



6CCS3PRJ

Emmy, The Game Boy Emulator

Final Project Report

Author: Oscar Sjöstedt

Supervisor: Ian Kenny

Student ID: K20040078

Programme of Study: Computer Science MSci

April 5, 2023

Abstract

This project aims to create a Game Boy emulator web-application, in other words a program capable of receiving Game Boy game files (commonly referred to as ROMs), and interpreting such ROM to play the game, or execute the program, it contains. The emulator will be usable in browsers, for both desktop computers and mobile devices that may not have access to a physical keyboard. The emulator will also contain debugging capacities, to allow other emulator developers to use it when comparing with their emulator and working on it.

The objective of this project is to create a piece of software that could be used by anyone wanting to emulate retro games, without the need for any technical knowledge on emulators or downloading anything (except the ROMs that need to be obtained separately).

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Oscar Sjöstedt

April 5, 2023

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Scope	3
1.3	Objectives	4
2	Background	5
2.1	Emulation	5
2.2	Video Game Emulation	6
2.3	Game Boy, Game Boy Color	6
2.4	Existing Literature	7
3	Requirements and Specification	10
3.1	Requirements	10
3.2	Specification	11
4	Design	14
4.1	Emulator Frontend	14
4.2	CPU	16
4.3	PPU, APU, Joypad	17
4.4	System Bus	19
4.5	Other Components	19
4.6	Useful Classes	20
5	Implementation	22
5.1	Emulator Frontend	23
5.2	Emulator-Frontend Interfaces	29
5.3	CPU	32
5.4	System	37
5.5	PPU	41
5.6	APU	46
5.7	MBCs and ROMs	47
5.8	Timer	49
5.9	Helpful Components	51

6	Evaluation	54
6.1	Absolute Accuracy	54
6.2	Relative Accuracy	56
6.3	Performance	60
6.4	Compliance to Specification	62
7	Legal, Social, Ethical and Professional Issues	64
7.1	Privacy	64
7.2	Legality	64
7.3	Integrity	65
8	Conclusion and Future Work	66
8.1	Conclusion	66
8.2	Future Work	66
	Acronyms	68
	Bibliography	71
A	User Guide	72
A.1	Disclaimer	72
A.2	Playing games on Emmy	72
A.3	Settings	73
A.4	Keybindings	74
A.5	Debugging	75
B	Source Code	77
B.1	Emulator Core Code	80
B.2	Frontend Code	171
C	GBEmulatorShootout Contribution	236
C.1	Emmy Testing Code	236

Chapter 1

Introduction

1.1 Motivation

Emulators are an area of computer science widely used today. Either implemented in hardware or software, they allow replicating the behaviour of one system on another. One of its applications is video game emulation, where a computer simulates a game console (usually a retro console). This allows users to play games that either may not be obtainable in stores anymore, or made for consoles that do not function properly anymore. A wide range of emulators already exist for most consoles. Emulation in general is also widely used in developing new systems, and is an active area of computer science.

This project will seek to create a new emulator for the Game Boy allowing users to play retro games on their computer or mobile device, through the browser. This report will document how the original console works and how the emulator imitates this behaviour to the best accuracy possible, as well as comparing the resulting emulator with other existing ones.

1.2 Scope

The scope of this project is creating a new Game Boy and Game Boy Color emulator, working for browsers. Emulation will be as accurate as achievable with the time available – there may be minor inaccuracies in the end product. Extra peripherals and features of the console may be omitted, to allow more focus to be put on the core part of the console.

The emulator will be usable across a range of devices. Debugging tools and additional features may be provided to the user to let them customise their experience to their needs.

1.3 Objectives

The resulting software will allow users to open a game file for the Game Boy (also called a ROM) and play it. They may use the emulator on a computer, controlling the console via the keyboard, or on a touch device, using on-screen buttons. The emulated features of the emulator include proper rendering of the screen, simulating the audio of the console, the different buttons, and support for a variety of chip controllers for game cartridges.

The frontend of the emulator may also contain additional quality of life features, such as custom themes, save states, and debugging options allowing to inspect the state of the Gameboy – a feature vital to emulator developers and retro game developers.

Chapter 2

Background

2.1 Emulation

An emulator is “hardware or software that permits programs written for one computer to be run on another computer” [1]. The imitated computer is the *guest*, and the one that imitates is the *host*. Emulators are nowadays mainly found in the form of software, and have many different uses, from preservation to hardware development.

Emulation was born with the first computers: the very first computer, the Colossus made in 1941, was built to imitate the Enigma machine [2]. However emulation was properly studied in the 1980s, when computing power started to steadily increase. One of the earliest instances of emulation as an actual feature is with the IBM System/360. This computer supported emulation of previous models, such as the IBM 709, 7090, 7094 and 7094 II [3].

Emulation is also vital for preservation: as transistors and motherboards age, old systems become unusable, and with them the software they ran. Companies may also stop producing the hardware to run this software on. Emulating these systems is often the only future-proof and sustainable way to keep this software usable [4].

Finally, another common use for emulation is virtual machines. These programs allow running another Operating System (OS) on a computer, which can be used for instance when developing for other systems, without needing to use the physical device directly, for instance when developing a Windows-compatible app with a Linux computer. In the case where the architectures of the guest and the host are the same, we call this *virtualisation* [5]. Virtualisation is nowadays extremely common, with tools like Docker¹, that allow multiple virtualised instances

¹See <https://www.docker.com/>

of computer systems running on the same host. This typically allows for great portability, as the same infrastructure can be copied and ran anywhere.

2.2 Video Game Emulation

Video game emulation is the art of emulation applied to video game hardware systems. This allows the host to run games destined for the original console. This usually requires precise understanding of the console’s hardware and functioning, as games may rely on specific behaviours and edge cases to function. This task is rendered harder by the fact that the only legally available source of information on these consoles comes from research and reverse-engineering done by hobbyists, and does not come from an official source, as it is proprietary hardware.

Video game emulation started in the 90s when computers were powerful enough to properly simulate console systems. Although precise dates are hard to get, the first console emulators seem to be either from 1990 or 1993 [6], and were able to run some NES games. The first Gameboy emulators were in the late 90s, with the Virtual GameBoy² in 1995 and NO\$GMB³ in 1997 (although its history page⁴ seems to indicate development started in 1993) [7].

The original game files and assembled code for video games are copyrighted material, and are referred to as the Read-Only Memory (ROM) of the game. Although distributing these ROMs is usually illegal, there also exist copyright free ROMs: games created by developers that chose to license them under Creative Commons licenses, for instance. Websites such as Retro Veteran⁵ host wide collections of legal ROMs.

Official emulators also exist, and are developed by the console manufacturers. These usually only allow playing from a selection of games, limiting options. The reason for this is that they are built to emulate these specific games, rather than the console as a whole, meaning games outside of the catalogue will often not work. They are thus usually less accurate than unofficial emulators – this is the case of the Virtual Console, the official emulator of the Nintendo 3DS, that fails many Game Boy test ROMs [8, Test ROMs].

2.3 Game Boy, Game Boy Color

The Game Boy (GB) is an 8-bit handheld video game console, released in 1989. It has a small 160×144 pixel screen, and has a Sharp LR35902 as its Central Processing Unit (CPU), clocked

²See <http://fms.komkon.org/VGB/>

³See <https://problemkaputt.de/gmb.htm>

⁴See <https://problemkaputt.de/gmbhist.htm>

⁵See <https://www.retroveteran.com/category/nintendo-game-boy-color/>

at 4.19MHz [9, Specifications]. In 1998, the Game Boy Color (GBC) was then released. Seen as the successor of the GB, it contains a screen of the same resolution, but supporting colour, from a palette of 32768 options (15 bits per colour). It contains the same CPU as its predecessor, a Sharp LR35902, with now two modes: a 4.19MHz mode and a 8.38MHz mode (double-speed mode). This allows the GBC to be backwards compatible with most GB games – there are a few exceptions to this, games that used hardware bugs of the original GB that were fixed in the GBC [10, STAT IRQ glitches].

From an emulation perspective, the Game Boy Color can thus be seen as an extension of the Game Boy – it has an identical CPU (although with a toggle-able double speed mode), and most of the memory layout is identical. To keep the remaining of this document simple, if not stated, “GB” will refer to both the original Game Boy and the Game Boy Color, as they are very similar. Dot Matrix Game (DMG) refers exclusively to the original Game Boy model.

2.4 Existing Literature

2.4.1 Gameboy Documentation

The Game Boy is one of the best documented consoles for emulation, and a large array of resources exist documenting it. Some useful resources explaining its behaviour are:

- Pandocs⁶ is a technical reference of how the GB works. It is extremely complete and covers a wide range of topics, so it is useful to get a global view of a problem. It is one of the most referenced pieces of literature on the console.
- GB CPU Instructions⁷ is a table containing all instructions its CPU has, as well as information on the amount of cycles taken by the instruction, the bytes of memory used, the flags affected by the operation, and a description of the instruction.
- Gameboy Complete Technical Reference⁸ (GBCTR) is an unfinished document that contains very detailed information on the CPU and other components of the GB. Although incomplete, it provides a much lower-level view of the details of the GB (compared to Pandocs), making it useful to emulate very specific behaviour like the cycle-by-cycle timing of the CPU.
- GB dev wiki⁹ is a wiki containing additional information on the GB, including guides to making games and explanations on some hardware quirks, and in particular a very precise

⁶See <https://gbdev.io/pandocs/>

⁷See <https://meganesu.github.io/generate-gb-opcodes/>

⁸See <https://gekkio.fi/files/gb-docs/gbctr.pdf>

⁹See <https://gbdev.gg8.se/wiki/>

description of the Audio Processing Unit (APU).

2.4.2 Existing Emulators

A wide range of emulators for the GB and GBC already exist, and many of them are open-source. These are useful when developing a new emulator, to see how they work internally. For performance reasons they are usually written in compiled languages, such as C++ and Rust, but some interpreted language alternatives exist. These emulators include:

- Game Boy Crust¹⁰ is a simple GB emulator written in Rust. It is quite incomplete but has a comprehensive structure, so its a good project to first figure out how emulators work.
- AccurateBoy¹¹ is a highly accurate emulator, in particular for its Picture Processing Unit (PPU) that has pixel-perfect accuracy.
- oxideboy¹² is another GB emulator written in Rust, that is much more complete and helpful for some edge cases.
- SameBoy¹³ is one of the most accurate open source GB and GBC emulators, written in C. It is much more technically complex but still useful to understand edge cases, especially since it is the emulator used as a reference when developing this project.
- Mooneye GB¹⁴ is a GB research emulator written in Rust. It passes most of the Mooneye test ROMs, making it helpful when encountering issues with these tests.
- GameBoy-Online¹⁵ is a high-accuracy JavaScript emulator, that is particularly useful to understand how to interface the emulator with the browser (notably for the APU).
- Gameboy.js¹⁶ is another JavaScript emulator. It is fairly simple and inaccurate, but is easily hackable, making it useful when starting a new emulator to compare execution traces.
- rboy¹⁷ is an emulator written in Rust that was used when developing the APU, as it passes some complex test ROMs with quite simple code.

The 8th of February 2023, Nintendo announced the release of a Game Boy and Game Boy Color emulator on the Nintendo Switch, via the Nintendo Switch Online subscription [11]. The recent release of official emulators as well as public enthusiasm for the latter prove the relevance

¹⁰See <https://github.com/mattbruv/Gameboy-Crust>

¹¹See <https://github.com/Atem2069/accurateboy>

¹²See <https://github.com/samcdays/oxideboy>

¹³See <https://github.com/LIJI32/SameBoy>

¹⁴See <https://github.com/Gekkio/mooneye-gb>

¹⁵See <https://github.com/taisel/GameBoy-Online/>

¹⁶See <https://github.com/juchi/gameboy.js/>

¹⁷See <https://github.com/mvdnes/rboy>

of this kind of emulator.

2.4.3 Gameboy Test ROMs

A core set of resources to develop an emulator are test ROMs. These are ROMs which instead of playing a game will run a series of tests on the console. These tests are first written to pass on the physical console itself, and are then used to ensure they also pass on the emulator. This means issues in specific components can be easily diagnosed (so long as the rest of the emulator responsible for running the test ROM works itself). These test ROMs also have the advantage of being open source, meaning their source code can be referred to in order to understand what they expect of the console.

An other advantage of using test ROMs is that they tend to re-use the same framework across a given test suite to report results. This means testing can easily be automated over multiple tests by inspecting specific registers/memory addresses, rather than having to store an “expected result” image for each test.

The test ROMs used for this project are:

- Blaarg test ROMs¹⁸ are some of the most well-known and used GB test ROMs. They include tests for the CPU, the timings of instructions, hardware bugs and the APU.
- Mooneye test ROMs¹⁹ is a very complete test suite, that verifies most components of the GB: CPU instructions, memory timings of specific instructions, behaviour of Memory Bank Controllers (MBCs), timings of the APU, Direct Memory Access (DMA), PPU and timer.
- Acid Test (DMG²⁰, GBC²¹) is a test that verifies that the PPU of the GB displays data properly (to line-rendering accuracy), for both Game Boy and Game Boy Color displays.
- SameSuite²² is a test suite that is valuable for its APU tests: it uses the PCM12 and PCM34 registers exclusive to the GBC to inspect the exact output of the APU (whereas other test ROMs tend to inspect the on/off status of the channels, which is much less accurate).

¹⁸See <https://github.com/retrio/gb-test-roms/>

¹⁹See <https://github.com/Gekkio/mooneye-test-suite>

²⁰See <https://github.com/mattcurrie/dmg-acid2>

²¹See <https://github.com/mattcurrie/cgb-acid2>

²²See <https://github.com/LIJI32/SameSuite/>

Chapter 3

Requirements and Specification

3.1 Requirements

3.1.1 User Requirements

- U1. Run Game Boy games to a satisfiable fidelity, with proper rendering and controls emulation.
- U2. Run Game Boy Color games to a satisfiable fidelity, with proper rendering and controls emulation.
- U3. Allow the user to run GB and GBC games on both Game Boy and a Game Boy Color.
- U4. Allow the user to save the state of the game, to continue their playthrough later. The state can simply be saved as a downloaded file, and re-uploaded later to continue the game.
- U5. Allow the user to change the speed at which the game is played: double speed mode, half speed mode, etc.
- U6. Have some debug functionality, to inspect the state of the console at any given time.
- U7. Allow users to pause the console, and add breakpoints to stop execution at specific moments.
- U8. Allow the user to switch between rendering modes (nearest-neighbour, LCD display, Scale2x, etc.)
- U9. Allow the user to switch the colour palette of the DMG emulation.

3.1.2 System Requirements

- F1. The system can receive a ROM file, construct an instance of the emulated console, and run the code inside said ROM.
- F2. The system emulates different components of the GB and GBC, with as much precision as possible (M-cycle precision).
- F3. The system renders the output of the emulator to a Web `<canvas />`.
- F4. The system creates the required DOM elements for the web-app, and updates them as needed.
- F5. The system listens to key presses and releases to emulate controls through the keyboard.
- F6. For touch devices, the system may render buttons to simulate the console's controls.

3.1.3 Non-Functional Requirements

- N1. The emulator should be accessible on computers through a web browser equipped with a recent version of JavaScript.
- N2. The emulator should be accessible on mobile devices through a web browser equipped with a recent version of JavaScript.
- N3. The emulator should be accessible on computers through a standalone app.
- N4. Maximise the tests passed by the emulator (see [Gameboy Test ROMs](#)).
- N5. Have the code be well documented, allowing new-comers to the project and to GB emulation to easily understand what is going on – if possible with links to relevant Game Boy emulation resources.

3.2 Specification

Code	Specification	Importance
U1	User can upload a GB ROM file (.gb), and the emulator will run the game. The keyboard can be used to control the game, and the output is displayed.	High

Code	Specification	Importance
U2	User can upload a GBC ROM file (<code>.gbc</code>), and the emulator will run the game. The keyboard can be used to control the game, and the output is displayed.	Medium
U3	User can upload a GB ROM file (<code>.gb</code>). The user can switch between a DMG and a GBC emulator.	Medium
U4	User can press a button to download a save of their game (or, alternatively, the save can be stored inside the browser with a technology like IndexedDB ¹).	Low
U5	User can select the speed of emulation, to dynamically accelerate/decelerate the game.	Medium
U6	User can see debug information of the emulator. This information includes the current tileset, background map, time to draw a frame, and register information.	Low
U7	User can pause the console emulation through a button. They can also input conditions for which the console should break execution.	Low
U8	User can dynamically switch the rendering filter via a dropdown button.	Low
U9	User can dynamically switch the colour palette of the GB via a dropdown button.	Low
F1	A ROM file can be uploaded, is transformed into an <code>UInt8Array</code> (because the GB is an 8-bit system), and the appropriate object is created to run the code.	High
F2	Different components exists as different classes, respecting typical OOP principles such as encapsulation and inheritance when relevant.	High
F3	A <code><canvas /></code> element is created, and is updated with the output of the emulator after every frame is drawn (ie. at the start of each VBlank mode).	High
F4	The Preact ² framework is used to handle the UI of the web-app.	High

¹See https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

²See <https://preactjs.com/>

Code	Specification	Importance
F5	Listeners are added to the environment's <code>window</code> to listen to all key presses and releases. The emulator can then request for a control update, by reading the state of keys.	High
F6	If a touch device is detected, button are added to the UI and are used by the emulator as inputs.	Medium
N1	A deployed version of the web-app is accessible on a desktop browser and provides full functionality, via keyboard and mouse inputs.	High
N2	A deployed version of the web-app is accessible on a mobile device browser and provides full functionality, via touch controls.	Medium
N3	A downloadable version of the web-app can be used on a computer and provides full functionality, via keyboard and mouse inputs.	Low
N4	As many possible tests as possible should be passed, while ensuring previously passing tests do not fail.	Medium
N5	Main methods and variables must be properly documented, and have links to appropriate online resources to documentation about said element.	High

Chapter 4

Design

In this chapter we will outline the main components of the Gameboy and of this emulator. For the different GB components, we will briefly go over their role, and how they interact with other components and to what end. The emulator’s name will be “Emmy”, short for emulator¹.

Because the emulator should not rely on the environment it is running in to work, the functionality of the project can be split into two parts: the emulator core, that is responsible for simulating the Game Boy, and the emulator’s frontend, that allows interfacing with the user, and may be changed to work for different platforms (see [Figure 4.1](#)).

In emulators, the different components are usually split into three groups:

- The CPU, responsible for reading instructions and changing the state of the console. This is what drives the emulation, as other components usually idle unless acted upon.
- Input and output components, such as the APU, the PPU and the joypad. These components interact with the outside user, by either outputting the game state, or reading inputs from the user.
- A memory system, that handles addressing within the console. This part is essential to ensure components are communicating between each other properly, since different addresses may map to different components.

4.1 Emulator Frontend

To allow the user to access the emulator, a frontend needs to be built with it. This frontend is responsible for outputting the graphics and audio data of the emulator, and also for receiving

¹The emulator was named before the realisation that an emulator-related project called “Emmy” already exists [12]. These two projects are unrelated.

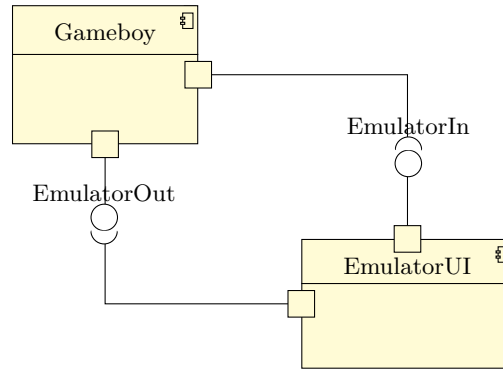


Figure 4.1: Split of emulator core and frontend

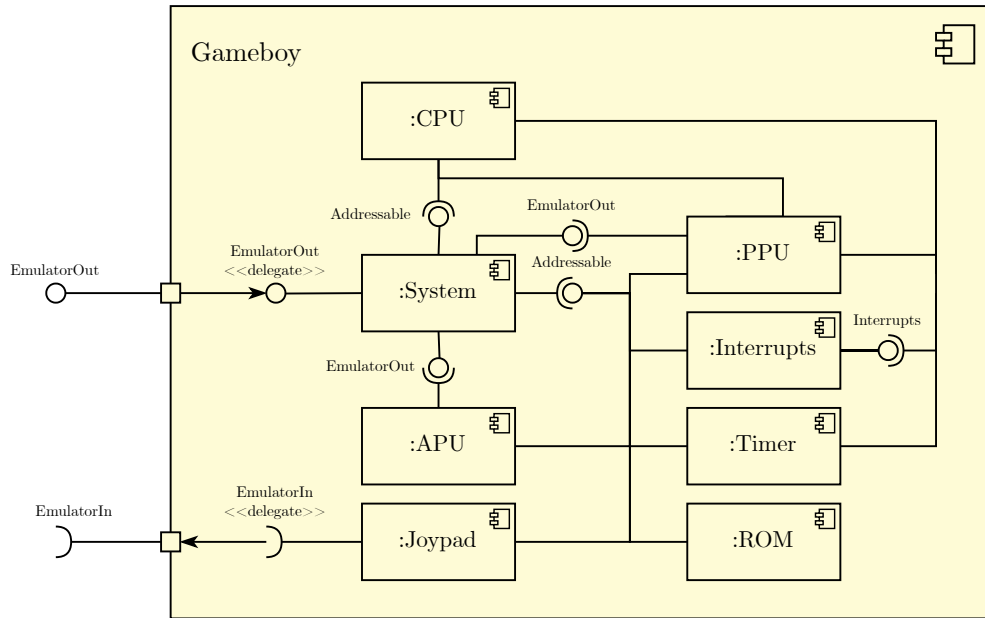


Figure 4.2: Simplified design of the emulator's core
Note utility classes have been excluded for readability purposes

input from the user to pass down to the emulator. Aside from that, extra UI components have been added, to make the usage of the emulator more convenient and add more features: a play and pause button, custom themes, debugging tools, upscaling filters and keybinding options. All of these are kept independent from the emulator, and control transformations of the input and output, or changes in the running of the emulator.

Because this implementation is merely to allow access to the emulator and does not have any outstanding functionality, its description will be brief – most of the focus will be put on the emulator's core.

4.2 CPU

The Central Processing Unit (CPU) is perhaps the most complex part of the GB. It is however also one of best documented parts of it, making it quite easy to emulate properly. The CPU model is a Sharp LR35902, which is a hybrid of the Intel 8080 chip and the Zilog Z80 [13].

Most emulator are typically “CPU-driven”. This means that the CPU is what drives the emulation. If an instruction takes multiple hardware cycles to execute, the CPU is responsible for asking the rest of the system to tick at the appropriate time. In the design chosen for this project however, the emulator will be “System-driven”, with the CPU only being part of the overall running of the emulator – it will not be responsible for ticking the rest of the system. This difference will be made obvious by the interfaces exposed to the CPU; it allows for nicer encapsulation and better separation of concerns.

The CPU is thus responsible for stepping through the program its given. It has 6 different 16-bit registers, with different roles. Some registers’ bytes can be accessed independently, operating as two 8-bit registers or one 16-bit register as needed [9, CPU Registers and Flags].

- **AF**: the accumulator-flag register. Most arithmetic operations can be done on **A**. **F** holds the CPU’s flags, and can only be altered indirectly, as a consequence of arithmetic operations.
- **PC**: the program counter register. It cannot be accessed directly, and is used to store the current position of the CPU in the program. It can be altered via **CALL**, **RET**, **JP** operations, for instance.
- **SP**: the stack pointer register. It can be modified or incremented, and stores the pointer to the top of the “stack”. A typical use of the stack is for handling functions, by pushing the current address to the stack whenever a procedure is called, and popping it back to the stack pointer when it returns.
- **BC, DE**: two simple registers that can be used for arithmetic operations.
- **HL**: similar to **BC** and **DE**, it can also be used as an address by some operations, allowing the use of indirect addressing.

Aside from its registers, the CPU needs to interact with system interrupts, and needs to access memory for reading/writing operations.

4.3 PPU, APU, Joypad

The Picture Processing Unit (PPU) is a complex part of the Gameboy, that handles outputting the game state to a 160×144 screen, that may either be monochrome on the DMG or with colour support on the GBC. It may raise interrupts, and needs to be able to access memory, due to it being responsible for the Object Attribute Memory (OAM) Direct Memory Access (DMA) – a feature allowing transfer of data from an arbitrary location in memory [9, OAM DMA Transfer].

The behaviour of the PPU changes significantly between the DMG and the GBC: it can, for the latter, support colours, and it also supports Video RAM (VRAM) bank switching, a feature the DMG lacks [9, CGB Registers]. It also has an Object Attribute Memory (OAM) DMA mechanism that is quite complex. As such, to make maintenance easier, the PPU is split into different smaller classes. This is not visible to the rest of the system, as only the PPU is exposed to it. See Figure 4.3 for a simplified class diagram of its structure.

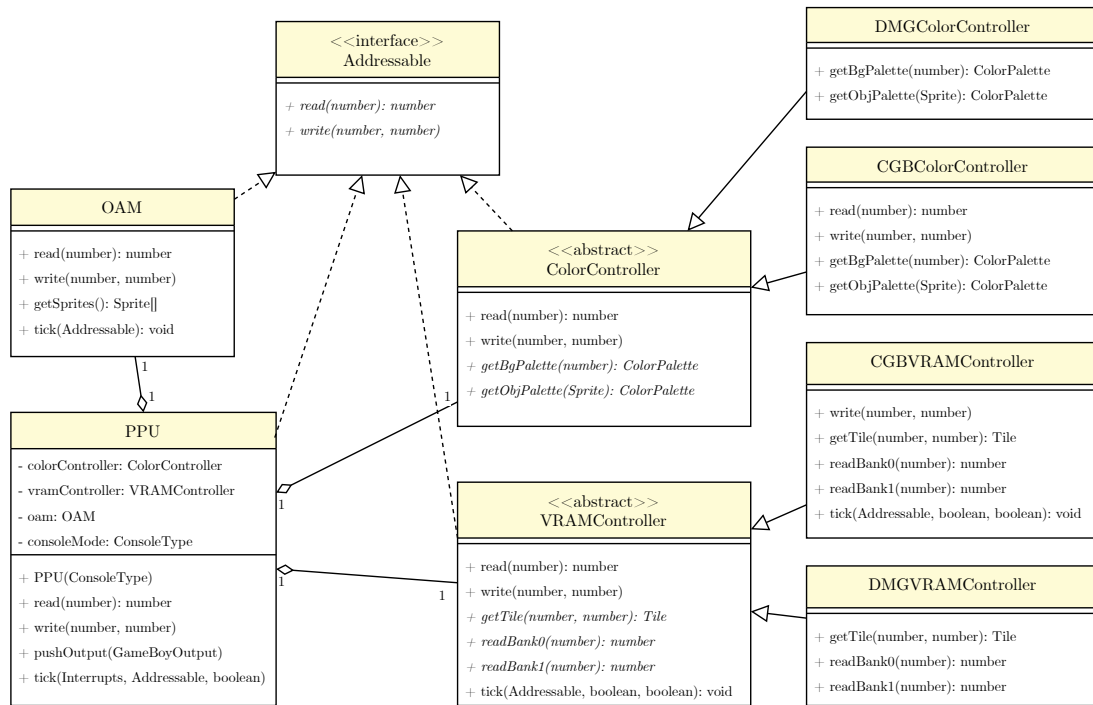


Figure 4.3: Simplified class diagram of the PPU

Note most private fields and methods have been excluded for readability purposes

The Audio Processing Unit (APU) is responsible for playing the audio output of the console. It has a few registers to control it, as well as a short memory area called the wave RAM. Its timings are controlled by the timer, which can be accessed via a memory read of the system.

Its sound output is derived from four separate channels: two square wave channels, a custom wave channel, and a noise channel [9, Audio]. These channels are here modelled as separate entities, but are purely internal (see [Figure 4.4](#)).

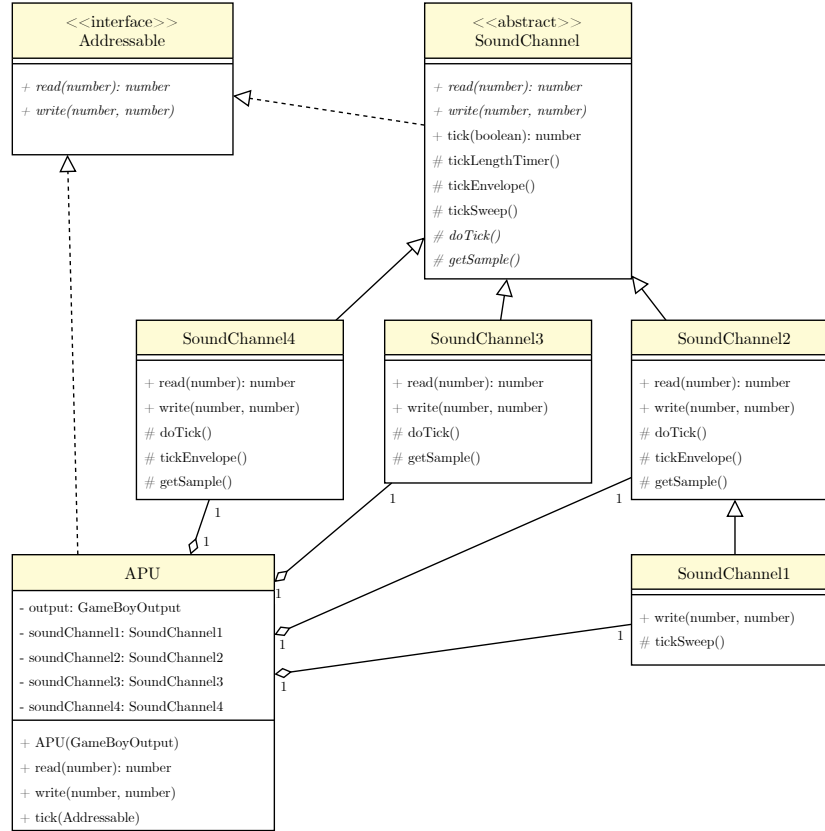


Figure 4.4: Simplified class diagram of the APU

Note most private fields and methods have been excluded for readability purposes

The former two components need to output information to the user, either graphic or audio data. To do this in a portable way, the emulator will expose an interface, that can be implemented by the front-end. This allows the emulator core to be platform-agnostic, and be easily re-usable.

The joystick, on the other hand, is the one input component the GB has. It contains 4 directional arrow buttons, and 4 input buttons: A, B, start and select (see [Figure 4.5](#)). To use these inputs, the CPU needs to use two different registers.

Similarly as for the output, the emulator core will expose an interface to read the user's input, ie. the state of all 8 buttons.

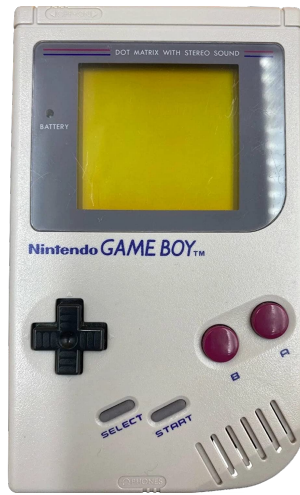


Figure 4.5: Picture of a Gameboy, with the screen and joypad visible

4.4 System Bus

Components within the Gameboy need to communicate. To do this, components typically have a set of registers that control how they behave – for instance, the timer has a **TAC** register at `0xFF07` to control its frequency. To allow this interaction without having all components directly depend on each other, a “system bus” component is created, responsible for handling all components and managing memory-addressing among them.

This component, called **System**, is thus responsible for instantiating the other components, and providing them access to each other, via a read and write method.

4.5 Other Components

Aside from the aforementioned components, the GB has a few components that allow it to run but are not part of the CPU or the input/output components. These will be briefly outlined here, as they have a lesser impact on the emulator’s design.

4.5.1 Timer

The timer, responsible for incrementing a counter (the **DIV**) at a set frequency. It can raise interrupts, and owns a few registers.

4.5.2 Interrupts

The interrupt system, that allows components such as the timer or the PPU to interrupt the CPU to run another process. It can also be enabled and disabled by the CPU, with the EI and DI instructions [9, Interrupts].

This sub-system is designed as a separate component to reduce coupling between other components, to instead have a small object that can be passed to components as needed.

4.5.3 MBC

The Memory Bank Controller (MBC) is an extra chip contained within some game cartridges, to allow access to more ROM data (as well as external RAM in some cases) via banking [9, MBCs]. There are multiple different MBCs, and so it is convenient to define a common interface for all of them, which can then be implemented according to specification for each MBC type.

The type of the MBC is in the header of the ROM [9, The Cartridge Header]. To separate this logic from the base system, a `ROM` class is used. It is responsible for reading the cartridge's header, and creating the appropriate MBC instance.

4.6 Useful Classes

To have similar interfaces over all components, we will also declare specific classes and interfaces to be implemented by them.

4.6.1 Addressable

The `Addressable` interface (see [Figure 4.6](#)) provides a read and write method. This allows all components to communicate between each other without needing to be aware of what the component they are communicating with is. Aside from the CPU, all components of the emulator implement this method.

```
1 interface Addressable {  
2     read(pos: number): number;  
3     write(pos: number, data: number): void;  
4 }
```

Figure 4.6: `Addressable` interface

4.6.2 Memory and registers

Simple utility classes can be declared to manage memory in a simple way.

