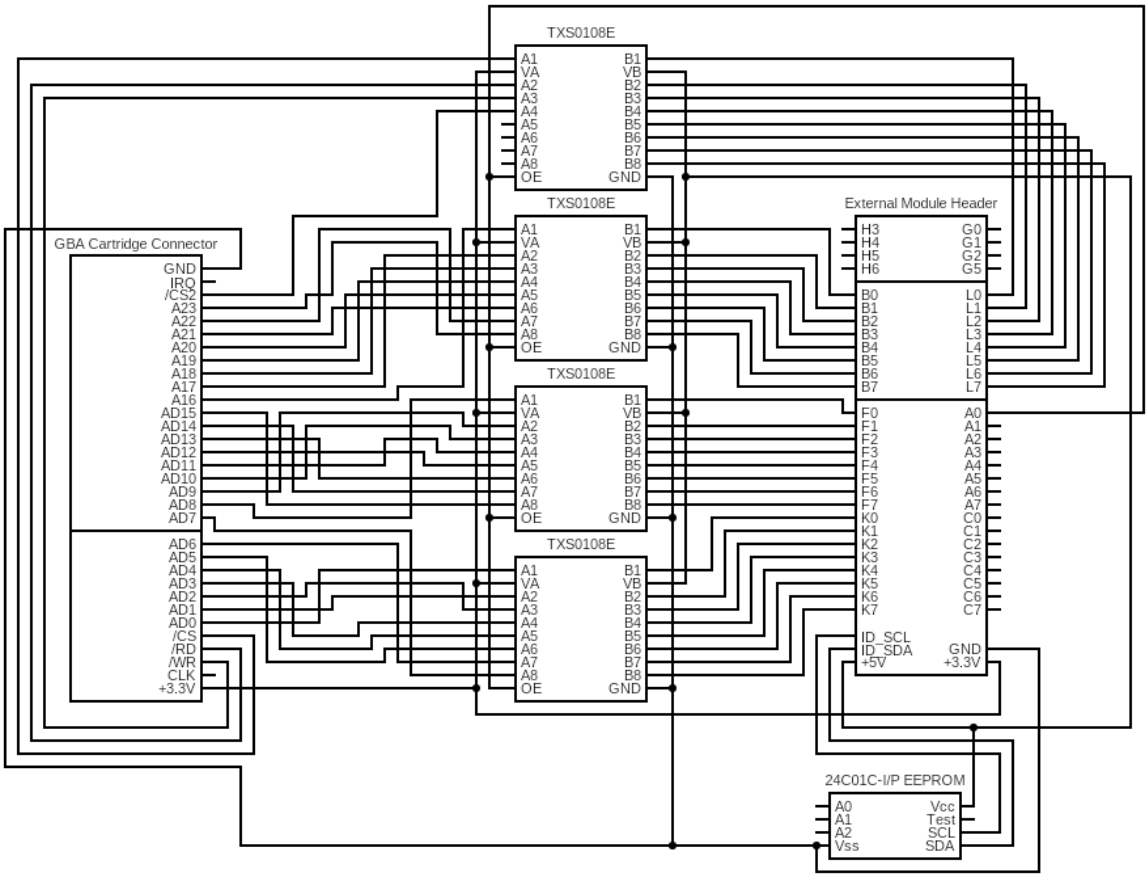
2.2.2 SNES Module



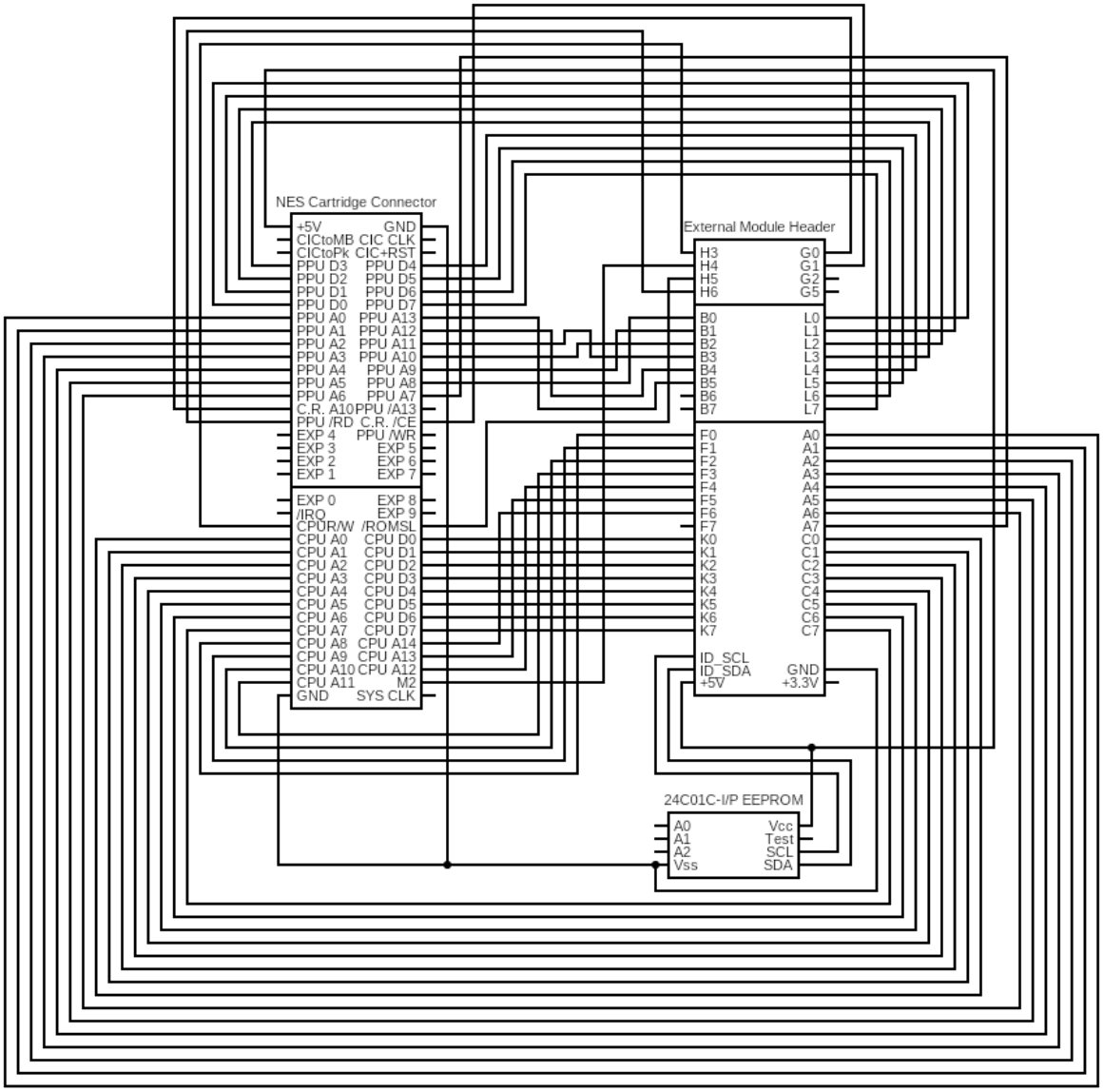
2.2.3 N64 Module



2.2.4 GBA Module



2.2.5 NES Module



3. Algorithm Pseudocode

3.1 SCARAB

3.1.1 Identify Module

```
func identifyModule() {
        Initialize EEPROM Connection
        value = Read From EEPROM
        Close EEPROM Connection
        switch (value) {
                case “SNES”:
                        globalFunctionPointers = snesFunctionPointers
                case “NES”:
                        globalFunctionPointers = nesFunctionPointers
                case “GBA”:
                        globalFunctionPointers = gbaFunctionPointers
                case “GBC”:
                        globalFunctionPointers = gbcFunctionPointers
                case “N64”:
                        globalFunctionPointers = n64FunctionPointers
        }
}
```

3.2 SNES

3.2.1 Read Cartridge

```
func readSnesCartridge(address) {
        Place address on address lines
        Set RD line to active low
        byte = Read from data lines
        return byte
}
```

3.2.2 Write Cartridge

```
func writeSnesCartridge(address, byte) {
        Place address on address lines
        Place byte on data lines
        Set WR line to active low
}
```

3.2.3 Dump Save

```
func dumpSnesSave(saveSize, romLayout) {
        Create buffer[1024] to store data
        switch (romLayout) {
                case “LoRom”:
                        startLocation = LoRomSaveLocation
                case “HiRom”:
                        startLocation = HiRomSaveLocation
                case “ExHiRom”:
                        startLocation = ExHiRomSaveLocation
        }
        for address in range startLocation to startLocation + saveSize {
                buffer[(address – startLocation % 1024)] = readSnesCartridge(address)
                checksum for buffer XOR new byte of data
                if buffer is full {
                        Send buffer to PC
                        Send checksum to PC
                        clear buffer
                        if PC returns OK then continue
                        else error
                }
        }
}
```

3.2.4 Restore Save

```
func restoreSnesSave(saveSize, romLayout) {
        Create buffer[1024] to store data
        switch (romLayout) {
                case “LoRom”:
                        startLocation = LoRomSaveLocation
                case “HiRom”:
                        startLocation = HiRomSaveLocation
                case “ExHiRom”:
                        startLocation = ExHiRomSaveLocation
}
        for address in range startLocation to startLocation + saveSize {
                if buffer is empty { //if (address – startLocation % 1024) == 0
                                Request data from PC
                                Request Checksum from PC
                                Validate Checksum
                                if OK then ACK
                                else ERR
                }
                writeSnesCartridge(address, buffer[(address – startLocation % 1024)])
        }
        Send OK to PC
}
```

