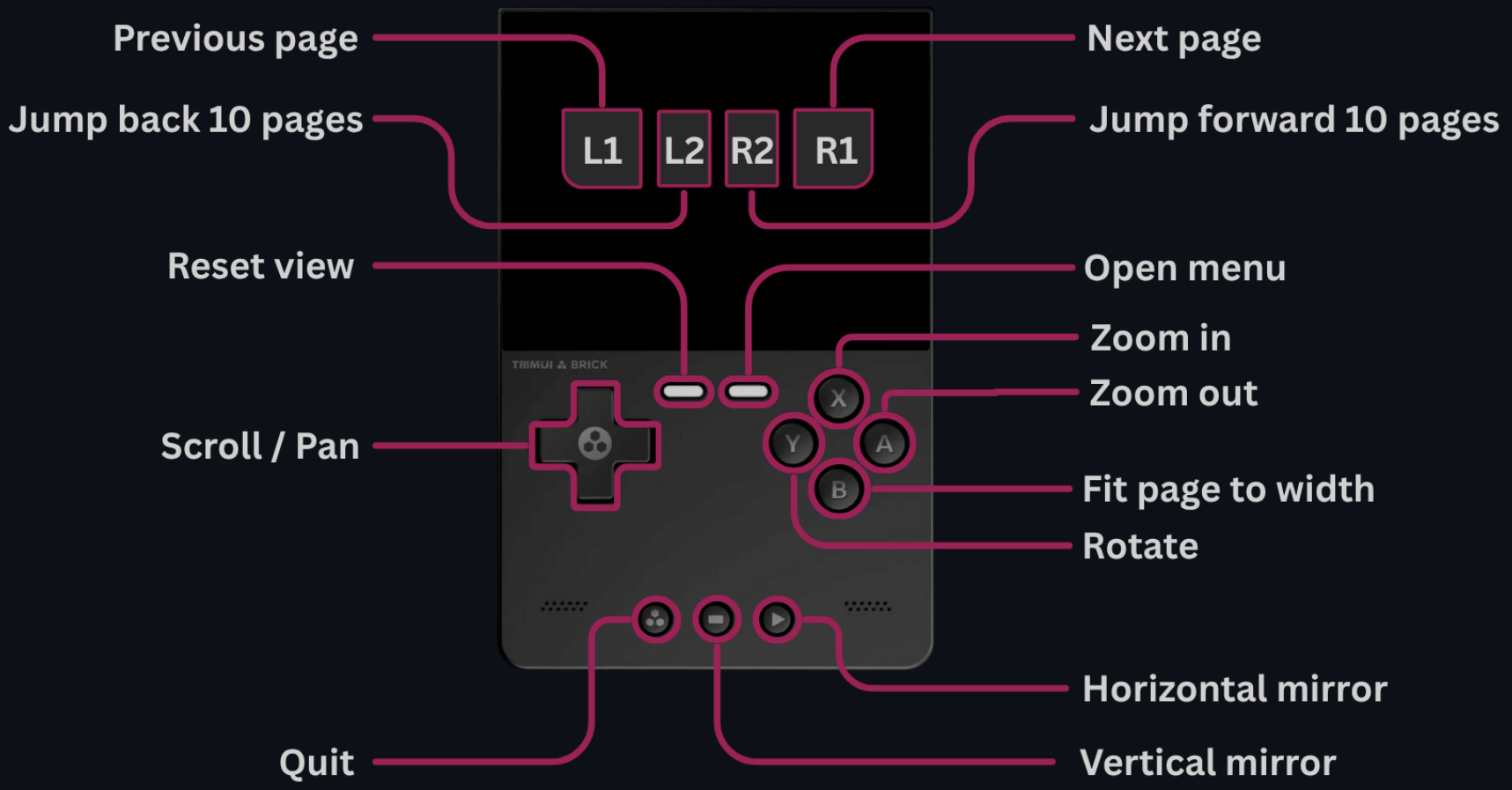
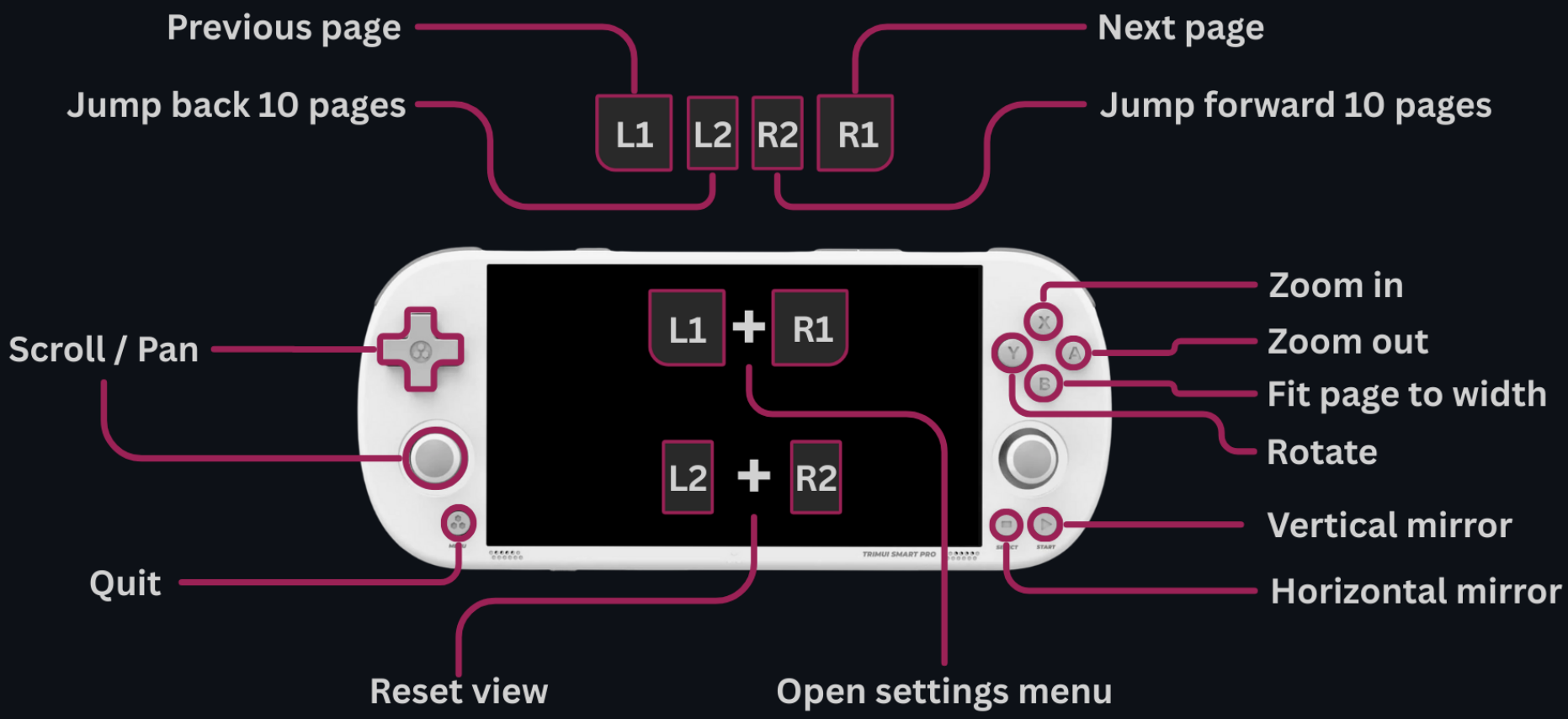
SDLReader TrimUI Brick Controls



SDLReader Smart Pro Controls



Contents

Welcome to the NextUI Docs!