The `RAM` class implements `Addressable`. It can be instantiated with a set size, and can be read and written to. It is used in components that have large blocks of writable data.

The `Register` class implements `Addressable`, but ignores the address parameter of the read and write operations, as it contains only one byte. A `DoubleRegister` class may also be implemented, backed by two `Registers`, to provide support for 16-bit registers, used for instance in the CPU.

Chapter 5

Implementation

The project uses Preact¹, a light-weight alternative to the more popular React² framework. Preact was chosen because the front-end of the web-app is extremely lightweight, so a smaller framework with less features is enough. This also avoids bloating the app with a heavy framework such as React: its GZipped and minified size is around 31.8Kb, while Preact is only 4Kb (87% less) [14].

The language used for the project is TypeScript³, a typed version of JavaScript. This is essential for the project, as ensuring the correctness of code can be extremely hard without proper typing constraints, in particular when the codebase becomes larger and more complex.

The project is divided into two parts:

- The `frontend/` directory contains the UI for the web-app. The main logic to create the emulator and run it is contained in `app.tsx`.
- The `emulator/` directory contains the actual GB emulator. Although most classes and interfaces used are exported, only three elements are needed to properly interact with the emulator:
 - `GameBoyColor.ts` handles the core loop of the system. It contains the `GameBoyColor` class, the emulator. Instantiating the emulator creates all the necessary sub-components, and calling the `drawFrame()` method runs the emulator for one frame (0.16 seconds).
 - `GameBoyOutput.ts` contains a simple interface, with optional methods to receive any output produced by the emulator (see Figure 5.1). The two main methods of this are `receiveGraphics` and `receiveSound`, which use the output of the actual console.

¹See <https://preactjs.com/>

²See <https://reactjs.org/>

³See <https://www.typescriptlang.org/>

- `GameBoyInput.ts` contains a simple interface with a required `read()` method that returns an object with the current inputs for the console (see [Figure 5.2](#)).

```
1 interface GameBoyOutput {
2   receiveGraphics?(data: Uint32Array): void;
3   receiveSound?(data: Float32Array): void;
4
5   // Debugging methods:
6   debugBackground?(data: Uint32Array): void;
7   debugTileset?(data: Uint32Array): void;
8   serialOut?(data: number): void;
9 }
```

Figure 5.1: `GameBoyOutput` interface methods

```
1 type GameBoyInputRead = {
2   up: boolean;
3   down: boolean;
4   left: boolean;
5   right: boolean;
6
7   a: boolean;
8   b: boolean;
9   start: boolean;
10  select: boolean;
11 };
12
13 interface GameBoyInput {
14   read(): GameBoyInputRead;
15 }
```

Figure 5.2: `GameBoyInput` interface method

5.1 Emulator Frontend

The frontend of the emulator is written in Preact, allowing for the creation of a simple, fast and lightweight UI to control it. It contains the emulator title, the control buttons and the emulator video output (see [Figure 5.3](#)). Along the left of the screen is a sidebar with more options, allowing the user to customise the emulator to their needs and debug the state of the console if needed.

5.1.1 Main Controls

The main controls for the emulator are the 6 buttons above the screen. These are, from left to right:

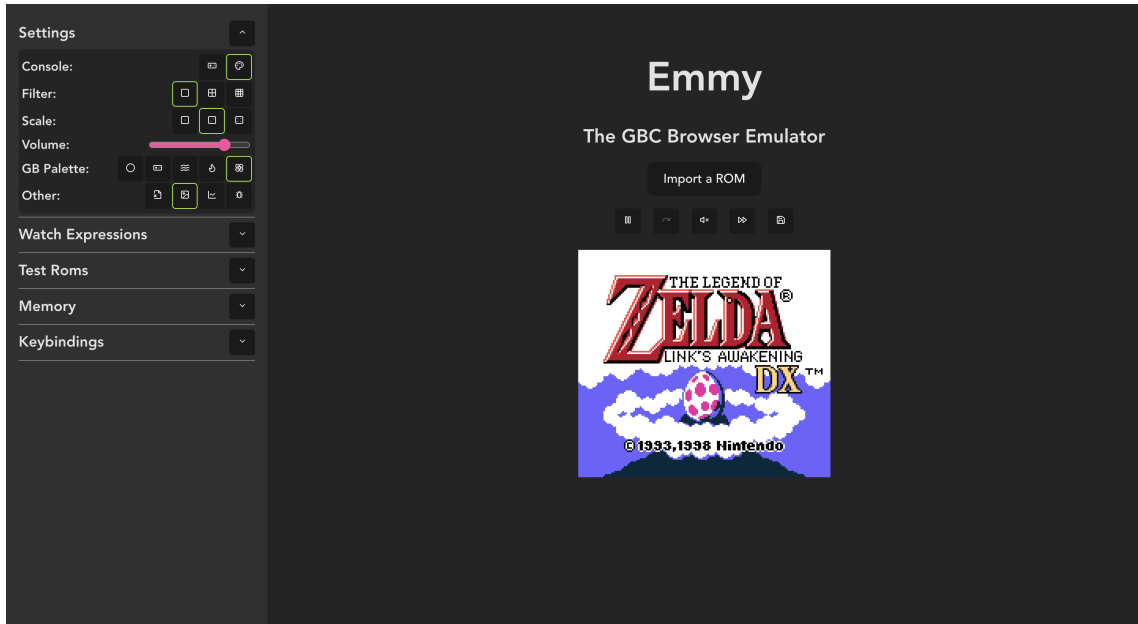


Figure 5.3: Home page of the emulator

- Pause/play: pauses or resumes the emulator.
- Step by one CPU instruction: this is a feature useful for debugging the emulator, when precise information of the state of the emulator is needed.
- Sound toggle: allows enabling or disabling the sound of the emulator. By default the sound is turned off, because modern browsers do not permit websites playing any sound before the user interacts with the website [15].
- Triple speed toggle: a toggle button that speeds up the emulator to emulate the GB thrice as fast as usual. This is a common feature found in most emulators.
- Save state: saves the current state of the cartridge in the browser's storage, allowing the user to resume playing the game later. Note this does not save the full state of the emulator, but that of the cartridge, making it equivalent to a real-life save on the GB where only the battery-backed storage of the cartridge persists through power-offs.

5.1.2 Settings

In the side drawer, the first tab is “Settings”. It contains general settings for the emulator, such as:

- the console used by the emulator. This may either be the DMG or the GBC.
- an extra filter to be applied to the output. This increases the resolution of the screen,

using the Scale2x or Scale4x⁴ algorithms (see Figure 5.4).

- the size of the emulator’s screen, allowing resizing to two or four times larger.
- a slider to change the volume of the emulator’s audio output.
- a palette selector, to change the four hues of the DMG’s screen.
- two buttons to upload the boot ROMs of the DMG and GBC. This is needed for users that wish to play with the full start-up screen of the console, as these two ROMs are copyrighted and cannot be distributed. If they are not provided, the emulator simulates their effect on the system.
- miscellaneous toggle-able settings, grouped together:
 - an option to play with or without the initial boot ROM of the emulator (a boot ROM must have been uploaded).
 - a toggle to enable frame-blending, meaning for every new frame the output is mixed with the previous frame. This is a nice addition to have, because certain games made some objects flicker on screen to make them appear translucent (since the flicker was not visible to the eye).
 - a button to show the performance statistics of the emulator. This is mainly useful when developing the emulator to make sure it is still efficient.
 - a toggle to enable the debug view of the emulator, where the currently loaded tileset and background map are displayed (see Figure 5.5).



Figure 5.4: Output of the GBC with, from left to right: no filter, Scale2x and Scale4x

5.1.3 Watch Expressions

The second drawer allows the user to define custom JavaScript functions to inspect the state of the emulator regularly (see Figure A.4). This requires knowledge of the inner structure of the emulator, as the field names need to be used, but is quite useful when needing to inspect parts

⁴See <https://www.scale2x.it/>

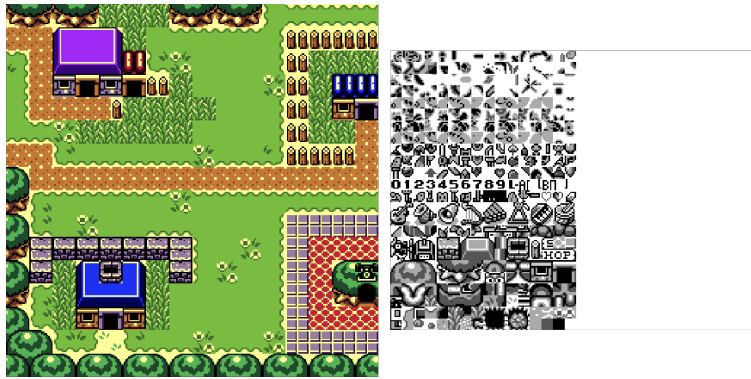


Figure 5.5: Debug view of the emulator

of the console that are not accessible via memory, like internal counters used for components, or register values.

To implement this, the `Function`⁵ constructor is used, which takes in a string with the function's code. This allows the user to dynamically change the expression, and the component will simply update the function, without needing to reload the whole application. These expressions are also automatically saved to `localStorage`⁶, meaning they will be kept between sessions.

The user-defined function is then repeatedly invoked, and the result output below the expression, allowing for a live-status of the emulator. If an error is thrown by the function (due to a null value, or invalid expression), the error is caught and 'Error' is displayed.

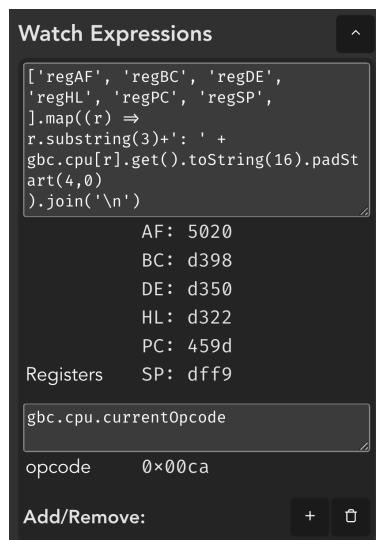


Figure 5.6: Watch expressions drawer

⁵See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/Function

⁶See <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

5.1.4 Test ROMs

To ensure emulators work properly, a variety of test ROMs have been made, that test most aspects of the GB (see [Gameboy Test ROMs](#)). The front-end of the emulator supports running a large number of them in an automated way (see [Figure 5.7](#)). The user can select the group of tests desired by ticking the associated checkbox, or select and unselect everything by pressing the top right button. They can then run them by pressing the “Test” button, making all selected tests run internally (without receiving any input or outputting anything directly). The status of each individual test is displayed, and the user can click on the test name to run it on the main emulator, for further debugging or inspection.

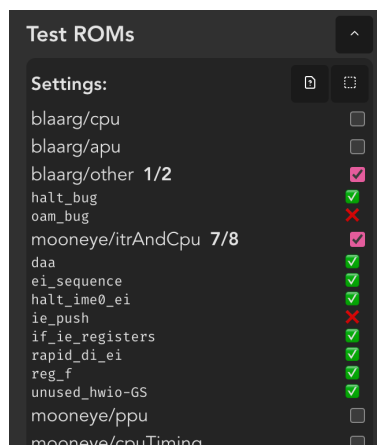


Figure 5.7: Test ROMs drawer

To run the tests, an emulator instance is created, with the test ROM as the input data and “spy” objects provided for input and output – these spies store the output of the emulator, to evaluate the state of the test. To stop execution, the emulator’s state is inspected regularly, and the outcome of the test is checked (with execution terminating if the test takes too long to end).

To detect the outcome of each test, these are grouped according to what test suite they belong to. Each test suite is then associated to a *success function*, that can either return “success”, “failure” or null if no outcome has been reached yet. As such, all tests in a test suite have a similar way of reporting success and failure.

The currently automated test suites are:

- The Blaarg test ROMs⁷. The success function of this suite is quite complex, as this suite does not have a standard way of outputting the result. As such multiple parts of the

⁷See <https://github.com/retrio/gb-test-roms>

emulator are inspected simultaneously:

- "Passed"⁸ and "Failed"⁹ may be output to the console's serial port.
- If the memory at 0xa001-0xa003 is equal to 0xdeb061, then the byte stored at 0xa000 is the status of the test¹⁰.
- For the `halt_bug` test, there does not seem to be anywhere where the result is output, so the graphical output of the emulator is verified.
- The Mooneye test ROMs¹¹ and SameSuite¹², of which all tests are verified the same way. In case of success, the B, C, D, E, H and L registers hold the values 3, 5, 8, 13, 21 and 34 respectively (Fibonacci's sequence). If they instead all hold the value 0x42, the test failed¹³.
- The Acid test ROMs (DMG¹⁴, GBC¹⁵). These need to be tested graphically, as their purpose is verifying the actual output of the emulator rather than its behaviour.

Thanks to this, the emulator's frontend supports a total of 191 automated test ROMs, that verify the behaviour of most of the emulator. Of these 191 tests, 101 pass.

5.1.5 Memory Inspect

When debugging an emulator, being able to inspect its memory is essential, as some bugs may be caused by the wrong mapping of components, or a fault when writing data. To help debugging this, the frontend comes with a basic memory inspection tool, that can show the entirety of the *addressable* data of the emulator (see [Figure A.6](#)). This means data that is inaccessible by the CPU (for instance, because the appropriate ROM bank is not selected) cannot be inspected here. A simple offset can also be indicated, to limit the data to a certain area.

5.1.6 Keybindings

The user can also customise their keybindings for the emulator, by mapping each input to a separate key (see [Figure A.3](#)). This is only relevant on keyboard-equipped devices.

⁸See lines 50-54 of `mem_timing/source/common/testing.s` in <https://github.com/retrio/gb-test-roms>

⁹See lines 112-139 of `mem_timing/source/common/runtime.s` in <https://github.com/retrio/gb-test-roms>

¹⁰See 'Output to memory' in `dmg_sound/readme.txt` of <https://github.com/retrio/gb-test-roms>

¹¹See <https://github.com/Gekkio/mooneye-test-suite>

¹²See <https://github.com/LIJI32/SameSuite/>

¹³See <https://github.com/Gekkio/mooneye-test-suite/#passfail-reporting> for the Mooneye test suite, see lines 265-281 of `include/base.inc` in <https://github.com/LIJI32/SameSuite/> for the SameSuite

¹⁴See <https://github.com/mattcurrie/dmg-acid2>

¹⁵See <https://github.com/mattcurrie/cgb-acid2>

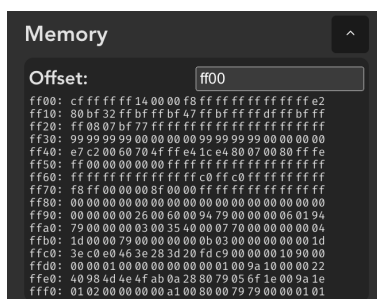


Figure 5.8: The memory inspection drawer

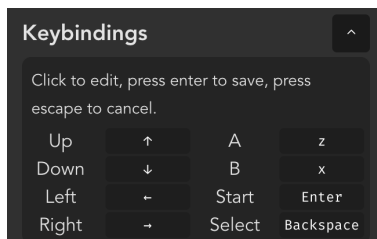


Figure 5.9: The keybindings settings

5.2 Emulator-Frontend Interfaces

To allow the emulator to communicate with the frontend, three components were created, each responsible for part of the output/input.

5.2.1 Screen output handler

To allow easily displaying graphics data from the emulator on the frontend, a **Screen** component was created. This highly configurable screen takes multiple parameters, to allow configuring the screen's size, scale, if it has any upscaling filters applied, if it has any colour palettes, and if it needs to blend frames together (see [Figure 5.10](#)).

```

1  type VideoReceiver = (data: Uint32Array) => void;
2  type ScreenProps = {
3    inputRef: MutableRef<VideoReceiver | undefined>;
4    width?: number;
5    height?: number;
6    scale?: number;
7    Filter?: ImageFilter;
8    blending?: boolean;
9    id?: string;
10   palette?: Partial<Record<number, number>>;
11 };

```

Figure 5.10: Parameters of the **Screen** component

All of the parameters are optional, and use sensible defaults when not specified. The only required parameter is `inputRef`, a modifiable “pointer” to the screen input function. This allows passing data to the screen without re-rendering the whole website. Because pointers do not exist in TypeScript, what is passed is actually an object with a modifiable `value` field.

Whenever any parameter changes, a new *render function* is generated, and is set in the given “pointer”. This function takes as input the graphics data (a `Uint32Array`¹⁶, containing ARGB values), and applies the required transformations to it. The function also holds a reference to two backing data buffers, for the previous and the current frame. This allows for blending between frames, and avoids creating new arrays repeatedly, lowering the memory consumption of the screen. The transformations it applies are:

1. Roll the buffers, setting the previous frame to the current frame, and the current frame to the newly received frame.
2. If needed, apply the colour palette to the input. The palette is a simple map, from source colour to target colour.
3. If it is the first frame to be drawn, also set the previous frame to the newly received frame (this avoid blending the new frame with a black frame, since the buffers are initialised with 0x000000 values).
4. If required, blend the previous frame and the current frame.
5. Apply the upscale filter to the image (this can be an identity filter, that simply returns the image, or upscale filters like `Scale2x` and `Scale4x`).
6. The image data is created from the `Uint32Array`, and drawn on the `<canvas />`.

This simple pipeline of functions makes the code really easy to understand and potentially modify, and allows for easy extension, as any filter could be given to the screen and it would work seamlessly. It is also of interest to note that the order in which the operations are applied maximises efficiency. For instance, the filter is applied last, to avoid having to do previous operations like blending on much larger images.

5.2.2 Audio output handler

To output the sound generated by the emulator, a utility class `AudioPlayer` was made. It is a simple class, that creates an `AudioContext`¹⁷ instance on creation. This class allows playing

¹⁶See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint32Array

¹⁷See <https://developer.mozilla.org/en-US/docs/Web/API/AudioContext>

sound output from the browser.

A difficulty with playing audio data stems from the fact that audio samples are quite short (they are output every frame, so every 16.6 milliseconds), and they must be constantly output, with tolerance for small delays in the output. If there happens to be even a small gap between two consecutive samples, a audible pop might play, which results in a very unpleasant experience.

To avoid this, **AudioPlayer** actually adds a small delay before starting to play audio. This ensures that by the time the first received sample is done playing, a new sample has already been received and queued for output, meaning there are never gaps between samples (except if there is an exceptionally long delay between samples, which should not happen).

A drawback from this solution however is that if samples are received too fast, a delay may create itself, as too many samples are enqueued, and new samples need to wait for all the previous ones to be played before being played itself. To avoid having this delay grow too large, a parameter of **AudioPlayer** allows setting the max size of the enqueued audio samples (the frontend uses a value of 8) – any samples that would make the queue exceed this size are ignored, meaning the max delay of the playback is $8 * \frac{1}{60} = 0.13$ seconds, a reasonable delay.

5.2.3 Input handler

To provide a simple way of handling inputs whether the device has a keyboard or uses touch controls, a **GameInput** component was created. It handles keyboard inputs, via a helper hook¹⁸, **useKeys**. To handle touch screen inputs, this component creates buttons mimicking those of the GB. These buttons all have the **mobile-only** class, which is defined in **main.css** to only be visible when no fine pointer is available on the device (ie. no mouse is detected), as can be seen on [Figure 5.11](#).

```
1 @media (any-pointer: fine) {  
2     .mobile-only {  
3         display: none;  
4     }  
5 }
```

Figure 5.11: **mobile-only** CSS class

This means that if the user is on a mobile devices, the buttons will show, allowing for input without the need for a keyboard (see [Figure 5.12](#)).

¹⁸See <https://reactjs.org/docs/hooks-intro.html>

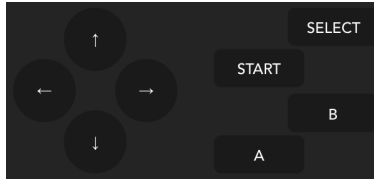


Figure 5.12: Buttons for mobile devices

The component itself then takes as a parameter a callback, that is called with the input function created by this component and returns an up to date object with the currently read inputs `GameBoyInputRead` (see [Figure 5.2](#)).

5.3 CPU

The CPU is the core of the emulator, and allows running the code from the ROM by reading the operation code (or opcode) and executing the matching action.

The Game Boy's clock ticks at $2^{22} \approx 4.19\text{MHz}$, with each of these clock cycles being commonly referred to as T-cycles or T-states [9, CPU Instruction Set]. However, because all of its components tick every four T-cycles, timings of the GB are divided by four and called M-cycles. In other words, four T-cycles make for one M-cycle, and M-cycles have a frequency of $\frac{2^{22}}{4} = 2^{20} \approx 1.05\text{Mhz}$ [13].

5.3.1 Instruction Set Decoding

Because the Game Boy is an 8-bit system, opcodes are 8 bits (or one byte) long, giving in theory a maximum of $2^8 = 256$ operations. However in the GB the operation `0xCB` gives access to an extended instruction set, meaning that when reading `0xCB` the CPU will read the next byte and use a different logic to execute the operation. This means there are now $2^8 - 1 + 2^8 = 511$ operations. The GB also has 11 'unused' opcodes, that will lock the console when used [9, CPU Comparison with Z80], meaning there are in total $2^8 - 12 + 2^8 = 500$ operations to implement.

Multiple techniques exist to handle this large number of operations:

- Have a large switch-statement for all possible operations. This is the most straightforward option, but can result in quite large switch-statements, especially for CPUs with more opcodes. The GB has comparatively few opcodes as it is an 8-bit system; the Gameboy Advance, the console that followed it, had 32-bit opcodes: way too many for a switch statement to be appropriate.
- Decode the operation by reading specific parts of the opcode, and generate the instructions

dynamically. This method is what is done for larger instruction sets, where opcodes can be split into separate parts to describe the operation’s behaviour [16, ARM CPU Reference]. It however comes with the cost of extra processing for each instruction, as it needs to be decoded. This method is applicable to the GB, as many instructions follow a pattern – see, for instance the LD instructions, that all follow the same order of registers: B, C, D, E, H, L, (HL) (the byte at address HL) and A [17].

- Create a dispatch table: a map that associates each opcode to a function to execute [18]. This technique is also only viable for small numbers of opcodes, as the map can result quite large. An advantage of this method is that the map does not have to be explicitly written out entirely – generators, or macros, can be used to populate some chunks of it, making it a hybrid of the two previous solutions: there is very little overhead to execute an instruction, as each opcode is associated to a function, but there is also no need to write out every instruction separately, as the map can be generated in parts or in its entirety (inducing a light setup cost).

Initially, the emulator had a large map, with all the opcodes as keys. The functions associated would then execute the instruction and return the number of cycles taken by the instruction (see Figure 5.13). This however proved quite repetitive and prone to errors. To solve this, a *generator function* was created. To use it, two parameters must be given: a map of opcodes to values (of an arbitrary type), and a helper function that for each arbitrary value returns a function that executes the instruction (see Figure 5.14). This allows generating repetitive instructions more easily, by only specifying what opcodes and objects are used, and not what the whole body of the instruction is (see Figure 5.15). This method is used in other emulators – for instance, SameBoy has a map that has an opcode-function mapping for all opcodes, and uses macros to generate the different functions¹⁹.

```

1  protected instructionSet: Partial<Record<number, InstructionMethod>> = {
2      // NOP
3      0x00: () => 1,
4      // LD BC/DE/HL/SP, d16
5      0x01: (s) => { this.regBC.set(this.nextWord(s)); return 3; },
6      0x11: (s) => { this.regDE.set(this.nextWord(s)); return 3; },
7      0x21: (s) => { this.regHL.set(this.nextWord(s)); return 3; },
8      0x31: (s) => { this.regSP.set(this.nextWord(s)); return 3; },
9      ...
10 }

```

Figure 5.13: Initial instruction set implementation

¹⁹See https://github.com/LIJI32/SameBoy/blob/master/Core/sm83_cpu.c

```

1  protected generateOperation<K extends number, T>(
2      items: Record<K, T>,
3      execute: (r: T) => InstructionMethod
4  ): Record<K, InstructionMethod> {
5      const obj: Record<K, InstructionMethod> = {};
6      for (const [opcode, item] of Object.entries(items)) {
7          obj[opcode] = execute(item);
8      }
9      return obj;
10 }

```

Figure 5.14: Generator function used for the CPU

```

1  protected instructionSet: Partial<Record<number, InstructionObject>> = {
2      // NOP
3      0x00: () => 1,
4      // LD BC/DE/HL/SP, d16
5      ...generateOperation(
6          {
7              0x01: this.regBC,
8              0x11: this.regDE,
9              0x21: this.regHL,
10             0x31: this.regSP,
11         },
12         (register) => (s) => {
13             register.set(this.nextWord(s));
14             return 3;
15         }
16     ),
17     ...
18 }

```

Figure 5.15: Improved instruction set implementation

5.3.2 A cycle accurate CPU

The Gameboy is a memory-bound system, meaning that it is limited by its memory accesses. The CPU can only execute either one read or one write per M-cycle [19, CPU core timing]. It also needs to retrieve the opcode for each instruction, which takes an additional cycle, meaning an instruction performing no memory accesses lasts one cycle, and an instruction performing n memory accesses lasts at least $n + 1$ cycles. Finally, the GB overlaps the last cycle of the execution with the fetching cycle for the next opcode – this will thus be of importance when implementing the fetching of the opcode. See [Figure 5.16](#) for the breakdown of an instruction.

The current implementation is not M-cycle accurate: the CPU instruction is executed as one monolithic block, rather than in different smaller parts. This becomes crucial when the timer, OAM and PPU are involved, as they run in parallel with the CPU, so memory accesses

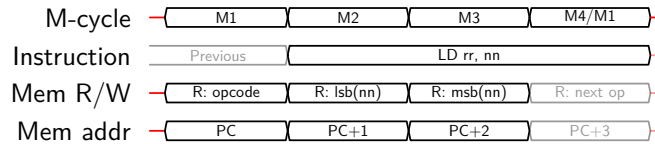


Figure 5.16: Timing of the LD `rr, nn` instruction
Taken from GBCTR [19, Sharp SM83 instruction set]

to these components may return different values depending on the M-cycle.

In all emulators that were found when researching for this project, M-cycle accuracy is reached by making the emulator “CPU-driven”. What this means is that inside each instruction, between each cycle, the CPU is responsible for ticking the rest of the system – the main loop is then only responsible for continuously running the CPU, and nothing else. This approach is probably the simplest and most straightforward one, as it is quite simple to implement: all one needs to do is call the system tick method when relevant (see Figure 5.17).

Some emulator where this method was found:

- In Mooneye GB, this can be seen by the usage of the `CPUContext.tick_cycle()` method when appropriate²⁰.
- In accurateboy, the `Bus.tick()` method is called from within the instructions (or it is done implicitly by methods such as `Bus.read()`)²¹.
- In Sameboy, the CPU uses functions like `cycle_read` or `cycle_no_access` to tick the rest of the system²².

```

1  protected ld_bc_d16(system: System) { // LD BC, d16
2      const lower = system.read(this.regPC.inc());
3      system.tick();
4      const upper = system.read(this.regPC.inc());
5      system.tick();
6      this.regBC.set(upper << 8 | lower);
7  }

```

Figure 5.17: CPU-driven LD `BC, (HL)`

This solution however comes with the cost of coupling the CPU to the system, or at least to a way of ticking said system. A particularity of this emulator is that the CPU is almost autonomous, in that it does not interact with any other functionality of the system directly, except the interrupts. The only other interface it requires is an `Addressable`, to allow access to memory (and thus the other components). This also means the emulator is no longer “CPU-

²⁰See <https://github.com/Gekkio/mooneye-gb/blob/master/core/src/cpu.rs>

²¹See <https://github.com/Atem2069/accurateboy/blob/master/accurateboy/CPU.cpp>

²²See https://github.com/LIJI32/SameBoy/blob/master/Core/sm83_cpu.c

driven”, since the CPU *cannot* tick the rest of the system. It is instead up to the root system to tick all of the components (including the CPU). This is thus a code quality improvement, that is not present in other emulators.

This was done by splitting all the instructions into their respective steps, and have each step return the next step (or `null` if it is the last step). The CPU now must simply store whatever the instruction returns: if it is `null` then it needs to fetch an instruction to prepare for the next cycle, otherwise it is a function and must be executed (and its result stored for the next step). See [Figure 5.18](#) and [Figure 5.19](#) for an example of this.

In the final version of the CPU’s tick method, a method `loadNextOp` is implemented. It not only handles getting an instruction associated with the desired opcode, but it also is responsible for checking if an interrupt was raised (in which case the interrupt handling procedure occurs [9, Interrupts]), and for ensuring the CPU is not halted before running an instruction.

```

1  protected ld_bc_d16(system: Addressable) { // LD BC, d16
2      const lower = system.read(this.regPC.inc());
3      return () => {
4          const upper = system.read(this.regPC.inc());
5          return () => {
6              this.regBC.set(upper << 8 | lower);
7              return null;
8          }
9      }
10 }
```

Figure 5.18: System-driven LD BC, d16

```

1  step(system: Addressable, interrupts: Interrupts) {
2      if (this.nextStep === null) {
3          // looks up opcode in instruction table
4          const nextStep = this.loadNextOp(system, interrupts);
5          if (nextStep === "halted") return;
6          this.nextStep = nextStep;
7      }
8      this.nextStep = this.nextStep(system, interrupts);
9      if (this.nextStep === null) {
10         // opcode is fetched on the last cycle of execution
11         this.currentOpcode = system.read(this.regPC.inc());
12     }
13 }
```

Figure 5.19: System-driven step of the CPU

Note how the opcode is fetched directly when the next instruction is over, to emulate the overlap between the fetch and the execute steps.

5.4 System

The **System** class implements the system bus, as well as the ticking of all other components. It handles the ticking of the PPU, timer, APU and interrupt logic. Furthermore, it is the component that links all of the data together: whenever a component is ticked (any of the above or the CPU), the **System** instance is passed, so that the components can read and write to the rest of the system. It implements **Addressable** (see [Figure 4.6](#)), and internally has a **getAddress** method that returns the **Addressable** at this specific address, to be accessed in the **read** and **write** methods. This ensures that the logic used to determine what component is accessed depending on the address is not duplicated.

All components of the Game Boy exist in the same address space. The *memory map* is what describes the allocation of different components to different areas of memory.

5.4.1 getAddress optimisation

Because **System** is a highly used component and is accessed for almost every read and write, the **getAddress** method is under a lot of pressure: for a game like “The Legend of Zelda: Link’s Awakening DX”, the method is called around 650,000 times per second. For a more complex and resource-intensive game such as “Alone in the Dark: The New Nightmare”, around 1,900,000 calls are made. This intensity in usage can easily be explained, as the GB is a memory-bound system: almost all interactions between components occur through memory reads and write. **getAddress** must thus be optimised as much as possible, as it is one of the main bottle-necks of the emulator.

An initial implementation of **getAddress** used the combination of a list of if-statements for different ranges of the memory map, as well as a map where all register addresses were mapped to the component responsible for them. The code would first check if the key exists in this map, and if not it would then go through a series of if-conditions (see [Figure 5.20](#)). It would return a tuple, containing both **Addressable** to use and the address within in – this was needed as some components had a particular mapping to memory. This is the case for the Work RAM (WRAM), whose memory starts at 0xC000 and ends at 0xFDFE (spanning over 15.5KB bytes) despite it only being 8KB long, because the last 7.5KB of its address range map back to the beginning of it (this is called the Echo RAM [20]).

This proved quite costly:

- This code creates a new map every time it is called, when checking for the registers’


```

1  protected getAddress(pos: number): AddressData {
2      const register = {
3          0xff00: this.joypad,
4          ...
5          0xffff: this.intEnable,
6      }[pos];
7      if (register !== undefined) return [register, pos];
8
9      if (0x0000 <= pos && pos <= 0x7fff) return [this.rom, pos]; // ROM Bank
10     ...
11     if (0xfe00 <= pos && pos <= 0xfe9f) return [this.oam, pos]; // OAM
12
13     // Unmapped area, return 'fake' register
14     return [{ read: () => 0xff, write: () => {} }, 0];
15 }

```

Figure 5.20: Initial implementation of `getAddress`

addresses.

- Having to return both an `Addressable` and an address is quite costly, as a new array with two items must be created every time. It also adds unnecessary complexity to the method, as the system bus should not be responsible for handling the details of address mapping, and instead simply delegate the task to the appropriate component.
- The if-conditions for the ranges of the biggest areas of memory (everything between `0x0000` and `0xEFFF`) happened after the register checks, which delayed response for these reads and writes. This is all the more important that this range represents $\frac{0xEFFF}{0xFFFF+1} \approx 93\%$ of the total address space.
- Chaining if-conditions is inefficient, as the JS engine must step through all conditions and check the values each time. Furthermore, although having both the lower and upper bound of the memory section indicated in the condition (e.g. `0x0000 <= pos && pos <= 0x7FFF` for `[0x0000; 0x7FFF]`) makes the translation from memory map to code easier, it is slower, since only the upper bound of the range is needed for the condition if all the ranges below have already been handled.

To fix these issues, we may first move the fine-grained addressing logic to sub-components like the WRAM. This removes the need to return an address from the method: only an `Addressable` is enough, as that component will then be responsible for decoding the address further.

The last two points may then be addressed, by removing the use of if-conditions, and moving this part of the code to the beginning of the function.