3.2.5 Calculate Checksum

```
func calcSnesChecksum(romSize) {
        checksum = 0
        for address in range romSize {
                checksum += readSnesCartridge(address)
                if checksum overflows 16 bits, checksum &= 0x00FF
        }
        return checksum
}
```

3.3 NES

3.3.1 Read Cartridge

```
func readNesCartridge(address, romArea) {
        if romArea is “PRG” {
                Place address on CPU address lines
                set CPU RD to active low
                byte = read from CPU data lines
        }
        else romArea is “CHR” {
                Place address on PPU address lines
                set PPU RD to active low
                byte = read from PPU data lines
        }
        return byte
}
```

3.3.2 Write Cartridge

```
func writeNesCartridge(address, romArea, byte) {
        if romArea is “PRG” {
                Place address on CPU address lines
                Place byte on CPU data lines
                set CPU WR to active low
        }
        else romArea is “CHR” {
                Place address on PPU address lines
                Place byte on PPU data lines
                set PPU WR to active low
        }
}
```

3.3.3 Dump Save

```
func dumpNesSave(saveSize, mapper, saveSize) {
        based on mapper, writeNesCartridge(BankControlAddress, PRG, RamEnableCode)
        declare buffer[1024] to store data
        for x in range saveSize {
                buffer[x % 1024] = readNesCartridge(x + baseSaveAddress, PRG)
                if buffer full {
                        send buffer to PC
                        calc Checksum on buffer
                        send Buffer to PC
                        clear buffer on OK
                }
        }
}
```

3.3.4 Restore Save

```
func restoreNesSave(saveSize, mapper, saveSize) {
        based on mapper, writeNesCartridge(BankControlAddress, PRG, RamEnableCode)
        declare buffer[1024] to store data
```

```
        for x in range saveSize {
                writeNesCartridge(x + baseSaveAddress, PRG, buffer[x%1024]
                if buffer empty { //if (x % 1024) == 0
                                Request data from PC
                                Request Checksum from PC
                                Validate Checksum
                                if OK then ACK
                                else ERR
                }
        }
}
```

3.3.5 Calculate Checksum

3.4 GBA

3.4.1 Read Cartridge

3.4.2 Write Cartridge

3.4.3 Dump Save

3.4.4 Restore Save

3.4.5 Calculate Checksum

3.5 GB/GBC

3.5.1 Switch Bank

```
func switchBank(bankNumber, romType, memType) {
        Determine Write address from romType
        if memType is RAM, write to enable RAM mode
        If bankNumber > 1F (needs more than 5 bits, never happens with RAM) {
                Write to enable ROM banking mode
                Write lower 5 bits
                Write upper 2 bits
        }
        else {
```

```
            Write bank to address
        }
}
```

3.5.2 Read Cartridge

```
func readGbCartridge(address) {
        Place address  on address lines
        Toggle READ to active low
        byte = Read from data pins
        return byte
}
```

3.5.3 Write Cartridge

```
func writeGbCartridge(address, bank, byte) {
        Place address on address lines
        Place byte on data lines
        Toggle WRITE to active low
}
```

3.5.4 Dump Save

```
func dumpGbSave(ramBanks) {
        Create buffer to store data
        for x in range ramBanks {
            //Banks are offset by 1. bank 0 is the first bank, never needs switching
            if bank > 1, switchBank(x – 1, romType, RAM), change offset from 0
            Write enable RAM
            for y in range 8KiB + offset {
                buffer.add(readGbCartridge(y)
                if buffer full {
                    Calculate checksum on buffer
                    Send buffer to PC
                    Send checksum to PC
                }
            }
```

```
            Write disable RAM
        }
}
```

3.5.5 Restore Save

```
func restoreGbSave(saveSize, ramBanks) {
        Create buffer[1024] to store data
        Request data from PC
        Request Checksum from PC
        Validate Checksum
        if OK then ACK
        else ERR
        for x in range ramBanks {
            //Banks are offset by 1. bank 0 is the first bank, never needs switching
            if bank > 1, switchBank(x – 1, romType, RAM), change offset from 0
            Write enable RAM
            for y in range 8KiB + offset {
                writeGbCartridge(
                if buffer empty {
                    Request data from PC into buffer
                    Request Checksum from PC
                    Validate Checksum
                    if OK then ACK
                    else ERR
                }
            }
            Write disable RAM
        }
        Send OK to PC
}
```

3.5.6 Calculate Checksum

3.6 N64

3.6.1 Read Cartridge

```
func readN64Cartridge(address) {
        Place upper half of address on address lines
        Latch upper half of address
        Place lower half of address on address lines
        Latch lower half of address
        Set RD line to active low
        byte = Read from data lines
        return byte
}
```

3.6.2 Write Cartridge

```
func writeN64Cartridge(address, byte) {
        Place upper half of address on address lines
        Latch upper half of address
        Place lower half of address on address lines
        Latch lower half of address
        Put data on data lines
        Set WR to active low
}
```