A powerful but understated CFW for TrimUI Brick & Smart Pro.

Installation Instructions

About NextUI

NextUI is a custom firmware (CFW) for the TrimUI Brick & Smart Pro retro gaming handhelds.

It was started by **@ro8inmorgan** as a fork of the popular MinUI CFW so they could implement changes that were determined to be outside the scope of the original project.

NextUI quickly took off and became especially popular on the TrimUI devices.

Features

NextUI officially supports the TrimUI Brick and TrimUI Smart Pro gaming handhelds.

It features a rebuilt emulation engine and tons of added features outlined below.

Note: For other devices we recommend checking out **MinUI**.

- Fixed both screen tearing and sync stutter problems of MinUI by rebuilding the emulator engine core
- Game switcher menu (OnionOS style) by **@frysee**
- High quality audio, due to advanced resampling engine using **libsamplerate** (with quality/performance setting per emulator)
- Much lower latency (average 20ms, 1 frame of 60fps)
- Shaders!
- Fully OpenGL/GPU based for faster performance!
- Native WiFi and Bluetooth support
- System-wide support for USB-C DACs via OTG port
- Game art/media support
- Game time tracker
- Cheats support
- Overlays support
- Broader zip file support (BZ2, LZMA)
- Dynamic CPU speed scaling (keeps your device cool and extends battery life, but gives the performance right when needed)
- Custom bootlogos contributed by **@SolvalouArt**
- Color and font settings to customize the NextUI interface
- Menu animations
- LED control, change colors, effects, brightness
- LED indicators, low battery, stand by, charging (brightness can be set seperately)
- Haptic feedback in the menu by **@ExonakiDev**
- Ambient LED mode—the LEDS act like ambient light TVs for extra immersion. Configurable per emulator
- Display controls (color temperature, brightness, contrast, saturation, exposure)
- Configurable FN/Mute switch that can trigger a "night mode"—toggle between two sets of custom display settings!
- Dpad/Analog stick/Turbo toggle via FN switch
- Support for automatic NTP time synchronization with timezones and realtime clock (RTC)
- Deep Sleep mode, gives instant ON and avoids the overheat bug on the Brick by **@zhaofengli**
- Customizable screen and sleep timeout (Including "Never")
- Battery monitoring, including history graph and time left prediction
- Scrolling animated titles for longer game names
- Updated and optimized build emulation cores
- Rumble strength fixed and is now variable as used by the games

- FBNeo arcade screen rotation
- PAL mode support
- Next font supports CJK for JP/CN/etc. ROM names
- Lot of other smaller fixes and optimizations

Future Plans

- [x] RetroAchievements
- [x] Configurable function buttons
- [x] More compatibility testing with different emulators and fix / improve if necessary

---

Discord Community

NextUI has a vibrant Discord community. Here you can talk about new and upcoming features, ask for help and contribute to the community.

Many of our members produce custom Paks add to the functionality of NextUI.

The more artistically inclined among us make custom emulator overlays and themes.

Don't be shy, come join us! | **Discord Invite**

Kudos

Many minds make us whole. NextUI is a product of its community.

Special thanks to **@shauninman** for their original work on MinUI and to **@ro8inmorgan** and **@frysee** for their tireless efforts improving the core NextUI experience.

Usage

Getting Started

> ⚠ **Read before continuing!**
>
> NextUI **officially supports** the TrimUI Brick, Smart Pro and Smart Pro S.
>
> The steps outlined here only apply to these two devices running the stock operating system.
>
> If you are planning to install NextUI after using a different custom firmware on your device, we recommend to first reinstall the TrimUI stock OS to revert any changes that might have been made without your knowledge. The only exception is MinUI.

What You Will Need

- A TrimUI Brick or TrimUI Smart Pro running the stock operating system.
- A fresh micro SD card from a reputable vendor (SanDisk, Samsung, etc.)
  - The size of the card depends on the size of the ROMs you plan to play
- Software on your computer to format the SD Card
  - On macOS, use built-in Disk Utility application
  - On Windows, use **Rufus**
- The latest release of NextUI from the **GitHub Repo**.
  - At minimum, you will need the `base.zip` file.
  - If you want additional goodies, download `extras.zip` as well (or download `all.zip` to get both files in one).

Installation Instructions

> ⚠ **Formatting will erase all data from the SD card. If there is anything important on the card, please back it up.**

> ⚠ **Do not unzip `MinUI.zip`! Copy it to the root of the SD card as is.**

1. Insert the micro SD card into your computer.
2. Ensure that the SD card is formatted as a FAT32 or exFAT filesystem, and using the "Master Boot Record" partition scheme.
   - If not, use the specified tools in **What you will need** to format the card as FAT32/exFAT.
   - Double check you selected "Mater Boot Record (MBR)" partition scheme option, otherwise your card will not boot correctly.
3. Find and unzip the file you downloaded from the **GitHub Releases Page** (`all.zip` or `base.zip`).
4. Open the unzipped directory and copy all content to the root of your SD card.
5. Eject your SD card safely from your computer and insert it back into your device.

6. Power on your device. A screen will display stating NextUI is being installed.
    - Be patient while it installs. Do not turn off the power to your device during installation.
    - Your device may turn off after installation is complete.
7. Turn on your device and enjoy NextUI!

Updating an Existing Installation

> ⚠ **Do not unzip `minui.zip`! Copy it to the root of the SD card as is.**

1. Eject your NextUI SD Card and insert it into your computer.
2. Download a fresh copy of `base.zip` from from the **GitHub Releases Page**.
3. Find and unzip `base.zip`.
4. Open the unzipped directory and copy `miniui.zip` and the `trimui` directory to the root of your SD Card. You can safely overwrite / replace the `trimui` directory when performing the copy.
5. Eject your SD card safely from your computer and insert it back into your device.
6. Power on your device. A screen will display stating NextUI is being updated.
    - Be patient while it updates. Do not turn off the power to your device during the update.
    - Your device may turn off after the update is complete.
7. Turn on your device and enjoy the update!

Adding ROMs

NextUI creates a `Roms` folder at the SD Card Root containing folders for each console currently supported.

You can rename these folders as you like; however, you must keep the uppercase tag name in parentheses in order to retain the mapping to the correct emulator.

> ✨ **Example: Nintendo Entertainment System (FC) can be renamed to Nintendo (FC) or NES (FC) or Famicom (FC)**

If multiple folders share the same name, eg. `Game Boy Advance (GBA)` and `Game Boy Advance (MGBA)`, they will be combined into a single menu item containing the ROMs from both folders.