With the memory map of the console (see [Table 5.1](#)) we can notice how the largest chunks

of memory all end with the last three nibbles of the address as `0xFFF`. This means that the component mapped to an address can be determined by simply looking at the first nibble of said address. This is probably done to simplify the circuitry responsible for addressing, as only the most significant 4 bits need to be compared. The emulator can take advantage of this.

Start	End	Description
0000	3FFF	16 KB ROM bank 00
4000	7FFF	16 KB ROM Bank 01~NN
8000	9FFF	8 KB Video RAM (VRAM)
A000	BFFF	8 KB External RAM
C000	CFFF	4 KB Work RAM (WRAM)
D000	DFFF	4 KB Work RAM (WRAM)
E000	FDFD	Mirror of 0xC000~0xDDFF

Table 5.1: Memory map for the largest chunks of memory [20]

For this area of memory (`0x000-0xFFFF`) the system bus may simply isolate the most significant nibble, and then use a map to associate each address block to a component. Only if the address corresponds to `0xFF--` do we try matching it to a register address. This removes the need for the if-conditions, and is also faster as it is evaluated much earlier on (see [Figure 5.21](#)). The map can be created on instantiation and kept for later use, to avoid unnecessary memory allocations.

```

1  protected getAddress(pos: number): Addressable {
2      // Checking last nibble
3      let addressable = this.addressesLastNibble[pos >> 12];
4      if (addressable) return addressable;
5
6      // Registers
7      if((address & 0xff00) === 0xff00) {
8          addressable = this.addressesRegisters[pos & 0xff];
9          if (addressable) return addressable;
10     }
11
12     if (pos <= 0xfdf) return this.wram; // Echo RAM
13     if (pos <= 0xfe9f) return this.ppu; // OAM
14     if (pos <= 0xfeff) return Register00; // Illegal Area
15
16     return RegisterFF; // fake register
17 }

```

Figure 5.21: Optimised implementation of `getAddress`

To ensure placing the “main block” first was the best choice, measurements have been taken of four different GB games. The emulator would log all memory accesses by groups of $100\,000\,000 = 10^8$, and categorise them by address “blocks”: the main block, the OAM block, the illegal block (called like this because this area of memory is restricted by Nintendo, and

only returns 0x00) and the register block. This separation is justified by the fact these are the five chunks of memory that must be checked separately, due to their irregular boundaries. The *second* set of 10^8 accesses was then used to gather the statistics of what blocks are used the most. The first 10^8 accesses are not used, as they may include setup operations that only happen when the game loads, and as such do not represent what the average execution will look like. See Table 5.2 for the results.

Name	Memory Area	Game 1	Game 2	Game 3	Game 4
Main Block	0x0000-0xEFFF	88.869%	95.679%	84.289%	95.740%
Echo RAM	0xF000-0xFDFF	0.000%	0.000%	0.000%	0.000%
OAM Block	0xFE00-0xFE9F	0.001%	0.000%	0.000%	0.000%
Illegal Block	0xFEAO-0xFEFF	0.001%	0.000%	0.000%	0.000%
Register Block	0xFF00-0xFFFF	11.129%	4.321%	15.711%	4.260%

Table 5.2: Access rate of memory blocks

Sample taken from the second set of 10,000,000 accesses of 4 different games.

Games are, respectively, “Alone in the Dark: The New Nightmare”, “The Legend of Zelda: Link’s Awakening DX”, “Tetris” and “Pokémon Silver”.

As we can see, the vast majority of memory accesses go to the main part of memory, with almost all of the rest going to the “register area”, the last 256 bytes of the address space. This can easily be explained by the fact all registers that allow interaction with other components (the PPU, the APU, the timer, etc.) are in this narrow range, so it is bound to have a high usage. It is also quite interesting to note that neither the Echo RAM or the OAM block are used at all, except very rarely for one of the tested games. It is thus safe to assume that the conditions responsible for mapping these areas can be left at the end of the function, as they will rarely match an address.

5.4.2 Evaluating the getAddress optimisation

A simple experiment was then run, to verify the performance improvement. The first 25 million instructions of the `cpu_instrs`²³ test ROM were run. This sample was chosen because it is considerably large and because the test itself requires around 25 million instructions to complete. For the measurement, `window.performance.now()`²⁴ was used before and after each drawn frame, and the values were then summed.

The result was the following: 33 955.9ms before the change, and 20 039.1ms after the change. The relative difference is thus $\frac{20\,039.1 - 33\,955.9}{33\,955.9} = -0.4098$, thus reducing time taken by 40.98%. By measuring the time spent within `getAddress` for these 25 million instructions, we get a total

²³See https://github.com/retrie/gb-test-roms/tree/master/cpu_instrs

²⁴See <https://developer.mozilla.org/en-US/docs/Web/API/Performance/now>

spent time in the method of 0.000151ms on average, showing that this method is no longer a bottleneck for the system, as its impact on performance is minimal.

5.5 PPU

The Picture Processing Unit (PPU) is the component responsible for rendering the game onto the Gameboy's screen. It is one of the most complex components of the GB, with intricate timings, and a behaviour that changes between the DMG and the GBC, due to the addition of colour-support, as well as Video RAM (VRAM) banking.

5.5.1 Presentation

The screen rendering is divided into three layers (see [Figure 5.22](#)), drawn on top of each other.

From bottom to top, these are:

- The background, a 256×256 image loaded into memory that support scrolling on both axis.
- The window, similar to the background, is a 256×256 image that can be moved around the screen, and is toggle-able. It however does not support scrolling.
- The objects, that are drawn at the very top, are smaller 8×8 tiles. These can be moved freely, and support some transformations, such as horizontal or vertical flipping. Their data is stored in the Object Attribute Memory (OAM).

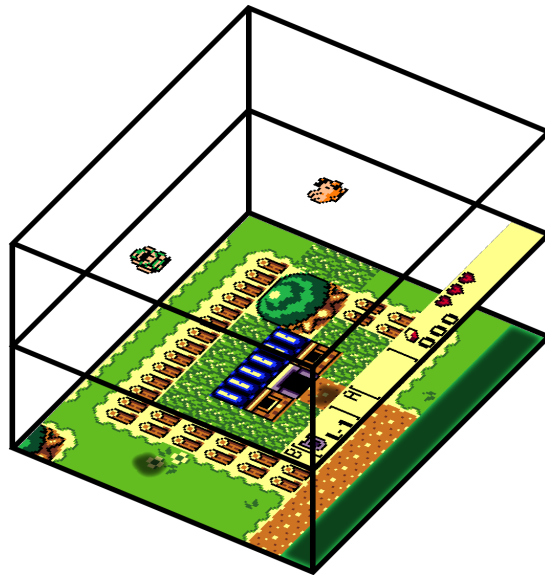


Figure 5.22: Layers of the PPU
From top to bottom, the objects, the window and the background

It renders onto the screen line by line, meaning it will draw a 1-pixel wide line (the scanline) from left to right, and then proceed to the line below. Drawing an entire line takes 114 M-cycles, and there is an 1140 M-cycle delay when the bottom of the screen is reached called the VBlank. The advantage of having per-line rendering is that by modifying the position of the background between each drawn line, complex visual effects can be obtained quite easily [21], see [Figure 5.23](#) for an example.

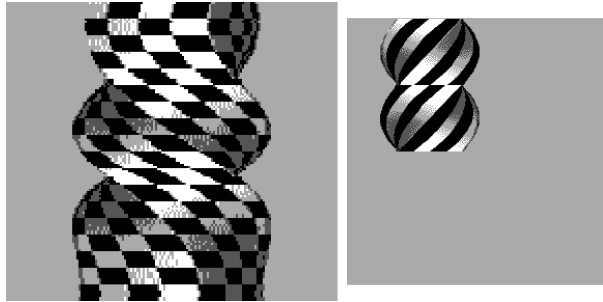


Figure 5.23: Visual effects made by scrolling the background Gameboy output (right), and loaded background data (right)

One of the main characteristics of the PPU is that it operates in modes. During each mode, it perform a different set of operations, and can only be interacted with in certain way [9, LCD Status Registers]. These four modes are, in order:

- Mode 2: the PPU looks through the OAM, to look for all the sprites to draw. During this time, the OAM is not accessible.
- Mode 3: the PPU reads both the VRAM and OAM, and draws the line. None of the video data is accessible here.
- Mode 0: the PPU does nothing for a short length of time after each line. An interrupt is raised at the start of this period, allowing the CPU to be notified that now is the time to update what's rendered if necessary. This is called the horizontal blank, or HBlank.
- Mode 1: the PPU does nothing for a long length of time after the bottom of the screen is loaded. This is usually where the bulk of the graphics update go, as it lasts a considerable amount of time (1140 M-cycles). An interrupt is also raised here, to notify the CPU. This is called the vertical blank, or VBlank.

Rendering of the line is thus done during mode 3, when no graphics data is accessible, using a FIFO queue (First In First Out) of pixels that can be pushed in and out as the data is read. Some of the registers are still writable, but most games do not use them during this mode because they would require very precise timing. An example of a game relying on this is “Prehistorik Man”, that changes the colour palette registers during the scanline in its

intro-scene [8, Tricky-to-emulate games].

Since this is only the case for a minority of games, we can go past this inaccuracy. We will thus take advantage of this to simplify greatly the rendering logic and make the renderer “scan-line based”, drawing an entire line all at once. This shortcut is commonly used by emulators that are not too accuracy-oriented.

To handle each mode separately, we have four distinct “mode” objects of type `PPUMode`, that hold some basic information on the operation of the mode: its length, its flag for the `STAT` register and the name of the method in PPU responsible for ticking said mode (see [Figure 5.24](#)). Note `KeyForType<T, V>` is the union of all keys `k` of `T` such that `T[k]` is of type `V`.

```
1 type PPUMode = {  
2   doTick: KeyForType<PPU, (interrupts: Interrupts) => void>;  
3   flag: number;  
4   cycles: number;  
5 };
```

Figure 5.24: Type definition of `PPUMode`

5.5.2 Rendering Logic

The first step needed to render the game is to select and order all the sprites to be shown on screen. One of the limitations of the GB is that despite having enough space in memory for 40 sprites, only 10 may be rendered at a time in a scanline [9, OAM]. The sprites are selected during mode 2 – as such, during the last cycle of mode 2, the emulator will look through the available sprites in the OAM, and select the appropriate ones. This can be done quite elegantly in a functional manner, using an array (see [Figure 5.25](#)).

```
1 this.readSprites = this.oam  
2   .getSprites()  
3   .filter((sprite) => sprite.y <= y && y < sprite.y + objHeight)  
4   .slice(0, 10)  
5   .map((sprite, index) => [sprite, index])  
6   .sort(objPrioritySort)  
7   .map(([sprite]) => sprite);
```

Figure 5.25: Retrieve the selected sprites for the scanline

First, we retrieve the sprites from the OAM. Internally, sprite data is cached between scanlines and invalidated when written to – `getSprites()` updates the dirty tiles in the cache and returns them. This avoids decoding the sprite every line if that tile has not received any changes, while also ensuring excess decoding is not done if the sprite is modified twice between

scanlines. We then select the first 10 sprites to be part of the scanline – this selection is done based on index: the PPU looks through the OAM sequentially to find matching sprites [9, OAM]. Finally, a re-ordering of the sprites needs to be done, to determine which sprite goes above which – this may depend on the sprite’s position in the OAM, or on its X coordinate. This is handled in `objPrioritySort`. To allow sorting based on both attributes, the sprites must be briefly remapped to a tuple with the sprite and its index (as JavaScript does not expose the indices of elements when ordering them). These sprites are then stored in the PPU object, to be used when rendering at the end of mode 3.

The PPU uses a form of indirect addressing to handle data (see [Figure 5.26](#)). It has an area in VRAM that stores tile data, a tile being an 8×8 image. Whenever one of these tiles needs to be used, for the background, or for an object, the identifier of the tile needs to be used, this identifier being derived from the tile’s address. This allows for the very simple re-use of tiles, which is particularly relevant for backgrounds where they may be repeated a lot. This background data is stored in a *tile map*, a 256×256 map of 1-byte indices to the actual tile data. The GB has two such maps, and both the background and window can display either of them – this is controlled via a flag in the LCDC register.

To render the scanline, the layers need to be drawn on top of each other: background, then window, then objects. Due to the similarities between background and window, a `drawLayer` method can be shared to handle the bulk of the work.

This method will loop over all tiles it needs to draw in the current scanline. It first needs to determine the tile’s index. This index can be used to retrieve the tile’s address in VRAM. It can also be used to fetch the attributes of the tile; this is a GBC-exclusive feature, that allows transforming tiles (for instance, flipping them, or selecting a different palette for the tile). These attributes are stored in the second bank of VRAM, at the same address, in a single byte of data. The DMG and GBC distinction is however not needed, because the attributes for a `0x00` value match the attributes used by the DMG. This means that instead of having two different methods split on the console versions, we can setup the DMG emulator to always return `0x00` for the second bank of VRAM, and keep the GBC way of handling of attributes.

Once the tile index has been acquired, the PPU may retrieve the tile data address from VRAM. This one-byte address needs to be converted to a valid address in the tile data range of VRAM, as it contains two partially overlapping tile data areas – a flag in the LCDC register controls this. Once the address where the tile data (ie. its texture) is obtained, the PPU may finally retrieve said data, and draw it in the scanline.

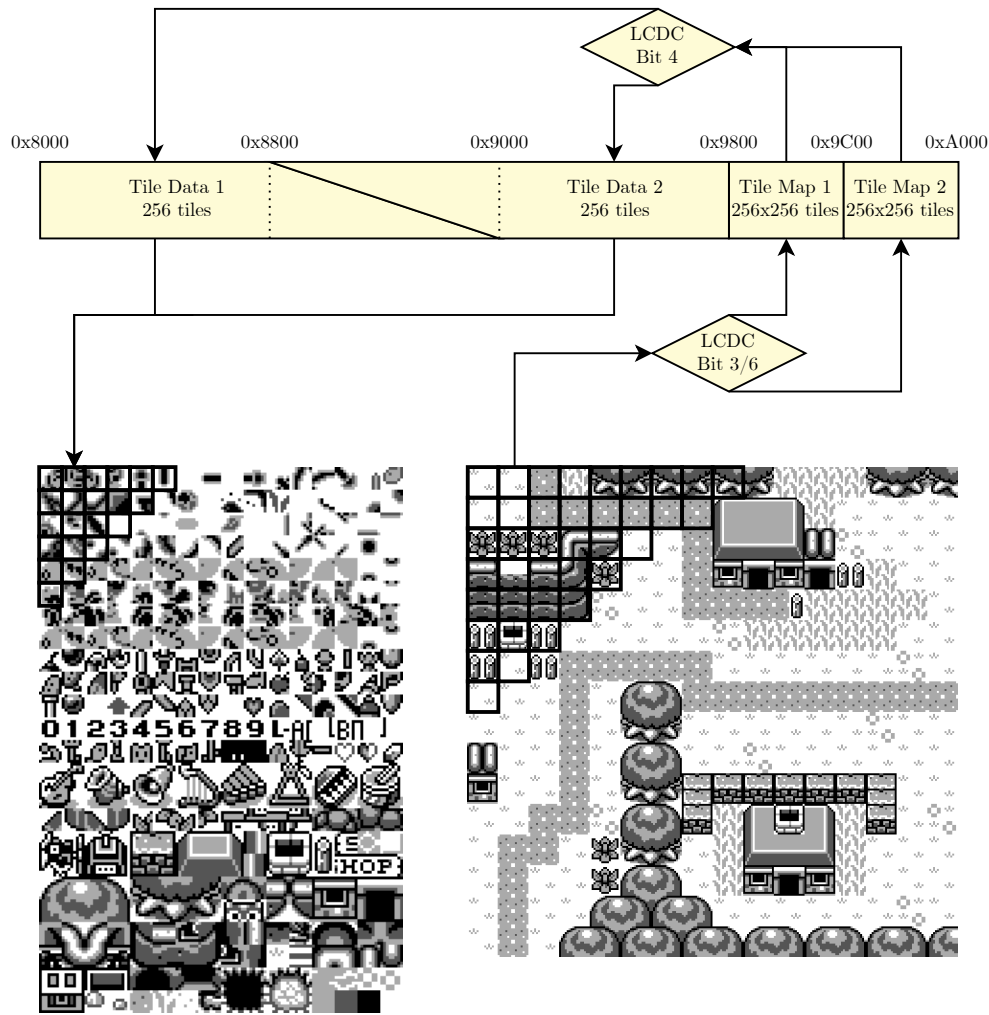


Figure 5.26: PPU logic to obtain tile data

This applies to the DMG – the CGB has two VRAM banks, thus having twice as many tile textures, and an additional tile attribute map.

This tile data, an 8×8 images where each pixel can be one of four shades is encoded in 16 bytes [9, VRAM Tile Data]. Because tile data is rarely changed and the procedure to extract the image data from these 16 bytes is quite complex and requires some bit manipulation, tiles are cached.

5.5.3 Subcomponents

Although it is presented to the system as a unit, the PPU is implemented as multiple smaller components, that handle different parts of the screen logic. This is all the more important that the PPU is extremely complex, so care is needed to ensure the code remains maintainable.

The Object Attribute Memory (OAM) is the memory area in the PPU that stores object

data. This small 160-byte long area has a feature called OAM Direct Memory Access (DMA), allowing for direct transfers from ROM or RAM to the OAM. Because this transfer runs in parallel to the other components, the OAM itself is its own component, contained inside the PPU. This allows for a better separation of concerns between the two: the PPU handles rendering, the OAM handles OAM DMA and sprite storing and decoding.

The colour management of the PPU also requires some extra logic that can be extracted into its own class. Although on the DMG this is limited to splitting a byte into 4 shades of 2 bits (white, light grey, dark grey and black), the GBC has 16 palettes of 4 colours, 8 palettes for the background and window, and 8 for sprites [9, Palettes]. As such, a `ColorController` class was created: it implements `Addressable`, and has a method to retrieve the palette for the background, and a method to retrieve the palette of a sprite. This class is extended by a `DMGColorController`, and a `CGBCColorController`. The PPU picks one of the two on instantiation, based on the emulated console, and can then use both, without needing to know if the screen is monochrome or coloured.

Similarly, the VRAM behaves slightly different between the DMG and the GBC: the latter's VRAM has two banks it can switch between, via an additional register, and also supports VRAM DMA transfers, similar to what is possible with the OAM. A `VRAMController` class is created: it also implements `Addressable`, and has a few extra methods, to allow getting tiles from it, and reading each bank individually (see [Figure 5.27](#)).

```
1 abstract class VRAMController implements Addressable {
2     read(pos: number): number;
3     write(address: number, value: number): void;
4
5     tick(system: Addressable, isInHblank: boolean, isLcdOn: boolean): boolean;
6
7     getTile(tileAddress: number, bankId: 0 | 1): Int2[] [];
8     readBank0(pos: number): number;
9     readBank1(pos: number): number;
10 }
```

Figure 5.27: Interface of `VRAMController`

5.6 APU

The Audio Processing Unit (APU) is the component responsible for producing the sounds and music of the GB. Its output is the combination of four different channels, each with their own properties: two square channels, a wave channel and a noise channel [9, Audio]. These channels

run independently, can be turned on or off individually, and are merged to form the output.

Most features of the channels can be derived to a counter, that ticks down from a value and has an effect when reaching 0. Although they differ in output, all channels have some attributes in common. For example, they all have a *length timer*, that turns off the channel when it reaches 0. Channels 1, 2 and 4 also have an *envelope*, that allows updating the volume of the channel at a set frequency. To have these behaviours work across all channels, an abstract class `SoundChannel` was created. It holds the registers all channels have in common, as well as methods like `tickLengthTimer` to handle common mechanisms across all channels.

To return their output, all channels have a `getSample` method that returns the current output value of the channel, a value between 0 and 15 (4 bits). This output can then be combined, and output to the frontend to handle. It is of interest to note that the GB is always outputting sound, with a resolution of 1 M-cycle, ie. $2^{20}\text{Hz} \approx 1.05\text{MHz}$. This is significantly more than the sample rate computers usually use, 44.1Hz. To fit to this standard, the emulator's APU has an internal counter that ticks down and only produces a sample every $\frac{2^{20}}{44.1 \cdot 10^3} = 23.8$ cycles. This is not the most accurate way of producing the audio data, but is far simpler and faster than downsampling the audio. An example of very accurate emulator that does downsampling is SameBoy²⁵.

5.7 MBCs and ROMs

A majority of GB cartridges came shipped with a Memory Bank Controller (MBC). This was done to circumvent the memory limit of the GB: due to its 16-bit addresses, only memory in 0x0000–0x7FFF is mapped to the ROM, limiting its size to 32KB. MBCs provided a way of extending this memory limit, by doing *bank switching* [9, MBCs]. The cartridge holds more data than it can address, and the CPU can modify which part of the memory it is accessing by writing to a register in the MBC. Because ROMs are read-only, the write-operations can be safely re-used to instead write to these built-in registers without losing any functionality.

On top of more memory space, MBCs can sometimes provide additional features, such as external RAM, a battery (to be able to keep the state of the RAM between sessions, which allows saving the game) [9, MBCs], a rumble motor [9, MBC5], or a battery-backed real time clock [9, MBC3], for example. The most common MBCs found were the MBC1, with space for a maximum of 2MB [9, MBC1], and the MBC5 – the only MBC to officially support the GBC's double speed mode [22]. See Table 5.3 for statistics of the usage of different MBCs.

²⁵See Accuracy, <https://sameboy.github.io/features/>

Name	ROM Count	Percentage
No MBC	2150	23.0%
MBC1	4010	43.0%
MBC2	227	2.4%
MBC3	367	3.9%
MBC5	2532	27.1%
Others	53	0.6%

Table 5.3: Statistics of different MBCs [23]

To ensure the emulator can run as many games as possible, the main MBCs have been implemented: MBC1, MBC2, MBC3 and MBC5 (although support for the MBC3’s real time clock has not been added). Since the GB interacts with the cartridge in the exact same way whether there is an MBC or not, the choice of making a `MBC` abstract class came quite naturally: its interface is that of `Addressable`, and it also supports a `save` and `load` method, for save files.

To decide which MBC to use, we have a `GameCartridge` class, that wraps around `MBC`. Its role is decoding the cartridge’s header, contained in `0x0100-0x014F` [9, The Cartridge Header], to decide on which MBC to use, as well as some of the properties of the cartridge: does it have RAM, is it battery backed (in which case it supports saving), what’s the game’s title and identifier.

The `read` and `write` methods received an optimisation similar to what was done with `System.getAddress`: because the internal addressing logic of the MBCs only relies on the most significant nibble, a simple switch statement can be made (see [Figure 5.28](#)).

Internally, the MBCs have a `ROM` instance to store the cartridge data (this is obtained from the ROM uploaded by the user), as well as an optional `RAM` instance. This RAM is what the game can edit. If it is battery backed, memory is kept when the GB is turned off, allowing data like scoreboards and progress to be saved. To replicate this saving behaviour, the emulator exposes a `save` and `load` method, that respectively return and set the data in the RAM. The emulator’s core is thus not responsible for handling these saves, and the frontend can decide freely how to store them.

In the implemented frontend, this is done by saving the RAM data in the browser’s available storage, using the “`localStorage`” library²⁶. When a ROM is loaded, the frontend checks if a save for this game exists, by using the ROM’s identifier decoded in the `GameCartridge` class. If it does, the save is then loaded. Similarly, when changing ROMs, closing the window, or pressing the “Save” button, the frontend retrieves the RAM data from the emulator and saves

²⁶See <https://github.com/localForage/localForage>

```

1  read(pos: number): number {
2      switch (pos >> 12) {
3          case 0x0: // ROM bank 00
4          case 0x1:
5          case 0x2:
6          case 0x3:
7              return this.data[pos & addressMask];
8          case 0x4: // ROM bank 01-ff
9          case 0x5:
10         case 0x6:
11         case 0x7: {
12             const address =
13                 (pos & ((1 << 14) - 1)) |
14                 (this.romBankLower8.get() << 14) |
15                 (this.romBankUpper1.get() << 22);
16             return this.data[address & addressMask];
17         }
18         case 0xa: // ERAM
19         case 0xb: {
20             if (this.ramEnable.get() !== RAM_ENABLED) return 0xff;
21             const address = this.resolveERAMAddress(pos);
22             return this.ram.read(address);
23         }
24     }
25     throw new Error(`Invalid address`);
26 }

```

Figure 5.28: read method of MBC5 [9, MBC5]

it, by setting the key of the entry in the local storage to the identifier of the ROM.

5.8 Timer

The timer is a component in the GB that ticks regularly. It allows the control of two independent mechanisms [9, Timer and Divider Registers].

First is the *divider counter*, accessed and controlled via the DIV register. This counter is internally 16-bits, although only the upper 8-bits can be accessed. It is incremented every clock cycle (ie. increased by 4 every M-cycle), can be read, and writing to the divider resets the counter (see [Figure 5.29](#)).

The second, more complex part of the timer is a *customisable timer*. It is made of three registers:

- **TIMA**: the timer counter. Every time a falling edge is detected on one of the bits of DIV (ie. the bit goes 1 to 0), this register is incremented. What bit is inspected can be controlled via the TAC register, effectively changing the frequency of the timer. When this register

```

1  protected divider = new DoubleRegister(0xab00);
2  protected addresses: Record<number, Register> = {
3      0xff04: this.divider.h, // we only ever read the upper 8 bits
4      ...
5  };
6
7  tick(interrupts: Interrupts): void {
8      const newDivider = wrap16(this.divider.get() + 4);
9      this.divider.set(newDivider);
10     ...
11 }
12
13 write(pos: number, data: number): void {
14     if (pos === 0xff04) { // Writing anything to DIV clears it.
15         this.divider.set(0);
16         return;
17     }
18     ...
19 }

```

Figure 5.29: Implementation of divider counter

overflows (is incremented when equal to 0xFF), an interrupt is raised.

- **TAC**: the timer control register. It allows enabling or disabling the timer, and changing the inspected bit of DIV.
- **TMA**: the timer modulo. It defines what value **TIMA** is reset to when overflowing.

Although it is a seemingly simple system made of 3 registers, the behaviour of this system is quite complex, as it has multiple edge cases [9, Timer obscure behaviour]. The code that handles the increase of **TIMA** is thus quite long, and required some re-factoring to be easily readable (see [Figure 5.30](#)). As can be seen from this implementation, the **TIMA** register can actually be incremented for two reasons (line 16). The first reason is the regular falling edge on the inspected bit. However, if the timer is disabled then the bit of **DIV** read is going to be 0, regardless of its actual value. This means that if the timer gets disabled while the read bit of **DIV** is 1, the timer detects a falling edge and increases **TIMA**. It is also of interest to note that **TIMA** is not set to the value of **TMA** when overflowing – this is actually done the following cycle (lines 2–9). This logic requires some extra fields to be present, recording the values of the registers on the previous tick. Some additional logic is required in the **write** method, because writes to **TIMA** are ignored when the timer overflows.

```

1  tick(interrupts: Interrupts): void {
2      this.previousTimerOverflowed = false;
3      if (this.timerOverflowed) {
4          const modulo = this.timerModulo.get();
5          this.timerCounter.set(modulo);
6          interrupts.requestInterrupt(IFLAG_TIMER);
7          this.timerOverflowed = false;
8          this.previousTimerOverflowed = true;
9      }
10
11     const timerControl = this.timerControl.get();
12     const timerIsEnabled = timerControl & TIMER_ENABLE_FLAG;
13     const speedMode = (timerControl & 0b11);
14     const checkedBit = TIMER_CONTROLS[speedMode];
15
16     const currentBitState = timerIsEnabled && (newDivider >> checkedBit) & 1;
17
18     if (this.previousBitState && !currentBitState) {
19         const result = (this.timerCounter.get() + 1) & 0xff;
20         this.timerCounter.set(result);
21         if (result === 0) this.timerOverflowed = true;
22     }
23
24     this.previousBitState = currentBitState;
25 }

```

Figure 5.30: Code to manage the TIMA increments

5.9 Helpful Components

To avoid repeating logic throughout the main components, small classes have been written. One of these is the **Register** class: it holds a single integer, that can be read or written. It also comes with a **flag** method, that returns whether a given flag is set in the register, as this is a frequent operation.

This class is extended by **MaskRegister**, a class to support registers of which all bits may not be used. This is the case for instance for TAC: bits 0–2 control the timer, and bits 3–7 are hardwired to 1, and cannot be reset. A **MaskRegister** allows defining this behaviour directly in the register, avoiding the need for additional logical in the class using the register (here, **Timer**). On instantiation, a *mask* is passed as an argument, and is applied whenever the value of the register needs to be changed (see [Figure 5.31](#)).

To accommodate the 16-bit registers of the CPU, a **DoubleRegister** class was implemented. It is made of two **Register** instances, and provides methods to get and set its value as if it held a 16-bit number (despite being implemented as two 8-bit numbers). This provides flexibility to the CPU, enabling both 16-bit and 8-bit arithmetic, without having to manage the way the

```

1  class MaskRegister extends Register {
2      protected mask: number;
3
4      constructor(mask: number, value: number = 0) {
5          super(value | mask);
6          this.mask = mask;
7      }
8
9      override set(value: number): void {
10         super.set(value | this.mask);
11     }
12 }

```

Figure 5.31: Implementation of MaskRegister

register is implemented.

Classes to manage memory were also created, to avoid having high-level components access data structures directly. For instance, the ROM class is backed by a `Uint8Array`²⁷, and implements `Addressable`. It is extended by `RAM`, to allow `write` operations.

To simplify the operation of certain components, a `CircularRAM` class was also created. It extends `RAM`, and must be provided an *offset* on creation. When reading or writing to it, the offset is subtracted from the address, and the modulo of the RAM's length is then applied to it (see Figure 5.32). This allows this part of memory to handle addressing in an independent way: the high-level component simply has to provide it with its address on creation.

```

1  class CircularRAM extends RAM {
2      protected offset: number;
3
4      constructor(size: number, offset: number, data?: Uint8Array) {
5          super(size, data);
6          this.offset = offset;
7      }
8      override read(pos: number): number {
9          return super.read((pos - this.offset) % this.size);
10     }
11     override write(pos: number, data: number): void {
12         super.write((pos - this.offset) % this.size, data);
13     }
14 }

```

Figure 5.32: Implementation of CircularRAM

This can be used to implement the WRAM. It can be addressed from `0xC000` to `0xFDFF`, but `0xE000–0xFDFF` maps back to `0xC000–DDFF`. As such, a `CircularRAM` with an offset of

²⁷See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint8Array

0xC000 and a length of 0x2000 will properly wrap address in the 0xE000–0xFDFE range back to the beginning of its memory. This avoids handling this logic in high-level components, and provides a generic solution to this kind of memory component. Other places where this class is used include the wave RAM of the APU, and the VRAM of the PPU.

Chapter 6

Evaluation

Evaluation of this emulator consists in verifying it simulates the guest console (the Game Boy) properly. This can be done via test ROMs, that verify the behaviour of parts of the console. In this chapter, we will both look at how the emulator performs by itself, and how it performs compared to other GB emulators. We will then also ensure that it is performant, and that it complies with the specification.

6.1 Absolute Accuracy

To measure the *absolute accuracy* of the emulator (how accurate it is, independently of other emulators), we will run different test ROMs and see what aspects of the emulator pass which tests. This will help us guarantee that specific parts are functional, and can be relied on.

Note that although on a functioning system test ROMs help diagnose issues, if the system is faulty then there may be false positives. Indeed, the test ROM is still a ROM that runs on the system, and relies on it somewhat working. For instance, a faulty CPU might make a test seem like it passes, despite it actually being wrong. As such, results should be taken with a grain of salt. Furthermore, test ROMs can only test known behaviour of the GB: if there exists a bug or behaviour that is unknown of, there is no way of writing a test ROM to verify it.

Using the “Test ROMs” panel from the emulator frontend, we can run a total of 191 tests automatically. Because each test focuses on a specific characteristic of a component, we can group them, to see the *success rate* this particular component has; see [Table 6.1](#).

We first may notice from this table that the accuracy of the CPU is quite good, the only test failing being `ie_push`¹, a test verifying an edge-case of interrupt handling. The success-rate

¹See https://github.com/Gekkio/mooneye-test-suite/blob/main/acceptance/interrupts/ie_push.s

Component	Category	Test count	Tests passed	Success rate
CPU	Instructions	20	19	95.0%
	Timing	23	23	100.0%
	Sum	43	42	97.7%
PPU	Rendering	2	2	100.0%
	Timing	12	5	41.6%
	DMA	12	7	58.3%
	Sum	26	14	53.8%
Timer		13	13	100.0%
MBCs	MBC1	12	12	100.0%
	MBC2	7	7	100.0%
	MBC5	8	8	100.0%
	Sum	27	27	100.0%
APU	General	17	3	17.6%
	Channel 1	21	0	0.0%
	Channel 2	15	0	0.0%
	Channel 3	16	2	12.5%
	Channel 4	13	0	0.0%
	Sum	82	5	6.1%
All		191	101	52.8%

Table 6.1: Results of the test ROMs on different components

of these tests being high is essential, since of course all tests run on the CPU and rely on it functioning to operate properly.

The PPU’s accuracy is good in terms of rendering, but the timings of it are wrong. When analysing the test ROMs further to understand what goes wrong, we can notice that some of the modes of the PPU are one cycle too long. This can be seen by inspecting the test’s output (see [Figure 6.1](#)), and noticing the error is in the D register, that has a value of 0x02 instead of 0x01. The part of the code responsible for this is in lines 35–40 of the test’s source². Although the timing of the emulator could be modified to make this test pass, the change makes other PPU timing tests fail; as such, it is reasonable to suspect the fault is in the logic itself of the PPU class and not the number of cycles only. Many modifications have been attempted to improve the accuracy of the timing, to no avail – more research would thus be needed to fix this.