3.6.3 Dump Save

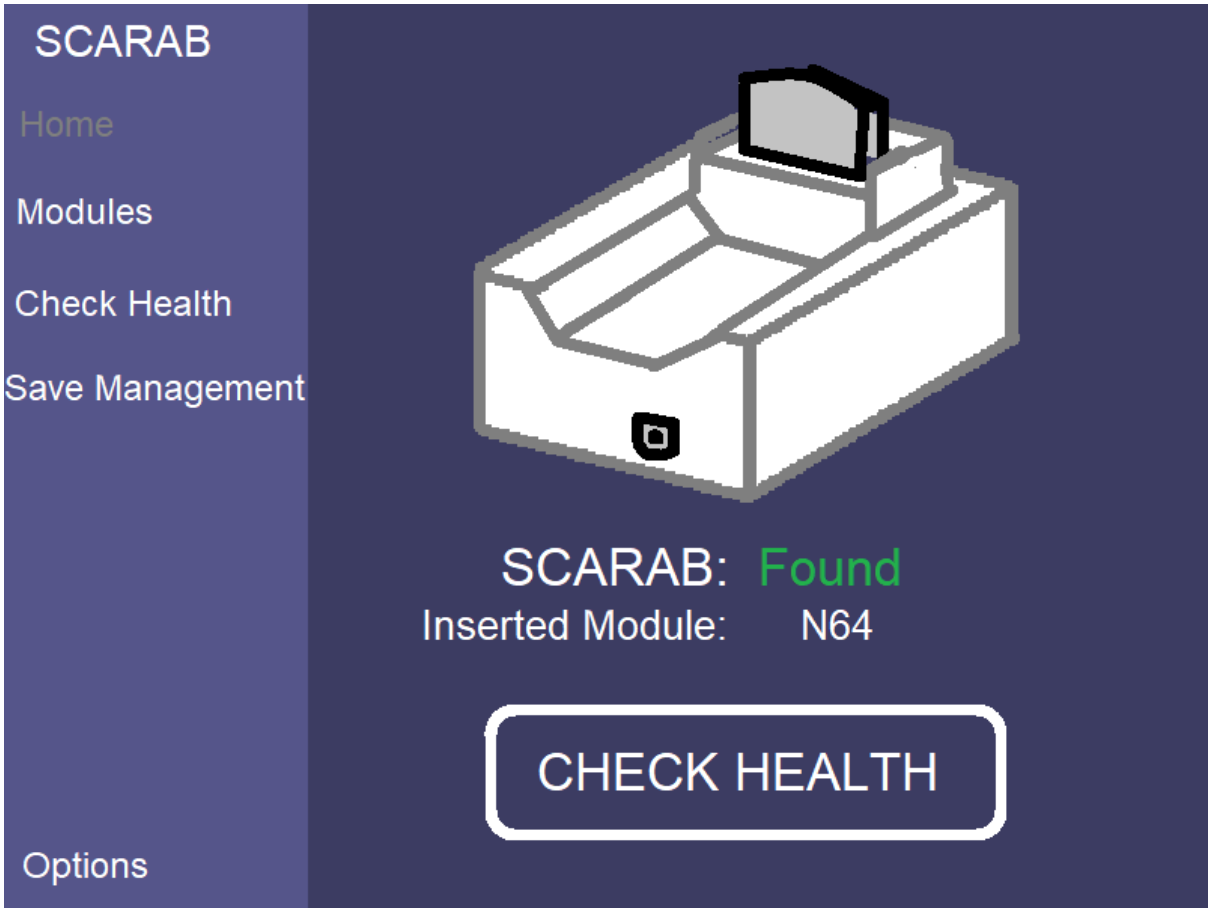
3.6.4 Restore Save

3.6.5 Calculate Checksum

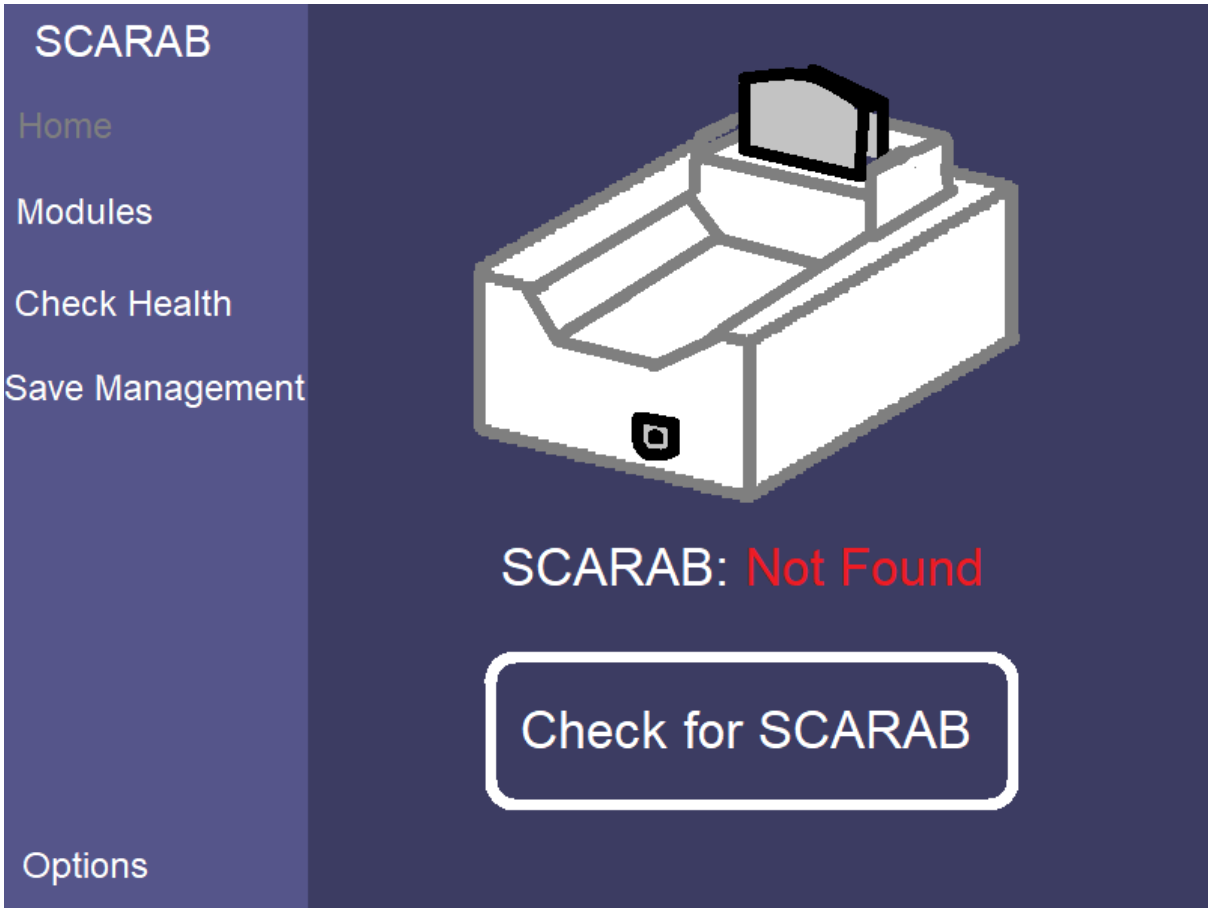
4. User Interface

4.1 Main Menu

4.1.1 SCARAB Found

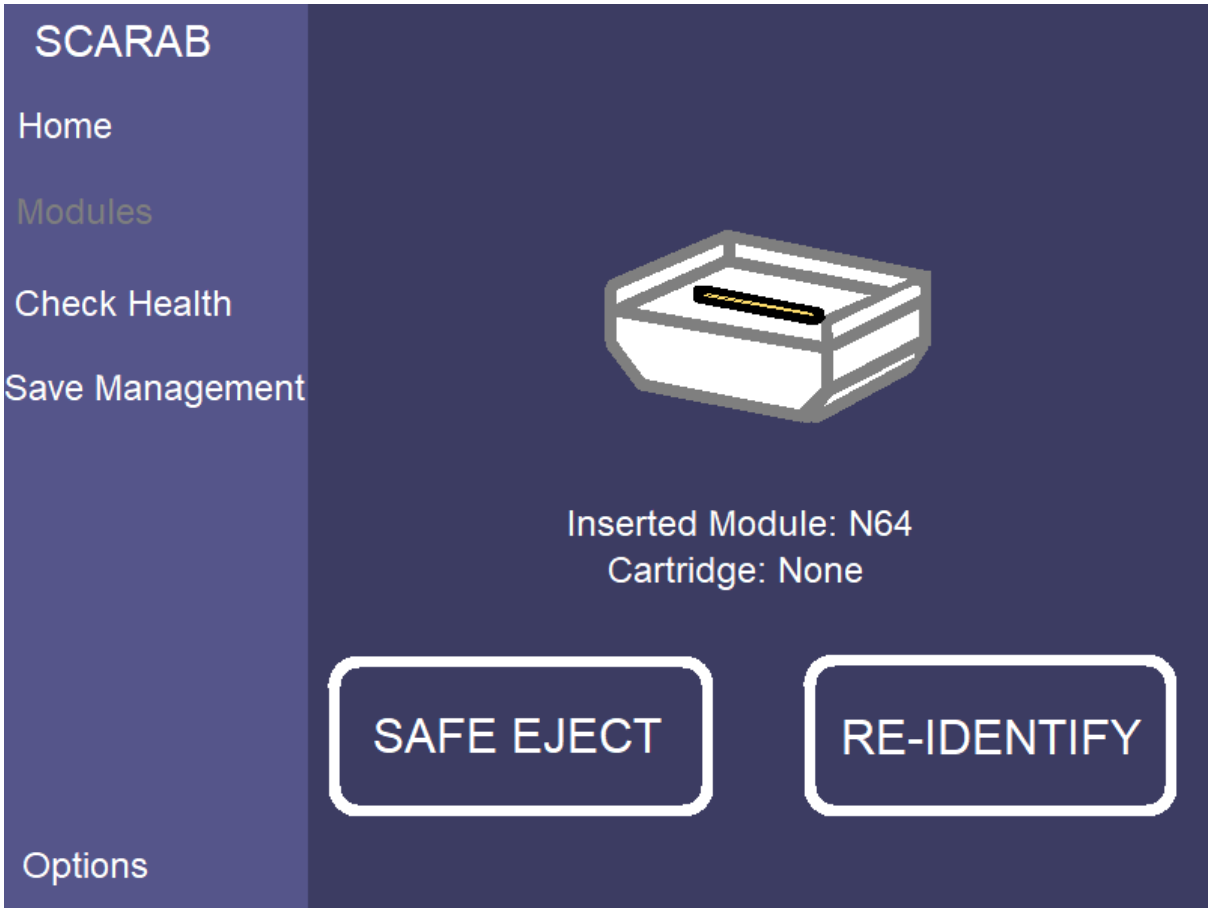


4.1.2 SCARAB Not Found

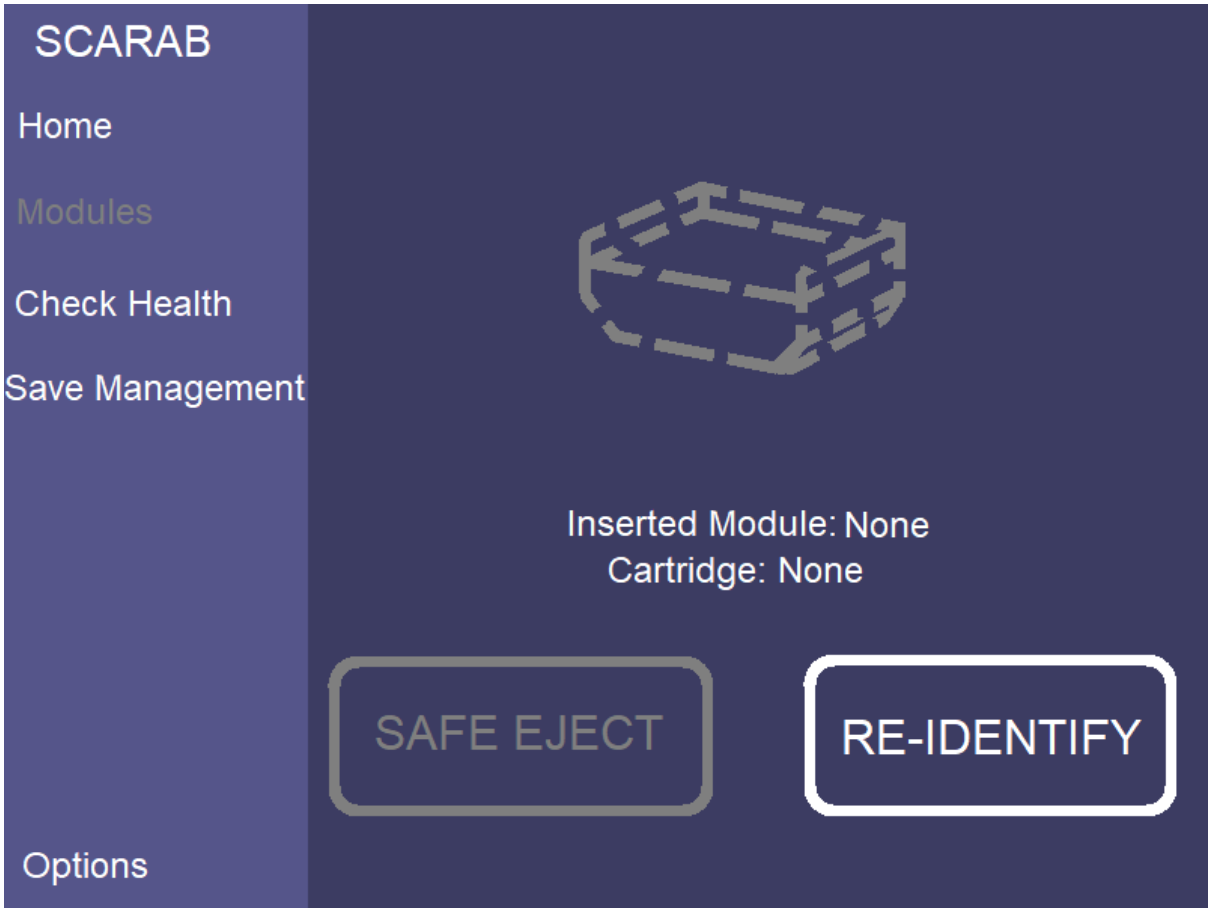


4.2 Modules

4.2.1 Module Inserted

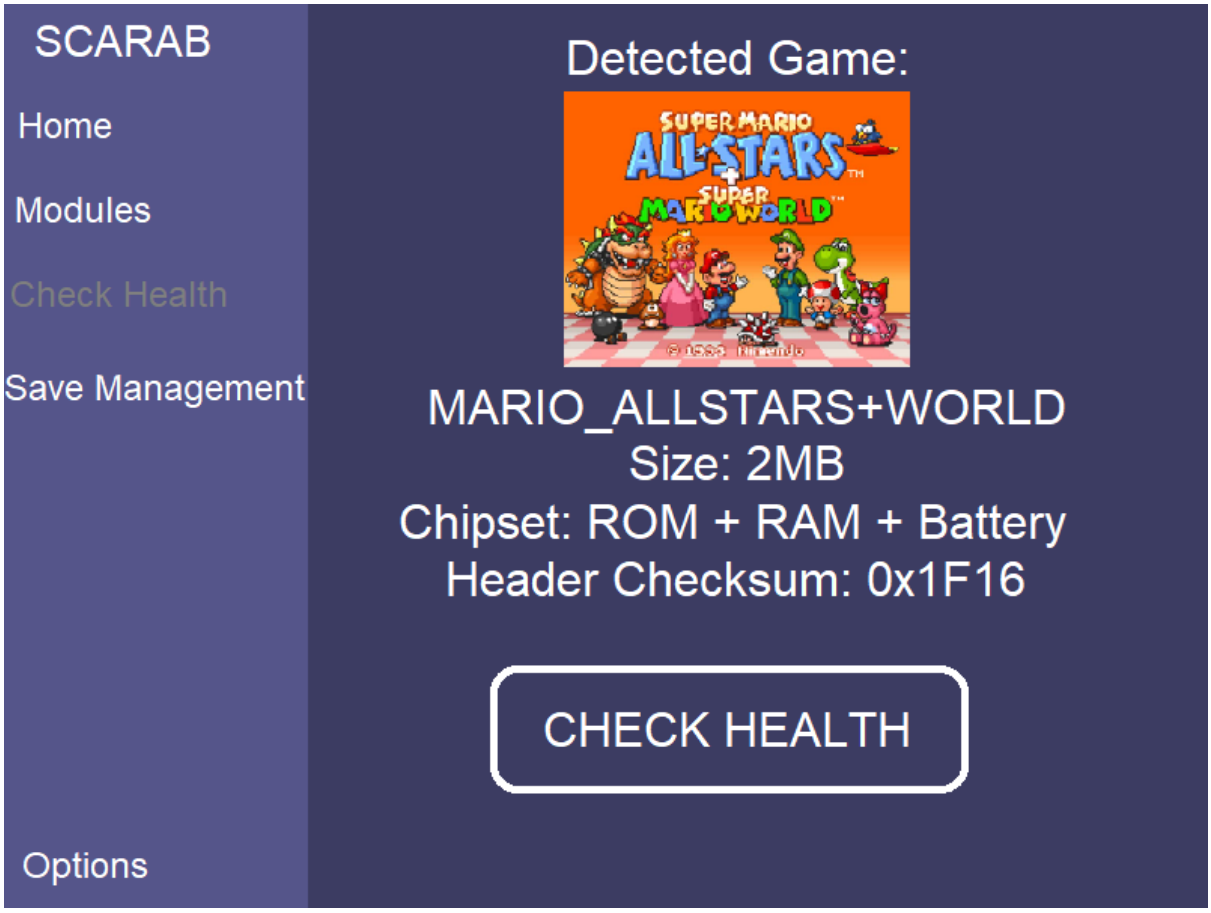