A ROM selected from this combined menu item will launch using the emulator in the tag of the folder is lives in.

Disc-based games

To streamline launching multi-file disc-based games, place your BIN / CUE files in a folder with the same name as the CUE file.

NextUI will automatically launch the CUE file instead of navigating into the folder when selected.

```
Tony Hawk's Pro Skater 2 (USA)/
  Tony Hawk's Pro Skater 2 (USA).bin
  Tony Hawk's Pro Skater 2 (USA).cue
```

For multi-disc games, follow these steps.

1. Create a folder for your disc files.
2. Put all the disc files into this folder.
3. Create a `.m3u` file that matches the name of the folder created in step one.
4. Edit the `.m3u` file and add the relative path to each disc's .cue file, one file per line.

NextUI will automatically launch the CUE file instead of navigating into the folder when selected.

For example, Final Fantasy VII has three discs:

```
Final Fantasy VII (USA)/
  Final Fantasy VII (USA).m3u
  Final Fantasy VII (USA) (Disc 1).bin
  Final Fantasy VII (USA) (Disc 1).cue
  Final Fantasy VII (USA) (Disc 2).bin
  Final Fantasy VII (USA) (Disc 2).cue
  Final Fantasy VII (USA) (Disc 3).bin
  Final Fantasy VII (USA) (Disc 3).cue
```

The `.m3u` file for FF7 would contain:

```
Final Fantasy VII (USA) (Disc 1).cue
Final Fantasy VII (USA) (Disc 2).cue
Final Fantasy VII (USA) (Disc 3).cue
```

When you are playing a multi-disc game, the NextUI in-game Menu will display the current disc.

Use `left` or `right` on the D-Pad to change discs.

NextUI also supports `.chd` files and `.pbp` files under 2GB.

> **Multi-disc games share the same memory card and save state slots across all discs.**

Collections

A collection is just a text file containing an ordered list of full paths to rom, cue, or m3u files. These text files live in the "Collections" folder at the root of your SD card, eg. `SDCARD_ROOT/Collections/Metroid series.txt` might look like this:

```
/Roms/GBA/Metroid Zero Mission.gba
/Roms/GB/Metroid II.gb
/Roms/SNES (SFC)/Super Metroid.sfc
/Roms/GBA/Metroid Fusion.gba
```

If you disable all visible folders under 'Roms', the 'Collections' folders contents will populate the main menu instead of being nested in the 'Collections' folder in the UI.

Display names

Certain (unsupported arcade) cores require roms to use arcane file names. You can override the display name used throughout NextUI by creating a map.txt in the same folder as the files you want to rename. One line per file, `rom.ext` followed by a single tab followed by `Display Name`. You can hide a file by adding a `.` at the beginning of the display name. eg. The 'Collections' folder needs its own map.txt file as well.

```
neogeo.zip  .Neo Geo Bios
mslug.zip   Metal Slug
sf2.zip     Street Fighter II
```

Doom PWADs

> **Warning**
>
> The PrBoom core *requires* the `prboom.wad` IWAD file - which is treated as a Bios file - and that file is available for download **here**. It can be placed in `/Bios/PRBOOM`.

NextUI supports Doom via the **PrBoom Libretro Core**, and loads Doom PWAD - or patch wad - files as it's game format. It also uses IWADs (internal Doom WADs) as the Bios files.

> **Note**
>
> The following documentation will use the fictional `NextUI Doom.wad` (Doom 1) and `NextUI Doom 2.wad` (`Doom 2) as the PWADs Megawads being loaded.

To setup a PWAD, place it in the `/Roms/Doom (PRBOOM)` folder on your SD Card.

```
/Roms/Doom (PRBOOM)/NextUI Doom.wad
```

PWADs all depend on a particular IWAD as the base for running the PWAD. IWADs are placed in the `/Bios/PRBOOM` folder, and a list of them is available in the **Optional BIOS** section. If all your PWADs use the same IWAD - for instance, `doom1.wad` then the IWAD can be placed directly in the `/Bios/PRBOOM` folder, and PrBoom will load the PWADs as expected.

```
/Bios/PRBOOM/doom1.wad
```

> **Note**
>
> All IWADs must be named using lowercase characters, including for the file extension.

In many cases, you will want to load specific PWADs with specific IWADs. Due to how PrBoom detects IWADs and lacking information about which IWAD is required by a PWAD, PrBoom may load the incorrect IWAD for your PWAD. To combat this, NextUI supports using a `doom.version` file to specify the correct `/Bios/PRBOOM` subdirectory to reference. Omitting a `doom.version` text file will result in the default PrBoom using the default IWAD detection algorithm. It is recommended to use a `doom.version` text file in conjuction with the `m3u` text file format commonly used for **Disc-based games** to tie PWADs to have a clean directory structure.

Using `NextUI Doom.wad` as an example, we would have the following directory structure in our Roms folder:

```
/Roms/Doom (PRBOOM)/NextUI Doom/NextUI Doom.wad
/Roms/Doom (PRBOOM)/NextUI Doom/NextUI Doom.m3u
/Roms/Doom (PRBOOM)/NextUI Doom/doom.version
```

The contents of the newly created `NextUI Doom.m3u` text would contain:

```
NextUI Doom.wad
```

While the newly created `doom.version` text file would contain the following:

```
doom1
```

The `doom.version` text file maps to a subfolder in the `/Bios/PRBOOM` folder that should be used to load dependencies, such as the IWAD or custom mp3 files. In the case of Doom 1 (Commercial), the commercial IWAD would be placed on the disk like so:

```
/Bios/PRBOOM/doom1/doom1.wad
```

Music for particular IWADs can also be customized by placing the **correctly named files** into the correct `/Bios/PRBOOM` subdirectory:

```
/Bios/PRBOOM/doom1/intro.mp3
/Bios/PRBOOM/doom1/e1m1.mp3
```

To load our `NextUI Doom 2.wad` megawad with only the title music changing, the following would be a sample file structure:

```
/Roms/Doom (PRBOOM)/NextUI Doom/NextUI Doom 2.wad
/Roms/Doom (PRBOOM)/NextUI Doom/NextUI Doom 2.m3u
/Roms/Doom (PRBOOM)/NextUI Doom/doom.version
/Bios/PRBOOM/doom2/doom1.wad
/Bios/PRBOOM/doom2/dm2ttl.mp3
```

The contents of `NextUI Doom 2.m3u` text file would be:

```
NextUI Doom 2.wad
```

And the `doom.version` text file would have the following as it's contents:

```
doom2
```

Optional BIOS

Some emulators require or perform better with the official BIOS.

NextUI is strictly BYOB and will not provide them nor links to them.

Google is your friend.

Use the table below to help you find and install the BIOS files you need.