The timer and MBCs have great accuracy, and do not have any problems as far as the tests can tell. This is important, in particular for the timer, because multiple timing tests rely on the timer functioning. As such, having this part of the emulator work reliably is vital to ensuring the other tests work properly.

Finally, the APU performs very poorly. This is firstly due to the complexity of the different channels: they all have several mechanisms working separately, and if only one of them behaves wrong then the whole set of tests will fail. Furthermore, the majority of these tests come from

²See https://github.com/Gekkio/mooneye-test-suite/blob/main/acceptance/ppu/intr_2_mode3_timing.s

```

Registers
A: 03 F: 00
B: 02 C: 00
D: 02 E: 02
H: FF L: 41

Assertions

D: 01! E: OK

Test failed

```

Figure 6.1: Output of PPU timing test `ppu_intr_2_mode3_timing`

the SameSuite³ test suite. These tests differ from other because they use the PCM12 and PCM34 registers of the GBC, to directly inspect the output of each channel [9, Audio Details]. These allows the tests to check with much more precision that the emulator works, making the test significantly more stringent. Although this is an obvious indicator that the APU needs to be reworked and fixed, this does not impact the user experience, or at least not that of a casual user, as the audio of the game does not sound off when being used.

Seeing the total of tests passed is also of interest. The total success rate is 52.8%, but this value is strongly influenced by the fact almost half of the tests are for the faulty APU (82 tests out of 191). If these tests are filtered out, we get a total of 109 tests, of which 96 pass, giving a success rate of 88%. A really good result, given the emulator was written in a short period of time with no prior experience.

6.2 Relative Accuracy

After verifying the accuracy of Emmy by itself, using automated tests, we may also want to compare it to other existing GB emulators, and see how it does in contrast with them.

Thankfully, a tool for this already exists: “GBEmulatorShootout”⁴. It automatically tests GB emulators, and displays their results in a large table, allowing for the comparison of emulators’ accuracy.

The way the tool works is that each emulator it supports must have a matching file, to allow downloading the emulator’s executable and running it with a given test ROM. The program then regularly screenshots the emulator’s window, and compares it with the expected visual output of the test, meaning it expects the emulator’s window to only contain the graphical output of the GB.

³See <https://github.com/LIJI32/SameSuite>

⁴See <https://github.com/daid/GBEmulatorShootout>

This means that adding this emulator to the tool required changes to the tool’s source code. Indeed, it has two notable distinctions to other emulators tested by the tool. First and foremost, this emulator is not based on an executable – it is instead *browser based*, meaning that simply running the emulator’s file is not possible. The second difference is that the emulator’s output does not cover the whole window, as there is an additional UI around it. We must thus instead find a way to extract the emulator’s output from the window.

To do this, we will use web development techniques to automate the tests. Selenium⁵ is a tool that allows programmatically opening a browser and running actions on it, mimicking those of a user. In this case, these actions would be opening the emulator’s website, and uploading the ROM to the emulator, as a user would do. Because the rest of the tool is written in Python⁶, we will use the `selenium` Python package⁷. See Figure 6.2 for the code to setup the emulator, and the code to run a ROM on it. Opening the window and accessing the emulator is quite simple (see lines 2–6), as we must just open the window at a given URL. We may note the script also does two additional actions: the first is enabling “triple speed mode” (line 5), to make the tests run faster. We also open the “Settings” tab of the sidebar (line 6), as this is where the console selection (DMG or GBC) is done. This is relevant because some tests are designed for a console in particular, and the only way to switch the console on this emulator is via this UI.

```
1 class Emmy(Emulator):
2     def setup(self):
3         self.driver = webdriver.Chrome("emu/chromedriver_win32/chromedriver.exe")
4         self.driver.get("https://emmy-gbc.vercel.app/")
5         self.driver.find_element(value="emu-speed").click()
6         self.driver.find_element(value="drawer-section-settings").click()
7
8     def startProcess(self, rom, *, model, required_features):
9         model_btn_id = {DMG: "dmg-mode", CGB: "cgb-mode"}.get(model)
10        if model_btn_id is None: # console not supported
11            return None
12        self.driver.find_element(value=model_btn_id).click()
13        rom_path = os.path.abspath(rom)
14        self.driver.find_element(value="rom-input").send_keys(rom_path)
15        try: # if an alert appeared, it means the rom is incompatible
16            self.driver.switch_to.alert.accept()
17            return None
18        except: # no alert, so error thrown, so the rom is compatible
19            return self.driver
```

Figure 6.2: Setup and run code to automate emulator testing

⁵See <https://www.selenium.dev/>

⁶See <https://www.python.org/>

⁷See <https://pypi.org/project/selenium/>

Once the emulator is setup, the test ROM must be handled. First, the desired console model is checked, and the appropriate console is selected in the UI (lines 9–12). This is done by matching the model ID to the ID of the button that selects said console, and clicking the button with that ID. The test ROM may then be uploaded, by pressing the “Import a ROM” button, and entering the ROM’s path. We must also check if an alert is raised in the browser (lines 15–19) – this occurs if an error occurs when reading a ROM, usually because the MBC is not supported.

The second part of automated testing requires monitoring the emulator, and verifying if the test succeeded or failed. GBEmulatorShootout requires taking screenshots of the emulator frequently, and comparing the screenshot to an image of the expected result. This method implies storing an additional image for each test ROM, but comes with the advantage that no complex code is required to look into the emulator’s state for other indicators of success (like what was done in this project’s frontend, see subsection [Test ROMs](#)). This makes it the ideal solution for this kind of tool, as the additional code to add an emulator to the test suite is minimal (all emulators will output the Game Boy’s screen, whereas not all emulators have the same internal structure).

To do this, we must extract the image from the `<canvas/>` element the emulator’s frontend draws in, and convert it to an appropriate Python object (see [Figure 6.3](#)). We first fetch the output canvas by its ID, and execute a small JavaScript function to retrieve the image inside the canvas, in a PNG format. This data is then decoded, and used to create an image, that is resized to the correct size (as the emulator’s screen may be upscaled by the browser).

```
1 def getScreenshot(self):
2     canvas = self.driver.find_element(value="emulator-frame")
3     canvas_base64 = self.driver.execute_script(
4         "return arguments[0].toDataURL('image/png').substring(21);",
5         canvas
6     )
7     canvas_png = base64.b64decode(canvas_base64)
8     large_image = PIL.Image.open(io.BytesIO(canvas_png))
9     small_image = large_image.resize((160, 144), PIL.Image.NEAREST)
10    return small_image
```

Figure 6.3: Code to get the emulator’s output with Selenium

After other minor tweaks to GBEmulatorShootout’s code to support web-based emulators, the test script can be run to compile the results and be able to finally compare our emulator to others. The results are accessible online at <https://nlark.github.io/GBEmulatorShootout/> (see [Figure 6.4](#)). The website to access the results is made of a large table, with on the X axis

the emulator running the test ROM, and on the Y axis the name of the test ROM. Emulators are ordered from most accurate (most tests passed) to least accurate, from left to right. The tool currently supports 239 test ROMs, for 14 emulators.















Updated On Fri, 03 Feb 2023 15:11:28 +0000	SameBoy (236/239)	Emulicious (236/237)	Beaten Dying Moon (225/239)	bgb (203/239)	GambatteSpeedrun (159/182)	binjgb (143/236)	ares (134/239)
acid/which.gb (DMG)	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR
acid/which.gb (GBC)	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR	INFO ===== TEST TO BE D++ SPR SPR
acid/dmg-acid2.gb	PASS HELLO WORLD!  dmg-acid2 by Matt Currie	PASS HELLO WORLD!  dmg-acid2 by Matt Currie	PASS HELLO WORLD!  dmg-acid2 by Matt Currie	PASS HELLO WORLD!  dmg-acid2 by Matt Currie	PASS HELLO WORLD!  dmg-acid2 by Matt Currie	PASS HELLO WORLD!  dmg-acid2 by Matt Currie	PASS HELLO WORLD!  dmg-acid2 by Matt Currie
acid/cgb-acid2.gb	PASS HELLO WORLD!  cgb-acid2 by Matt Currie	PASS HELLO WORLD!  cgb-acid2 by Matt Currie	PASS HELLO WORLD!  cgb-acid2 by Matt Currie	PASS HELLO WORLD!  cgb-acid2 by Matt Currie	PASS HELLO WORLD!  cgb-acid2 by Matt Currie	PASS HELLO WORLD!  cgb-acid2 by Matt Currie	PASS HELLO WORLD!  cgb-acid2 by Matt Currie
	PASS	PASS	PASS	FAIL	FAIL	FAIL	FAIL

Figure 6.4: Screenshot of the UI of GBEmulatorShootout

We may first notice that Emmy passes 118 out of 236 tests (3 tests are skipped because the emulator does not support the “Super Game Boy”), ranking it 9th of the 14 tested emulators. It is relevant here to note that most of these other emulators have existed for several years now, and have been worked on by many people.

We may also use this table to compare the accuracy of this emulator to others, by looking at the success rate for tests of different components (see Table 6.2). Note that if an emulator cannot run a test (because it lacks support for a feature needed by the test), the test is skipped, rather than counted as failed. This explains why “VisualBoyAdvance-M” has a success rate of 57.1% but is ranked lower than Emmy.

This table is a good indicator of where our emulator, Emmy, does and does not perform well, compared to other GB emulators. For instance, it has a really accurate timer, compared to emulators ranked around it. This also emphasises the fact that its CPU is of an accuracy similar to that of the higher accuracy emulators, passing almost 80% of tests. It also helps see that only very high accuracy emulators manage to pass the APU tests. The only outlier seems to be VisualBoyAdvance-M, with an APU success rate of 60.7%. However, when inspecting the test results, we notice that most of the APU tests were not run on it – this is because it lacks

Emulator	Tests Run	CPU	PPU	Timer	MBCs	APU	All
SameBoy	239	100.0%	91.9%	100.0%	100.0%	100.0%	98.7%
Emulicious	237	98.5%	100.0%	100.0%	100.0%	100.0%	99.6%
Beaten Dying Moon	239	100.0%	97.3%	100.0%	97.1%	85.9%	94.1%
bgb	239	74.6%	91.9%	100.0%	97.1%	82.4%	84.9%
GambatteSpeedrun	182	98.5%	81.1%	100.0%	57.1%	100.0%	87.4%
binjgb	236	84.6%	63.9%	92.3%	74.3%	29.4%	60.6%
ares	239	82.1%	51.4%	38.5%	80.0%	29.4%	56.1%
mGBA	233	83.1%	52.8%	92.3%	90.6%	20.0%	57.1%
Emmy	236	76.9%	45.9%	100.0%	79.4%	10.6%	50.0%
VisualBoyAdvance-M	182	53.7%	45.9%	76.9%	62.9%	60.7%	57.1%
PyBoy	143	32.1%	17.2%	38.5%	90.6%	0.0%	40.6%
Goomba	232	30.8%	8.6%	7.7%	59.4%	3.5%	20.7%
no\$gmb	236	35.4%	22.2%	0.0%	20.0%	2.4%	17.8%
KiGB	231	29.2%	20.0%	0.0%	12.9%	2.4%	14.7%

Table 6.2: Success rate of emulators for different components, sorted by total passed tests

support for the PCM12 and PCM34 registers these tests use. If we add the un-run tests to the percentage, we end with 17 tests passed out of 85, or a 20% success rate, in line with the success rate of similar emulators. This further shows that creating an accurate APU is challenging, especially given the standards of the used test ROMs.

6.3 Performance

Aside from accuracy, we may want to measure how performant the emulator is. This is important, because most emulators offer a “turbo mode”, allowing the emulator to run the GB faster. Furthermore, this emulator is designed to run on mobile devices, that may have less computing power – as such, the emulator being efficient with the resources it has is vital for these platforms.

To measure performance, the most relevant metric is frame time: the time needed in milliseconds to draw a frame (this, of course, includes the rendering of the frame but also all the processing that goes before). The emulator’s frontend already comes with a measure of frame time – we thus just need to select what ROM the test will be performed on, and how the measure will be taken.

For the ROM, the performance will be measured on 4 distinct ROMs, all outlining different uses of the emulator. These are:

- “The Legend of Zelda: Link’s Awakening”, a DMG game. It is quite simple and should not be too memory intensive. It will help outline the average performance on the DMG.
- The `cpu_instrs` test ROM. Its advantage is that it is quite long, and its purpose is to

test the entirety of the CPU, meaning it will be very processor-intensive.

- “The Legend of Zelda: Oracle of Ages”, a GBC game. It is similar to the previous Zelda ROM, but was built for the GBC, meaning it likely requires more resources to run.
- “Alone in the Dark: The New Nightmare”, a GBC game. This is a very complex game released for the GBC, that supports “3D-scenes”, and very regularly changes the colour palette registers [8, Tricky-to-emulate games]. Both of these properties combined should make this ROM slower, making it a hypothetical “upper-bound” on frame time.

As for how to measure the performance, the frame time of every frame will be summed and averaged, over the first 10 million cycles, to give a more accurate measure. See Table 6.3 for the results.

Console	ROM	Frame time (ms)	Speed increase
DMG	The Legend of Zelda: Link’s Awakening	4.39	279%
	cpu_instrs	5.43	207%
GBC	The Legend of Zelda: Oracle of Ages	4.89	241%
	Alone in the Dark: The New Nightmare	5.78	188%

Table 6.3: Average frame time of different ROMs

From this table we can notice two things. First, DMG games run faster than GBC games. This can be due to the double speed mode, that forces the emulator to run twice as many instructions, or to the fact that the PPU logic with colour is more complex and needs to be optimised. Secondly we may look at the average frame times of these ROMs.

The GB runs at 60 frames per second, meaning a frame must last at most $\frac{1}{60} = 16.6\text{ms}$. We may calculate the maximum speed increase attainable for a frame time via the formula $\frac{1000}{\text{frame time} \times 60}$, or $(\frac{1000}{\text{frame time} \times 60} - 1) \times 100$ to get an increase percentage. For instance, for “The Legend of Zelda: Link’s Awakening”, the maximum speed increase could be of around 279%.

This is a satisfiable value: a speed increase of 200% is good enough for most uses. Many emulators provide speed increases much superior to this, however this is because they are often written in compiled languages like C++ or Rust, and will thus run much faster than an interpreted language like JavaScript.

WebAssembly⁸ is a low-level language made for browsers, resembling assembly language. It is designed for efficiency, and would thus be a great pick for a browser-based emulator as it runs faster than JavaScript [24]. Languages like C, C++ and Rust can be compiled to WebAssembly via tools like emscripten⁹, allowing for great performance with portable code. Examples of such

⁸See <https://webassembly.org/>

⁹See <https://emscripten.org/>

emulators playable from the browser include GBEmu¹⁰, an emulator written in Rust.

This would however require a full rewrite of the code, since TypeScript and C++ have little in common in terms of syntax. Another simpler alternative to convert the emulator’s code to a WebAssembly-compileable language is using AssemblyScript¹¹, a language similar to TypeScript.

Research has been done to convert the project’s code to AssemblyScript, and the progress so far can be seen on the `assembly-script` branch¹². However, because AssemblyScript targets such a low-level language, many TypeScript constructs are not (yet) available in it, such as arrow functions, or dynamic objects. Although the implemented emulator does not use these features extensively and most components were successfully migrated to AssemblyScript, the CPU uses arrow functions for all instructions – a successful conversion of the entire emulator would thus require a lengthy rewrite of the CPU. This project was thus abandoned in favour of additional features in the emulator’s core and frontend. An example of GB emulator in AssemblyScript is wasmboy¹³.

6.4 Compliance to Specification

The presented emulator fulfils most of the specification, with some specifications of lower importance having been left out. These include:

- U4: User can press a button to download a save of their game (or, alternatively, the save can be stored inside the browser with a technology like IndexedDB.

The user cannot, currently, download the save of their game. The save file is instead stored in the browser. Some research was done to allow saving the emulator’s state to the BESS¹⁴ save format, however this required a lot more effort so priority was shifted on other features, as the existing save system was deemed sufficient.

- U7: User can pause the console emulation through a button. They can also input conditions for which the console should break execution.

The emulator’s frontend does not provide a way to add breakpoints to the emulation. This feature was initially supported from the browser’s console, but its implementation was unsatisfactory. A possible improvement to the emulator would thus be support for such breakpoints, maybe in a format similar to that of the current “Watch Expressions”

¹⁰See <https://github.com/BlueBlazin/gbemu>

¹¹See <https://www.assemblyscript.org/>

¹²See <https://github.com/Niark/gbc-emulator/tree/assembly-script>

¹³See <https://github.com/torch2424/wasmboy/>

¹⁴See <https://github.com/LIJI32/SameBoy/blob/master/BESS.md>

menu.

- N3: A downloadable version of the web-app can be used on a computer and provides full functionality, via keyboard and mouse inputs.

The implemented emulator cannot currently be downloaded as a local app. This could however be added quite easily, using a tool like Electron¹⁵ that allows converting web applications to desktop apps. Another option is converting the web-app into a Progressive Web App (PWA)¹⁶, allowing users to “download” the web-app and use it while offline.

All of these specifications are however of low importance, and are mainly small quality of life features. Other possible improvements could be adding more debugging tools (such as a way to inspect the APU’s raw output), or more features added to the emulator’s core, like support for other MBCs (like the MBC6, MBC7, or MMM01 [9, MBCs]), the real time clock of the MBC3, additional outputs of the GB like the Game Boy Printer or the Game Boy Camera [9], and support for the Link Cable, allowing multiple Gameboys to play together [9, Serial Data Transfer].

¹⁵See <https://www.electronjs.org/>

¹⁶See <https://web.dev/progressive-web-apps/>

Chapter 7

Legal, Social, Ethical and Professional Issues

7.1 Privacy

This piece of software is safe to use in terms of privacy – it runs entirely locally, with no data ever being sent from the user to the server. There are no cookies, and the only forms of storage used are `localStorage`¹ and `localForage`², both of which are local and offline.

`localStorage` is handled by the browser, so it is the user’s responsibility to ensure that they use a secure browser.

`localForage` is an open source library, allowing for more transparency – one could go through the source code to verify that it is safe as well. Whenever a new update is released, we may simply verify that the modified code is still safe, and then configure the project to use the new version in the `package.json` file.

7.2 Legality

The Game Boy is a copyrighted console, owned by Nintendo, but making an emulator for it is deemed legal as long as it is done following a “clean room” design [25]. Overall, it is legal to make emulators, as long as they follow this method. This was ruled via a series of court appeals between console manufacturers and groups that produce emulators, with the latter consistently winning the appeal. This was the case for the Sony Computer Entertainment America v. Con-

¹See <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

²See <https://github.com/localForage/localForage>

nectix Corporation trial, that deemed that the “Virtual Game Station” PlayStation emulator was not a copyright infringement on Sony, and that Connectix’s reverse engineering of the original PlayStation’s BIOS was fair use [26]. Emmy was developed using online resources about the GB compiled by people who reverse engineered the GB, and as such was developed with a clean room design too.

Although distributing the emulator is legal, distributing the BIOS (Basic Input/Output System) of the console with the emulator is not, as it is still copyrighted software [25]. Because acquiring such BIOS is however not illegal as long as the user owns the console, a user is free to use their own BIOS in the emulator (see subsection [Settings](#)). The emulator otherwise simulates the effect of the BIOS on the system, by setting up all the necessary memory and registers.

Distributing copyrighted game ROMs is also not allowed – this is why the user must upload the ROM they want to use themselves. The software does not upload the ROM, and simply stores it in local storage.

7.3 Integrity

This piece of software was developed with integrity, always thinking critically on what was developed, and with no intention to harm others. This report has also been written with honesty, without attempting to withhold information on the resulting software. No acquired data was falsified or modified to fit a narrative.

The produced software is open source, and available at <https://github.com/Nlark/gbc-emulator>. This means it can be used by others to create similar emulators, or for educational purposes. Other users may also contribute to the project, or fix issues it has.

Overall, this software was developed with the British Computer Society’s Code of Conduct & Code of Good Practice³ in mind, and all of its relevant rules were followed.

³See <https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf>

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This project helped outline the typical structure of a an emulator, and in particular that of a Game Boy emulator. It describes how the console works in detail, what components it is made of, and how they interact with each other. It showcased optimisation methods for emulating different components of an emulator, like the CPU or the system bus. The resulting software is a fully playable Game Boy and Game Boy Color emulator, that has multiple quality of life features to make the experience more pleasant while also providing debugging tools for retro game and emulator developers. This emulator can be played on both computer and mobile devices, and is of good accuracy compared to other existing emulators.

8.2 Future Work

As it stands the emulator has three main properties that could be improved, each independently one of the other.

The first is improving the accuracy of the emulator. As seen in the evaluation of the project, its APU should be reworked. It is currently lacklustre, and represents a part of the system that could be improved without requiring a full re-write, as it is self contained. Other small improvements could be done to improve the accuracy of other components like the MBCs or the PPU, or to add currently unsupported MBCs and accessories. All of these improvements can be done separately and in small increments, and do not entail any breaking changes outside of the emulator's core.

A second important point to be improved in the emulator is its performance. As seen in its evaluation, an average maximum of a 200% speed increase is reachable. To improve the performance of the emulator (which will be needed if more features are to be added), its performance must be improved substantially. Although code optimisations and caching could improve some of the performance issues, a significant bottleneck is the language used itself. If the emulator were to be entirely rewritten using a language such as AssemblyScript, it would likely run much faster. This would be a much more time-consuming change, and would require updating the front-end to work properly with it, but could yield great performance improvements if done properly.

Finally, the third point to improve on this project is its frontend. Although it has a good range of available features, it could still be improved to be up to par with other technical emulators. More and better debugging tools could be provided to the user, as well as more customisation options, such as new screen filters, a full-screen mode or support for input macros.

Acronyms

APU Audio Processing Unit.

CPU Central Processing Unit.

DMA Direct Memory Access.

DMG Dot Matrix Game.

GB Game Boy.

GBC Game Boy Color.

MBC Memory Bank Controller.

OAM Object Attribute Memory.

OS Operating System.

PPU Picture Processing Unit.

ROM Read-Only Memory.

VRAM Video RAM.

WRAM Work RAM.

Bibliography

- [1] Merriam-Webster. “Emulator definition and meaning.” (Mar. 2022), [Online]. Available: <https://www.merriam-webster.com/dictionary/emulator> (visited on 03/02/2023).
- [2] A. Kaluszka. “Computer emulation, history.” (Dec. 2001), [Online]. Available: <https://kaluszka.com/vt/emulation/history.html> (visited on 03/03/2023).
- [3] IBM. “709/7090/7094/7094 II compatibility feature for IBM System/370 models 165, 165 II, and 168.” (1973), [Online]. Available: http://bitsavers.org/pdf/ibm/370/compatibility_feature/GA22-6955-1_709x_Compatibility_Feature_for_IBM-370_165_168.pdf (visited on 03/03/2023).
- [4] S. Granger, “Emulation as a digital preservation strategy,” *D Lib Mag.*, vol. 6, no. 10, Oct. 2000. DOI: [10.1045/october2000-granger](https://doi.org/10.1045/october2000-granger). [Online]. Available: <https://doi.org/10.1045/october2000-granger> (visited on 04/02/2023).
- [5] L. Boley. “Emulation or virtualization: What’s the difference?” (Mar. 2014), [Online]. Available: <https://www.dell.com/en-us/blog/emulation-or-virtualization-what-s-the-difference/> (visited on 04/02/2023).
- [6] MyaMyaMya. “First Famicom/NES emulator? - Zophar’s Domain.” (Jan. 2009), [Online]. Available: <https://www.zophar.net/forums/index.php?threads/first-famicom-nes-emulator.10169/> (visited on 03/07/2023).
- [7] Emulation General Wiki. “History of emulation.” (Jan. 2023), [Online]. Available: https://emulation.gametechniki.com/index.php/History_of_emulation#Game_Boy_2FColor (visited on 03/07/2023).
- [8] “Gameboy development wiki.” (2023), [Online]. Available: <https://gbdev.gg8.se/wiki/> (visited on 03/28/2023).
- [9] “Pandocs.” (2023), [Online]. Available: <https://gbdev.io/pandocs/> (visited on 03/07/2023).

- [10] endrift. “"holy grail" bugs in emulation, part 1.” (May 2017), [Online]. Available: <https://mgba.io/2017/05/29/holy-grail-bugs/> (visited on 04/03/2023).
- [11] T. Phillips, “Game Boy and Game Boy Color titles headed to Nintendo Switch Online,” *Eurogamer*, Feb. 2023. [Online]. Available: <https://www.eurogamer.net/game-boy-and-game-boy-color-titles-headed-to-nintendo-switch-online-1> (visited on 04/02/2023).
- [12] M. J. Flynn, C. Neuhauser, and R. M. McClure, “EMMY: An emulation system for user microprogramming,” in *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, ser. AFIPS ’75, Anaheim, California: Association for Computing Machinery, 1975, pp. 85–89, ISBN: 9781450379199. DOI: [10.1145/1499949.1499968](https://doi.org/10.1145/1499949.1499968). [Online]. Available: <https://doi.org/10.1145/1499949.1499968>.
- [13] Pan of Anthrox, GABY, M. Fayzullin, *et al.* “Game Boy CPU manual.” (May 2008), [Online]. Available: <http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf> (visited on 03/22/2023).
- [14] A. Vynogradenko. “Framework-sizes.md.” (2023), [Online]. Available: <https://gist.github.com/Restuta/cda69e50a853aa64912d> (visited on 04/03/2023).
- [15] MDN Web Docs. “Autoplay guide for media and web audio APIs.” (Mar. 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Media/Autoplay%5C_guide (visited on 03/23/2023).
- [16] M. Korth. “GBATEK.” (2014), [Online]. Available: <https://problemkaputt.de/gbatek.htm> (visited on 03/26/2023).
- [17] M. Sullivan. “Game boy CPU intructions.” (Oct. 2022), [Online]. Available: <https://meganesu.github.io/generate-gb-opcodes/> (visited on 03/22/2023).
- [18] A. Goldfuss. “Function dispatch tables in C.” (Mar. 2019), [Online]. Available: <https://blog.alicegoldfuss.com/function-dispatch-tables/> (visited on 03/29/2023).
- [19] Joonas “Gekkio” Javanainen. “Game Boy: Complete technical reference.” (Mar. 2023), [Online]. Available: <https://gekkio.fi/files/gb-docs/gbctr.pdf> (visited on 03/07/2023).
- [20] R. Mongenel. “GameBoy memory map.” (2010), [Online]. Available: <http://gameboy.mongenel.com/dmg/asmmemmap.html> (visited on 03/27/2023).
- [21] M. Steil. “The ultimate game boy talk (33c3).” (Dec. 2016), [Online]. Available: <https://youtu.be/HyzD8pNlpwI?t=2460> (visited on 03/27/2023).

- [22] Great Hierophant. “MBC5 backwards compatibility.” (Nov. 2009), [Online]. Available: https://forums.nesdev.org/viewtopic.php?p=52575#post_content52603 (visited on 03/29/2023).
- [23] M. Achibet. “Game boy/game boy color roms.” (Jul. 2015), [Online]. Available: <http://merwanachibet.net/gameboy-roms> (visited on 03/29/2023).
- [24] B. Velghe. “Is WebAssembly really faster than JavaScript? a hands-on experiment!” (May 2022), [Online]. Available: <https://www.tmssoftware.com/site/blog.asp?post=933> (visited on 04/01/2023).
- [25] J. Conley, E. Andros, P. Chinai, E. Lipkowitz, and D. Perez, “Use of a game over: Emulation and the video game industry, a white paper,” *Northwestern Journal of Technology and Intellectual Property*, Tech. Rep., 2004. [Online]. Available: <https://scholarlycommons.law.northwestern.edu/njtip/vol2/iss2/3/> (visited on 03/31/2023).
- [26] *Sony Computer Entertainment America, Inc. v. Connectix Corporation*, 203 F.3d 596, 2000. [Online]. Available: <https://web.archive.org/web/20070228070634/http://www.ca9.uscourts.gov/ca9/newopinions.nsf/0/66b3a352ea33712988256952007578c2?OpenDocument> (visited on 03/31/2023).

Appendix A

User Guide

In this guide, we will walk you through how to use the Emmy emulator, a Game Boy and Game Boy Color emulator for the browser.

A.1 Disclaimer

Emmy is not affiliated with Nintendo in any way. You are responsible for obtaining copies of the ROMs you wish to play yourself, and must ensure you are allowed to have these ROMs. Open source ROMs are available online, on websites like Retro Veteran¹. We do not condone the usage of illegally obtained ROMs.

One feature of the emulator involves using boot ROMs for the console. These cannot be distributed by Emmy, and must be obtained legally to be used. These can be found and downloaded online. They are not necessary for the usage of the emulator. Open source versions of the boot ROMs can be found online.

A.2 Playing games on Emmy

First, open the emulator from your browser – a hosted version is available at <https://emmy-gbc.vercel.app/>. You will get to the Emmy emulator (see [Figure A.1](#)).

The interface is divided into two parts:

- The main area, on the right. This is where the console's screen will appear. It has extra buttons which we will talk about later.

¹See <https://www.retroveteran.com/category/nintendo-game-boy-color/>

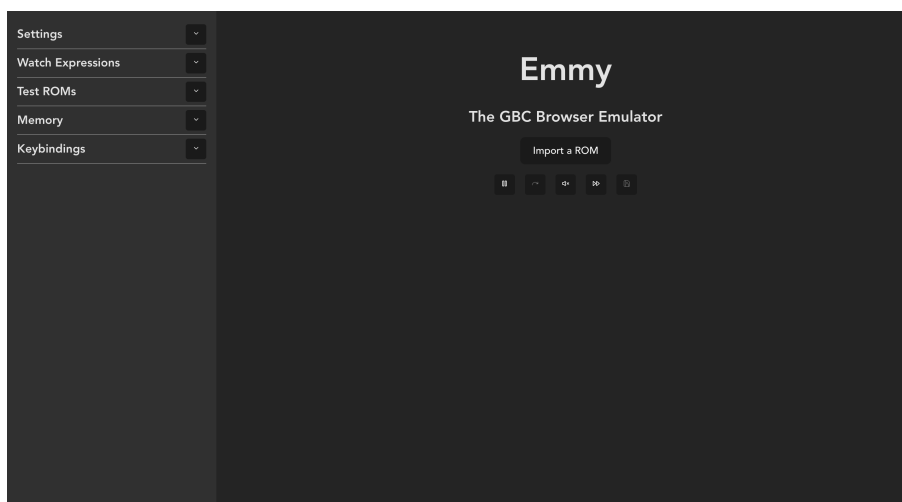


Figure A.1: Start screen of Emmy

- The sidebar, on the left. This is where all of the settings of the emulator are. All submenus can be opened and closed. You can also resize the sidebar by dragging on its right side.

Press the “Import a ROM” button, to select the ROM you wish to use. A file picker dialogue should open – select the ROM, and start playing! The game should directly load into the emulator.

A.3 Settings

The behaviour of Emmy can be customised to your needs. In the main area, you can find buttons to freeze the play-through, or resume it. You can also decide to enable or disable the game’s sound, and speed up the emulation, making it go three times as fast. If the game supports save files, press the “Save” button to save the current state of the game. The next time you open the ROM, your save will automatically be loaded.

In the sidebar, you will find more settings to customise the emulator (see [Figure A.2](#)).

From top to bottom, these settings are:

- The emulated console. This may either be the classic Game Boy, or the Game Boy Color.
- The upscaling filter chosen. This option modifies the output of the screen. The first option leaves it as it is, without altering it. The two other options, Scale2x and Scale4x², allow upscaling the image, by adding more details to it, increasing its resolution by two and four respectively.
- The scale at which the screen is displayed. This allows increasing the size of the screen –

²See <https://www.scale2x.it/>

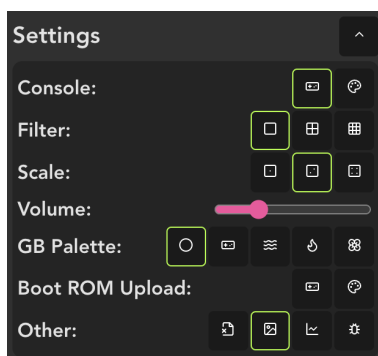


Figure A.2: Settings of Emmy

not its resolution!

- The volume at which the emulator outputs sound (if audio is enabled).
- The palette used for the Game Boy. By default, it will use shades of grey. Options include the classic green hues, a blue palette, a darker red palette, and a flowery pink theme.
- Two buttons to upload the boot ROM of the emulator. This is not needed to play, but can be nice if you want the original startup screen.
- A set of miscellaneous settings, including:
 - If the boot ROM is enabled. For this to work, you must have first uploaded a boot ROM for the current console.
 - Enabling frame blending, meaning frames are blended together. This may cause some slight visual artefacts when the scene is moving around.
 - Displaying the current performance of the emulator.
 - Displaying the tileset and tile data of the emulator. This is intended for debug purposes.

A.4 Keybindings

Emmy comes with default keybindings for the controls, that are common in other emulators (see [Table A.1](#)).