4.2.2 Module Not Inserted

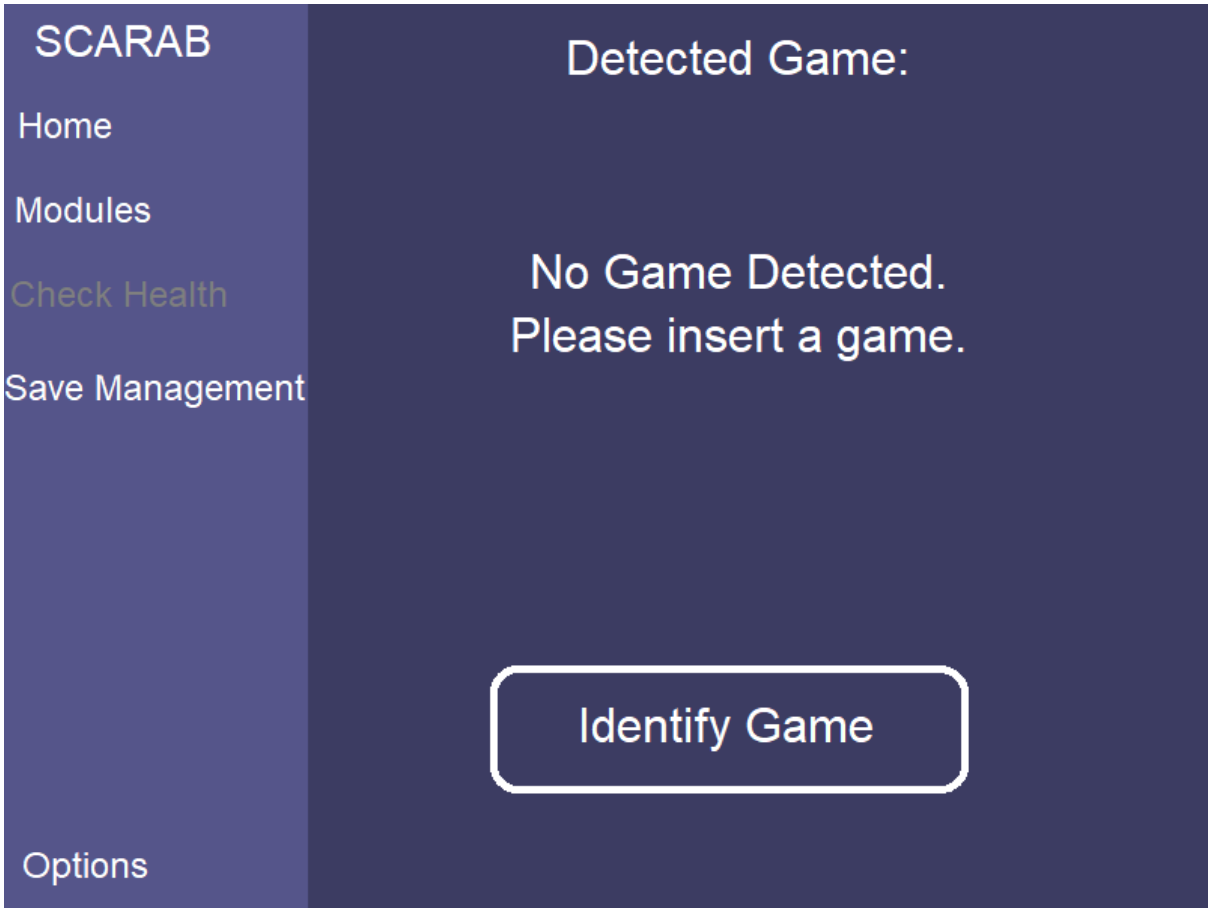


4.3 Check Health

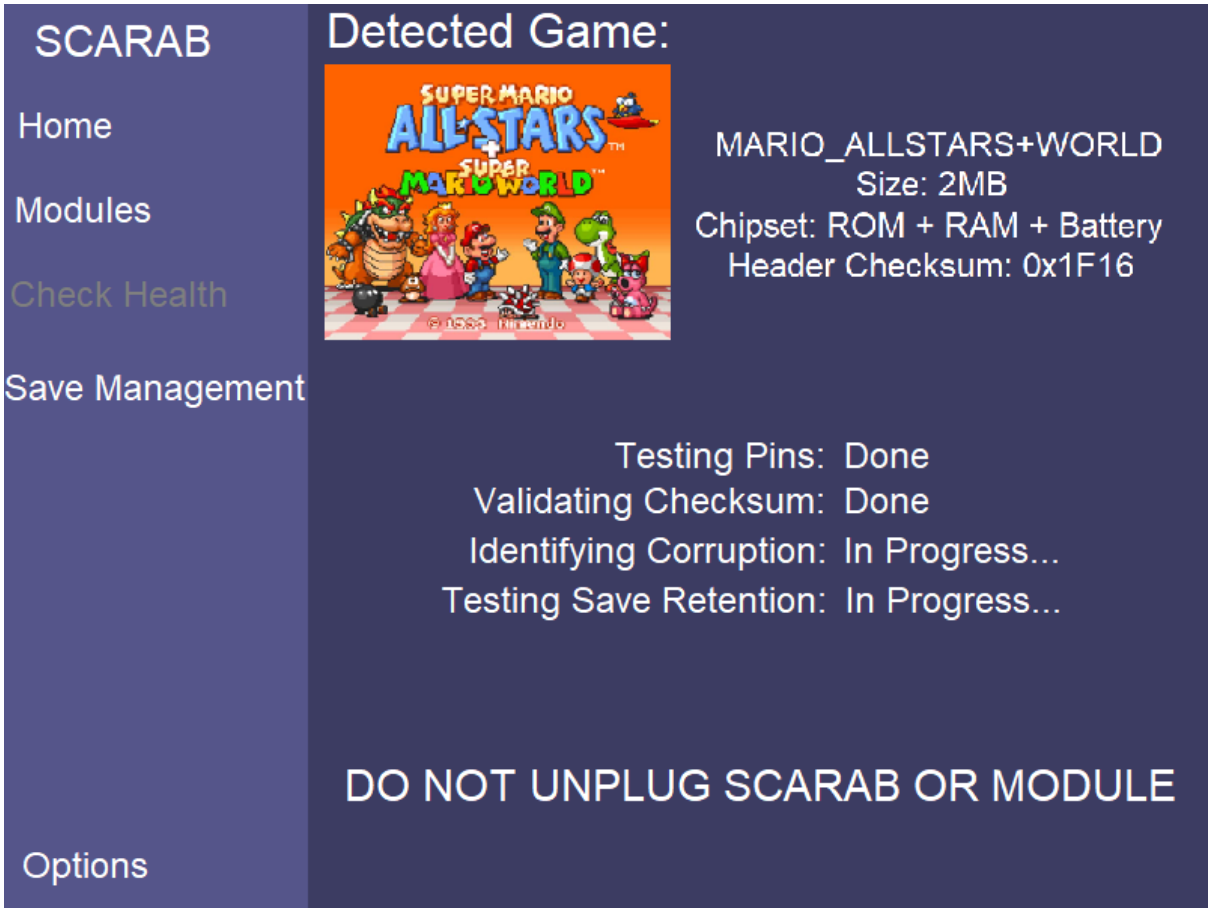
4.3.1 Cartridge Inserted



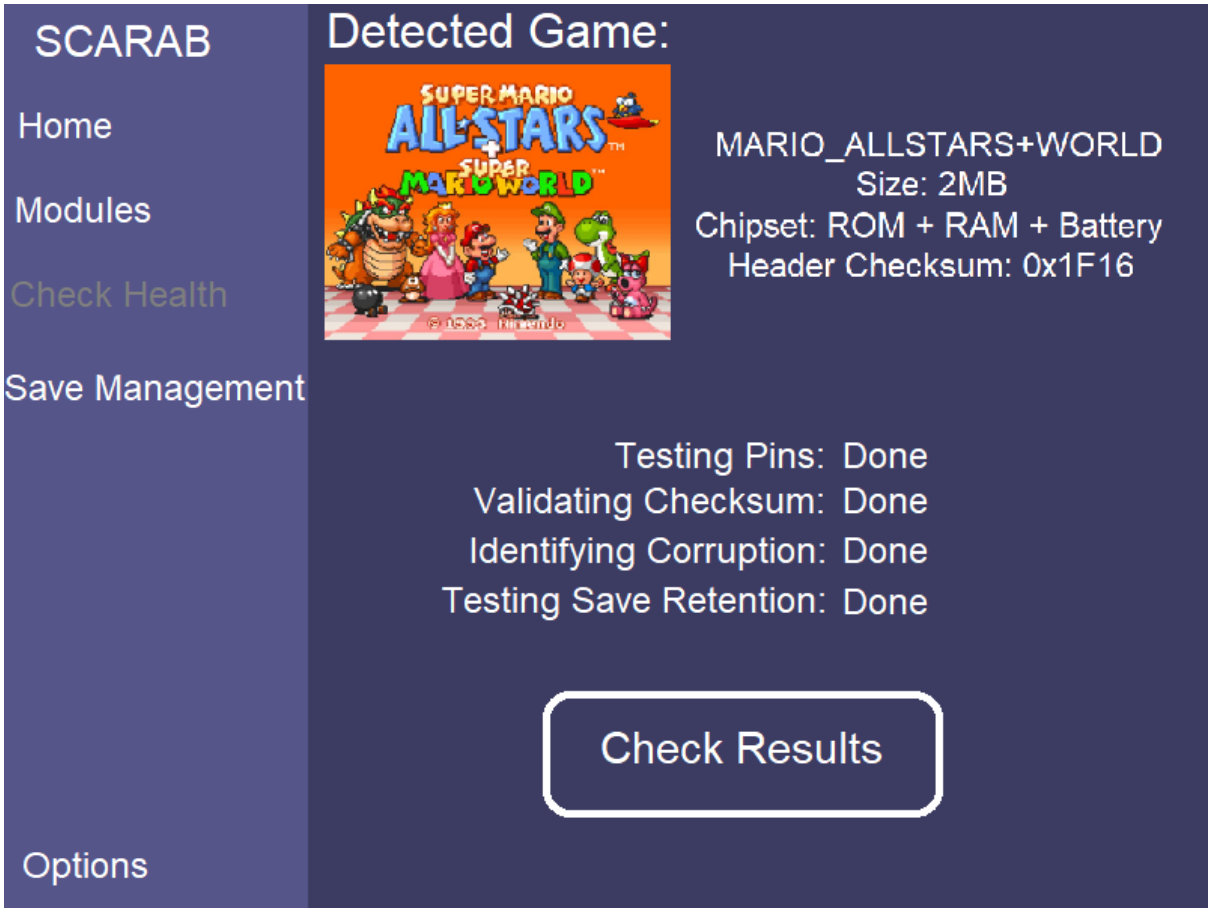
4.3.2 Cartridge Not Inserted



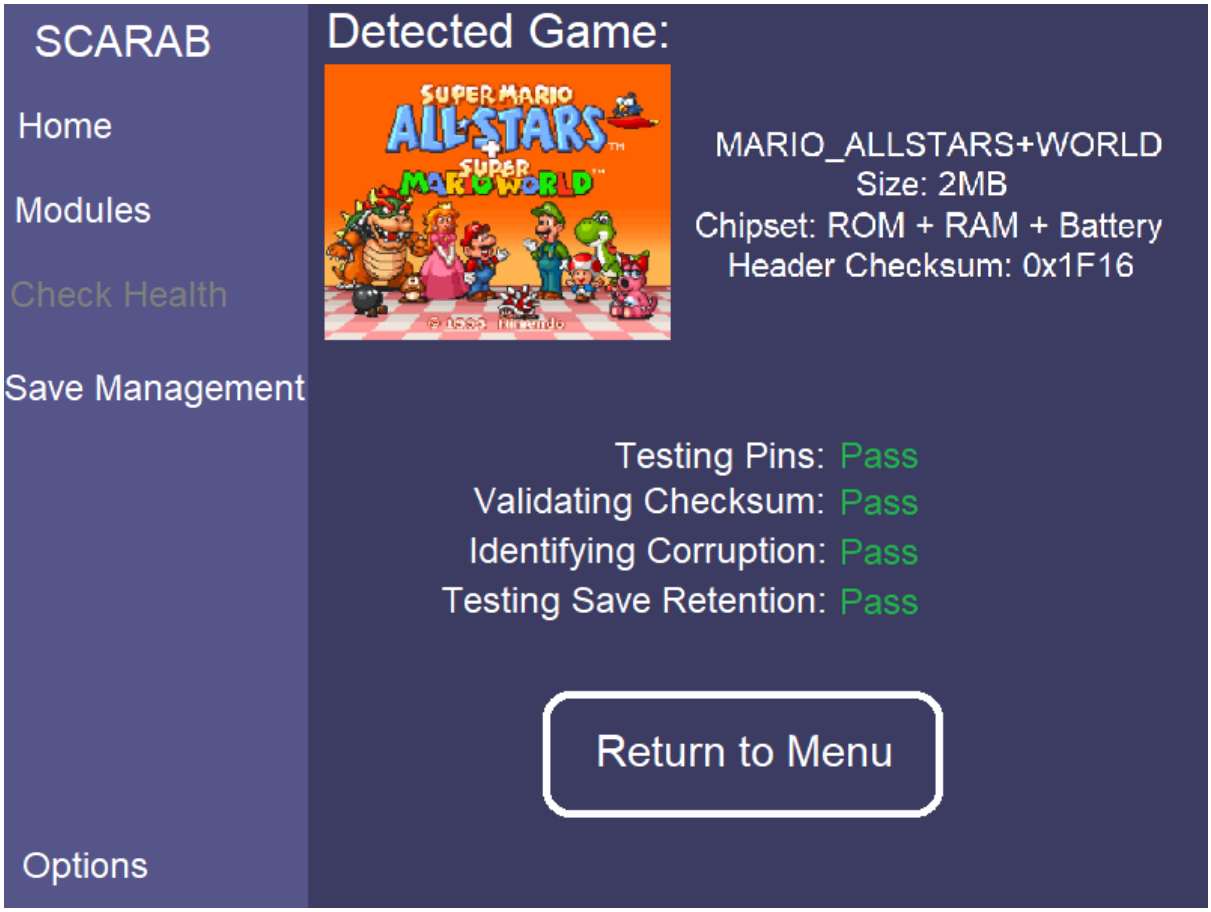
4.3.3 Health Check In Progress



4.3.4 Health Check Done



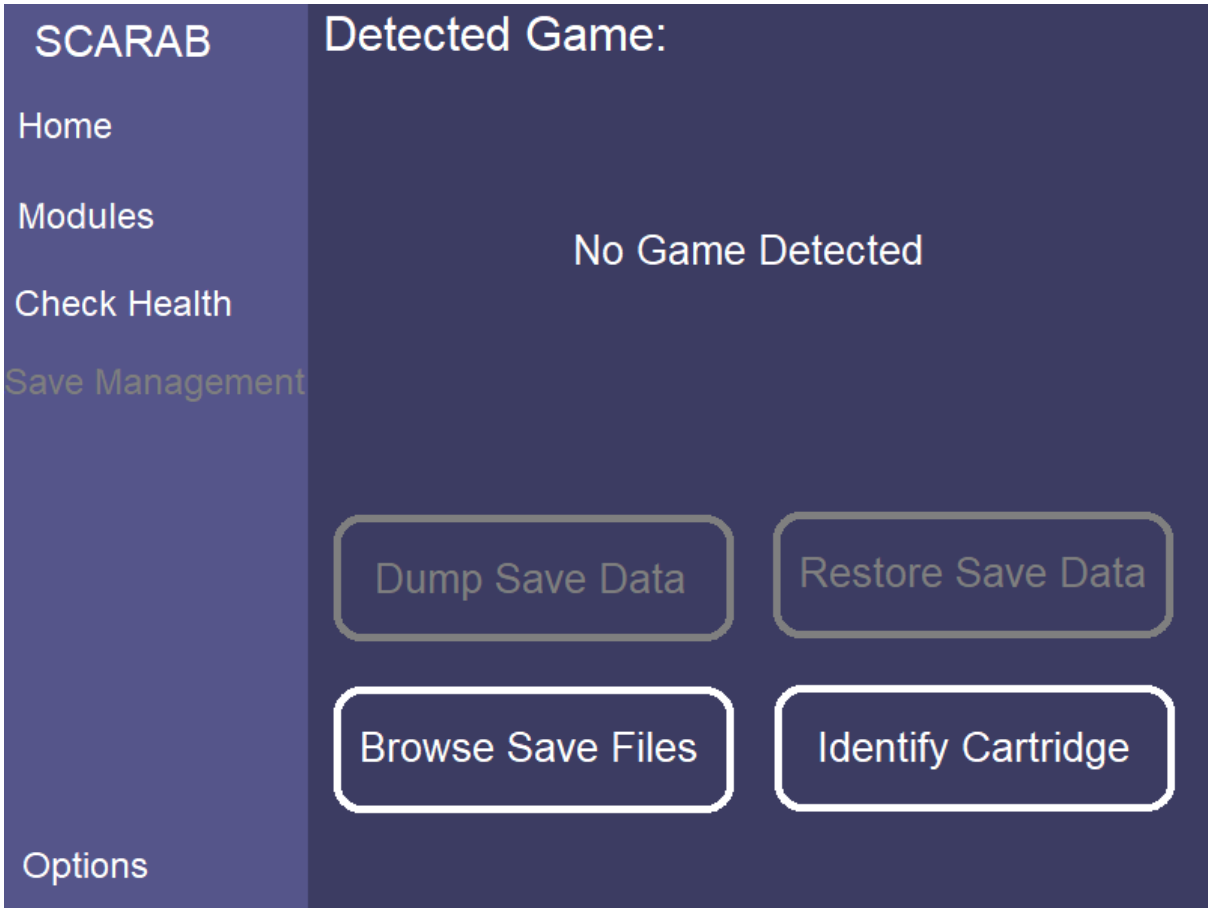