> ⚠ **BIOS file names are case-sensitive!**

System	Tag	File Name(s)	BIOS Directory
Doom	PRBOOM	prboom.wad doom.wad doom2.wad doomu.wad plutonia.wad tnt.wad freedoom.wad freedoom1.wad freedoom2.wad	SDCARD_ROOT/Bios/PRBOOM/prboom.wad SDCARD_ROOT/Bios/PRBOOM/doom/doom.wad SDCARD_ROOT/Bios/PRBOOM/doom2/doom2.wad SDCARD_ROOT/Bios/PRBOOM/doom-ultimate/doomu.wad SDCARD_ROOT/Bios/PRBOOM/plutonia/plutonia.wad SDCARD_ROOT/Bios/PRBOOM/tnt/tnt.wad SDCARD_ROOT/Bios/PRBOOM/freedoom/freedoom.wad SDCARD_ROOT/Bios/PRBOOM/freedoom1/freedoom1.wad SDCARD_ROOT/Bios/PRBOOM/freedoom2/freedoom2.wad
Famicom	FC	disksys.rom	SDCARD_ROOT/Bios/FC/disksys.rom
Game Boy	GB	gb_bios.bin	SDCARD_ROOT/Bios/GB/gb_bios.bin
Game Boy Color	GBC	gbc_bios.bin	SDCARD_ROOT/Bios/GBC/gbc_bios.bin
Game Boy Advance	GBA	gba_bios.bin	SDCARD_ROOT/Bios/GBA/gba_bios.bin
Game Boy Advance	MGBA	gba_bios.bin	SDCARD_ROOT/Bios/MGBA/gba_bios.bin
Mega Drive / Genesis	MD	bios_CD_E.bin bios_CD_J.bin bios_CD_U.bin	SDCARD_ROOT/Bios/MD/bios_CD_E.bin SDCARD_ROOT/Bios/MD/bios_CD_J.bin SDCARD_ROOT/Bios/MD/bios_CD_U.bin
Amiga	PUAE	kick33180.A500 kick34005.A500 kick34005.CDTV kick37175.A500	SDCARD_ROOT/Bios/PUAE/kick33180.A500 SDCARD_ROOT/Bios/PUAE/kick34005.A500

System	Tag	File Name(s)	BIOS Directory
		kick37350.A600 kick39106.A1200 kick39106.A4000 kick40060.CD32 kick40060.CD32.ext kick40063.A600 kick40068.A1200 kick40068.A4000	SDCARD_ROOT/Bios/PUAE/kick34005.CDTV SDCARD_ROOT/Bios/PUAE/kick37175.A500 SDCARD_ROOT/Bios/PUAE/kick37350.A600 SDCARD_ROOT/Bios/PUAE/kick39106.A1200 SDCARD_ROOT/Bios/PUAE/kick39106.A4000 SDCARD_ROOT/Bios/PUAE/kick40060.CD32 SDCARD_ROOT/Bios/PUAE/kick40060.CD32.ext SDCARD_ROOT/Bios/PUAE/kick40063.A600 SDCARD_ROOT/Bios/PUAE/kick40068.A1200 SDCARD_ROOT/Bios/PUAE/kick40068.A4000
PC Engine	PCE	syscard3.pce	SDCARD_ROOT/Bios/PCE/syscard3.pce
Pokémon Mini	PKM	bios.min	SDCARD_ROOT/Bios/PKM/bios.min
Sony PlayStation	PS	psxonpsp660.bin	SDCARD_ROOT/Bios/PS/psxonpsp660.bin
Super Game Boy	SGB	sgb.bios	SDCARD_ROOT/Bios/SGB/sgb.bios

Managing Saves

NextUI places all save files by system in `SDCARD_ROOT/Saves`.

RetroArch .srm Support

By default, NextUI uses the emulator's default save file format.

RetroArch .srm save support can be enabled in Settings.

After making this change any existing save files will have to be renamed.

Directories and files that start with a period are hidden by default.

Windows Users: Open File Explorer, click the View Tab, look for the Show / Hide group and check the "show hidden files" box.

macOS Users: With a Finder window open, use the keyboard shortcut `Command + Shift + A`.

Managing Cheats

Cheats use RetroArch .cht file format. libretro maintains a **database of cheats**.

Cheat filenames needs to match the ROM name, including the ROM extension.

Place the cheat file in `SDCARD_ROOT/{System}/`.

Cheat filepath example

ROM filename: `Super Mario Land (World).zip`

ROM location: `SDCARD_ROOT/Roms/GB/`

Cheat file path: `/Cheats/GB/Super Mario Land (World).zip.cht`

When a cheat file is detected, it will show up in the "cheats" menu item in game.

Cheats are not supported by all cores.

LED Controls

Device support

NextUI's LED Controls currently only support the TrimUI Brick.

Support for the TrimUI Smart Pro will be addressed in a future update.

A comment on color reproduction

The TrimUI Brick is a budget device with budget LEDs. Don't expect the colors to display accurately.

LED Control App

On home screen, select `Tools`, then select `LedControl`.

The app provides the following options.

LED Selection

The TrimUI Brick has the following configurable LEDs.

- Two LEDs on the front of the device for each function button (F1 & F2)
- Two LEDs under the triggers (one under L1 & L2 the other under R1 & R2)
- One LED bar on the top of the device

When in the LED Control App you can use `Left Trigger` and `Right Trigger` to cycle between these LEDs.

Effects

Each LED can be configured with the following effects.

Effect Name	Description
Static	Keep the LED(s) on and static
Blink 1	Quickly blink 1 time
Blink 2	Quickly blinks 2 times
Linear	Slowly increase the brightness and then fall back to off
Breathe	Slowly increase the brightness and slowly decrease the brightness

Effect Name	Description
Interval Breathe	Slowly increase the brightness and slowly decrease the brightness, with a longer pause between "breaths"

Color

The color of LEDs. Use `Left` and `Right` on the `D-Pad` to cycle through the colors.

Speed

The speed of breathing effect in milliseconds.

Brightness

The brightness level of the LEDs. Setting this to `0` will turn the LED off.

Info Brightness

The brightness of LED when informing you about something.

Currently, this is only supported by:

- Power Button turning red alerting for low battery
- Front Function Button LED blinking when entering sleep

Setting brightness to `0` will turn the LED off.

Ambient Mode

> **Only supported By certain emulator cores**
>
> Ambient light effects are only supported by the built in Libretro cores.

Ambient light mode makes your LEDs change color to match the dominant color on screen during gameplay.

To enable ambient mode (in a supported emulator) follow these steps:

1. While in game, press the `Menu` button
2. Select `Options`
3. Select `Frontend`
4. Scroll down to the Ambient Mode line and turn it on. You can select to use all LEDs or just a specific one.

Ambient mode sets the LED Brightness to Maximum

We found that lower brightness levels will result in displaying an incorrect color.

Emulator Overlays

Overlays are only supported by libretro emulators

Standalone emulators installed via Paks do not support NextUI overlays.

NextUI looks for accompanying media for each emulator under the `/Overlays/[System]` folder. The `System` corresponds to the name of the Emu Pak specified in your Emulator folder name within parenthesis. In this folder you can add as many overlay PNGs as you want!