Game Boy Button	Default Key
Directional arrows	Directional arrows
A	Z
B	X
Start	Enter
Select	Backspace

Table A.1: Default keybindings of Emmy

These can also be customised, in the “Keybindings” submenu (see [Figure A.3](#)). To change a keybinding, click on the button you want to remap, and then press the key you want to bind to it. Press enter to confirm, and escape to cancel.

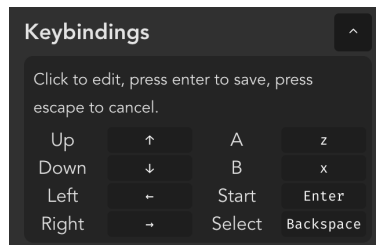


Figure A.3: Keybindings menu

A.5 Debugging

Emmy also provides a set of debugging tools - if you are a retro game developer, an emulator developer or want to study Game Boy ROMs, these tools will be of great help to you.

First, in the sidebar there is a “Watch Expressions” submenu (see [Figure A.4](#)). It allows writing expressions and seeing their output directly from the UI. The syntax is the JavaScript syntax. The emulator is defined as the variable `gbc`. To access any data on the emulator, refer yourself to the project’s code, available at <https://github.com/Nlark/gbc-emulator>. The emulator is in the `src/emulator` folder.

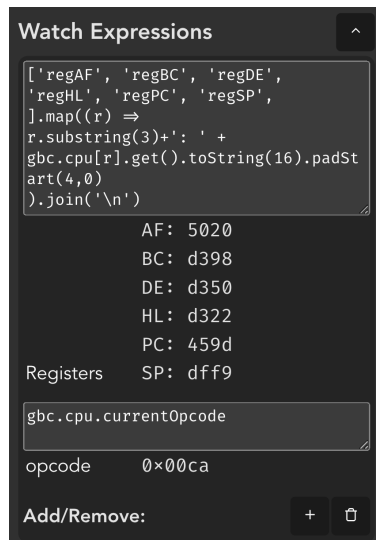


Figure A.4: Settings of Emmy

For instance, to access the DIV register of the timer, use the expression `gbc.system.timer.divider`. By default, numbers are output in hexadecimal. If you want to see the number in a decimal

format, format it into a string: `num.toString(10)`. You can add and remove as many watch expressions as you want! If one of them is not valid, the message “Error” will appear.

The emulator also comes with a builtin set of test ROMs, to test the emulator live (see [Figure A.5](#)). This allows you to easily see how accurate it is, and what tests the emulator passes and fails - this is very useful if you are working on the emulator’s code, to ensure you are not breaking anything as you modify it. You can select the set of tests you wish to run, and there is also a button to select or unselect everything.

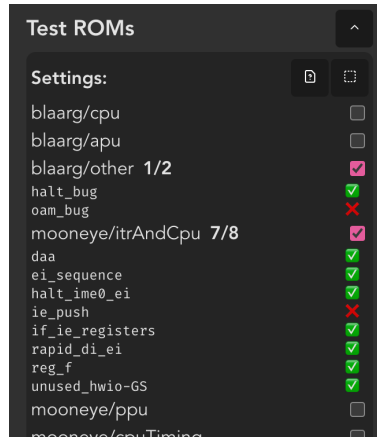


Figure A.5: Automated test ROMs

To inspect the behaviour of the emulator in detail, the emulator can be paused, and ticked instruction by instruction by pressing the “Step” button in the main area. This allows you to see what each instruction does, which is very helpful when a bug arises.

Finally, a memory inspection tool is available, to see the data contained in all of memory. To see it, open the “Memory” submenu (see [Figure A.6](#)). It will display every single accessible byte of memory, from 0x0000 to 0xFFFF. If you are only interested in a part of memory, an offset can be input (for instance, if you only want to see the data from 0xFF00 to 0xFFFF).

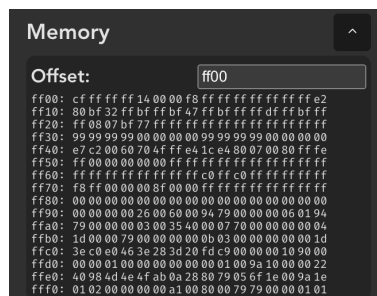


Figure A.6: Memory inspection tool

Appendix B

Source Code

B.1	Emulator Core Code	80
B.1.1	Emulator Main Interfaces.	80
	GameBoyColor (src/emulator/GameBoyColor.ts)	80
	GameBoyInput (src/emulator/GameBoyInput.ts)	83
	GameBoyOutput (src/emulator/GameBoyOutput.ts)	84
B.1.2	CPU	85
	CPU (src/emulator/CPU.ts)	85
B.1.3	System	104
	System (src/emulator/System.ts)	104
B.1.4	PPU	109
	PPU (src/emulator/ppu/PPU.ts)	109
	OAM (src/emulator/ppu/OAM.ts)	122
	ColorController (src/emulator/ppu/ColorController.ts)	124
	VRAMController (src/emulator/ppu/VRAMController.ts)	127
B.1.5	APU	131
	APU (src/emulator/apu/APU.ts)	131
	SoundChannel (src/emulator/apu/SoundChannel.ts).	134
	SoundChannel1 (src/emulator/apu/SoundChannel1.ts)	137
	SoundChannel2 (src/emulator/apu/SoundChannel2.ts)	139
	SoundChannel3 (src/emulator/apu/SoundChannel3.ts)	141
	SoundChannel4 (src/emulator/apu/SoundChannel4.ts)	143
B.1.6	MBCs.	145
	MBC (src/emulator/mbc/MBC.ts)	145

NoMBC (src/emulator/mbc/NoMBC.ts)	146
MBC1 (src/emulator/mbc/MBC1.ts)	147
MBC2 (src/emulator/mbc/MBC2.ts)	149
MBC3 (src/emulator/mbc/MBC3.ts)	151
MBC5 (src/emulator/mbc/MBC5.ts)	154
B.1.7 Other Components	156
GameCartridge (src/emulator/GameCartridge.ts)	156
JoypadInput (src/emulator/JoypadInput.ts)	158
Timer (src/emulator/Timer.ts)	160
Interrupts (src/emulator/Interrupts.ts)	162
WRAM (src/emulator/WRAM.ts)	164
B.1.8 Helpers	165
Memory (src/emulator/Memory.ts)	165
Register (src/emulator/Register.ts)	167
constants (src/emulator/constants.ts)	169
util (src/emulator/util.ts)	170
B.2 Frontend Code	171
B.2.1 Main App	171
main (src/frontend/main.tsx)	171
app (src/frontend/app.tsx)	172
B.2.2 Main Styling	178
index (src/frontend/index.css)	178
mobile (src/frontend/mobile.css)	181
B.2.3 Components	182
Alerts (src/frontend/components/Alerts.tsx)	182
Alerts (src/frontend/components/Alerts.css)	184
GameInput (src/frontend/components/GameInput.tsx)	185
GameInput (src/frontend/components/GameInput.css)	187
IconButton (src/frontend/components/IconButton.tsx)	188
KeybindingInput (src/frontend/components/KeybindingInput.tsx)	189
KeybindingInput (src/frontend/components/KeybindingInput.css)	191
Resizable (src/frontend/components/Resizable.tsx)	192
Resizable (src/frontend/components/Resizable.css)	193
RomInput (src/frontend/components/RomInput.tsx)	194
Screen (src/frontend/components/Screen.tsx)	195

B.2.4	Drawer Components	198
	Drawer (src/frontend/components/Drawer/Drawer.tsx).	198
	Drawer (src/frontend/components/Drawer/Drawer.css).	199
	DrawerSection (src/frontend/components/Drawer/DrawerSection.tsx)	202
	ExpressionDrawer (src/frontend/components/Drawer/ExpressionDrawer.tsx)	203
	ExpressionWatch (src/frontend/components/Drawer/ExpressionWatch.tsx).	204
	Grid2x (src/frontend/components/Drawer/Grid2x.tsx).	206
	KeysDrawer (src/frontend/components/Drawer/KeysDrawer.tsx)	208
	MemoryDrawer (src/frontend/components/Drawer/MemoryDrawer.tsx)	210
	SettingsDrawer (src/frontend/components/Drawer/SettingsDrawer.tsx).	211
	TestDrawer (src/frontend/components/Drawer/TestDrawer.tsx).	216
B.2.5	Helpers	220
	AudioPlayer (src/frontend/helpers/AudioPlayer.ts)	220
	ConfigContext (src/frontend/helpers/ConfigContext.tsx)	221
	useKeys (src/frontend/helpers/useKeys.ts)	224
B.2.6	Image Filters	225
	index (src/frontend/helpers/ImageFilter/index.ts)	225
	Base (src/frontend/helpers/ImageFilter/Base.ts).	226
	Identity (src/frontend/helpers/ImageFilter/Identity.ts)	227
	Scale2x (src/frontend/helpers/ImageFilter/Scale2x.ts).	228
	Scale4x (src/frontend/helpers/ImageFilter/Scale4x.ts).	229
B.2.7	Automated Test Config	230
	testConfig (src/frontend/testConfig.ts)	230

B.1 Emulator Core Code

B.1.1 Emulator Main Interfaces

GameBoyColor (src/emulator/GameBoyColor.ts)

```
1 import { ConsoleType, CYCLES_PER_FRAME, SpeedMode } from "../constants";
2 import CPU from "../CPU";
3 import GameBoyInput from "../GameBoyInput";
4 import System from "../System";
5 import GameBoyOutput from "../GameBoyOutput";
6
7 export type GameBoyColorOptions = {
8     bootRom: null | Uint8Array;
9 };
10
11 const DEFAULT_OPTIONS: GameBoyColorOptions = {
12     bootRom: null,
13 };
14
15 class GameBoyColor {
16     protected options: GameBoyColorOptions;
17     protected mode: ConsoleType;
18
19     protected cpu: CPU;
20     protected system: System;
21
22     protected cpuIsHalted = false;
23     protected cycles: number = 0;
24     protected isFullCycle = true; // used for double speed mode
25
26     protected output: GameBoyOutput;
27
28     constructor(
29         modeStr: "DMG" | "CGB",
30         rom: Uint8Array,
31         input: GameBoyInput,
32         output: GameBoyOutput,
33         options?: Partial<GameBoyColorOptions>
34     ) {
35         this.mode = modeStr === "DMG" ? ConsoleType.DMG : ConsoleType.CGB;
36         this.system = new System(rom, input, output, this.mode);
37         this.cpu = new CPU(() => this.system.didStopInstruction());
38         this.output = output;
39         this.options = { ...DEFAULT_OPTIONS, ...options };
40
41         this.setup();
42     }
43
44     protected setup() {
45         if (this.options.bootRom !== null) {
46             if (this.mode === ConsoleType.DMG && this.options.bootRom.length !== 256) {
47                 throw new Error("DMG boot ROM must be 256 bytes long");
48             }
49             if (this.mode === ConsoleType.CGB && this.options.bootRom.length !== 2304) {
50                 throw new Error("CGB boot ROM must be 2304 bytes long");
51             }
52             this.system.loadBootRom(this.options.bootRom);
53         }
54
55         // Setup registers as if the boot ROM was executed
56         else {
57             // CPU
58             if (this.mode === ConsoleType.DMG) {
59                 this.cpu["regAF"].set(0x01b0);
```

```

60         this.cpu["regBC"].set(0x0013);
61         this.cpu["regDE"].set(0x00d8);
62         this.cpu["regHL"].set(0x014d);
63     } else {
64         this.cpu["regAF"].set(0x1180);
65         this.cpu["regBC"].set(0x0000);
66         this.cpu["regDE"].set(0xff56);
67         this.cpu["regHL"].set(0x000d);
68     }
69     this.cpu["regPC"].set(0x0100);
70     this.cpu["regSP"].set(0xffff);
71
72     // PPU
73     this.system["ppu"]["ppu"]["lcdControl"].set(0x91);
74
75     // Emulate GBC compatibility check
76     if (this.mode === ConsoleType.CGB) {
77         const compat = this.system.read(0x0143);
78         if (compat === 0x80 || compat === 0xc0) {
79             this.system.write(0xff4c, compat);
80         } else {
81             const colorControl = this.system["ppu"]["ppu"]["colorControl"];
82             // reset palette controls
83             colorControl.write(0xff68, 0x80);
84             colorControl.write(0xff6a, 0x80);
85
86             // load palettes
87             const bgrPalette = [0xff, 0x7f, 0xef, 0x1b, 0x80, 0x61, 0x00, 0x00];
88             const objPalette = [0xff, 0x7f, 0x1f, 0x42, 0xf2, 0x1c, 0x00, 0x00];
89             bgrPalette.forEach((value) => colorControl.write(0xff69, value)); // bg
90             objPalette.forEach((value) => colorControl.write(0xff6b, value)); // obj0
91             objPalette.forEach((value) => colorControl.write(0xff6b, value)); // obj1
92
93             this.system.write(0xff4c, 0x04); // change to DMG mode
94             this.system["ppu"]["ppu"]["objPriorityMode"].write(0, 0x01); // OPRI
95         }
96     }
97
98     // End initialisation
99     this.system["bootRomLocked"] = true;
100 }
101
102 /**
103  * @returns whether the emulator supports saving the current ROM state.
104  */
105 supportsSaves(): boolean {
106     return this.system.supportsSaves();
107 }
108
109 /** Saves the current ROM state (null if no save support). */
110 save(): Uint8Array | null {
111     return this.system.save();
112 }
113
114 /** Loads the given ROM data. */
115 load(data: Uint8Array): void {
116     this.system.load(data);
117 }
118
119 /** Returns the title of the current ROM. */
120 getTitle(): string {
121     return this.system.getTitle();
122 }
123
124 /** Returns the identifier of the current ROM. */
125 getIdentifier(): string {
126     return this.system.getIdentifier();
127 }
128 }

```

```

129
130 /** The current mode of the emulator. */
131 getMode(): "DMG" | "CGB" {
132     return this.mode === ConsoleType.DMG ? "DMG" : "CGB";
133 }
134
135 /**
136  * Draws a full frame
137  * @param frames number of frames to draw (defaults to 1 to draw every frame - can be used
138  * to speed up emulation).
139  * @param isDebugging whether the emulator is in debugging mode (goes CPU step by step,
140  * prints verbose CPU logs).
141  * @returns the number of T-cycles executed.
142  */
143 drawFrame(frames: number = 1, isDebugging: boolean = false): number {
144     const cycleTarget = CYCLES_PER_FRAME * frames;
145     const interrupts = this.system.getInterrupts();
146     const cyclesStart = this.cycles;
147
148     while (this.cycles < cycleTarget) {
149         const normalSpeedMode = this.system.getSpeedMode() === SpeedMode.Normal;
150         const cycles = normalSpeedMode ? 4 : 2;
151         this.isFullCycle = normalSpeedMode || !this.isFullCycle;
152
153         // one CPU step, convert M-cycles to CPU cycles
154         let cpuIsDone: boolean;
155         if (!this.cpuIsHalted)
156             cpuIsDone = this.cpu.step(this.system, interrupts, isDebugging);
157         else cpuIsDone = true;
158
159         this.cpuIsHalted = this.system.tick(this.isFullCycle);
160
161         this.cycles += cycles;
162
163         // If instruction finished executing and we're debugging
164         if (cpuIsDone && isDebugging) return this.cycles - cyclesStart;
165     }
166
167     // Keep leftover cycles
168     const cyclesRan = this.cycles - cyclesStart;
169     this.cycles %= cycleTarget;
170
171     // Read input
172     this.system.readInput();
173
174     // Output
175     this.system.pushOutput(this.output);
176
177     return cyclesRan;
178 }
179 }
180
181 export default GameBoyColor;

```

GameBoyInput (src/emulator/GameBoyInput.ts)

```
1 type GameBoyInputRead = {
2   up: boolean;
3   down: boolean;
4   left: boolean;
5   right: boolean;
6
7   a: boolean;
8   b: boolean;
9   start: boolean;
10  select: boolean;
11 };
12
13 interface GameBoyInput {
14   read(): GameBoyInputRead;
15 }
16
17 export default GameBoyInput;
```

GameBoyOutput (src/emulator/GameBoyOutput.ts)

```
1 interface GameBoyOutput {
2   /**
3    * @param data the GameBoy's screen output, as an array of RGBA values, making up for a
4    * 160x144 image.
5    */
6   receiveGraphics?(data: Uint32Array): void;
7
8   /**
9    * @param data a sample of sound. The emulator produces samples at a
10   * 44.1Hz rate, and outputs them every 60th of a second (ie. every frame).
11   */
12   receiveSound?(data: Float32Array): void;
13
14   // Debugging methods:
15
16   /**
17    * @param data an array of RGBA values, with the image of the currently loaded background
18    * data in 256x256.
19    */
20   debugBackground?(data: Uint32Array): void;
21
22   /**
23    * @param data an array of RGBA values, with the image of the current tileset in 256x192.
24    */
25   debugTileset?(data: Uint32Array): void;
26
27   /**
28    * @param data the serial output of the Gameboy - called everytime a character is pushed.
29    */
30   serialOut?(data: number): void;
31 }
32
33 export default GameBoyOutput;
```

B.1.2 CPU

CPU (src/emulator/CPU.ts)

```
1 import Interrupts from "../Interrupts";
2 import { Addressable } from "../Memory";
3 import { DoubleRegister, Register } from "../Register";
4 import { asSignedInt8, combine, high, low, wrap16, wrap8 } from "../util";
5
6 const FLAG_ZERO = 1 << 7;
7 const FLAG_SUBTRACTION = 1 << 6;
8 const FLAG_HALFCARRY = 1 << 5;
9 const FLAG_CARRY = 1 << 4;
10
11 type InstructionMethod = (system: Addressable, interrupts: Interrupts) => InstructionReturn;
12 type InstructionReturn = InstructionMethod | null;
13
14 /**
15  * The CPU of the GBC, responsible for reading the code and executing instructions.
16  */
17 class CPU {
18     // All registers are 16 bits long.
19     // AF: lower is flags: ZNHC (zero, subtraction, half-carry, carry)
20     protected regAF = new DoubleRegister();
21     protected regBC = new DoubleRegister();
22     protected regDE = new DoubleRegister();
23     protected regHL = new DoubleRegister();
24     protected regPC = new DoubleRegister(); // program counter
25     protected regSP = new DoubleRegister(); // stack pointer
26
27     // If the CPU is halted
28     protected halted: boolean = false;
29     // If the CPU was halted when IME=0
30     protected haltBug: boolean = false;
31
32     // 8-bit registers, for convenience sake
33     protected srA = this.regAF.h;
34     protected srB = this.regBC.h;
35     protected srC = this.regBC.l;
36     protected srD = this.regDE.h;
37     protected srE = this.regDE.l;
38     protected srH = this.regHL.h;
39     protected srL = this.regHL.l;
40
41     // Next instruction callable
42     protected nextStep: InstructionMethod | null = null;
43
44     // The opcode for the next instruction
45     // This is needed because the CPU actually fetches the opcode on the last M-cycle of the
46     // previous instruction. This emulator stores the value at the PC at the end of each
47     // instruction to use for the next instruction (this opcode is invalidated if an interrupt
48     // happens)
49     protected currentOpcode: number | null = null;
50
51     // for debug purposes
52     protected stepCounter: number = 0;
53
54     // STOP instruction - relies on the rest of the system to stop
55     protected stopInstruction: () => void;
56
57     constructor(stopInstruction: () => void) {
58         this.stopInstruction = stopInstruction;
59     }
60
61     // Returns the next opcode
62     protected nextOpCode(system: Addressable): number {
63         if (this.currentOpcode === null) {
64             return system.read(this.regPC.inc());
```



```

65     }
66     const op = this.currentOpcode;
67     this.currentOpcode = null;
68     return op;
69 }
70
71 /**
72  * Reads the given address and returns it to the receiver.
73  * Takes 1 cycle.
74  */
75 protected readAddress(
76     address: number | (() => number),
77     receiver: (value: number) => InstructionMethod
78 ): InstructionMethod {
79     return (system) => {
80         const effectiveAddress = typeof address === "number" ? address : address();
81         const value = system.read(effectiveAddress);
82         return receiver(value);
83     };
84 }
85
86 /**
87  * Reads the next byte from the PC and increases it.
88  * Takes 1 cycle.
89  */
90 protected nextByte(receiver: (value: number) => InstructionMethod): InstructionMethod {
91     return (system) => {
92         const value = system.read(this.regPC.inc());
93         return receiver(value);
94     };
95 }
96
97 /**
98  * Reads the next word (two bytes) from the PC and increases it.
99  * Takes 2 cycles.
100  */
101 protected nextWord(receiver: (value: number) => InstructionMethod): InstructionMethod {
102     return this.nextByte((low) => this.nextByte((high) => receiver(combine(high, low))));
103 }
104
105 getStepCounts() {
106     return this.stepCounter;
107 }
108
109 getPC() {
110     return this.regPC.get();
111 }
112
113 /**
114  * Steps through one line of the code, and returns the M-cycles required for the
115  * operation
116  * @param system The system to execute the instruction on
117  * @param verbose If true, prints the executed instruction to the console
118  * @returns true if the CPU is in a "set" state (ie. it's halted or just finished an
119  * instruction), false if it is mid-instruction.
120  */
121 step(system: Addressable, interrupts: Interrupts, verbose?: boolean): boolean {
122     if (this.nextStep === null) {
123         const nextStep = this.loadNextOp(system, interrupts, verbose);
124         if (nextStep === "halted") return true;
125         this.nextStep = nextStep;
126     }
127
128     this.nextStep = this.nextStep(system, interrupts);
129     if (this.nextStep === null) {
130         this.currentOpcode = system.read(this.regPC.inc());
131     }
132     return this.nextStep === null;
133 }

```

```

134
135     protected loadNextOp(
136         system: Addressable,
137         interrupts: Interrupts,
138         verbose?: boolean
139     ): InstructionMethod | "halted" {
140         // Check if any interrupt is requested. This also stops HALting.
141         if (interrupts.hasPendingInterrupt) {
142             this.halted = false;
143             if (interrupts.interruptsEnabled) {
144                 const execNext = interrupts.handleNextInterrupt();
145                 // Interrupt handling takes 5 cycles
146                 const nextStep = () => () => this.call(execNext, () => null);
147                 this.currentOpcode = null;
148                 this.regPC.dec(); // undo the read done at the end of the previous instruction
149                 if (verbose)
150                     console.log("[CPU] interrupt execute, goto", execNext.toString(16));
151                 return nextStep;
152             }
153         }
154
155         // Do nothing if halted
156         if (this.halted) {
157             return "halted";
158         }
159
160         // Execute next instruction
161         const opcode = this.nextOpCode(system);
162         ++this.stepCounter;
163         if (verbose)
164             console.log(
165                 `[CPU] ${this.stepCounter} - (0x${(this.regPC.get() - 1).toString(
166                     16
167                 ))} executing op 0x${opcode.toString(16)}`
168             );
169
170         if (this.haltBug) {
171             this.haltBug = false;
172             this.regPC.dec();
173         }
174
175         const instruction = this.instructionSet[opcode];
176         if (instruction === undefined) {
177             throw Error(
178                 `Unrecognized opcode ${opcode?.toString(16)} at address ${((
179                     this.regPC.get() - 1
180                 ).toString(16))}`
181             );
182         }
183         return instruction;
184     }
185
186     /**
187     * A list of all 8-bit opcodes.
188     * Each callable executes the instruction, and returns the number of M-cycles that the
189     * instruction took.
190     * @link https://meganesulli.com/generate-gb-opcodes/
191     */
192     protected instructionSet: Partial<Record<number, InstructionMethod>> = {
193         // NOP
194         0x00: () => null,
195         // STOP
196         0x10: () => {
197             this.stopInstruction();
198             return null;
199         },
200         // extended instructions
201         0xcb: this.nextByte((opcode) => (system, interrupts) => {
202             const instruction = this.extendedInstructionSet[opcode];

```

```

203         if (instruction === undefined) {
204             throw Error(
205                 `Unrecognized extended opcode ${opcode.toString(16)} at address ${
206                     this.regPC.get() - 1
207                 }.toString(16)`)
208         );
209     }
210     return instruction(system, interrupts);
211 },
212 // LD BC/DE/HL/SP, d16
213 ...this.generateOperation(
214     {
215         0x01: this.regBC,
216         0x11: this.regDE,
217         0x21: this.regHL,
218         0x31: this.regSP,
219     },
220     (register) =>
221         this.nextWord((value) => () => {
222             register.set(value);
223             return null;
224         })
225 ),
226 // INC BC/DE/HL/SP
227 ...this.generateOperation(
228     {
229         0x03: this.regBC,
230         0x13: this.regDE,
231         0x23: this.regHL,
232         0x33: this.regSP,
233     },
234     (r) => () => () => {
235         r.inc();
236         return null;
237     }
238 ),
239 // DEC BC/DE/HL/SP
240 ...this.generateOperation(
241     {
242         0x0b: this.regBC,
243         0x1b: this.regDE,
244         0x2b: this.regHL,
245         0x3b: this.regSP,
246     },
247     (r) => () => () => {
248         r.dec();
249         return null;
250     }
251 ),
252 // ADD HL, BC/DE/HL/SP
253 ...this.generateOperation(
254     {
255         0x09: this.regBC,
256         0x19: this.regDE,
257         0x29: this.regHL,
258         0x39: this.regSP,
259     },
260     (register) => () => () => {
261         const hl = this.regHL.get();
262         const n = register.get();
263         const result = wrap16(hl + n);
264         this.regHL.set(result);
265         this.setFlag(FLAG_SUBTRACTION, false);
266         this.setFlag(FLAG_HALFCARRY, (((hl & 0xffff) + (n & 0xffff)) & 0x1000) != 0);
267         this.setFlag(FLAG_CARRY, hl > 0xffff - n);
268         return null;
269     }
270 ),
271 // INC B/D/H/C/E/L/A

```

```

272     ...this.generateOperation(
273     {
274         0x04: this.srB,
275         0x0c: this.srC,
276         0x14: this.srD,
277         0x1c: this.srE,
278         0x24: this.srH,
279         0x2c: this.srL,
280         0x3c: this.srA,
281     },
282     (r) => () => {
283         const result = this.incN(r.get());
284         r.set(result);
285         return null;
286     }
287 ),
288 // INC (HL)
289 0x34: this.readAddress(
290     () => this.regHL.get(),
291     (value) => (s) => {
292         const result = this.incN(value);
293         s.write(this.regHL.get(), result);
294         return () => null;
295     }
296 ),
297 // DEC B/D/H/C/E/L/A
298 ...this.generateOperation(
299     {
300         0x05: this.srB,
301         0x0d: this.srC,
302         0x15: this.srD,
303         0x1d: this.srE,
304         0x25: this.srH,
305         0x2d: this.srL,
306         0x3d: this.srA,
307     },
308     (r) => () => {
309         const result = this.decN(r.get());
310         r.set(result);
311         return null;
312     }
313 ),
314 // DEC (HL)
315 0x35: this.readAddress(
316     () => this.regHL.get(),
317     (value) => (s) => {
318         const result = this.decN(value);
319         s.write(this.regHL.get(), result);
320         return () => null;
321     }
322 ),
323 // LD (BC/DE/HL+/HL-), A
324 ...this.generateOperation(
325     {
326         0x02: () => this.regBC.get(),
327         0x12: () => this.regDE.get(),
328         0x22: () => this.regHL.inc(),
329         0x32: () => this.regHL.dec(),
330     },
331     (getAddress) => (system) => {
332         const address = getAddress();
333         system.write(address, this.srA.get());
334         return () => null;
335     }
336 ),
337 // LD A, (BC/DE/HL+/HL-)
338 ...this.generateOperation(
339     {
340         0x0a: () => this.regBC.get(),

```

```

341         0x1a: () => this.regDE.get(),
342         0x2a: () => this.regHL.inc(),
343         0x3a: () => this.regHL.dec(),
344     },
345     (getAddress) =>
346         this.readAddress(getAddress, (value) => () => {
347             this.srA.set(value);
348             return null;
349         })
350 ),
351 // LD B/C/D/E/H/L/A, d8
352 ...this.generateOperation(
353     {
354         0x06: this.srB,
355         0x0e: this.srC,
356         0x16: this.srD,
357         0x1e: this.srE,
358         0x26: this.srH,
359         0x2e: this.srL,
360         0x3e: this.srA,
361     },
362     (r) =>
363         this.nextByte((value) => () => {
364             r.set(value);
365             return null;
366         })
367 ),
368 // LD (HL), d8
369 0x36: this.nextByte((value) => (s) => {
370     s.write(this.regHL.get(), value);
371     return () => null;
372 }),
373 // LD (a16), SP
374 0x08: this.nextWord((value) => (s) => {
375     s.write(value, this.regSP.l.get());
376     return () => {
377         s.write(value + 1, this.regSP.h.get());
378         return () => null;
379     };
380 }),
381 // LD B/C/D/E/H/L/A, B/C/D/E/H/L/A
382 ...this.generateOperation<number, [Register, Register]>(
383     {
384         0x40: [this.srB, this.srB],
385         0x41: [this.srB, this.srC],
386         0x42: [this.srB, this.srD],
387         0x43: [this.srB, this.srE],
388         0x44: [this.srB, this.srH],
389         0x45: [this.srB, this.srL],
390         0x47: [this.srB, this.srA],
391
392         0x48: [this.srC, this.srB],
393         0x49: [this.srC, this.srC],
394         0x4a: [this.srC, this.srD],
395         0x4b: [this.srC, this.srE],
396         0x4c: [this.srC, this.srH],
397         0x4d: [this.srC, this.srL],
398         0x4f: [this.srC, this.srA],
399
400         0x50: [this.srD, this.srB],
401         0x51: [this.srD, this.srC],
402         0x52: [this.srD, this.srD],
403         0x53: [this.srD, this.srE],
404         0x54: [this.srD, this.srH],
405         0x55: [this.srD, this.srL],
406         0x57: [this.srD, this.srA],
407
408         0x58: [this.srE, this.srB],
409         0x59: [this.srE, this.srC],

```

```

410         0x5a: [this.srE, this.srD],
411         0x5b: [this.srE, this.srE],
412         0x5c: [this.srE, this.srH],
413         0x5d: [this.srE, this.srL],
414         0x5f: [this.srE, this.srA],
415
416         0x60: [this.srH, this.srB],
417         0x61: [this.srH, this.srC],
418         0x62: [this.srH, this.srD],
419         0x63: [this.srH, this.srE],
420         0x64: [this.srH, this.srH],
421         0x65: [this.srH, this.srL],
422         0x67: [this.srH, this.srA],
423
424         0x68: [this.srL, this.srB],
425         0x69: [this.srL, this.srC],
426         0x6a: [this.srL, this.srD],
427         0x6b: [this.srL, this.srE],
428         0x6c: [this.srL, this.srH],
429         0x6d: [this.srL, this.srL],
430         0x6f: [this.srL, this.srA],
431
432         0x78: [this.srA, this.srB],
433         0x79: [this.srA, this.srC],
434         0x7a: [this.srA, this.srD],
435         0x7b: [this.srA, this.srE],
436         0x7c: [this.srA, this.srH],
437         0x7d: [this.srA, this.srL],
438         0x7f: [this.srA, this.srA],
439     },
440     ([to, from]) =>
441     () => {
442         to.set(from.get());
443         return null;
444     }
445 ),
446 // LD B/C/D/E/H/L/A (HL)
447 ...this.generateOperation(
448     {
449         0x46: this.srB,
450         0x4e: this.srC,
451         0x56: this.srD,
452         0x5e: this.srE,
453         0x66: this.srH,
454         0x6e: this.srL,
455         0x7e: this.srA,
456     },
457     (r) =>
458         this.readAddress(
459             () => this.regHL.get(),
460             (value) => {
461                 r.set(value);
462                 return () => null;
463             }
464         )
465 ),
466 // LD (HL), B/C/D/E/H/L/A
467 ...this.generateOperation(
468     {
469         0x70: this.srB,
470         0x71: this.srC,
471         0x72: this.srD,
472         0x73: this.srE,
473         0x74: this.srH,
474         0x75: this.srL,
475         0x77: this.srA,
476     },
477     (r) => (system) => {
478         const address = this.regHL.get();

```

```

479         system.write(address, r.get());
480         return () => null;
481     }
482 ),
483 // ADD A, B/C/D/E/H/L/A/(HL)/d8
484 ...this.generateOperation(
485     {
486         0x80: this.srB,
487         0x81: this.srC,
488         0x82: this.srD,
489         0x83: this.srE,
490         0x84: this.srH,
491         0x85: this.srL,
492         0x87: this.srA,
493     },
494     (r) => () => {
495         this.addNTToA(r.get(), false);
496         return null;
497     }
498 ),
499 0x86: this.readAddress(
500     () => this.regHL.get(),
501     (value) => () => {
502         this.addNTToA(value, false);
503         return null;
504     }
505 ),
506 0xc6: this.nextByte((value) => () => {
507     this.addNTToA(value, false);
508     return null;
509 }),
510 // ADDC A, B/C/D/E/H/L/A/(HL)/d8
511 ...this.generateOperation(
512     {
513         0x88: this.srB,
514         0x89: this.srC,
515         0x8a: this.srD,
516         0x8b: this.srE,
517         0x8c: this.srH,
518         0x8d: this.srL,
519         0x8f: this.srA,
520     },
521     (r) => () => {
522         this.addNTToA(r.get(), true);
523         return null;
524     }
525 ),
526 0x8e: this.readAddress(
527     () => this.regHL.get(),
528     (value) => () => {
529         this.addNTToA(value, true);
530         return null;
531     }
532 ),
533 0xce: this.nextByte((value) => () => {
534     this.addNTToA(value, true);
535     return null;
536 }),
537 // SUB A, B/C/D/E/H/L/A/(HL)/d8
538 ...this.generateOperation(
539     {
540         0x90: this.srB,
541         0x91: this.srC,
542         0x92: this.srD,
543         0x93: this.srE,
544         0x94: this.srH,
545         0x95: this.srL,
546         0x97: this.srA,
547     },