4.3.5 Health Check Results



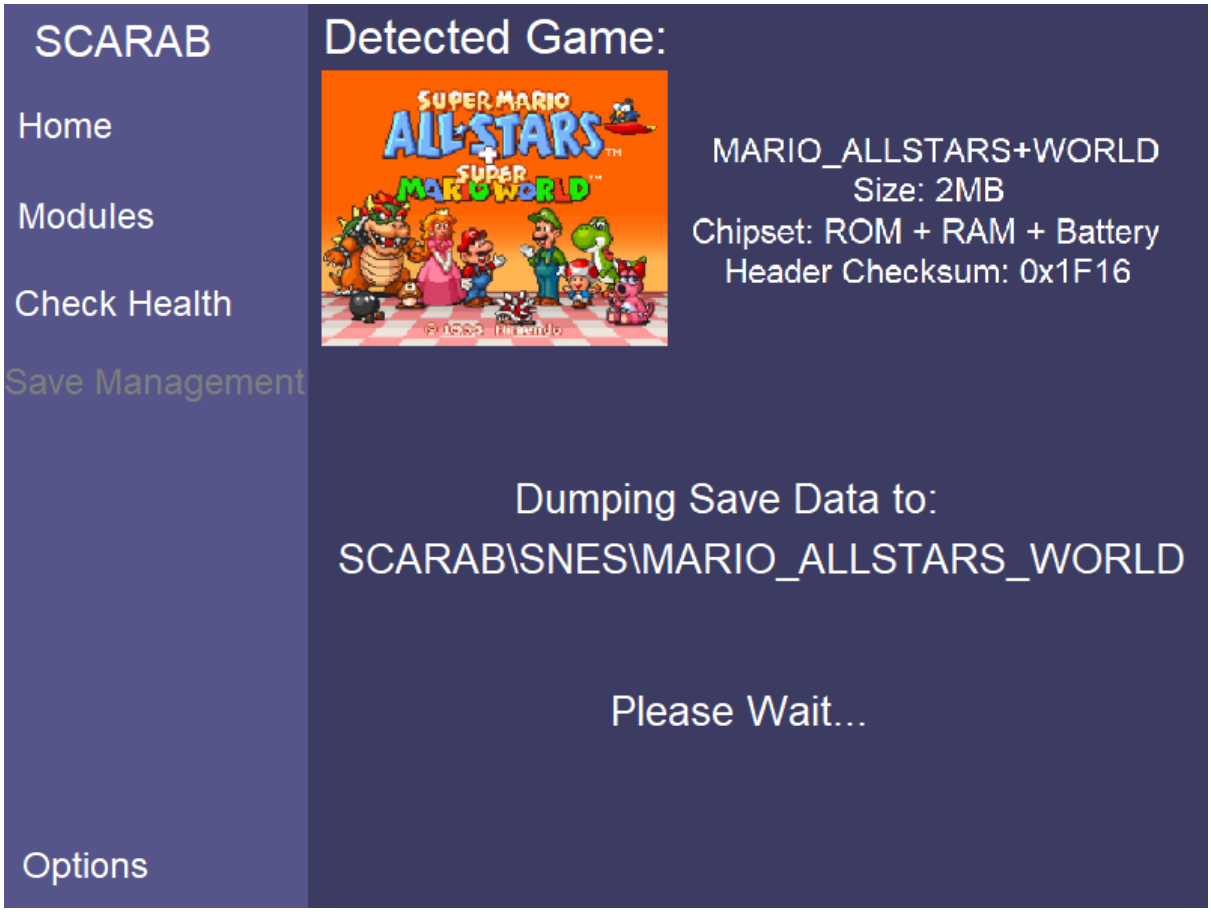
4.4 Save Management

4.4.1 Main Menu

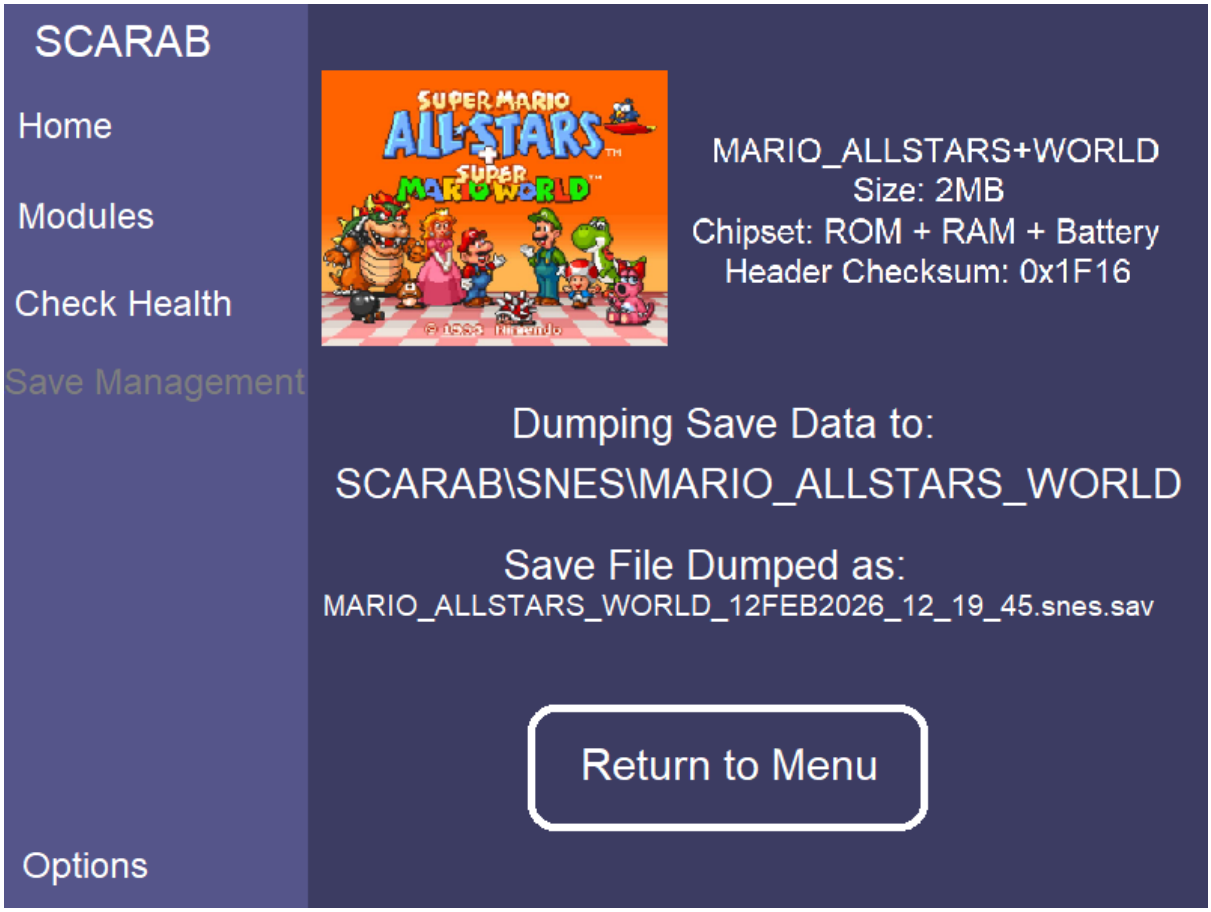
4.4.2 Main Menu No Cart



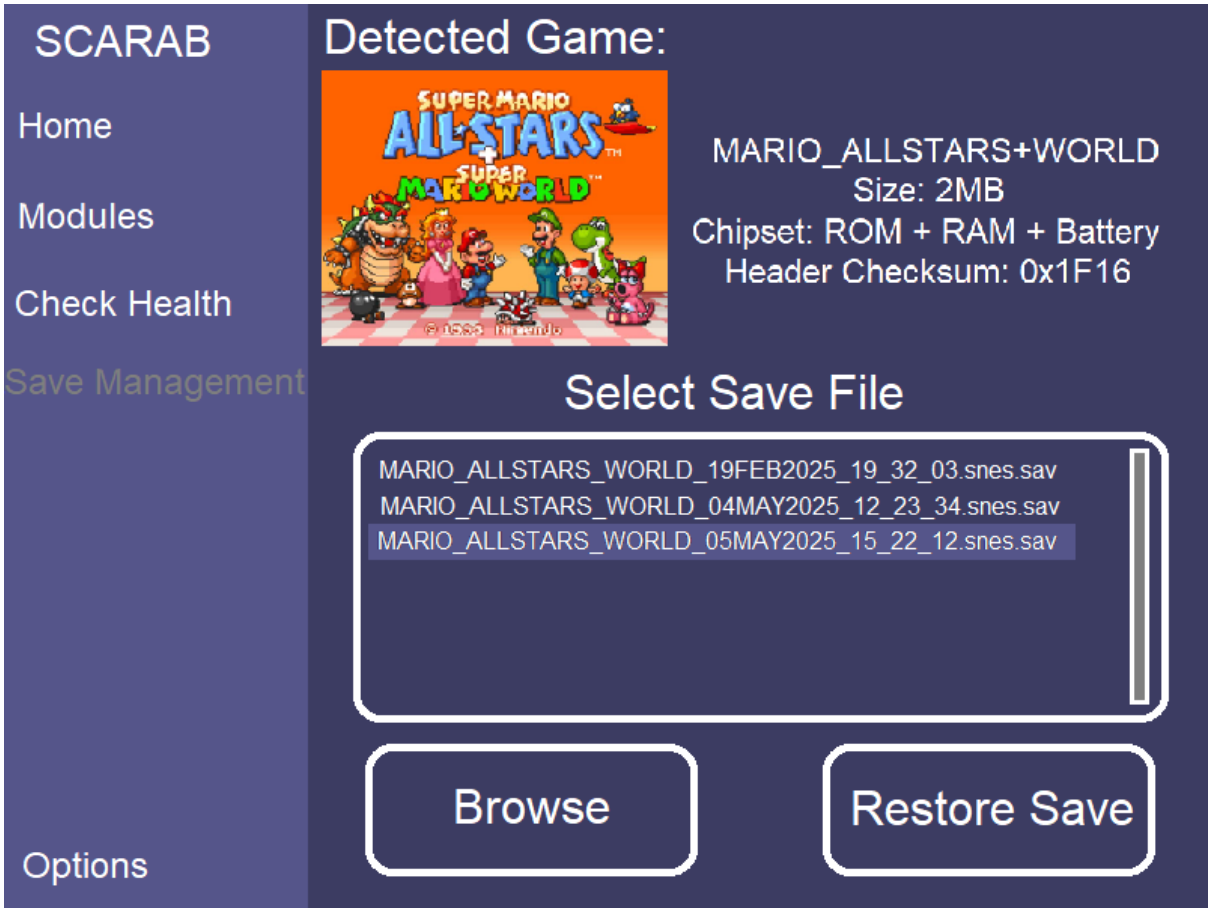
4.4.3 Dumping Save



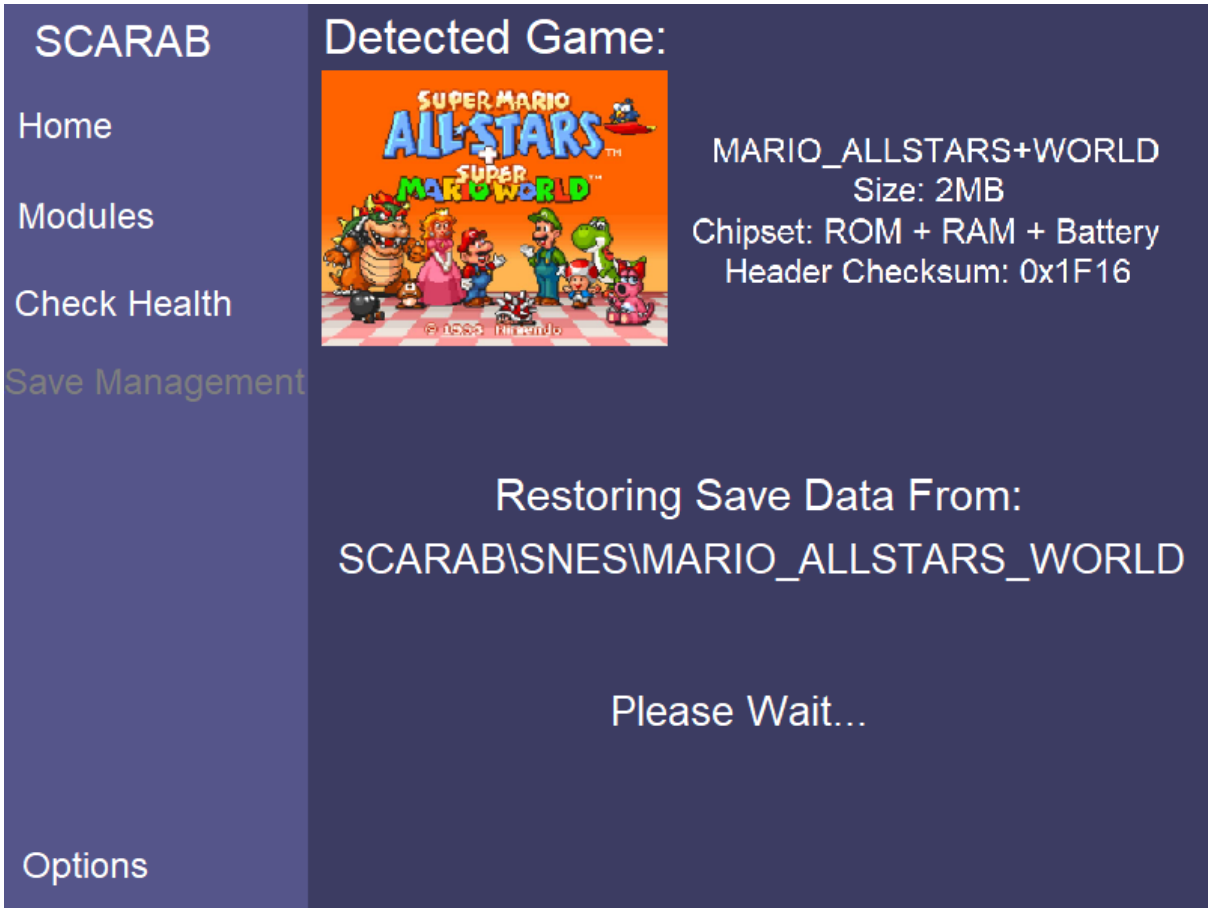
4.4.4 Save Dumped



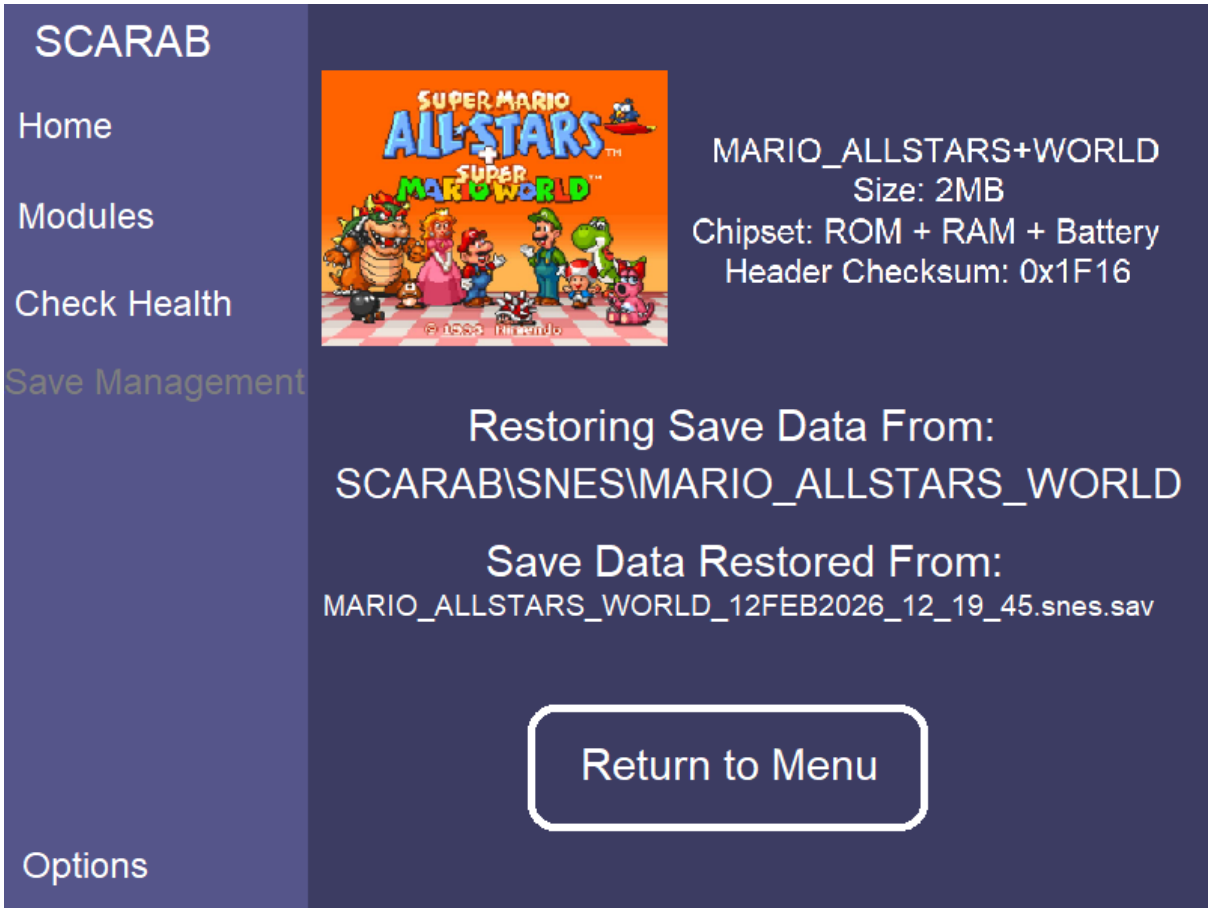
4.4.5 Restore Save



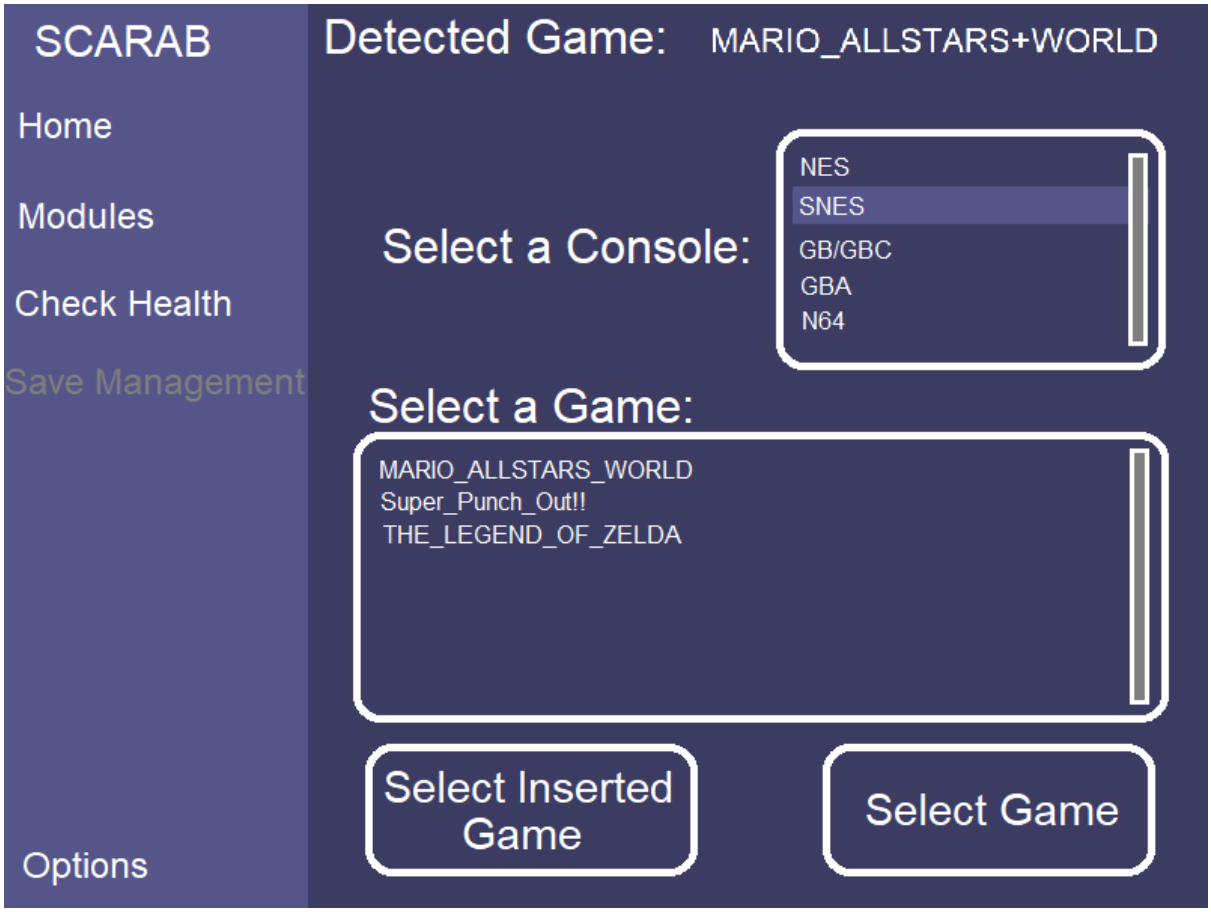