For example:

```
# For /Roms/Game Boy Advance (GBA)
/Overlays/GBA/overlay1.png
/Overlays/GBA/overlay2.png
/Overlays/GBA/overlay3.png
/Overlays/GBA/overlay4.png
/Overlays/GBA/overlay5.png

You do not need to follow this naming convention. You can name your overlay image files however
you'd like.
```

When in game, hit the `Menu` button and navigate to `Options -> Frontend`. Scroll to the `Overlay` setting and pick from the ones you added to the corresponding system's folder.

Adding Quick Menu Png's

Custom backgrounds for the Quick Menu are loaded from `mnt/SDCARD/.media/quick_<settings>.png` with the following valid names.

Quick Menu Background PNG Directory Structure

```
SD_CARD
├─ .media/
│   ├─ quick_Collections.png                    *Collections List background
│   ├─ quick_Games.png                 *Games List background
│   ├─ quick_Poweroff.png              *System Power Off background
│   ├─ quick_Reboot.png                *System Reboot background
│   ├─ quick_Settings.png              *Settings background
│   ├─ quick_Sleep.png                 *Deep Sleep background
│   ├─ quick_Tools.png                 *Tools list background
│   ├─ quick_Wifi.png                  * Toggle Wifi OFF background (Shown if Wifi is currently ON)
│   ├─ quick_Wifi_off.png              *Toggle Wifi ON background (Shown if Wifi is currently OFF)
│   ├─ quick_Recents.png               *Recently Played List background
│   ├─ quick_Pak Store.png             *Pak Store background
│   ├─ quick.png                    *Generic fallback background for all slots
```

Additional Customization notes

Icon assets (small and big) can be replaced as usual in `.system/res`. Please note that these are present in multiple sizes, e.g. "3x" for Brick and "2x" for TSP. The option to completely hide the UI elements of the quick switcher and build your own UI (with just the background images) is located in Appearance Settings. Also note that the Pak Store and Collections background will only appear if each are present.

Adding Emulator Icons

Create a `.media` directory under the corresponding folder to create icons for Emulators, Collections and Tools.

Emulator Icon Directory Structure

```
Example System Used: Game Boy Advance (MGBA)
Example Rom Used:    My Awesome Game.gba
Example Pak Used:    Artwork Scraper.pak


SD_CARD
├─ bg.png                                  *Default Menu Background
├─ .media/
│  ├─ Collections.png                      *Collections List Icon in Main Menu
│  ├─ Recently Played.png                  *Recently Played List Icon In Main Menu
├─ Collections/
│  ├─ .media/
│  │  ├─ bg.png                            *Collections List Background
│  │  ├─ Collection 1.png                  *Collection Icon In Collections List
│  │  ├─ Collection 2.png                  *Collection Icon In Collections List
├─ Overlays/
│  ├─ MGBA/
│  │  ├─ overlay1.png                      *MGBA System Overlay #1
│  │  ├─ overlay2.png                      *MGBA System Overlay #2
│  │  ├─ overlay3.png                      *MGBA System Overlay #3
│  │  ├─ overlay4.png                      *MGBA System Overlay #4
│  │  ├─ overlay5.png                      *MGBA System Overlay #5
├─ Recently Played/
│  ├─ .media/
│  │  ├─ bg.png                            *Recently Played List Menu Background
├─ Roms/
│  ├─ .media/
│  │  ├─ Game Boy Advance (MGBA).png      *Main Menu Console Icon
│  ├─ Game Boy Advance (MGBA)/
│  │  ├─ .media/
│  │  │  ├─ bg.png                         *MGBA Rom List Background
│  │  │  ├─ My Awesome Game.png            *Box Art / In-game Preview
│  │  ├─ My Awesome Game.gba               *Rom Location
├─ Screenshots/
│  ├─ screenshot.png                       *Screenshot Monitor Pak PNG Output Location
├─ Tools/
│  ├─ .media/
│  │  ├─ tg5040.png                        *Tools list Icon in Main Menu
│  ├─ tg5040/
│  │  ├─ .media/
│  │  │  ├─ bg.png                         *Tools List Main Menu Background
│  │  │  ├─ Artwork Scraper.png            *Individual Tool Icon in Tools List
```

Add Custom Boot Logo

To add a custom boot logo. Place a 24-bit color `.bmp` image under the corresponding folder for your device. Then, select **Tools** > **Bootlogo** to set your custom logo

Directory Location

```
SD_CARD
├─ Tools/
│  ├─ tg5040/
│  │  ├─ Bootlogo.pak/
│  │  │  ├─ brick/
│  │  │  │  ├─ your_logo.bmp                    *For Brick devices - Add your IMAGE here
│  │  │  ├─ smartpro/
│  │  │  │  ├─ your_logo.bmp                    *For SmartPro devices - Add your IMAGE here
```

Recomended Specs:

- Depth: 24-bit color
- Dimensions: 1024 x 768px (for Brick)

N.B. You can use any of the existing logo options as a template for yoru custom logo if you wish to maintain the same sizing.

Game Artwork

NextUI looks for accompanying media for each emulator under `/Roms/[Emulator]/.media`.

To add artwork:

1. Create the `.media` folder if it does not exist.
2. In the `.media` folder, add an image in `PNG` format with the exact same name as the ROM file (NextUI will automatically scale the artwork).

Here's an example:

```
# With an SFC game located at:
# /Roms/SFC/My Awesome Game.smc
# the box art or in-game preview is located at:

/Roms/SFC/.media/My Awesome Game.png
```

The **Artwork Scraper pak** can also be used to automatically download artwork for your device.

Custom Categories

If folders have the same name prior to the brackets for the emulator, they will be merged. So make one for each emulator, then create subfolders for the systems that use that emulator (note in the second pic you can still use numbers to order the consoles.