```

```

548         (r) => () => {
549             this.subNFromA(r.get(), false);
550             return null;
551         }
552     ),
553     0x96: this.readAddress(
554         () => this.regHL.get(),
555         (value) => () => {
556             this.subNFromA(value, false);
557             return null;
558         }
559     ),
560     0xd6: this.nextByte((value) => () => {
561         this.subNFromA(value, false);
562         return null;
563     })),
564     // SBC A, B/C/D/E/H/L/A/(HL)/d8
565     ...this.generateOperation(
566         {
567             0x98: this.srB,
568             0x99: this.srC,
569             0x9a: this.srD,
570             0x9b: this.srE,
571             0x9c: this.srH,
572             0x9d: this.srL,
573             0x9f: this.srA,
574         },
575         (r) => () => {
576             this.subNFromA(r.get(), true);
577             return null;
578         }
579     ),
580     0x9e: this.readAddress(
581         () => this.regHL.get(),
582         (value) => () => {
583             this.subNFromA(value, true);
584             return null;
585         }
586     ),
587     0xde: this.nextByte((value) => () => {
588         this.subNFromA(value, true);
589         return null;
590     })),
591     // AND/XOR/OR B/C/D/E/H/L/A
592     ...this.generateOperation<number>, [Register, "&" | "|" | "^"]>(
593         {
594             0xa0: [this.srB, "&"],
595             0xa1: [this.srC, "&"],
596             0xa2: [this.srD, "&"],
597             0xa3: [this.srE, "&"],
598             0xa4: [this.srH, "&"],
599             0xa5: [this.srL, "&"],
600             0xa7: [this.srA, "&"],
601
602             0xa8: [this.srB, "^"],
603             0xa9: [this.srC, "^"],
604             0xaa: [this.srD, "^"],
605             0xab: [this.srE, "^"],
606             0xac: [this.srH, "^"],
607             0xad: [this.srL, "^"],
608             0xaf: [this.srA, "^"],
609
610             0xb0: [this.srB, "|"],
611             0xb1: [this.srC, "|"],
612             0xb2: [this.srD, "|"],
613             0xb3: [this.srE, "|"],
614             0xb4: [this.srH, "|"],
615             0xb5: [this.srL, "|"],
616             0xb7: [this.srA, "|"],

```



```

617         },
618         ([r, op]) =>
619             () => {
620                 this.boolNToA(r.get(), op);
621                 return null;
622             }
623     ),
624     // AND/XOR/OR (HL)
625     ...this.generateOperation(
626         {
627             0xa6: "&" as const,
628             0xae: "^" as const,
629             0xb6: "|" as const,
630         },
631         (op) =>
632             this.readAddress(
633                 () => this.regHL.get(),
634                 (value) => () => {
635                     this.boolNToA(value, op);
636                     return null;
637                 }
638             )
639     ),
640     // AND/XOR/OR d8
641     ...this.generateOperation(
642         {
643             0xe6: "&" as const,
644             0xee: "^" as const,
645             0xf6: "|" as const,
646         },
647         (op) =>
648             this.nextByte((value) => () => {
649                 this.boolNToA(value, op);
650                 return null;
651             })
652     ),
653     // CP B/C/D/E/H/L/A/(HL)/d8
654     ...this.generateOperation(
655         {
656             0xb8: this.srB,
657             0xb9: this.srC,
658             0xba: this.srD,
659             0xbb: this.srE,
660             0xbc: this.srH,
661             0xbd: this.srL,
662             0xbf: this.srA,
663         },
664         (r) => () => {
665             this.compNToA(r.get());
666             return null;
667         }
668     ),
669     0xbe: this.readAddress(
670         () => this.regHL.get(),
671         (value) => () => {
672             this.compNToA(value);
673             return null;
674         }
675     ),
676     0xfe: this.nextByte((value) => (s) => {
677         this.compNToA(value);
678         return null;
679     }),
680     // LD (a8), A
681     0xe0: this.nextByte((address) => (system) => {
682         const value = this.srA.get();
683         system.write(0xff00 | address, value);
684         return () => null;
685     }),

```

```

686 // LD A, (a8)
687 0xf0: this.nextByte((address) =>
688     this.readAddress(0xff00 | address, (data) => () => {
689         this.srA.set(data);
690         return null;
691     })
692 ),
693 // LD (C), A
694 0xe2: (s) => {
695     s.write(0xff00 | this.srC.get(), this.srA.get());
696     return () => null;
697 },
698 // LD A, (C)
699 0xf2: this.readAddress(
700     () => 0xff00 | this.srC.get(),
701     (value) => () => {
702         this.srA.set(value);
703         return null;
704     })
705 ),
706 // LD (a16), A
707 0xea: this.nextWord((address) => (system) => {
708     const value = this.srA.get();
709     system.write(address, value);
710     return () => null;
711 }),
712 // LD A, (a16)
713 0xfa: this.nextWord((value) => (s) => {
714     const address = s.read(value);
715     return () => {
716         this.srA.set(address);
717         return null;
718     };
719 }),
720 // RST 0/1/2/3/4/5/6/7
721 ...this.generateOperation(
722     {
723         0xc7: 0x00,
724         0xcf: 0x08,
725         0xd7: 0x10,
726         0xdf: 0x18,
727         0xe7: 0x20,
728         0xef: 0x28,
729         0xf7: 0x30,
730         0xff: 0x38,
731     },
732     (jumpAdr) => this.call(jumpAdr, () => () => null)
733 ),
734 // CALL a16
735 0xcd: this.nextWord((value) => this.call(value, () => () => null)),
736 // CALL NZ/Z/NC/C a16
737 ...this.generateOperation(
738     {
739         0xc4: () => !this.flag(FLAG_ZERO),
740         0xcc: () => this.flag(FLAG_ZERO),
741         0xd4: () => !this.flag(FLAG_CARRY),
742         0xdc: () => this.flag(FLAG_CARRY),
743     },
744     (condition) =>
745         this.nextWord((value) =>
746             condition() ? this.call(value, () => () => null) : () => null
747         )
748 ),
749 // RET
750 0xc9: this.return(() => () => null),
751 // RETI
752 0xd9: this.return((s, i) => {
753     i.enableInterrupts();
754     return () => null;

```

```

755     }),
756     // RET Z/C/NZ/NC
757     ...this.generateOperation(
758         {
759             0xc0: () => !this.flag(FLAG_ZERO),
760             0xc8: () => this.flag(FLAG_ZERO),
761             0xd0: () => !this.flag(FLAG_CARRY),
762             0xd8: () => this.flag(FLAG_CARRY),
763         },
764         (condition) => () => condition() ? this.return(() => () => null) : () => null
765     ),
766     // JP a16
767     0xc3: this.nextWord((value) => this.jump(value, () => () => null)),
768     // JP HL
769     0xe9: this.jump(
770         () => this.regHL.get(),
771         () => null
772     ),
773     // JP Z/C/NZ/NC, a16
774     ...this.generateOperation(
775         {
776             0xc2: () => !this.flag(FLAG_ZERO),
777             0xca: () => this.flag(FLAG_ZERO),
778             0xd2: () => !this.flag(FLAG_CARRY),
779             0xda: () => this.flag(FLAG_CARRY),
780         },
781         (condition) =>
782             this.nextWord(
783                 (value) => () => condition() ? this.jump(value, () => null) : null
784             )
785     ),
786     // JR s8
787     0x18: this.nextByte((value) => this.jumpr(asSignedInt8(value), () => () => null)),
788     // JR NZ/Z/NC/C, s8
789     ...this.generateOperation(
790         {
791             0x20: () => !this.flag(FLAG_ZERO),
792             0x28: () => this.flag(FLAG_ZERO),
793             0x30: () => !this.flag(FLAG_CARRY),
794             0x38: () => this.flag(FLAG_CARRY),
795         },
796         (condition) =>
797             this.nextByte(
798                 (value) => () =>
799                     condition() ? this.jumpr(asSignedInt8(value), () => null) : null
800             )
801     ),
802     // POP BC/DE/HL/AF
803     ...this.generateOperation(
804         {
805             0xc1: this.regBC,
806             0xd1: this.regDE,
807             0xe1: this.regHL,
808         },
809         (r) =>
810             this.pop((value) => () => {
811                 r.set(value);
812                 return null;
813             })
814     ),
815     // We need to mask lower 4 bits bc hardwired to 0
816     0xf1: this.pop((value) => () => {
817         this.regAF.set(value & 0xff0);
818         return null;
819     }),
820     // PUSH BC/DE/HL/AF
821     ...this.generateOperation(
822         {
823             0xc5: this.regBC,

```

```

824         0xd5: this.regDE,
825         0xe5: this.regHL,
826         0xf5: this.regAF,
827     },
828     (register) =>
829         this.push(
830             () => register.get(),
831             () => () => null
832         )
833     ),
834     // RLCA / RLA / RRCA / RRA
835     0x07: () => {
836         const result = this.rotateL(this.srA.get(), false, false);
837         this.srA.set(result);
838         return null;
839     },
840     0x17: () => {
841         const result = this.rotateL(this.srA.get(), true, false);
842         this.srA.set(result);
843         return null;
844     },
845     0x0f: () => {
846         const result = this.rotateR(this.srA.get(), false, false);
847         this.srA.set(result);
848         return null;
849     },
850     0x1f: () => {
851         const result = this.rotateR(this.srA.get(), true, false);
852         this.srA.set(result);
853         return null;
854     },
855     // ADD SP, s8
856     0xe8: this.nextByte((value) => () => {
857         const s8 = asSignedInt8(value);
858         const sp = this.regSP.get();
859         this.regSP.set(this.perfAdd(s8, sp));
860         return () => () => null; // 3 cycles (idk the timing yet)
861     }),
862     // LD HL, SP+s8
863     0xf8: this.nextByte((value) => () => {
864         const s8 = asSignedInt8(value);
865         const sp = this.regSP.get();
866         this.regHL.set(this.perfAdd(s8, sp));
867         return () => null;
868     }),
869     // LD SP, HL
870     0xf9: () => {
871         this.regSP.set(this.regHL.get());
872         return () => null;
873     },
874     // DI / EI
875     0xf3: (s, i) => {
876         i.disableInterrupts();
877         return null;
878     },
879     0xfb: (s, i) => {
880         i.enableInterrupts();
881         return null;
882     },
883     // HALT
884     0x76: (s, interrupts) => {
885         this.halted = true;
886         if (!interrupts.fastEnableInterrupts() && interrupts.hasPendingInterrupt) {
887             this.haltBug = true; // halt bug triggered on HALT when IME == 0 & IE&IF != 0
888         }
889         return null;
890     },
891     // SCF / CCF
892     0x37: () => {

```

```

893         this.setFlag(FLAG_SUBTRACTION, false);
894         this.setFlag(FLAG_HALFCARRY, false);
895         this.setFlag(FLAG_CARRY, true);
896         return null;
897     },
898     0x3f: () => {
899         this.setFlag(FLAG_SUBTRACTION, false);
900         this.setFlag(FLAG_HALFCARRY, false);
901         this.setFlag(FLAG_CARRY, !this.flag(FLAG_CARRY));
902         return null;
903     },
904     // DAA
905     0x27: () => {
906         let a = this.srA.get();
907         let adjust = this.flag(FLAG_CARRY) ? 0x60 : 0x00;
908         if (this.flag(FLAG_HALFCARRY)) {
909             adjust |= 0x06;
910         }
911         if (!this.flag(FLAG_SUBTRACTION)) {
912             if ((a & 0x0f) > 0x09) adjust |= 0x06;
913             if (a > 0x99) adjust |= 0x60;
914         }
915
916         a = wrap8(a + (this.flag(FLAG_SUBTRACTION) ? -adjust : adjust));
917         this.srA.set(a);
918         this.setFlag(FLAG_CARRY, adjust >= 0x60);
919         this.setFlag(FLAG_HALFCARRY, false);
920         this.setFlag(FLAG_ZERO, a === 0);
921         return null;
922     },
923     // CPL
924     0x2f: () => {
925         this.srA.set(~this.srA.get() & 0xff);
926         this.setFlag(FLAG_SUBTRACTION, true);
927         this.setFlag(FLAG_HALFCARRY, true);
928         return null;
929     },
930 };
931
932 /**
933  * A list of all 16-bit opcodes. Works the same as instructionSet.
934  */
935 protected extendedInstructionSet: Partial<Record<number, InstructionMethod>> = {
936     // RLC ...
937     ...this.generateExtendedOperation(0x00, ({ get, set }) =>
938         get((value) => set(this.rotateL(value, false, true)))
939     ),
940     // RRC ...
941     ...this.generateExtendedOperation(0x08, ({ get, set }) =>
942         get((value) => set(this.rotateR(value, false, true)))
943     ),
944     // RL ...
945     ...this.generateExtendedOperation(0x10, ({ get, set }) =>
946         get((value) => set(this.rotateL(value, true, true)))
947     ),
948     // RC ...
949     ...this.generateExtendedOperation(0x18, ({ get, set }) =>
950         get((value) => set(this.rotateR(value, true, true)))
951     ),
952     // SLA ...
953     ...this.generateExtendedOperation(0x20, ({ get, set }) =>
954         get((value) => {
955             const result = (value << 1) & 0xff;
956             this.setFlag(FLAG_ZERO, result === 0);
957             this.setFlag(FLAG_SUBTRACTION, false);
958             this.setFlag(FLAG_HALFCARRY, false);
959             this.setFlag(FLAG_CARRY, ((value >> 7) & 0b1) === 1);
960             return set(result);
961         })

```

```

962     ),
963     // SRA ...
964     ...this.generateExtendedOperation(0x28, ({ get, set }) =>
965         get((value) => {
966             ↪ unchanged
967             const result = ((value >> 1) & 0xff) | (value & (1 << 7)); // bit 7 left
968             this.setFlag(FLAG_ZERO, result === 0);
969             this.setFlag(FLAG_SUBTRACTION, false);
970             this.setFlag(FLAG_HALFCARRY, false);
971             this.setFlag(FLAG_CARRY, (value & 0b1) === 1);
972             return set(result);
973         })
974     ),
975     // SRL ...
976     ...this.generateExtendedOperation(0x38, ({ get, set }) =>
977         get((value) => {
978             const result = (value >> 1) & 0xff;
979             this.setFlag(FLAG_ZERO, result === 0);
980             this.setFlag(FLAG_SUBTRACTION, false);
981             this.setFlag(FLAG_HALFCARRY, false);
982             this.setFlag(FLAG_CARRY, (value & 0b1) === 1);
983             return set(result);
984         })
985     ),
986     // SWAP ...
987     ...this.generateExtendedOperation(0x30, ({ get, set }) =>
988         get((value) => {
989             const result = ((value & 0x0f) << 4) | ((value & 0xf0) >> 4);
990             this.setFlag(FLAG_ZERO, result === 0);
991             this.setFlag(FLAG_SUBTRACTION, false);
992             this.setFlag(FLAG_HALFCARRY, false);
993             this.setFlag(FLAG_CARRY, false);
994             return set(result);
995         })
996     ),
997     // BIT 0/1/2/.../7, ...
998     ...[...new Array(8)].reduce(
999         (previous, _, bit) => ({
1000             ...previous,
1001             ...this.generateExtendedOperation(0x40 + bit * 8, ({ get }) =>
1002                 get((value) => {
1003                     const out = (value >> bit) & 0b1;
1004                     this.setFlag(FLAG_ZERO, out === 0);
1005                     this.setFlag(FLAG_SUBTRACTION, false);
1006                     this.setFlag(FLAG_HALFCARRY, true);
1007                     return null;
1008                 })
1009             ),
1010             {} as Partial<Record<number, InstructionMethod>>
1011         })
1012     ),
1013     // RES 0/1/2/.../7, ...
1014     ...[...new Array(8)].reduce(
1015         (previous, _, bit) => ({
1016             ...previous,
1017             ...this.generateExtendedOperation(0x80 + bit * 8, ({ get, set }) =>
1018                 get((value) => {
1019                     const result = value & ~(1 << bit);
1020                     return set(result);
1021                 })
1022             ),
1023             {} as Partial<Record<number, InstructionMethod>>
1024         })
1025     ),
1026     // SET 0/1/2/.../7, ...
1027     ...[...new Array(8)].reduce(
1028         (previous, _, bit) => ({
1029             ...previous,
1030             ...this.generateExtendedOperation(0xc0 + bit * 8, ({ get, set }) =>

```

```

1030         get((value) => {
1031             const result = value | (1 << bit);
1032             return set(result);
1033         })
1034     ),
1035 },
1036 {} as Partial<Record<number, InstructionMethod>>
1037 ),
1038 };
1039
1040 // Helper functions for instructions
1041 /** Reads flags */
1042 protected flag(flag: number): boolean {
1043     return this.regAF.l.flag(flag);
1044 }
1045 /** Sets flags */
1046 protected setFlag(flag: number, state: boolean) {
1047     this.regAF.l.sflag(flag, state);
1048 }
1049 /** Increments an 8bit value (wrapping), updates flags Z/O/H */
1050 protected incN(n: number): number {
1051     const result = wrap8(n + 1);
1052     this.setFlag(FLAG_ZERO, result === 0);
1053     this.setFlag(FLAG_SUBTRACTION, false);
1054     this.setFlag(FLAG_HALFCARRY, (result & 0xf) < (n & 0xf));
1055     return result;
1056 }
1057
1058 /** Decrements an 8bit value (wrapping), updates flags Z/I/H */
1059 protected decN(n: number): number {
1060     const result = wrap8(n - 1);
1061     this.setFlag(FLAG_ZERO, result === 0);
1062     this.setFlag(FLAG_SUBTRACTION, true);
1063     this.setFlag(FLAG_HALFCARRY, (result & 0xf) > (n & 0xf));
1064     return result;
1065 }
1066
1067 /** Adds a value to register A, updates flags Z/O/H/CY */
1068 protected addNToA(n: number, carry: boolean) {
1069     const a = this.srA.get();
1070     const carryVal = carry && this.flag(FLAG_CARRY) ? 1 : 0;
1071     const result = wrap8(a + n + carryVal);
1072     this.srA.set(result);
1073     this.setFlag(FLAG_ZERO, result === 0);
1074     this.setFlag(FLAG_SUBTRACTION, false);
1075     this.setFlag(FLAG_HALFCARRY, (a & 0xf) + (n & 0xf) + carryVal > 0xf);
1076     this.setFlag(FLAG_CARRY, a + n + carryVal > 0xff);
1077 }
1078 /** Adds the two given 16-bit values (updating flags), returns the result */
1079 protected perfAdd(a: number, b: number) {
1080     const result = wrap16(a + b);
1081     this.setFlag(FLAG_ZERO, false);
1082     this.setFlag(FLAG_SUBTRACTION, false);
1083     this.setFlag(FLAG_CARRY, (a & 0xff) > 0xff - (b & 0xff));
1084     this.setFlag(FLAG_HALFCARRY, (a & 0xf) > 0xf - (b & 0xf));
1085     return result;
1086 }
1087 /** Subtracts a value from register A, updates flags Z/I/H/CY */
1088 protected subNFromA(n: number, carry: boolean) {
1089     const a = this.srA.get();
1090     const carryVal = carry && this.flag(FLAG_CARRY) ? 1 : 0;
1091     const result = wrap8(a - n - carryVal);
1092     this.srA.set(result);
1093     this.setFlag(FLAG_ZERO, result === 0);
1094     this.setFlag(FLAG_SUBTRACTION, true);
1095     this.setFlag(FLAG_HALFCARRY, (a & 0xf) - (n & 0xf) - carryVal < 0);
1096     this.setFlag(FLAG_CARRY, a - n - carryVal < 0);
1097 }
1098 /** Stores the given boolean operation of A and the given value in A, updates Z/O/H/O */

```

```

1099 protected boolNTToA(n: number, op: "&" | "|" | "^") {
1100     const a = this.srA.get();
1101     const result = op === "&" ? a & n : op === "|" ? a | n : a ^ n;
1102     this.srA.set(result);
1103     this.setFlag(FLAG_ZERO, result === 0);
1104     this.setFlag(FLAG_SUBTRACTION, false);
1105     this.setFlag(FLAG_HALFCARRY, op === "&");
1106     this.setFlag(FLAG_CARRY, false);
1107 }
1108 /** Compares the given number with the value in A without changing A, updates Z/1/H/CY */
1109 protected compNToA(n: number) {
1110     const a = this.srA.get();
1111     this.subNFromA(n, false);
1112     this.srA.set(a);
1113 }
1114 /**
1115  * Pushes the given data to the stack pointer's position, and moves it back by two
1116  * Takes 3 cycles
1117  */
1118 protected push(
1119     data: number | (() => number),
1120     receiver: () => InstructionReturn
1121 ): InstructionMethod {
1122     return () => (system) => {
1123         const effectiveData = typeof data === "number" ? data : data();
1124         this.regSP.dec();
1125         system.write(this.regSP.get(), high(effectiveData));
1126         return (system) => {
1127             this.regSP.dec();
1128             system.write(this.regSP.get(), low(effectiveData));
1129             return receiver();
1130         };
1131     };
1132 }
1133 /**
1134  * Pops a 16bit address from the stack pointer's position, and moves it forward by two.
1135  * Takes two cycles.
1136  */
1137 protected pop(receiver: (value: number) => InstructionMethod): InstructionMethod {
1138     return (s) => {
1139         const low = s.read(this.regSP.inc());
1140         return (s) => {
1141             const high = s.read(this.regSP.inc());
1142             return receiver(combine(high, low));
1143         };
1144     };
1145 }
1146 /**
1147  * Pushes the current PC to memory, and jump to the given address.
1148  * Takes 3 cycles.
1149  */
1150 protected call(address: number, receiver: () => InstructionReturn): InstructionMethod {
1151     return this.push(
1152         () => this.regPC.get(),
1153         () => {
1154             this.regPC.set(address);
1155             return receiver();
1156         }
1157     );
1158 }
1159 /**
1160  * Returns the current call (ie. consumes a pointer at SP and sets it to PC).
1161  * Takes 3 cycles.
1162  */
1163 protected return(receiver: InstructionMethod): InstructionMethod {
1164     return this.pop((value) => (s, i) => {
1165         this.regPC.set(value);
1166         return receiver(s, i);
1167     });

```



```

1168     });
1169 }
1170 /**
1171  * Jumps to the given 16bit address
1172  * Takes one cycle
1173  */
1174 protected jump(
1175     n: number | (() => number),
1176     receiver: () => InstructionReturn
1177 ): InstructionMethod {
1178     return () => {
1179         const address = typeof n === "number" ? n : n();
1180         this.regPC.set(address);
1181         return receiver();
1182     };
1183 }
1184 /**
1185  * Relative-jumps by the given 8-bit value
1186  * Takes one cycle
1187  */
1188 protected jumpr(
1189     n: number | (() => number),
1190     receiver: () => InstructionReturn
1191 ): InstructionMethod {
1192     return () => {
1193         const address = typeof n === "number" ? n : n();
1194         this.regPC.set(wrap16(this.regPC.get() + address));
1195         return receiver();
1196     };
1197 }
1198 /** Rotates the given number left. Sets flags Z|0|0|0|N? */
1199 protected rotateL(n: number, useCarry: boolean, setZero: boolean) {
1200     const bit7 = (n >> 7) & 0b1;
1201     const cflag = this.flag(FLAG_CARRY) ? 1 : 0;
1202     const result = ((n << 1) & 0xff) | (useCarry ? cflag : bit7);
1203     this.setFlag(FLAG_ZERO, setZero && result === 0);
1204     this.setFlag(FLAG_SUBTRACTION, false);
1205     this.setFlag(FLAG_HALFCARRY, false);
1206     this.setFlag(FLAG_CARRY, bit7 === 1);
1207     return result;
1208 }
1209 /** Rotates the given number right. Sets flags Z|0|0|0|NO */
1210 protected rotateR(n: number, useCarry: boolean, setZero: boolean) {
1211     const bit0 = n & 0b1;
1212     const cflag = this.flag(FLAG_CARRY) ? 1 : 0;
1213     const result = ((n >> 1) & 0xff) | ((useCarry ? cflag : bit0) << 7);
1214     this.setFlag(FLAG_ZERO, setZero && result === 0);
1215     this.setFlag(FLAG_SUBTRACTION, false);
1216     this.setFlag(FLAG_HALFCARRY, false);
1217     this.setFlag(FLAG_CARRY, bit0 === 1);
1218     return result;
1219 }
1220
1221 /**
1222  * Helper function for instructions that do the same operations for a set of objects.
1223  * @param items The object the operation runs on, matched to its opcode.
1224  * @param execute A function that executes the instruction for a given object.
1225  * @returns An object with the completed instructions
1226  */
1227 protected generateOperation<K extends number, T>(
1228     items: Record<K, T>,
1229     execute: (r: T) => InstructionMethod
1230 ): Record<K, InstructionMethod> {
1231     const obj: Record<K, InstructionMethod> = {} as any;
1232     for (const [opcode, item] of Object.entries(items) as any as [K, T][]) {
1233         obj[opcode] = execute(item);
1234     }
1235     return obj;
1236 }

```

```

1237
1238 /**
1239  * Helper function for instructions that follow the same B-C-D-E-H-L-(HL)-A pattern
1240  * @param baseCode The base code of the instruction (e.g. 0x50)
1241  * @param execute A function that executes the instruction for a given register
1242  * @returns An object with the completed instructions (e.g. 0x50, 0x51, ..., 0x57)
1243  */
1244 protected generateExtendedOperation(
1245     baseCode: number,
1246     execute: (r: {
1247         get: (r: (value: number) => InstructionReturn) => InstructionReturn;
1248         set: (x: number) => InstructionReturn;
1249     }) => InstructionReturn
1250 ): Partial<Record<number, InstructionMethod>> {
1251     const make = (sr: Register) => ({
1252         get: (r: (value: number) => InstructionReturn) => {
1253             const value = sr.get();
1254             return r(value);
1255         },
1256         set: (x: number) => {
1257             sr.set(x);
1258             return null;
1259         },
1260     });
1261     const regB = make(this.srB);
1262     const regC = make(this.srC);
1263     const regD = make(this.srD);
1264     const regE = make(this.srE);
1265     const regH = make(this.srH);
1266     const regL = make(this.srL);
1267     const regA = make(this.srA);
1268
1269     // order matters: B/C/D/E/H/L/(HL)/A
1270     return {
1271         [baseCode + 0]: (s) => execute(regB),
1272         [baseCode + 1]: (s) => execute(regC),
1273         [baseCode + 2]: (s) => execute(regD),
1274         [baseCode + 3]: (s) => execute(regE),
1275         [baseCode + 4]: (s) => execute(regH),
1276         [baseCode + 5]: (s) => execute(regL),
1277         [baseCode + 6]: (s) =>
1278             execute({
1279                 get: (r: (value: number) => InstructionReturn) => {
1280                     const value = s.read(this.regHL.get());
1281                     return () => r(value);
1282                 },
1283                 set: (x: number) => {
1284                     s.write(this.regHL.get(), x);
1285                     return () => null;
1286                 },
1287             }),
1288         [baseCode + 7]: (s) => execute(regA),
1289     };
1290 }
1291 }
1292
1293 export default CPU;

```

B.1.3 System

System (src/emulator/System.ts)

```
1 import APU from "../apu/APU";
2 import { CGBMode, ConsoleType, HRAM_SIZE, SpeedMode } from "../constants";
3 import GameBoyInput from "../GameBoyInput";
4 import PPU from "../ppu/PPU";
5 import JoypadInput from "../JoypadInput";
6 import { CircularRAM, RAM, Addressable, ROM } from "../Memory";
7 import { MaskRegister, Register00, RegisterFF, Register } from "../Register";
8 import GameCartridge from "../GameCartridge";
9 import Timer from "../Timer";
10 import GameBoyOutput from "../GameBoyOutput";
11 import { Int4, rangeObject } from "../util";
12 import { DMGWRAM, GBCWRAM } from "../WRAM";
13 import Interrupts from "../Interrupts";
14
15 const KEY0_DISABLE_ALL = 1 << 2;
16 const KEY0_DISABLE_SOME = 1 << 3;
17
18 class System implements Addressable {
19     // General use
20     protected mode: ConsoleType;
21
22     // Devices
23     protected timer = new Timer();
24     protected apu: APU;
25     protected joypad: JoypadInput;
26     protected ppu: PPU;
27     protected interrupts: Interrupts = new Interrupts();
28
29     // Memory
30     protected bootRom: ROM;
31     protected cartridge: GameCartridge;
32     protected wram: Addressable;
33     protected hram: RAM = new CircularRAM(HRAM_SIZE, 0xff80);
34
35     // System registers
36     protected bootRomLocked = false;
37     protected bootRomRegister: Addressable = {
38         read: () => (this.bootRomLocked ? 0xff : 0xfe),
39         write: (pos, value) => (this.bootRomLocked || (value & 1) === 1),
40     };
41
42     // KEY0: CGB features toggle (CGB Register)
43     protected cgbMode: CGBMode = CGBMode.CGB;
44     protected key0Register: Addressable = {
45         read: () => {
46             return 0x0;
47         },
48         write: (_, value) => {
49             if (this.bootRomLocked) return; // becomes read-only after boot rom is disabled
50
51             if (value & KEY0_DISABLE_SOME) this.cgbMode = CGBMode.DMGExtended;
52             else if (value & KEY0_DISABLE_ALL) this.cgbMode = CGBMode.DMG;
53             else this.cgbMode = CGBMode.CGB;
54
55             this.ppu.setCGBMode(this.cgbMode);
56             this.addressesRegisters[0x4d] =
57                 this.cgbMode === CGBMode.CGB ? this.key1Register : undefined;
58         },
59     };
60
61     // KEY1: Speed switch register (CGB Register)
62     protected speedMode: SpeedMode = SpeedMode.Normal;
63     protected wantsSpeedModeChange = false;
64     protected key1Register: Addressable = {
```

```

65     read: () =>
66         (this.speedMode === SpeedMode.Double ? 1 << 7 : 0) |
67         (this.wantsSpeedModeChange ? 1 << 0 : 0),
68     write: (_, value) => (this.wantsSpeedModeChange = (value & 1) === 1),
69 };
70
71 constructor(
72     rom: Uint8Array,
73     input: GameBoyInput,
74     output: GameBoyOutput,
75     mode: ConsoleType
76 ) {
77     this.mode = mode;
78     this.bootRom = new ROM(mode === ConsoleType.DMG ? 0x100 : 0x900);
79     this.cartridge = new GameCartridge(rom);
80     this.ppu = new PPU(mode);
81     this.wram = mode === ConsoleType.DMG ? new DMGWRAM() : new GBCWRAM();
82     this.joyypad = new JoypadInput(input);
83     this.apu = new APU(output);
84
85     const registerSerial: Addressable = {
86         read: () => 0xff,
87         write: (pos, value) => output.serialOut && output.serialOut(value),
88     };
89
90     this.addressesLastNibble = {
91         ...rangeObject(0x0, 0x7, this.cartridge),
92         ...rangeObject(0x8, 0x9, this.ppu),
93         ...rangeObject(0xa, 0xb, this.cartridge),
94         ...rangeObject(0xc, 0xe, this.wram), // wram and echo
95         0xf: undefined, // handled separately
96     };
97
98     this.addressesRegisters = {
99         0x00: this.joyypad, // joyypad
100        0x01: registerSerial, // SB - serial data
101        0x02: RegisterFF, // CB - serial control
102        ...rangeObject(0x04, 0x07, this.timer), // timer registers
103        0x0f: this.interrupts, // IF
104        ...rangeObject(0x10, 0x26, this.apu), // actual apu registers
105        ...rangeObject(0x30, 0x3f, this.apu), // wave ram
106        ...rangeObject(0x40, 0x4b, this.ppu), // ppu registers
107        0x4c: mode === ConsoleType.CGB ? this.key0Register : undefined, // KEY0 -
108        0x4d: mode === ConsoleType.CGB ? this.key1Register : undefined, // KEY1 - speed
109        ↪ switch
110        0x4f: this.ppu, // ppu vram bank register
111        0x50: this.bootRomRegister, // boot rom register
112        ...rangeObject(0x51, 0x55, this.ppu), // ppu vram dma registers
113        ...rangeObject(0x68, 0x6b, this.ppu), // ppu palette registers (CGB only)
114        0x70: mode === ConsoleType.CGB ? this.wram : undefined, // wram bank register
115        0x72: mode === ConsoleType.CGB ? new Register() : undefined, // undocumented register
116        0x73: mode === ConsoleType.CGB ? new Register() : undefined, // undocumented register
117        0x74: mode === ConsoleType.CGB ? new Register() : undefined, // undocumented register
118        0x75: mode === ConsoleType.CGB ? new MaskRegister(0b1000_1111) : undefined, //
119        ↪ undocumented register
120        0x76: mode === ConsoleType.CGB ? this.apu : undefined, // PCM12
121        0x77: mode === ConsoleType.CGB ? this.apu : undefined, // PCM34
122        ...rangeObject(0x80, 0xfe, this.hram), // hram
123        0xff: this.interrupts, // IE
124    };
125
126    /**
127     * Ticks the whole system for one M-cycle.
128     * @param isMCycle if the tick is a regular M-cycle (everything runs) or a double-speed mode
129     * cycle (the APU and PPU don't run).
130     * @returns if the CPU should be halted (because a VRAM-DMA is in progress).
131     */
132    tick(isMCycle: boolean): boolean {