4.4.6 Restoring Save



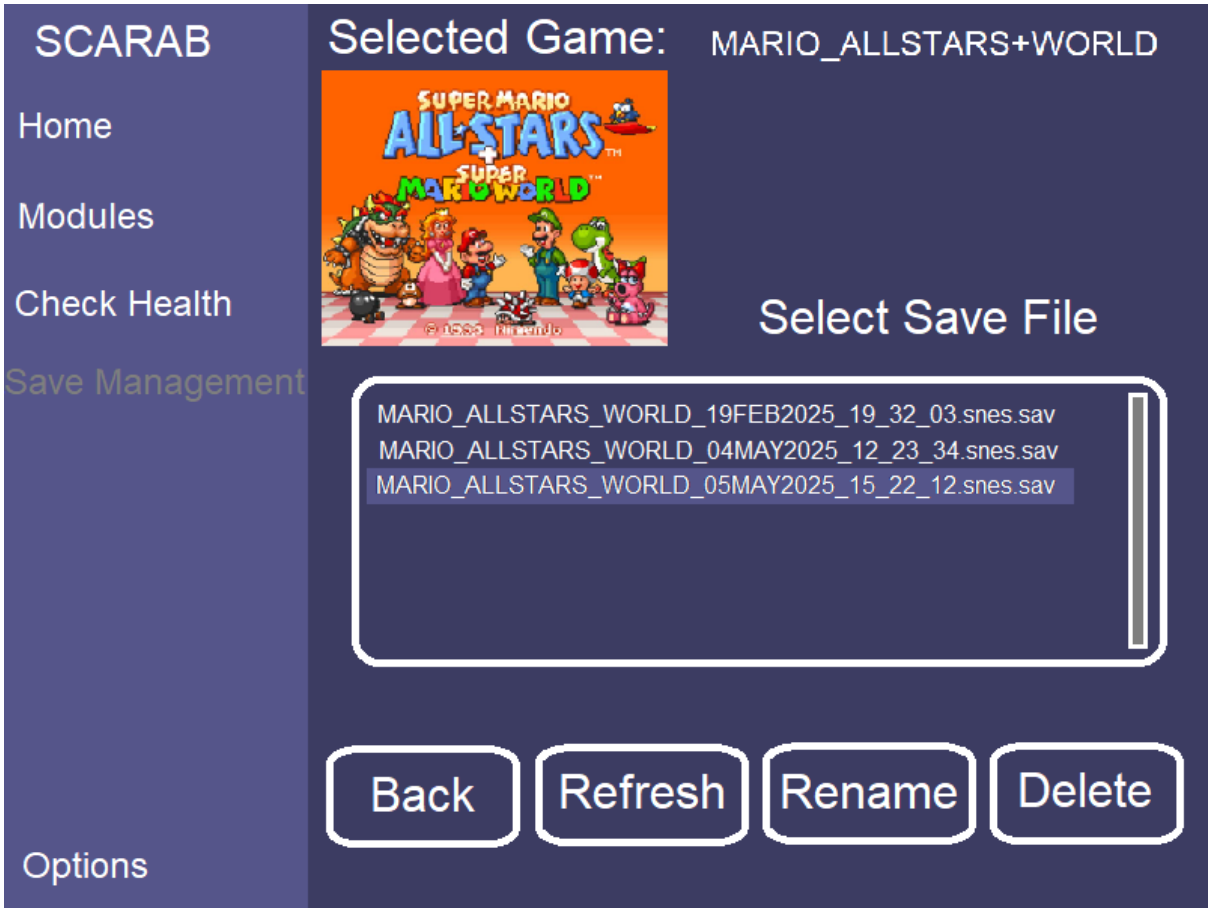
4.4.7 Save Restored



4.4.8 Browse Saves – Select Game



4.4.9 Browse Saves



5. Bibliography

Arduino, 2025. *Arduino® MEGA 2560 Rev3*. [Online]
Available at: https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf
[Accessed 10 February 2026].

pyserial, 2022. *pySerial*. [Online]
Available at: https://github.com/pyserial/pyserial
[Accessed 10 February 2026].

pyside, 2021. *GitHub - pyside/pyside-setup: Git super project for PySide*. [Online]
Available at: https://github.com/pyside/pyside-setup
[Accessed 10 February 2026].

W3Schools, n.d. *Introduction to Python*. [Online]
Available at: https://www.w3schools.com/python/python_intro.asp
[Accessed 9 November 2025].

W3Schools, n.d. *Introduction to Python*. [Online]
Available at: https://www.w3schools.com/python/python_intro.asp
[Accessed 10 February 2026].

Yates, J., 2024. *Understanding Buck and Boost Converters and the Capacitors Behind Them*. [Online]
Available at: https://blog.knowlescapacitors.com/blog/understanding-buck-and-boost-converters-and-the-capacitors-behind-them
[Accessed 10 February 2026].