Emulator Directory Structure

```
Example Systems Used: Game Boy (GB), Game Boy Advance (MGBA), NES (FC), Sega Genesis (MD)
Example Rom Used:    My Awesome Game.ext
Example Pak Used:    Artwork Scraper.pak


SD_CARD
├─ bg.png                                    *Default Menu Background
├─ .media/
│  ├─ Collections.png                        *Collections List Icon in Main Menu
│  ├─ Recently Played.png                    *Recently Played List Icon In Main Menu
```

```
├─ Collections/
│  ├─ .media/
│  │  ├─ bg.png                          *Collections List Background
│  │  ├─ Collection 1.png                *Collection Icon In Collections List
│  │  ├─ Collection 2.png                *Collection Icon In Collections List
├─ Overlays/
│  ├─ MGBA/
│  │  ├─ overlay1.png                    *MGBA System Overlay #1
│  │  ├─ overlay2.png                    *MGBA System Overlay #2
│  │  ├─ overlay3.png                    *MGBA System Overlay #3
│  │  ├─ overlay4.png                    *MGBA System Overlay #4
│  │  ├─ overlay5.png                    *MGBA System Overlay #5
├─ Recently Played/
│  ├─ .media/
│  │  ├─ bg.png                          *Recently Played List Menu Background
├─ Roms/
│  ├─ .media/
│  │  ├─ 01)Handheld (GB).png              *Main Menu Handheld Icon (match first folder in group)
│  │  ├─ 02)Console (FC).png            *Main Menu Console Icon (match first folder in group)
│  │  ├─ 03)Arcade (FBN).png            *Main Menu Console Icon
│  │
│  ├─ 01)Handheld (GB)/             *Uses (GB) to store systems using the GB emulator
│  │  ├─ .media/
│  │  │  ├─ 01)Game Boy.png         *GB system icon
│  │  │  ├─ bg.png              *Background used when highlighting Handheld in the main menu (only
required for first folder in group)
│  │  │
│  │  ├─ 01)Game Boy                    *The number here sorts the systems in the Handheld screen
│  │  │  ├─ .media/
│  │  │  │  ├─ bg.png               *Background for GB that will be shown in the Handheld menu
│  │  │  │  ├─ My Awesome Game.png          *Rom boxart
│  │  │  ├─ My Awesome Game.gb          *Rom file
│  │
│  ├─ 01)Handheld (MGBA)/           *Uses (MGBA) to store systems using the MGBA emulator
│  │  ├─ .media/
│  │  │  ├─ 02)Game Boy Advance.png     *GBA system icon
│  │  │
│  │  ├─ 02)Game Boy Advance                     *The number here sorts the systems in the Handheld
screen
│  │  │  ├─ .media/
│  │  │  │  ├─ bg.png               *Background for GBA that will be shown in the Handheld menu
│  │  │  │  ├─ My Awesome Game.png          *Rom boxart
│  │  │  ├─ My Awesome Game.gba         *Rom file
│  │
│  ├─ 02)Console (FC)/
│  │  ├─ .media/
│  │  │  ├─ 01)NES.png              *NES system icon
│  │  │
│  │  ├─ 01)NES                         *The number here sorts the systems in the Console screen
│  │  │  ├─ .media/
│  │  │  │  ├─ bg.png               *Background for NES shown in console menu
│  │  │  │  ├─ My Awesome Game.png          *Rom boxart
│  │  │  ├─ My Awesome Game.nes         *Rom file
│  │
│  ├─ 02)Console (MD)/
│  │  ├─ .media/
│  │  │  ├─ 01)Sega Genesis.png         *Sega Genesis system icon shown in Console menu
│  │  │  ├─ 01)Sega Master System.png       *Sega Master system icon shown in Console menu
│  │  │
│  │  ├─ 02)Sega Genesis                    *The number here sorts the systems in the Console screen
│  │  │  ├─ .media/
│  │  │  │  ├─ bg.png               *Background for Sega Genesis shown in Console menu
│  │  │  │  ├─ My Awesome Game.png          *Rom boxart
│  │  │  ├─ My Awesome Game.md          *Rom file
│  │  │
│  │  ├─ 03)Sega Master System                  *The number here sorts the systems in the Console
screen
```

```
│   │   │   ├── .media/
│   │   │   │   ├── bg.png                *Background for Sega Master System shown in Console menu
│   │   │   │   ├── My Awesome Game.png          *Rom boxart
│   │   │   ├── My Awesome Game.sms          *Rom file
│   │
│   ├── 03)Arcade(FBN)/
│   │   ├── .media/
│   │   │   ├── bg.png               *Background for FBN that will be shown in the Main menu
│   │   │   ├── My Awesome Game.png             *Rom boxart
│   │   ├── My Awesome Game.zip
```

Notes for Merged Folders

Folders with the same name (with the exception of the brackets) will be merged. To use this functionality, create folders with matching names (ex. Handheld, Console, Nintendo, Sega, etc.) and in brackets, every emulator you will need for the systems you want sorted into here.

Think of the folders in 3 tiers SD/Roms/CategoryName(emulator)/Gamesystem/Games

In tier 2, Create folders for each system that uses the emulator listed in its parent folder ex. for Console (MD), MD can emulate Sega Genesis, Sega 32x, Sega CD & Sega Master System, so all those system folders should be placed in Console (MD)

In tier 3, place all game files. Additionally at this point subfolders can be easily made for subcategories like rom hacks, translations, etc.

Icons and background for the merged folders only need to be made for the first folder alphabetically. In the example directory structure above, the icon for Game Boy was titled 01)Handheld (GB).png and the background was placed in Roms/01)Handheld (GB)/.media/bg.png

```
├── Screenshots/
│   ├── screenshot.png                          *Screenshot Monitor Pak PNG Output Location
├── Tools/
│   ├── .media/
│   │   ├── tg5040.png                          *Tools List Icon in Main Menu
│   ├── tg5040/
│   │   ├── .media/
│   │   │   ├── bg.png                          *Tools List Main Menu Background
│   │   │   ├── Artwork Scraper.png             *Individual Tool Icon in Tools List
```

Shaders Explanation

A Little Introduction

In today's world, most of us have screens with resolutions of at least 1920×1080 and higher. However, back in the days of our favorite retro games, this was far from the case. Older TVs operated at around 720×525 resolution, and handheld devices like Game Boys had even lower resolutions.

While we enjoy our collections of retro games today, there's one big problem: our modern screens usually have resolutions far beyond what these games were originally designed for. In practice, this means we either play our games in a tiny square in the middle of our screens to preserve their original resolution, or (what most of us prefer) we scale the image up to better fit our modern, high-resolution displays.

There are three main scaling modes you can choose from in the frontend options under Screen Scaling:

- Native—This keeps the game at its original resolution, resulting in that tiny box in the center of your screen.
- Aspect—This scales the image to your screen while maintaining the original aspect ratio, so nothing looks distorted (no "fat" Super Mario). It scales the image up until either the width or height reaches the screen size. This often leaves empty space on the sides or top and bottom, unless you're lucky enough to have a screen that matches the original aspect ratio exactly.
- Full Screen Stretch—This simply stretches the image to fill the entire screen without preserving the aspect ratio. While it fills the screen nicely, it can make things look strange if the original and screen aspect ratios don't match.

Ultimately, the choice is yours based on what you prefer. However, understanding screen scaling is important because it plays a big role in how shaders are used — although scaling isn't the only reason shaders are popular, it is a major one.

Lets Get Shady!

Options → Frontend → Screen Sharpness

First, let's talk a little about this option.
The GPU in your device basically has two modes to upscale an image to the final screen size.

The first is **NEAREST**, which is faster because it's less complex. It simply upscales the image by doubling each pixel. This looks very sharp, but it can make an image look *too* sharp, especially if the input (the output of your emulator core) is at a very small resolution. NEAREST just doubles the pixels until it reaches the desired size.

The second option is **LINEAR**. This costs a little more performance-wise, but instead of just doubling pixels, it applies linear interpolation between pixels, creating a smoother, softer look.
This can definitely create a more even, pleasant image — but again, it depends heavily on what the original source image looks like. If your emulator is outputting a very small image, LINEAR interpolation has to invent a lot of new pixels to fill the screen, and the result can look very blurry.