```

```

132     const haltCpu = this.ppu.tick(this, this.interrupts, isMCycle);
133     this.timer.tick(this.interrupts);
134     if (isMCycle) this.apu.tick(this.timer);
135     this.interrupts.tick();
136
137     return haltCpu;
138 }
139
140 /**
141  * Mapping of addressables depending on the last (most significant) nibble of an address
142  * e.g. 0x0 to 0x7 maps to ROM, etc.
143  */
144 protected addressesLastNibble: Partial<Record<Int4, Addressable>>;
145 /**
146  * Mapping of addressables depending on the first (most significant) byte of an address -
147  * this is only applicable to the 0xff00 to 0xffff range.
148  * e.g. 0x00 maps to joystick, 0x01 maps to serial, 0x04 to 0x07 maps to timer, etc.
149  */
150 protected addressesRegisters: Partial<Record<number, Addressable>>;
151
152 /**
153  * Responsible for following the memory map.
154  * @link https://gbdev.io/pandocs/Memory_Map.html#memory-map
155  */
156 protected getAddress(pos: number): Addressable {
157     if (pos < 0x0000 || pos > 0xffff)
158         throw new Error(`Invalid address to read from ${pos.toString(16)}`);
159
160     // Boot ROM
161     if (!this.bootRomLocked) {
162         if (pos < 0x100) return this.bootRom;
163         // (the CGB's boot rom extends to 0x900, but leaves a gap for the header)
164         if (this.mode === ConsoleType.CGB && 0x200 <= pos && pos < 0x900)
165             return this.bootRom;
166     }
167
168     // Checking last nibble
169     let addressable = this.addressesLastNibble[(pos >> 12) as Int4];
170     if (addressable) return addressable;
171
172     // Registers
173     if ((pos & 0xff00) === 0xff00) {
174         addressable = this.addressesRegisters[pos & 0xff];
175         if (addressable) return addressable;
176     }
177
178     // Echo RAM
179     if (pos <= 0xfdef) return this.wram;
180     // OAM
181     if (pos <= 0xfe9f) return this.ppu;
182     // Illegal Area
183     if (pos <= 0xfeff) return Register00;
184
185     console.debug(
186         `Accessed unmapped area ${pos
187             .toString(16)
188             .padStart(4, "0")}`, return a fake 0xff register`
189     );
190     return RegisterFF;
191 }
192
193 /**
194  * Reads the data at the given 16-bit address. This method will follow the memory map and
195  * return the data belonging to the right component.
196  */
197 read(pos: number): number {
198     return this.getAddress(pos).read(pos);
199 }
200 /**

```

```

201     * Write the 8-bit data at the given 16-bit address. This method will follow the memory map
202     * and write the data in the right component.
203     */
204     write(pos: number, data: number): void {
205         this.getAddress(pos).write(pos, data);
206     }
207
208     /**
209     * When the STOP (0x10) instruction is executed, the system clock will stop. If a speed
210     * mode change is requested, this will be applied and the system will continue.
211     */
212     didStopInstruction() {
213         if (this.wantsSpeedModeChange) {
214             this.wantsSpeedModeChange = false;
215             this.speedMode =
216                 this.speedMode === SpeedMode.Double ? SpeedMode.Normal : SpeedMode.Double;
217         }
218     }
219
220     getSpeedMode(): SpeedMode {
221         return this.speedMode;
222     }
223
224     getInterrupts(): Interrupts {
225         return this.interrupts;
226     }
227
228     /**
229     * Sets the boot ROM data. This will be used on the start up of the system, if the boot ROM
230     * is not skipped.
231     */
232     loadBootRom(data: Uint8Array): void {
233         this.bootRom.rawSet(data);
234     }
235
236     /**
237     * Returns the chain of bytes at the given address, for the given length.
238     * @param pos The start position of the inspection
239     * @param length The number of inspected bytes
240     * @param format The formatting of the values (e.g. 16 for hexadecimal)
241     */
242     inspect(pos: number, length: number = 16, format: number = 16): string {
243         return [...new Array(length)]
244             .map((_, index) =>
245                 this.read(pos + index)
246                     .toString(format)
247                     .padStart((255).toString(format).length, "0")
248             )
249             .join(" ");
250     }
251
252     /** Reads user input */
253     readInput() {
254         this.joypad.readInput();
255     }
256
257     /** Pushes output data if needed */
258     pushOutput(output: GameBoyOutput) {
259         this.ppu.pushOutput(output);
260     }
261
262     /**
263     * @returns if the system supports saving
264     */
265     supportsSaves(): boolean {
266         return this.cartridge.supportsSaves();
267     }
268
269     /** Saves the current ROM state (null if no save support). */

```

```

270     save(): Uint8Array | null {
271         return this.cartridge.save();
272     }
273
274     /** Loads the given ROM data. */
275     load(data: Uint8Array): void {
276         this.cartridge.load(data);
277     }
278
279     /** Returns the title of the current ROM. */
280     getTitle(): string {
281         return this.cartridge.getTitle();
282     }
283
284     /** Returns the identifier of the current ROM */
285     getIdentifier(): string {
286         return this.cartridge.getIdentifier();
287     }
288 }
289
290 export default System;

```

B.1.4 PPU

PPU (src/emulator/ppu/PPU.ts)

```
1 import {
2     CGBMode,
3     ConsoleType,
4     IFLAG_LCDC,
5     IFLAG_VBLANK,
6     SCREEN_HEIGHT,
7     SCREEN_WIDTH,
8 } from "../constants";
9 import { Addressable } from "../Memory";
10 import { MaskRegister, RegisterFF, Register } from "../Register";
11 import { asSignedInt8, Int3, wrap8 } from "../util";
12 import GameBoyOutput from "../GameBoyOutput";
13 import OAM, { Sprite } from "../OAM";
14 import { CGBColorControl, ColorController, DMGColorControl } from "../ColorController";
15 import { VRAMController, CGBVRAMController, DMGVRAMController } from "../VRAMController";
16 import Interrupts from "../Interrupts";
17
18 type KeyForType<T, V> = NonNullable<
19     {
20         [k in keyof T]: T[k] extends V ? k : never;
21     }[keyof T]
22 >;
23
24 type PPUMode = {
25     doTick: KeyForType<PPU, (interrupts: Interrupts) => void>;
26     flag: number;
27     cycles: number;
28 };
29
30 type PPUModeI = PPUMode & { interrupt: number };
31
32 /*
33  * All modes, with:
34  * - flag: corresponding STAT flag
35  * - cycles: cycles until completion (including previous steps)
36  * - interrupt?: optional corresponding STAT interrupt flag
37  */
38 const MODE_HBLANK_FIRST: PPUModeI = {
39     doTick: "tickHBlankFirst",
40     flag: 0b00,
41     cycles: 18,
42     interrupt: 1 << 3,
43 };
44 const MODE_HBLANK: PPUModeI = {
45     doTick: "tickHBlank",
46     flag: 0b00,
47     cycles: 51,
48     interrupt: 1 << 3,
49 };
50 const MODE_VBLANK: PPUModeI = {
51     doTick: "tickVBlank",
52     flag: 0b01,
53     cycles: 114,
54     interrupt: 1 << 4,
55 };
56 const MODE_SEARCHING_OAM: PPUModeI = {
57     doTick: "tickSearchingOam",
58     flag: 0b10,
59     cycles: 20,
60     interrupt: 1 << 5,
61 };
62 const MODE_TRANSFERRING: PPUMode = {
63     doTick: "tickTransferring",
64     flag: 0b11,
```



```

65     cycles: 43,
66 };
67
68 // Helpful constants
69 const SCREEN_HEIGHT_WOFFSCREEN = 154;
70
71 // LCD control flags
72 const LCDC_BG_WIN_PRIO = 1 << 0;
73 const LCDC_OBJ_ENABLE = 1 << 1;
74 const LCDC_OBJ_SIZE = 1 << 2;
75 const LCDC_BG_TILE_MAP_AREA = 1 << 3;
76 /** @link https://gbdev.io/pandocs/LCDC.html#lcdc4--bg-and-window-tile-data-area */
77 const LCDC_BG_WIN_TILE_DATA_AREA = 1 << 4;
78 const LCDC_WIN_ENABLE = 1 << 5;
79 const LCDC_WIN_TILE_MAP_AREA = 1 << 6;
80 const LCDC_LCD_ENABLE = 1 << 7;
81
82 // LCD status flags
83 const STAT_MODE = 0b11;
84 const STAT_LYC_LY_EQ_FLAG = 1 << 2;
85 const STAT_LYC_LY_EQ_INT = 1 << 6;
86
87 // VRAM2 Attributes
88 const VRAM2_ATTR_BG_OAM_PRIORITY = 1 << 7;
89 const VRAM2_ATTR_V_FLIP = 1 << 6;
90 const VRAM2_ATTR_H_FLIP = 1 << 5;
91 const VRAM2_ATTR_VRAM_BANK = 1 << 3;
92 const VRAM2_ATTR_PALETTE = 0b111;
93
94 /**
95  * The PPU of the GBC, responsible for rendering the current state of the console.
96  * @link https://gbdev.io/pandocs/Rendering.html
97  */
98 class PPU implements Addressable {
99     // Internal counter for cycles
100     cycleCounter: number = 0;
101     windowLineCounter: number = 0;
102     mode: PPUMode = MODE_VBLANK;
103
104     interruptStateBefore: boolean = false;
105     interruptLineState = {
106         lycLyMatch: false,
107         oamActive: false,
108         vblankActive: false,
109         hblankActive: false,
110     };
111     nextInterruptLineUpdate: Partial<typeof this.interruptLineState> | null = null;
112
113     // OAM
114     oam = new OAM();
115     canReadOam: boolean = true;
116     canWriteOam: boolean = true;
117
118     // Variable extra cycles during pixel transfer
119     transferExtraCycles: number = 0;
120
121     // Read sprites
122     readSprites: Sprite[] = [];
123
124     // Data Store
125     vramControl: VRAMController;
126     canReadVram: boolean = true;
127     canWriteVram: boolean = true;
128
129     // Temporary buffer when drawing line by line
130     videoBuffer = new Uint32Array(SCREEN_HEIGHT * SCREEN_WIDTH).fill(0xffffffff);
131     // Complete buffer with the last fully drawn frame
132     lastVideoOut = new Uint32Array(SCREEN_HEIGHT * SCREEN_WIDTH);
133     // Debug video output/storage

```

```

134 backgroundVideoBuffer?: Uint32Array;
135 tilesetVideoBuffer?: Uint32Array;
136
137 // General use
138 /** @link https://gbdev.io/pandocs/LCDC.html */
139 lcdControl = new Register(0x00);
140 /** @link https://gbdev.io/pandocs/STAT.html */
141 lcdStatus = new MaskRegister(0b1000_0000, 0x85);
142 /** Only for GBC @link
143 ↪ https://gbdev.io/pandocs/CGB_Registers.html#ff6c--opri-cgb-mode-only-object-priority-mode */
144 objPriorityMode: Addressable;
145
146 // Positioning
147 screenY = new Register(0x00); // these two indicate position of the viewport
148 screenX = new Register(0x00); // in the background map
149
150 lcdY = new Register(0x00); // indicates currently drawn horizontal line
151 lcdYCompare = new Register(0x00);
152
153 windowY = new Register(0x00); // position of the window
154 windowX = new Register(0x00);
155
156 // Color control
157 colorControl: ColorController;
158
159 // General use
160 consoleMode: ConsoleType;
161 cgbMode: CGBMode = CGBMode.DMG; // for cgb only
162 isCgbMode: boolean = false;
163 protected registerAddresses: Record<number, Addressable> = {};
164
165 constructor(mode: ConsoleType) {
166     if (mode === ConsoleType.CGB) {
167         this.cgbMode = CGBMode.CGB;
168         this.vramControl = new CGBVRAMController(true);
169         this.colorControl = new CGBColorControl();
170         this.objPriorityMode = new MaskRegister(0b1111_1110);
171         this.isCgbMode = true;
172     } else {
173         this.vramControl = new DMGVRAMController();
174         this.colorControl = new DMGColorControl();
175         this.objPriorityMode = RegisterFF;
176         this.isCgbMode = false;
177     }
178
179     this.consoleMode = mode;
180     this.updateAddresses();
181 }
182
183 updateAddresses(): void {
184     this.registerAddresses = {
185         0xff40: this.lcdControl,
186         0xff41: this.lcdStatus,
187         0xff42: this.screenY,
188         0xff43: this.screenX,
189         0xff44: this.lcdY,
190         0xff45: this.lcdYCompare,
191         0xff46: this.oam,
192         0xff47: this.colorControl, // DMG Palettes
193         0xff48: this.colorControl, // |
194         0xff49: this.colorControl, // |
195         0xff4a: this.windowY,
196         0xff4b: this.windowX,
197         0xff4f: this.vramControl, // VRAM Bank
198         0xff51: this.vramControl, // CGB VRAM DMA
199         0xff52: this.vramControl, // |
200         0xff53: this.vramControl, // |
201         0xff54: this.vramControl, // |
202         0xff55: this.vramControl, // |

```

```

202         0xff68: this.colorControl, // CGB Palettes
203         0xff69: this.colorControl, // |
204         0xff6a: this.colorControl, // |
205         0xff6b: this.colorControl, // |
206         0xff6c: this.objPriorityMode,
207     };
208 }
209
210 setCGBMode(mode: CGBMode): void {
211     this.cgbMode = mode;
212     this.isCgbMode = mode !== CGBMode.DMG;
213
214     switch (mode) {
215         case CGBMode.CGB:
216             this.objPriorityMode = new MaskRegister(0b1111_1110);
217             this.vramControl = new CGBVRAMController(true);
218             break;
219
220         case CGBMode.DMGExtended:
221             this.objPriorityMode = new MaskRegister(0b1111_1110);
222             this.vramControl = new CGBVRAMController(false);
223             break;
224
225         case CGBMode.DMG:
226             this.objPriorityMode = RegisterFF;
227             this.vramControl = new DMGVRAMController();
228             break;
229     }
230
231     this.colorControl.changeCBGMode(mode);
232
233     this.updateAddresses();
234 }
235
236 protected haltCpu: boolean = false;
237
238 /**
239  * This the PPU, effectively updating the screen-buffer and rendering it if it's done.
240  * @param system The system that links all components together
241  * @returns Whether the CPU should be halted (a GBC VRAM-DMA is in progress)
242  * @link https://gbdev.io/pandocs/pixel\_fifo.html
243  */
244 tick(system: Addressable, interrupts: Interrupts, isMCycle: boolean): boolean {
245     const isLcdOn = this.lcdControl.flag(LCDC_LCD_ENABLE);
246
247     this.oam.tick(system);
248
249     this.haltCpu = this.vramControl.tick(
250         system,
251         this.mode === MODE_HBLANK && this.lcdY.get() < SCREEN_HEIGHT,
252         isLcdOn
253     );
254
255     if (!isMCycle || !isLcdOn) return this.haltCpu;
256
257     // Update interrupt line from previous write operations?
258     if (this.nextInterruptLineUpdate !== null) {
259         this.updateInterrupt(interrupts, this.nextInterruptLineUpdate);
260         this.nextInterruptLineUpdate = null;
261     }
262
263     this.cycleCounter++;
264
265     if (this.cycleCounter === 1) {
266         this.setMode(this.mode);
267     }
268
269     this[this.mode.doTick](interrupts);
270
271

```

```

271     return this.haltCpu;
272 }
273
274 tickHBlankFirst() {
275     if (this.cycleCounter === MODE_HBLANK_FIRST.cycles) {
276         this.cycleCounter = 0;
277         this.mode = MODE_TRANSFERRING;
278     }
279 }
280
281 tickHBlank(interrupts: Interrupts) {
282     if (this.cycleCounter === 1) {
283         this.updateInterrupt(interrupts, { hblankActive: true });
284
285         this.canReadOam = true;
286
287         // For LY <= 1 there is a delay of one extra cycle
288         if (this.lcdY.get() > 1) {
289             this.canReadVram = true;
290         }
291     }
292
293     if (this.cycleCounter === 2) {
294         this.canReadVram = true;
295         this.canWriteOam = true;
296         this.canWriteVram = true;
297     }
298
299     if (this.cycleCounter === MODE_HBLANK.cycles - this.transferExtraCycles) {
300         this.cycleCounter = 0;
301         const lcdY = this.lcdY.get() + 1;
302         this.lcdY.set(lcdY);
303
304         if (lcdY !== this.lcdYCompare.get()) {
305             this.updateInterrupt(interrupts, { lycLyMatch: false });
306         }
307
308         if (lcdY === SCREEN_HEIGHT) {
309             this.mode = MODE_VBLANK;
310         } else {
311             this.mode = MODE_SEARCHING_OAM;
312         }
313     }
314 }
315
316 tickVBlank(interrupts: Interrupts) {
317     if (this.cycleCounter === 1) {
318         const isVblankStart = this.lcdY.get() === 144;
319         this.updateInterrupt(interrupts, {
320             lycLyMatch: this.lcdY.get() === this.lcdYCompare.get(),
321             vblankActive: isVblankStart || this.interruptLineState.vblankActive,
322             oamActive: isVblankStart || this.interruptLineState.oamActive,
323         });
324
325         if (this.lcdY.get() === 144) {
326             interrupts.requestInterrupt(IFLAG_VBLANK);
327             this.lastVideoOut.set(this.videoBuffer);
328         }
329     } else if (this.cycleCounter === 20) {
330         this.updateInterrupt(interrupts, { oamActive: false });
331     } else if (this.cycleCounter === MODE_VBLANK.cycles) {
332         this.cycleCounter = 0;
333         const lcdY = this.lcdY.get() + 1;
334         if (lcdY === SCREEN_HEIGHT_WOFFSCREEN) {
335             this.lcdY.set(0);
336             this.windowLineCounter = 0;
337             this.mode = MODE_SEARCHING_OAM;
338         } else {
339             this.lcdY.set(lcdY);

```

```

340     }
341   }
342 }
343
344 tickSearchingOam(interrupts: Interrupts) {
345   if (this.cycleCounter === 1) {
346     this.updateInterrupt(interrupts, {
347       oamActive: true,
348       hblankActive: false,
349       vblankActive: false,
350       lycLyMatch: this.lcdY.get() === this.lcdYCompare.get(),
351     });
352     this.canReadOam = false;
353   }
354
355   if (this.cycleCounter === 2) {
356     this.canWriteOam = false;
357   }
358
359   if (this.cycleCounter === MODE_SEARCHING_OAM.cycles) {
360     this.cycleCounter = 0;
361     this.mode = MODE_TRANSFERRING;
362
363     // Read the sprite data here ! this should create a copy !!
364     const y = this.lcdY.get();
365     // Height of objects in pixels
366     const objHeight = this.lcdControl.flag(LCDC_OBJ_SIZE) ? 16 : 8;
367     // This is only relevant in GBC: priority by position or by index
368     let objPrioritySort: (a: [Sprite, number], b: [Sprite, number]) => number;
369     if (this.objPriorityMode.read(0) & 1) {
370       // priority by X position, then by index
371       objPrioritySort = ([spriteA, indexA], [spriteB, indexB]) =>
372         spriteA.x !== spriteB.x ? spriteA.x - spriteB.x : indexA - indexB;
373     } else {
374       // priority only by index
375       objPrioritySort = ([spriteA, indexA], [spriteB, indexB]) => indexA - indexB;
376     }
377     // We select the sprites the following way:
378     // - must be visible
379     // - max 10 per line
380     // - sorted, first by X position then by index
381     this.readSprites = this.oam
382       .getSprites()
383       // only get selected sprites
384       .filter((sprite) => sprite.y <= y && y < sprite.y + objHeight)
385       // only 10 sprites per scanline, lower index first
386       .slice(0, 10)
387       // need to add the index, for sorting
388       .map((sprite, index) => [sprite, index] as [Sprite, number])
389       .sort(objPrioritySort)
390       .map(([sprite]) => sprite);
391   }
392 }
393
394 tickTransferring() {
395   if (this.cycleCounter === 1) {
396     this.updateInterrupt(null, { oamActive: false });
397
398     this.canReadOam = false;
399     this.canReadVram = false;
400
401     this.transferExtraCycles = 0;
402
403     // Extra cycles are spent during transfer depending on scroll, because the tile is
404     // still loaded (the pixels are simply thrown away)
405     const offsetX = this.screenX.get() % 8;
406     this.transferExtraCycles += Math.ceil(offsetX / 4);
407
408     // When drawing sprites we delay extra 6 cycles per sprite.

```

```

409         // We may also delay longer if the sprite is towards the left of the screen, because
410         // the PPU must wait for those pixels to be drawn from the background before
411         // taking care of the sprites. This delay can thus add up to 5 cycles per
412         // X-position.
413         // https://gbdev.io/pandocs/STAT.html#properties-of-stat-modes
414         if (this.lcdControl.flag(LCDC_OBJ_ENABLE)) {
415             let extraSpriteTCycles = 0;
416             let lastPenaltyX = NaN;
417             let lastPenaltyPaid = false;
418             for (let sprite of this.readSprites) {
419                 if (lastPenaltyX !== sprite.x || !lastPenaltyPaid) {
420                     lastPenaltyX = sprite.x;
421                     lastPenaltyPaid = true;
422                     extraSpriteTCycles +=
423                         5 - Math.min(5, (sprite.x + this.screenX.get()) % 8);
424                 }
425                 extraSpriteTCycles += 6;
426             }
427             // Re-convert to M-cycles
428             this.transferExtraCycles += Math.floor(extraSpriteTCycles / 4);
429         }
430
431         this.updateScanline();
432     }
433
434     if (this.cycleCounter === 2) {
435         this.canWriteOam = false;
436         this.canWriteVram = false;
437     }
438
439     if (this.cycleCounter === MODE_TRANSFERRING.cycles + this.transferExtraCycles) {
440         this.cycleCounter = 0;
441         this.mode = MODE_HBLANK;
442     }
443 }
444
445 pushOutput(output: GameBoyOutput) {
446     if (output.receiveGraphics) {
447         output.receiveGraphics(this.lastVideoOut);
448     }
449     if (output.debugBackground) {
450         const backgroundImg = this.debugBackground();
451         output.debugBackground(backgroundImg);
452     }
453     if (output.debugTileset) {
454         const tilesetImg = this.debugTileset();
455         output.debugTileset(tilesetImg);
456     }
457 }
458
459 /** Sets the current mode of the PPU, updating the STAT register. */
460 setMode(mode: PPUMode) {
461     this.lcdStatus.set((this.lcdStatus.get() & ~STAT_MODE) | mode.flag);
462 }
463
464 /**
465  * Will update the STAT interrupt line, raise an interrupt if there is a high to low
466  * transition and the passed in System isn't null (ie. pass null to disable interrupts).
467  */
468 updateInterrupt(
469     interrupts: Interrupts | null,
470     data: Partial<typeof this.interruptLineState>
471 ) {
472     Object.assign(this.interruptLineState, data);
473     const lcdStatus = this.lcdStatus.get();
474     const interruptState =
475         (lcdStatus & STAT_LYC_LY_EQ_INT && this.interruptLineState.lycLyMatch) ||
476         (lcdStatus & MODE_HBLANK.interrupt && this.interruptLineState.hblankActive) ||
477         (lcdStatus & MODE_VBLANK.interrupt && this.interruptLineState.vblankActive) ||

```

```

478         (lcdStatus & MODE_SEARCHING_OAM.interrupt && this.interruptLineState.oamActive);
479
480         this.lcdStatus.sflag(STAT_LYC_LY_EQ_FLAG, this.interruptLineState.lycLyMatch);
481
482         // LCDC Interrupt only happens on rising edges (if allowed)
483         if (interrupts && interruptState && !this.interruptStateBefore) {
484             interrupts.requestInterrupt(IFLAG_LCDC);
485         }
486         this.interruptStateBefore = !!interruptState;
487     }
488
489     protected bgPriorities = new Uint8Array(SCREEN_WIDTH);
490
491     /** Updates the current scanline, by rendering the background, window and then objects. */
492     updateScanline() {
493         this.bgPriorities.fill(0);
494         // The BG/WIN priority flag acts as a toggle only in DMG
495         if (this.isCgbMode || this.lcdControl.flag(LCDC_BG_WIN_PRIO)) {
496             this.drawBackground();
497
498             if (this.lcdControl.flag(LCDC_WIN_ENABLE)) {
499                 this.drawWindow();
500             }
501         } else {
502             this.fillWhite();
503         }
504
505         if (this.lcdControl.flag(LCDC_OBJ_ENABLE)) {
506             this.drawObjects();
507         }
508     }
509
510     /** Function to get access to the tile data, ie. the shades of a tile */
511     getTileAddress(n: number): number {
512         return this.lcdControl.flag(LCDC_BG_WIN_TILE_DATA_AREA)
513             ? // Unsigned regular, 0x8000-0x8fff
514               0x8000 + n * 16
515             : // Signed offset, 0x9000-0x97ff for 0-127 and 0x8800-0x8fff for 128-255
516               0x9000 + asSignedInt8(n) * 16;
517     }
518
519     debugBackground() {
520         const width = 256;
521         const height = 256;
522         if (this.backgroundVideoBuffer === undefined)
523             this.backgroundVideoBuffer = new Uint32Array(width * height);
524
525         // The tilemap used (a map of tile *pointers*)
526         const tileMapLoc = this.lcdControl.flag(LCDC_BG_TILE_MAP_AREA) ? 0x9c00 : 0x9800;
527
528         for (let i = 0; i < 1024; i++) {
529             // Tile positions (0 <= n < 32)
530             const posX = i % 32; // 32 tiles on width
531             const posY = Math.floor(i / 32); // 32 tiles on height
532
533             const tileIndex = tileMapLoc + i;
534
535             // On CGB, the attributes of the tile
536             // Note we can do this even in DMG mode, because VRAM2 in DMG is just a 00 register,
537             // and all the 0 attributes match the normal behaviour of the DMG
538             const tileAttributes = this.vramControl.readBank1(tileIndex);
539             const flipX = (tileAttributes & VRAM2_ATTR_H_FLIP) !== 0;
540             const flipY = (tileAttributes & VRAM2_ATTR_V_FLIP) !== 0;
541             const vramBank = (tileAttributes & VRAM2_ATTR_VRAM_BANK) !== 0 ? 1 : 0;
542             const tilePalette = (tileAttributes & VRAM2_ATTR_PALETTE) as Int3;
543
544             // Map of colors for each shade
545             const palette = this.colorControl.getBgPalette(tilePalette);
546

```

```

547     // The ID (pointer) of the tile
548     const tileAddress = this.vramControl.readBank0(tileIndex);
549     // Convert the ID to the actual address
550     const tileDataAddress = this.getTileAddress(tileAddress);
551     // Get the tile data
552     const tileData = this.vramControl.getTile(tileDataAddress, vramBank);
553
554     // Draw the 8 lines of the tile
555     for (let tileY = 0; tileY < 8; tileY++) {
556         for (let tileX = 0; tileX < 8; tileX++) {
557             const arrayX = flipX ? 7 - tileX : tileX;
558             const arrayY = flipY ? 7 - tileY : tileY;
559             const colorId = tileData[arrayX][arrayY];
560             const index = posX * 8 + posY * width * 8 + tileX + tileY * width;
561             this.backgroundVideoBuffer[index] = palette[colorId];
562         }
563     }
564 }
565
566 return this.backgroundVideoBuffer;
567 }
568
569 debugTileset() {
570     const width = 256; // 16 * 8 * 2;
571     const height = 192; // 24 * 8;
572     if (this.tilesetVideoBuffer === undefined)
573         this.tilesetVideoBuffer = new Uint32Array(width * height);
574
575     // The colors used
576     const palette = {
577         0b00: 0xffffffff,
578         0b01: 0xffaaaaaa,
579         0b10: 0xff555555,
580         0b11: 0xff000000,
581     };
582
583     for (let i = 0; i < 0x180; i++) {
584         const tileAddress = 0x8000 + i * 16;
585         // Tile positions (0 ≤ n < 32)
586         const posX = i % 16; // 20 tiles on width
587         const posY = Math.floor(i / 16);
588
589         // Get tile data (VRAM 0 and 1)
590         const tileData1 = this.vramControl.getTile(tileAddress, 0);
591         const tileData2 = this.vramControl.getTile(tileAddress, 1);
592         // Draw the 8 lines of the tile
593         for (let tileX = 0; tileX < 8; tileX++) {
594             for (let tileY = 0; tileY < 8; tileY++) {
595                 const colorId1 = tileData1[tileX][tileY];
596                 const colorId2 = tileData2[tileX][tileY];
597
598                 const index = posX * 8 + posY * width * 8 + tileX + tileY * width;
599                 this.tilesetVideoBuffer[index] = palette[colorId1];
600                 this.tilesetVideoBuffer[index + width / 2] = palette[colorId2];
601             }
602         }
603     }
604     return this.tilesetVideoBuffer;
605 }
606
607 fillWhite() {
608     const y = this.lcdY.get();
609     const bgPalette = this.colorControl.getBgPalette(0);
610     const white = bgPalette[0]; // needed for CGB in DMG-mode with custom palette
611     for (let x = 0; x < SCREEN_WIDTH; x++) {
612         this.videoBuffer[y * SCREEN_WIDTH + x] = white;
613     }
614 }
615

```



```

616  /**
617   * Handles drawing a single line of the screen, for the background and window.
618   * @param startX The X position to start drawing at.
619   * @param y The Y position to draw at.
620   * @param locationFlag The flag to use to determine the tilemap location.
621   * @param scrollOffsetX The X offset to use for scrolling.
622   * @param getX The function to use to get the X position of the tile.
623   */
624  drawLayer(
625      startX: number,
626      y: number,
627      locationFlag: number,
628      scrollOffsetX: number,
629      getX: (x: number) => number
630  ) {
631      // Global BG priority bit (CGB only)
632      const bgPrioCgb = this.lcdControl.flag(LCDC_BG_WIN_PRIO);
633      // The tilemap used (a map of tile *pointers*)
634      const tileMapLoc = this.lcdControl.flag(locationFlag) ? 0x9c00 : 0x9800;
635      // The currently read Y position of the corresponding tile (one tile is 8 pixels long)
636      const tileY = Math.floor(y / 8);
637      // The currently read Y position *inside* the tile
638      const tileInnerY = y % 8;
639      // Start of video buffer for this line
640      const bufferStart = this.lcdY.get() * SCREEN_WIDTH;
641
642      for (let i = startX; i < SCREEN_WIDTH + scrollOffsetX; i += 8) {
643          // The currently read X pixel of the tile map
644          const x = getX(i);
645          // The currently read X position of the corresponding tile
646          // this determines the tile of the next 8 pixels
647          const tileX = Math.floor(x / 8);
648          // Index of the tile in the current tile data
649          const tileIndex = tileMapLoc + tileX + tileY * 32;
650
651          // On CGB, the attributes of the tile
652          // Note we can do this even in DMG mode, because VRAM2 in DMG is filled with 0x00,
653          // and all the 0 attributes match the normal behaviour of the DMG
654          const tileAttributes = this.vramControl.readBank1(tileIndex);
655          const bgToOamPrio = (tileAttributes & VRAM2_ATTR_BG_OAM_PRIORITY) !== 0;
656          const flipX = (tileAttributes & VRAM2_ATTR_H_FLIP) !== 0;
657          const flipY = (tileAttributes & VRAM2_ATTR_V_FLIP) !== 0;
658          const vramBank = (tileAttributes & VRAM2_ATTR_VRAM_BANK) !== 0 ? 1 : 0;
659          const tilePalette = (tileAttributes & VRAM2_ATTR_PALETTE) as Int3;
660
661          // Map of colors for each shade
662          const palette = this.colorControl.getBgPalette(tilePalette);
663          // The ID (pointer) of the tile
664          const tileAddress = this.vramControl.readBank0(tileIndex);
665          // Convert the ID to the actual address
666          const tileDataAddress = this.getTileAddress(tileAddress);
667          // Get the tile data
668          const tileData = this.vramControl.getTile(tileDataAddress, vramBank);
669
670          for (let innerX = 0; innerX < 8; innerX++) {
671              const posX = i + innerX - scrollOffsetX;
672              if (posX < 0) continue;
673
674              const arrayX = flipX ? 7 - innerX : innerX;
675              const arrayY = flipY ? 7 - tileInnerY : tileInnerY;
676
677              // Get the RGBA color, and draw it!
678              const colorId = tileData[arrayX][arrayY];
679              this.videoBuffer[bufferStart + posX] = palette[colorId];
680
681              // Update priorities
682              this.bgPriorities[posX] = 0;
683              if (colorId > 0) {
684                  if (!this.isCgbMode) {