The takeaway here is simple:
The smaller the original image the GPU receives to upscale to your screen's resolution, the worse it will look.
Neither NEAREST nor LINEAR can magically make a tiny 320×240 image look amazing on a big 1024×768 screen.
That's just been a fact of life since the early days of computers.

Now, if you really just want to *rawdog* the tiny emulator output directly to the screen for maximum performance, I recommend using **NEAREST**.
LINEAR usually just ends up looking too blurry, while NEAREST faithfully replicates the pixels from the source, simply making the image bigger — simple and sharp.

So, does that mean old games will *always* look like crap on modern screens?

Well, no, not really! Welcome to the world of shaders!

A Note on the Word "Shader"

Before we dive in, a quick note:
The word *shader* is actually kind of wrong in this context. In emulation, "shaders" have become synonymous with "effects," but technically speaking, shaders are just small programs that tell your GPU what to do with pixel data.
Every image you see on your screen — even a simple one without effects — is being drawn by a shader.
Even the most basic output still runs a shader that simply says: "draw a rectangle the size of the screen and fill it with the emulator's pixel data."

Okay, with that out of the way — since we already use shaders to draw the screen, **we can also use them to alter the look of the image!**

Making LINEAR Look Good

Let's start with a very basic idea that already looks great for most systems and might even be all you need:

Remember how I said NEAREST is better because LINEAR gets too blurry?
Well, **LINEAR can actually be awesome — if we prepare the image a little first.**

Here's the problem:
When you scale a small image directly up to a big resolution with LINEAR interpolation, the GPU has to invent a *lot* of made-up pixels, which makes everything blurry.

(For example, scaling a 320×240 image up to 1024×768 — almost two-thirds of the final pixels are *imaginary!* No wonder it looks so bad.)

So what if we first scale the image sharply with NEAREST — and *then* apply LINEAR interpolation afterward?

In other words:
First, double (or triple) the sharp pixels with NEAREST, then let LINEAR smooth out only the small leftover gaps.
Best of both worlds!

Example:

- Emulator outputs 320×240.
- First, upscale to 640×480 using NEAREST (or even 960×720 if you want).
- Then, apply LINEAR interpolation when stretching it the rest of the way to your screen's full resolution.

Result?
It looks *way* better! This is because LINEAR now only has to invent half as many pixels as it's working with a bigger starting point.

Pay special attention to how all the pixels on the ground are now equeally square sized, compared to the previous example without shaders where the pixels where unequal because of the stretching. The shaders now made sure everything stretches nicely and the pixels don't deform even though the image is stretched to a different aspect ratio/size than the game's original ones.

How to Set This Up

First, let's set a shader to upscale the image 2× using NEAREST:

Go to:

> Options → Shaders

And set the following:
- **Number of shaders:** 1 (we only need one)
- **Shader 1:** `stock.glsl` (basic shader that just outputs the input image)
- **Filter:** NEAREST (very important — we want a sharp, clean NEAREST upscale first!)
- **Source type:** Relative or Source (doesn't matter — they're the same for the first shader)
- **Texture type:** Relative or Source
- **Scale:** 2 or 3 (Use 2× for SNES, MD, etc., or 3× for very small images like GB/GBC.)

What this shader does:
It takes the emulator's output, scales it up 2× or 3× using NEAREST, and passes it on to the next stage (in this case, directly to the screen, since we're only using one shader).

Now, let's apply the final LINEAR smoothing when outputting to the screen:

Go to:

> Options → Frontend → Screen Sharpness

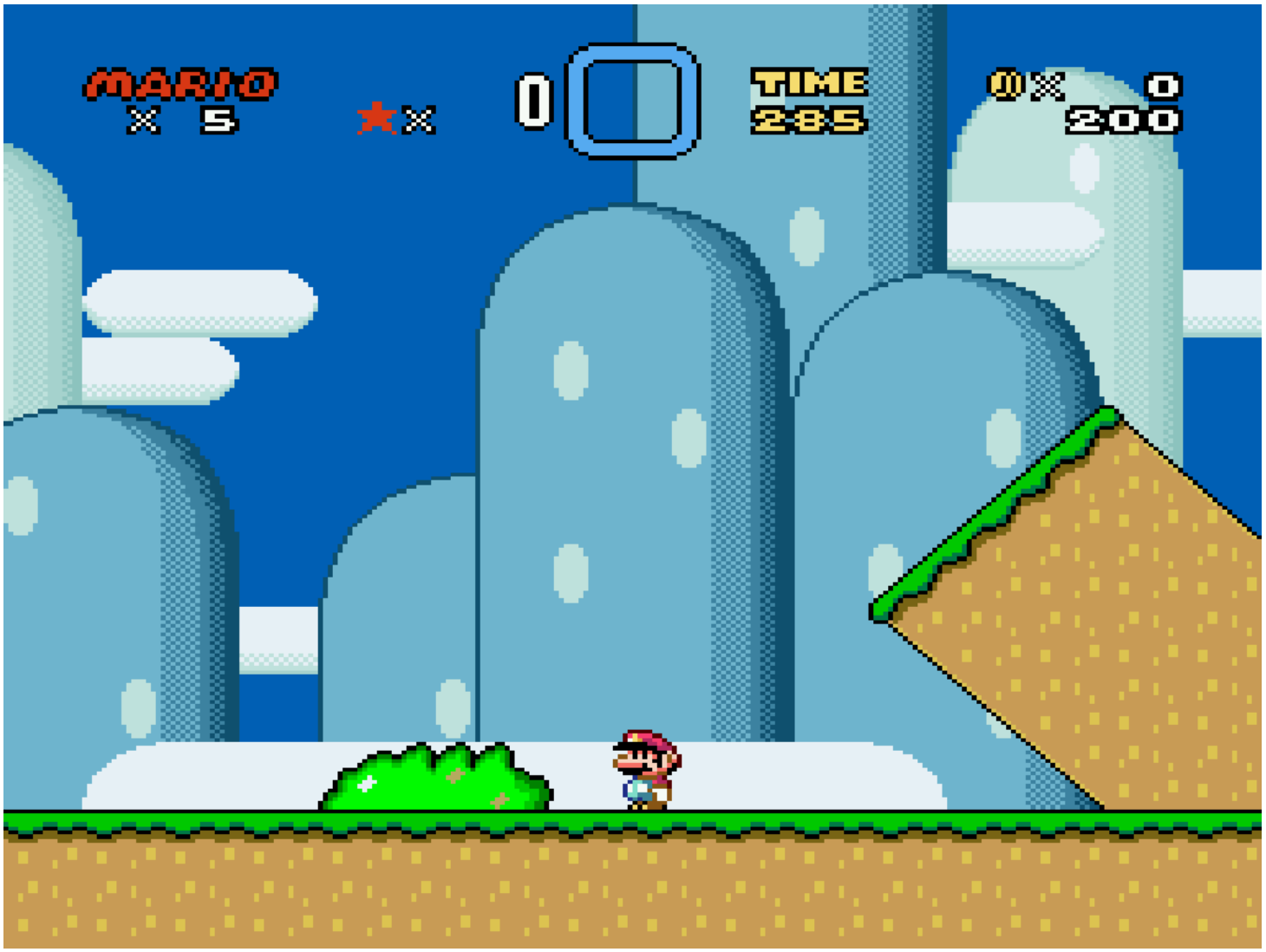
And set this to **LINEAR**.

So now:

- First, the shader upscales the image 2× or 3× sharply with NEAREST.
- Then, the frontend upscales the rest of the way using LINEAR to smooth it out slightly.

And the result already looks MUCH better, right?

The first image below is without shaders, just straight NEAREST upscaling to fullscreen. The second one uses the above settings to make things a little smoother and nicer. This looks really good on the Brick's high DPI screen.


MARIO
X 5
0
TIME
285
0
200

A Few Final Notes

The "world of shaders" is all about **chaining small steps together** — each shader alters the image slightly before passing it along.

- **Choosing NEAREST or LINEAR only matters when the step involves resizing.**
  If a shader step uses a Scale of 1 (no scaling), the NEAREST or LINEAR setting has no effect at all. The same goes for the screen sharpness setting in frontend options. If an image already is at screen size before it reaches this final step then the final filter does nothing because it doesn't have to up or downscale anymore. NEAREST and LINEAR only apply when the image has to resize either up or down. It's not a filter applying to the image, it just tells the GPU what method to use when up (or down) scaling an image.

But like I said shaders are not only used for scaling, since they are just little programs that run on your GPU they can do anything. There are tons of shaders out there (`.glsl` files) that do all kinds of fun things:

- Simulate CRT screens
- Mimic Game Boy Advance LCD looks
- Apply different smoothing/sharpening algorithms
- Add scanlines, fake grid patterns
- Distort the screen or enhance colors, etc.

Feel free to experiment!
Try different shaders, different combinations, and find the look that feels best for your games. You can download any .glsl file and place it in the `/Shaders` folder on your SD card to use it within NextUI. NextUI does not yet support .glsp shader preset files, though these are simple text files and you can open them to view the settings and replicate them in the shaders menu.

Enjoy! 🎮

BONUS: GB/GBC/GBA Tip!

Want to make your Game Boy, Game Boy Color, and Game Boy Advance games look even cooler? Here's a quick bonus trick!

Set everything up the same way as above, **but now set the number of shaders to 2** and configure the second shader like this:

- **Number of shaders:** 2
- **Shader 2:** `lcd3x.glsl` (a shader that overlays a pixel grid, simulating old handheld screens)
- **Filter:** NEAREST (NEAREST recommended for a sharp grid, but LINEAR could work more for you)
- **Source type:** Source (important — I'll explain why below)
- **Texture type:** Source
- **Scale:** Screen

What This Does

This setup applies an additional shader called `lcd3x`, which overlays a grid on the image to mimic the look of original Game Boy, Game Boy Color, and GBA screens.
It's a simple effect, but it adds a lot of nostalgic charm!

Why Use Source and Screen?

Here's the idea:

- Setting **Source** as the **Source/Texture Types** tells the shader to use the original dimensions from the emulator output — the real, original pixel layout. However, for the actual pixel data, it still uses the output from the previous shader step.
- While we're at it: **Relative** means "use the dimensions of the previous shader step," and **Screen** means "use the dimensions of the screen" (with aspect ratio correction applied).
- Setting the **Scale** to **Screen** means the shader will output an image with a grid drawn over it (where the grid itself is based on the original pixel size) at the screen's resolution. The upscaling happens here, and no further upscaling is needed.

So, the `lcd3x` shader creates a grid based on the original pixel size but with an output size that covers the entire screen.
You end up with an LCD-style grid that perfectly matches the original pixel layout, giving each pixel a distinct look —

while in reality, the image is being upscaled to your screen's full resolution.
It looks like your Game Boy screen suddenly has four times as many pixels. If that ain't cool I don't know what is!

Important Note About Filtering

Since Shader 2 is already stretching the image up to full screen, **the frontend's Screen Sharpness setting no longer matters.**
At this point, the image is already at its final size before it even reaches the frontend's final filter.

That means:

> **The final image sharpness is controlled inside Shader 2 itself, not by the frontend option anymore.**

That's it — now you've got a sharp, beautiful pixel grid overlay just like an old-school Game Boy screen! 🎮🖤

Frequently Asked Questions

Here are the answers to questions that come up often. If you have a question that isn't answered here, hit up the **Discord** .

Where Can I Download NextUI?

You can always find the latest NextUI release on our **GitHub repo**.

How Do I Install / Update?

For a fresh install of NextUI, follow our **installation guide**.

Already a NextUI user? Follow our **updating guide** to grab the latest release.

I am currently using classic MinUI, how can I migrate?

Since MinUI and NextUI share a lot of DNA, you can just install NextUI over an existing MinUI installation.

There is no need to format your exising card, just follow the **installation guide** from step 3, using the `all.zip` archive from the latest NextUI release.

> **While this is a pretty straightforward process, it never hurts to have a backup of your savegames.**

What Systems are Supported?

Included in the Base CFW

- Game Boy (gambatte)
- Game Boy Color (gambatte)
- Game Boy Advance (gbsp, mgba)
- Nintendo Entertainment System (fceumm)
- Sega 32X (picodrive)
- Sega CD (picodrive)
- Sega Game Gear (picodrive)
- Sega Genesis (picodrive)
- Sega Master System (picodrive)
- Sega SG-1000 (picodrive)
- Sony PlayStation (pcsx_rearmed)
- Super Nintendo Entertainment System (snes9x, mednafen_supafaust)
- Super Game Boy (mgba)

Additional emulators (extra.zip)

- Amiga (puae2021)
- Amstrad CPC (cap32)
- Arcade (CPS, MAME, etc - fbneo)
- Atari 2600 (stella2014)
- Atari 5200 (a5200)
- Atari 7800 (prosystem)
- Colecovidion (gearcoleco)
- Commodore 64 (vice_x64)
- Commodore 128 (vice_x128)
- Commodore PET (vice_xpet)
- Commodore Plus4 (vice_xplus4)
- Commodore VIC20 (vice_xvic)
- Doom (prboom)
- Microsoft MSX/MSX2 (bluemsx)
- Neo Geo Pocket/Color (race)
- Pico-8 (fake08)
- Pokémon mini (pokemini)
- TurboGrafx-16/TurboGrafx-CD (mednafen_pce_fast)
- Virtual Boy (mednafen_vb)

Available as community paks

- DOS (dosbox)
- Nintendo 64 (mupen64plus)
- Nintendo DS (DraStic)
- ScummVM
- Sega Dreamcast (flycast)
- Sony PlayStation Portable (PPSSPP)
- Portmaster
- Wonderswan (mednafen_wonderswan)

Why Am I Only Seeing Tools?

Ensure that you have a `Roms` folder at the root of your SD card with at least one ROM file.

For more info, visit the **Adding ROMs** page.