```

```

685         this.bgPriorities[posX] = 1;
686     } else {
687         if (bgPrioCgb) this.bgPriorities[posX] += 2;
688         if (bgToOamPrio) this.bgPriorities[posX] += 1;
689     }
690 }
691 }
692 }
693 }
694
695 drawBackground() {
696     // The top-left corner of the 160x144 view area
697     const viewX = this.screenX.get();
698     const viewY = this.screenY.get();
699     // Current Y position in the map
700     const y = wrap8(viewY + this.lcdY.get());
701     // The offset of the current line in the map
702     const scrollOffsetX = viewX % 8;
703
704     this.drawLayer(
705         0, // start x
706         y, // y
707         LCDC_BG_TILE_MAP_AREA, // location flag
708         scrollOffsetX, // scroll offset x
709         (x) => wrap8(x + viewX) // get x
710     );
711 }
712
713 drawWindow() {
714     // The top-left corner of the 160x144 view area
715     const windowX = this.windowX.get() - 7;
716     const windowY = this.windowY.get();
717
718     // If the window is not visible, return
719     if (this.lcdY.get() < windowY || windowX >= SCREEN_WIDTH) return;
720
721     // The currently read Y pixel of the window map
722     const y = this.windowLineCounter++;
723
724     this.drawLayer(
725         windowX, // start x
726         y, // y
727         LCDC_WIN_TILE_MAP_AREA, // location flag
728         0, // scroll offset x
729         (x) => wrap8(x - windowX) // get x
730     );
731 }
732
733 drawObjects() {
734     const y = this.lcdY.get();
735     const doubleObjects = this.lcdControl.flag(LCDC_OBJ_SIZE);
736     const sprites = this.readSprites;
737
738     for (const sprite of sprites.reverse()) {
739         // Get tile id (to get the actual data pointer)
740         let tileId = sprite.tileIndex;
741         if (doubleObjects) {
742             // We ignore bit 0 for 8x16 objects
743             tileId &= ~0b1;
744             // if below tile and not flipped, or upper tile but flipped
745             if (y - sprite.y >= 8 !== sprite.yFlip) tileId += 1;
746         }
747         // We need to check if we have double height sprites and this is the lower half of
748         // the sprite, in which case the actual tile address is the next byte
749         const tileAddress = 0x8000 + tileId * 16;
750         // The currently read Y position inside the corresponding tile
751         let tileY = (y - sprite.y) % 8;
752         tileY = sprite.yFlip ? 7 - tileY : tileY;
753         // Get the palette for the object

```

```

754     const palette = this.colorControl.getObjPalette(sprite);
755
756     // Get tile colors
757     const tileData = this.vramControl.getTile(tileAddress, sprite.cgbVramBank);
758
759     for (let innerX = 0; innerX < 8; innerX++) {
760         const x = innerX + sprite.x;
761         // The X value of the sprite is offset by 8 to the left, so we skip off-screen
762         if (x < 0 || x >= SCREEN_WIDTH) continue;
763         const tileX = sprite.xFlip ? 7 - innerX : innerX;
764         const colorId = tileData[tileX][tileY];
765
766         // if transparent, skip
767         // also skip if bg/win should be above, and priority is set
768         if (colorId === 0) continue;
769         if (this.isCgbMode) {
770             if (this.bgPriorities[x] + (sprite.bgAndWinOverObj ? 1 : 0) > 2) continue;
771         } else {
772             if (this.bgPriorities[x] && sprite.bgAndWinOverObj) continue;
773         }
774
775         this.videoBuffer[y * SCREEN_WIDTH + x] = palette[colorId];
776     }
777 }
778 }
779
780 protected address(pos: number): Addressable {
781     // VRAM
782     if (0x8000 <= pos && pos <= 0x9fff) return this.vramControl;
783     // OAM
784     if (0xfe00 <= pos && pos <= 0xfe9f) return this.oam;
785     // Registers
786     const register = this.registerAddresses[pos];
787     if (register) return register;
788
789     throw new Error(`Invalid address given to PPU: ${pos.toString(16)}`);
790 }
791
792 read(address: number): number {
793     const component = this.address(address);
794     if (component === this.oam && !this.canReadOam) return 0xff;
795     if (0x8000 <= address && address <= 0x9fff && !this.canReadVram) return 0xff;
796     return component.read(address);
797 }
798
799 write(address: number, data: number): void {
800     const component = this.address(address);
801
802     if (component === this.oam && !this.canWriteOam) return;
803     if (0x8000 <= address && address <= 0x9fff && !this.canWriteVram) return;
804
805     if (component === this.lcdControl) {
806         const isEnabled = this.lcdControl.flag(LCDC_LCD_ENABLE);
807         const willEnable = (data & LCDC_LCD_ENABLE) === LCDC_LCD_ENABLE;
808
809         // Will disable LCD
810         if (isEnabled && !willEnable) {
811             // console.warn("disabled LCD");
812             this.lcdY.set(0);
813             this.setMode(MODE_HBLANK_FIRST);
814
815             this.canWriteOam = true;
816             this.canReadOam = true;
817             this.canWriteVram = true;
818             this.canReadVram = true;
819         }
820         // Will enable LCD
821         else if (!isEnabled && willEnable) {
822             // console.warn("enabled LCD");

```

```

823         this.lcdY.set(0);
824         this.mode = MODE_HBLANK_FIRST;
825         this.setMode(MODE_HBLANK_FIRST);
826         this.cycleCounter = 0;
827
828         this.nextInterruptLineUpdate = {
829             lycLyMatch: this.lcdY.get() === this.lcdYCompare.get(),
830             vblankActive: false,
831             hblankActive: false,
832             oamActive: false,
833         };
834     }
835 }
836 if (component === this.lcdStatus) {
837     this.nextInterruptLineUpdate = {};
838     // 3 first bits are read-only
839     data = (data & 0b1111_1000) | (this.lcdStatus.get() & 0b0000_0111);
840 }
841
842 component.write(address, data);
843
844 // Writing to LYC updates interrupt line if screen is on only
845 if (component === this.lcdYCompare && this.lcdControl.flag(LCDC_LCD_ENABLE)) {
846     this.nextInterruptLineUpdate = {
847         lycLyMatch: this.lcdYCompare.get() === this.lcdY.get(),
848     };
849 }
850 }
851 }
852
853 /**
854  * To have nice TypeScript types we need to be able to refer to the protected attributes of the
855  * PPU class. However this isn't possible. As such we instead create an exported class that
856  * has all the public attributes of PPU, and choose to not export the PPU that has all its
857  * fields as public.
858  * This means other classes can keep using PPU as it was and don't have access to anything else,
859  * but inside of the file everything is public and usable.
860  */
861 class PPUExported implements Addressable {
862     protected ppu: PPU;
863
864     constructor(mode: ConsoleType) {
865         this.ppu = new PPU(mode);
866     }
867
868     tick(system: Addressable, interrupts: Interrupts, isMCycle: boolean): boolean {
869         return this.ppu.tick(system, interrupts, isMCycle);
870     }
871
872     setCGBMode(mode: CGBMode): void {
873         this.ppu.setCGBMode(mode);
874     }
875
876     pushOutput(output: GameBoyOutput): void {
877         this.ppu.pushOutput(output);
878     }
879
880     read(pos: number): number {
881         return this.ppu.read(pos);
882     }
883
884     write(pos: number, data: number): void {
885         return this.ppu.write(pos, data);
886     }
887 }
888
889 export default PPUExported;

```

OAM (src/emulator/ppu/OAM.ts)

```
1 import { Addressable, RAM } from "../Memory";
2 import { Register } from "../Register";
3 import { Int1, Int3 } from "../util";
4
5 export type Sprite = {
6   x: number;
7   y: number;
8   tileIndex: number;
9   // From attributes:
10  xFlip: boolean;
11  yFlip: boolean;
12  bgAndWinOverObj: boolean;
13  // DMG only
14  dmgPaletteNumber: Int1;
15  // CGB only
16  cgbPaletteNumber: Int3;
17  cgbVramBank: Int1;
18 };
19
20 const ATTRIB_DMG_PALETTE_NUM_IDX = 4;
21 const ATTRIB_CGB_PALETTE_NUM = 0b111;
22 const ATTRIB_CGB_VRAM_BANK_IDX = 3;
23 const ATTRIB_X_FLIP = 1 << 5;
24 const ATTRIB_Y_FLIP = 1 << 6;
25 const ATTRIB_BG_AND_WIN_OVER_OBJ = 1 << 7;
26
27 const NOT_TRANSFERRING = -3;
28 const SHOULD_TRANSFER = -2;
29 const TRANSFER_END = 160;
30
31 /**
32  * The OAM (Object Attribute Memory) used to store sprite data. It is the same as RAM, but has
33  * an extra method to more easily retrieve sprite data, and to do OAM DMA transfers.
34  * @link https://gbdev.io/pandocs/OAM.html
35  */
36 class OAM implements Addressable {
37   /**
38    * There is a 2-cycle delay before any transfer.
39    * -3 = not transferring
40    * -2 = transfer starts in 2 cycles
41    * -1 = transfer starts in 1 cycle
42    * 0-159 = next byte to transfer
43    */
44   protected transferStep: number = NOT_TRANSFERRING;
45   protected transferStart = new Register(0xff);
46   protected data = new RAM(160);
47
48   /** @link https://gbdev.io/pandocs/OAM\_DMA\_Transfer.html */
49   tick(system: Addressable) {
50     // If we're transferring...
51     if (this.transferStep >= 0) {
52       const baseAddress = this.transferStart.get() << 8;
53
54       // Copy a byte
55       const transferredByte = system.read(baseAddress + this.transferStep);
56       this.data.write(this.transferStep, transferredByte);
57       this.spriteCache[this.transferStep >> 2].valid = false;
58     }
59
60     // Tick the transfer and the start delay
61     if (this.transferStep !== NOT_TRANSFERRING) {
62       this.transferStep++;
63       // Transfer ended
64       if (this.transferStep === TRANSFER_END) {
65         this.transferStep = NOT_TRANSFERRING;
66       }
67     }
68   }
69 }
```

```

68     }
69
70     read(pos: number): number {
71         if (pos === 0xff46) return this.transferStart.get();
72         if (0xfe00 <= pos && pos <= 0xfe9f) {
73             if (this.transferStep >= 0) return 0xff; // read disabled
74             return this.data.read(pos - 0xfe00);
75         }
76         throw new Error(`Invalid address for OAM: 0x${pos.toString(16)}`);
77     }
78
79     write(pos: number, data: number): void {
80         if (pos === 0xff46) {
81             this.transferStart.set(data);
82             if (this.transferStep === NOT_TRANSFERRING || this.transferStep > 0)
83                 this.transferStep = SHOULD_TRANSFER;
84         } else if (0xfe00 <= pos && pos <= 0xfe9f) {
85             if (this.transferStep !== NOT_TRANSFERRING) return;
86             const address = pos - 0xfe00;
87             this.data.write(address, data);
88             this.spriteCache[address >> 2].valid = false;
89         }
90     }
91
92     protected spriteCache: (Sprite & { valid: boolean })[] = [...new Array(40)].map(() => ({
93         y: 0,
94         x: 0,
95         tileIndex: 0,
96         xFlip: false,
97         yFlip: false,
98         bgAndWinOverObj: false,
99         dmgPaletteNumber: 0,
100        cgbPaletteNumber: 0,
101        cgbVramBank: 0,
102        valid: false,
103    }));
104
105     getSprites(): Readonly<Sprite>[] {
106         this.spriteCache
107             .filter((x) => !x.valid)
108             .forEach((sprite, index) => {
109                 const address = index << 2;
110                 const attribs = this.data.read(address + 3);
111                 sprite.y = this.data.read(address + 0) - 16;
112                 sprite.x = this.data.read(address + 1) - 8;
113                 sprite.tileIndex = this.data.read(address + 2);
114                 sprite.xFlip = (attribs & ATTRIB_X_FLIP) !== 0;
115                 sprite.yFlip = (attribs & ATTRIB_Y_FLIP) !== 0;
116                 sprite.bgAndWinOverObj = (attribs & ATTRIB_BG_AND_WIN_OVER_OBJ) !== 0;
117                 sprite.dmgPaletteNumber = ((attribs >> ATTRIB_DMG_PALETTE_NUM_IDX) & 1) as Int1;
118                 sprite.cgbPaletteNumber = (attribs & ATTRIB_CGB_PALETTE_NUM) as Int3;
119                 sprite.cgbVramBank = ((attribs >> ATTRIB_CGB_VRAM_BANK_IDX) & 1) as Int1;
120                 sprite.valid = true;
121             });
122         return this.spriteCache;
123     }
124 }
125
126 export default OAM;

```

ColorController (src/emulator/ppu/ColorController.ts)

```
1 import { CGBMode } from "../constants";
2 import { Addressable, RAM } from "../Memory";
3 import { Sprite } from "../OAM";
4 import { Register, MaskRegister } from "../Register";
5 import { Int3 } from "../util";
6
7 export type ColorPalette = [number, number, number, number];
8
9 // Palette flags
10 const PALETTE_AUTO_INCREMENT = 1 << 7;
11 const PALETTE_INDEX = 0b0011_1111;
12
13 abstract class ColorController implements Addressable {
14     protected abstract readonly addresses: Record<number, Addressable>;
15     abstract getBgPalette(id: Int3): ColorPalette;
16     abstract getObjPalette(sprite: Sprite): ColorPalette;
17
18     changeCGBMode(mode: CGBMode): void {}
19
20     read(pos: number): number {
21         const component = this.addresses[pos];
22         if (!component) return 0xff;
23         return component.read(pos);
24     }
25
26     write(pos: number, value: number): void {
27         const component = this.addresses[pos];
28         if (!component) return;
29         component.write(pos, value);
30     }
31 }
32
33 class DMGColorControl extends ColorController {
34     static readonly colorOptions = [
35         0xffffffff, // white
36         0xffaaaaaa, // light gray
37         0xff555555, // dark gray
38         0xff000000, // black
39     ];
40
41     // Background palette
42     protected bgPalette = new Register(0x00);
43     // Object palettes
44     protected objPalette0 = new Register(0x00);
45     protected objPalette1 = new Register(0x00);
46
47     protected addresses = {
48         0xff47: this.bgPalette,
49         0xff48: this.objPalette0,
50         0xff49: this.objPalette1,
51     };
52
53     getBgPalette(): ColorPalette {
54         const palette = this.bgPalette.get();
55         return [
56             DMGColorControl.colorOptions[(palette >> 0) & 0b11],
57             DMGColorControl.colorOptions[(palette >> 2) & 0b11],
58             DMGColorControl.colorOptions[(palette >> 4) & 0b11],
59             DMGColorControl.colorOptions[(palette >> 6) & 0b11],
60         ];
61     }
62
63     getObjPalette(sprite: Sprite): ColorPalette {
64         const palette =
65             sprite.dmgPaletteNumber === 0 ? this.objPalette0.get() : this.objPalette1.get();
66         return [
67             0x00000000, // unused, color 0b00 is transparent
68         ];
69     }
70 }
```

```

68         DMGColorControl.colorOptions[(palette >> 2) & 0b11],
69         DMGColorControl.colorOptions[(palette >> 4) & 0b11],
70         DMGColorControl.colorOptions[(palette >> 6) & 0b11],
71     ];
72 }
73 }
74
75 class CGBCColorControl extends ColorController {
76     // Background palette
77     protected bgPaletteOptions = new MaskRegister(0b0100_0000);
78     protected bgPaletteData = new RAM(64);
79     // Object palettes
80     protected objPaletteOptions = new MaskRegister(0b0100_0000);
81     protected objPaletteData = new RAM(64);
82     // Palette cache
83     protected paletteCache: { data: ColorPalette; valid: boolean }[] = [...Array(16)].map(
84         () => ({
85             data: [0, 0, 0, 0],
86             valid: false,
87         })
88     );
89     // Compatibility mode
90     protected dmgBgPalette = new Register(0x00);
91     protected dmgObj0Palette = new Register(0x00);
92     protected dmgObj1Palette = new Register(0x00);
93     protected dmgMode = false;
94
95     protected addresses = {
96         0xff68: this.bgPaletteOptions,
97         0xff69: this.bgPaletteData,
98         0xff6a: this.objPaletteOptions,
99         0xff6b: this.objPaletteData,
100
101         0xff47: this.dmgBgPalette, // Unused
102         0xff48: this.dmgObj0Palette, // Unused
103         0xff49: this.dmgObj1Palette, // Unused
104     };
105
106     changeCBGMode(mode: CGBMode): void {
107         this.dmgMode = mode !== CGBMode.CGB;
108         this.paletteCache.forEach((x) => (x.valid = false)); // force-reload color 0 for objects
109     }
110
111     override read(pos: number): number {
112         if (pos === 0xff69)
113             return this.bgPaletteData.read(this.bgPaletteOptions.get() & PALETTE_INDEX);
114
115         if (pos === 0xff6b)
116             return this.objPaletteData.read(this.objPaletteOptions.get() & PALETTE_INDEX);
117
118         return super.read(pos);
119     }
120
121     override write(pos: number, value: number): void {
122         if (pos === 0xff69) {
123             const bgPaletteOptions = this.bgPaletteOptions.get();
124             const index = bgPaletteOptions & PALETTE_INDEX;
125             this.bgPaletteData.write(index, value);
126             this.paletteCache[index >> 3].valid = false;
127             if (bgPaletteOptions & PALETTE_AUTO_INCREMENT)
128                 this.bgPaletteOptions.set(
129                     (bgPaletteOptions & ~PALETTE_INDEX) | ((index + 1) & PALETTE_INDEX)
130                 );
131         } else if (pos === 0xff6b) {
132             const objPaletteOptions = this.objPaletteOptions.get();
133             const index = objPaletteOptions & PALETTE_INDEX;
134             this.objPaletteData.write(index, value);
135             this.paletteCache[(index >> 3) + 8].valid = false;
136             if (objPaletteOptions & PALETTE_AUTO_INCREMENT)

```



```

137         this.objPaletteOptions.set(
138             (objPaletteOptions & ~PALETTE_INDEX) | ((index + 1) & PALETTE_INDEX)
139         );
140     } else {
141         super.write(pos, value);
142     }
143 }
144
145 protected decodePalette(data: RAM, id: Int3, offset: 0 | 1, cacheOffset: 0 | 8) {
146     const palette = this.paletteCache[id + cacheOffset];
147     if (palette.valid) return palette.data;
148
149     for (let colorIdx = offset; colorIdx < 4; colorIdx++) {
150         const colorLow = data.read(id * 8 + colorIdx * 2);
151         const colorHigh = data.read(id * 8 + colorIdx * 2 + 1);
152         const fullColor = (colorHigh << 8) | colorLow;
153
154         const red5 = (fullColor >> 0) & 0b0001_1111;
155         const green5 = (fullColor >> 5) & 0b0001_1111;
156         const blue5 = (fullColor >> 10) & 0b0001_1111;
157
158         const red8 = (red5 << 3) | (red5 >> 2);
159         const green8 = (green5 << 3) | (green5 >> 2);
160         const blue8 = (blue5 << 3) | (blue5 >> 2);
161
162         palette.data[colorIdx] = (0xff << 24) | (blue8 << 16) | (green8 << 8) | red8;
163     }
164     palette.valid = true;
165
166     return palette.data;
167 }
168
169 protected applyDmgTransform(
170     palette: ColorPalette,
171     register: Register,
172     isObj: boolean
173 ): ColorPalette {
174     const dmgPalette = register.get();
175     return [
176         isObj ? 0 : palette[(dmgPalette >> 0) & 0b11],
177         palette[(dmgPalette >> 2) & 0b11],
178         palette[(dmgPalette >> 4) & 0b11],
179         palette[(dmgPalette >> 6) & 0b11],
180     ];
181 }
182
183 getBgPalette(id: Int3): ColorPalette {
184     if (!this.dmgMode) return this.decodePalette(this.bgPaletteData, id, 0, 0);
185
186     const palette = this.decodePalette(this.bgPaletteData, 0, 0, 0);
187     const dmgPalette = this.applyDmgTransform(palette, this.dmgBgPalette, false);
188     return dmgPalette;
189 }
190
191 getObjPalette(sprite: Sprite): ColorPalette {
192     if (!this.dmgMode)
193         return this.decodePalette(this.objPaletteData, sprite.cgbPaletteNumber, 1, 8);
194
195     const palette = this.decodePalette(this.objPaletteData, sprite.dmgPaletteNumber, 0, 8);
196     const dmgPalette = this.applyDmgTransform(
197         palette,
198         sprite.dmgPaletteNumber ? this.dmgObj1Palette : this.dmgObj0Palette,
199         true
200     );
201     return dmgPalette;
202 }
203 }
204
205 export { ColorController, DMGColorControl, CGBCColorControl };

```

VRAMController (src/emulator/ppu/VRAMController.ts)

```

1 import { Addressable, CircularRAM } from "../Memory";
2 import { MaskRegister, Register } from "../Register";
3 import { combine, high, Int2, low } from "../util";
4
5 type TileCache = Record<number, { valid: boolean; data: Int2[] [] }>;
6
7 const HDMA5_LENGTH = 0b0111_1111;
8 const HDMA5_MODE = 0b1000_0000;
9
10 abstract class VRAMController implements Addressable {
11     protected abstract readonly addresses: Record<number, Addressable>;
12     protected abstract get currentBank(): Addressable;
13     protected abstract get currentCache(): TileCache;
14
15     protected static makeCache(): TileCache {
16         return [...new Array(0x180)].map(() => ({
17             valid: false,
18             data: Array.from(Array(8), () => new Array(8)),
19         }));
20     }
21
22     protected _getTile(tileAddress: number, bank: Addressable, cache: TileCache): Int2[] [] {
23         const cachedTile = cache[(tileAddress >> 4) & 0x1ff];
24         if (!cachedTile.valid) {
25             // Draw the 8 lines of the tile
26             for (let tileY = 0; tileY < 8; tileY++) {
27                 const tileDataH = bank.read(tileAddress + tileY * 2);
28                 const tileDataL = bank.read(tileAddress + tileY * 2 + 1);
29                 for (let tileX = 0; tileX < 8; tileX++) {
30                     const shadeL = (tileDataH >> (7 - tileX)) & 0b1;
31                     const shadeH = (tileDataL >> (7 - tileX)) & 0b1;
32                     const shade = ((shadeH << 1) | shadeL) as Int2;
33                     cachedTile.data[tileX][tileY] = shade;
34                 }
35             }
36             cachedTile.valid = true;
37         }
38         return cachedTile.data;
39     }
40
41     abstract getTile(tileAddress: number, bankId: 0 | 1): Int2[] [];
42
43     abstract readBank0(pos: number): number;
44     abstract readBank1(pos: number): number;
45
46     /**
47      * Ticks the VRAM. This does nothing on DMG, but ticks the VRAM-DMA on CGB.
48      * @param system The system of the Gameboy. Used for the DMA.
49      * @param isInHblank If the PPU is in a HBlank and LY=0-143.
50      * @param isLcdOn If the LCD is on.
51      * @returns If the CPU should be halted (because a DMA is in progress)
52      */
53     tick(system: Addressable, isInHblank: boolean, isLcdOn: boolean): boolean {
54         return false;
55     }
56
57     read(pos: number): number {
58         if (0x8000 <= pos && pos <= 0x9fff) return this.currentBank.read(pos);
59         const component = this.addresses[pos];
60         if (component) return component.read(pos);
61         return 0xff;
62     }
63
64     write(address: number, value: number): void {
65         if (0x8000 <= address && address <= 0x9fff) {
66             if (
67                 // if in tile memory, dirty tile

```

```

68         0x8000 <= address &&
69         address < 0x9800 &&
70         value != this.currentBank.read(address)
71     ) {
72         this.currentCache[(address >> 4) & 0x1ff].valid = false;
73     }
74     return this.currentBank.write(address, value);
75 }
76 const component = this.addresses[address];
77 if (component) return component.write(address, value);
78 }
79 }
80
81 class DMGVRAMController extends VRAMController {
82     protected vram = new CircularRAM(8192, 0x8000);
83     protected tileCache = VRAMController.makeCache();
84     protected currentBank = this.vram;
85     protected currentCache = this.tileCache;
86     protected readonly addresses: Record<number, Addressable> = {};
87
88     readBank0(pos: number): number {
89         return this.vram.read(pos);
90     }
91     readBank1(pos: number): number {
92         return 0;
93     }
94
95     getTile(tileAddress: number): Int2[] [] {
96         return this._getTile(tileAddress, this.vram, this.tileCache);
97     }
98 }
99
100 enum DMAState {
101     NONE,
102     HBLANK,
103     GENERAL,
104 }
105
106 class CGBVRAMController extends VRAMController {
107     protected vram0 = new CircularRAM(8192, 0x8000);
108     protected vram1 = new CircularRAM(8192, 0x8000);
109     protected tileCache0 = VRAMController.makeCache();
110     protected tileCache1 = VRAMController.makeCache();
111     protected vramBank = new MaskRegister(0b1111_1110);
112
113     protected dmaInProgress: DMAState = DMAState.NONE;
114     /** Number of bytes in the current block that were transferred [0-16] */
115     protected dmaSubSteps: number = 0;
116     protected isFirstDmaBlock: boolean = false;
117
118     protected hdma1 = new Register();
119     protected hdma2 = new Register();
120     protected hdma3 = new Register();
121     protected hdma4 = new Register();
122     protected hdma5 = new Register();
123
124     protected get currentBank() {
125         return this.vramBank.get() & 0b1 ? this.vram1 : this.vram0;
126     }
127     protected get currentCache() {
128         return this.vramBank.get() & 0b1 ? this.tileCache1 : this.tileCache0;
129     }
130
131     protected hdmaEnabled: boolean;
132     protected readonly addresses: Record<number, Addressable>;
133
134     constructor(hdmaEnabled: boolean) {
135         super();
136

```

```

137     this.hdmaEnabled = hdmaEnabled;
138     if (this.hdmaEnabled) {
139         this.addresses = {
140             0xff4f: this.vramBank,
141             0xff51: this.hdma1,
142             0xff52: this.hdma2,
143             0xff53: this.hdma3,
144             0xff54: this.hdma4,
145             0xff55: this.hdma5,
146         };
147     } else {
148         this.addresses = {
149             0xff4f: this.vramBank,
150         };
151     }
152 }
153
154 override tick(system: Addressable, isInHblank: boolean, isLcdOn: boolean): boolean {
155     if (
156         (this.dmaInProgress === DMAState.HBLANK && isInHblank) ||
157         this.dmaInProgress === DMAState.GENERAL
158     ) {
159         const source = combine(this.hdma1.get(), this.hdma2.get()) & 0xffff;
160         const dest = combine(this.hdma3.get(), this.hdma4.get()) & 0xffff;
161
162         const byte1 = system.read(source + this.dmaSubSteps);
163         const byte2 = system.read(source + this.dmaSubSteps + 1);
164         this.write(0x8000 + dest + this.dmaSubSteps, byte1);
165         this.write(0x8000 + dest + this.dmaSubSteps + 1, byte2);
166
167         this.dmaSubSteps += 2;
168
169         if (this.dmaSubSteps === 16) {
170             this.dmaSubSteps = 0;
171             this.isFirstDmaBlock = false;
172
173             const newSource = source + 16;
174             const newDest = dest + 16;
175
176             this.hdma1.set(high(newSource));
177             this.hdma2.set(low(newSource));
178             this.hdma3.set(high(newDest));
179             this.hdma4.set(low(newDest));
180
181             const length = this.hdma5.get() & HDMA5_LENGTH;
182             if (length > 0) {
183                 this.hdma5.set(length - 1);
184             } else {
185                 this.dmaInProgress = DMAState.NONE;
186                 this.hdma5.set(0xff);
187             }
188         }
189
190         return true;
191     }
192     return false;
193 }
194
195 override write(address: number, value: number): void {
196     super.write(address, value);
197
198     if (address === 0xff55) {
199         // Interrupts the transfer
200         if (this.dmaInProgress !== DMAState.NONE) {
201             this.dmaInProgress = DMAState.NONE;
202             this.hdma5.set(this.hdma5.get() | HDMA5_MODE);
203         }
204         // Starts the transfer
205         else {

```

```

206         this.dmaInProgress = value & HDMA5_MODE ? DMAState.HBLANK : DMAState.GENERAL;
207         this.dmaSubSteps = 0;
208         this.isFirstDmaBlock = true;
209         this.hdma5.set(this.hdma5.get() & ~HDMA5_MODE);
210     }
211 }
212 }
213
214 readBank0(pos: number): number {
215     return this.vram0.read(pos);
216 }
217
218 readBank1(pos: number): number {
219     return this.vram1.read(pos);
220 }
221
222 getTile(tileAddress: number, bank: 0 | 1): Int2[][] {
223     return this._getTile(
224         tileAddress,
225         bank ? this.vram1 : this.vram0,
226         bank ? this.tileCache1 : this.tileCache0
227     );
228 }
229 }
230
231 export { VRAMController, DMGVRAMController, CGBVRAMController };

```

B.1.5 APU

APU (src/emulator/apu/APU.ts)

```
1 import { Addressable } from "../Memory";
2 import { CLOCK_SPEED, FRAME_RATE } from "../constants";
3 import GameBoyOutput from "../GameBoyOutput";
4 import { MaskRegister, Register } from "../Register";
5 import { Int4, rangeObject } from "../util";
6 import SoundChannel1 from "./SoundChannel1";
7 import SoundChannel2 from "./SoundChannel2";
8 import SoundChannel3 from "./SoundChannel3";
9 import SoundChannel4 from "./SoundChannel4";
10
11 const SAMPLE_RATE = 44100;
12
13 /**
14  * Cycles for one full sample at 44.1Hz
15  * - We divide clock speed by 4 to get M-cycles
16  */
17 const CYCLES_PER_SAMPLE = CLOCK_SPEED / 4 / SAMPLE_RATE;
18 /** Number of values in a "frame-wide" sample */
19 const SAMPLE_SIZE = Math.floor(SAMPLE_RATE / FRAME_RATE);
20
21 const NR52_APU_TOGGLE = 1 << 7;
22 const NR52_CHAN1_ON = 1 << 0;
23 const NR52_CHAN2_ON = 1 << 1;
24 const NR52_CHAN3_ON = 1 << 2;
25 const NR52_CHAN4_ON = 1 << 3;
26
27 const DIV_ADDRESS = 0xff04;
28 const DIV_TICK_BIT = 1 << 4;
29
30 /**
31  * Converts a digital value in 0 - F into an analog value in -1 - 1 (negative slope)
32  */
33 function DAC(n: Int4): number {
34     return (-n / 0xf) * 2 + 1;
35 }
36
37 /**
38  * The APU (Audio Processing Unit) of the Gameboy - it handles producing sound.
39  */
40 export class APU implements Addressable {
41     protected channel1 = new SoundChannel1((s) => this.nr52.sflag(NR52_CHAN1_ON, s));
42     protected channel2 = new SoundChannel2((s) => this.nr52.sflag(NR52_CHAN2_ON, s));
43     protected channel3 = new SoundChannel3((s) => this.nr52.sflag(NR52_CHAN3_ON, s));
44     protected channel4 = new SoundChannel4((s) => this.nr52.sflag(NR52_CHAN4_ON, s));
45
46     /** Master volume and stereo mix control */
47     protected nr50 = new Register(0x77);
48     /** Stereo mix control register */
49     protected nr51 = new Register(0xf3);
50     /** Status and control register */
51     protected nr52 = new MaskRegister(0b0111_0000, 0xf1);
52
53     /** PCM registers (CGB only) */
54     protected pcm12 = new Register();
55     protected pcm34 = new Register();
56
57     /** Ticking control */
58     protected oldDivBitState = false;
59
60     /** Audio output */
61     protected cyclesForSample: number = 0;
62     protected sampleIndex: number = 0;
63     protected audioBuffer = new Float32Array(SAMPLE_SIZE);
64     protected output: GameBoyOutput;
```

```

65
66     constructor(output: GameBoyOutput) {
67         this.output = output;
68     }
69
70     /**
71      * Ticks the APU system.
72      */
73     tick(system: Addressable): void {
74         // Turned off
75         if (!this.nr52.flag(NR52_APU_TOGGLE)) return;
76
77         const divBitState = (system.read(DIV_ADDRESS) & DIV_TICK_BIT) === DIV_TICK_BIT;
78         const divChanged = !divBitState && this.oldDivBitState;
79         this.oldDivBitState = divBitState;
80
81         const chan1Out = this.channel1.tick(divChanged);
82         const chan2Out = this.channel2.tick(divChanged);
83         const chan3Out = this.channel3.tick(divChanged);
84         const chan4Out = this.channel4.tick(divChanged);
85
86         this.pcm12.set(chan1Out | (chan2Out << 4));
87         this.pcm34.set(chan3Out | (chan4Out << 4));
88
89         if (++this.cyclesForSample >= CYCLES_PER_SAMPLE) {
90             this.cyclesForSample -= CYCLES_PER_SAMPLE;
91
92             // Get all variables for processing audio
93             const nr51 = this.nr51.get();
94             const nr52 = this.nr52.get();
95
96             // Get output from each channel
97             const out1 = DAC(chan1Out);
98             const out2 = DAC(chan2Out);
99             const out3 = DAC(chan3Out);
100            const out4 = DAC(chan4Out);
101
102            // Mix right stereo side, enabling relevant channels
103            const rightAudio =
104                out1 * ((nr51 >> 0) & 1) +
105                out2 * ((nr51 >> 1) & 1) +
106                out3 * ((nr51 >> 2) & 1) +
107                out4 * ((nr51 >> 3) & 1);
108
109            // Mix left stereo side, enabling relevant channels
110            const leftAudio =
111                out1 * ((nr51 >> 4) & 1) +
112                out2 * ((nr51 >> 5) & 1) +
113                out3 * ((nr51 >> 6) & 1) +
114                out4 * ((nr51 >> 7) & 1);
115
116            // Get volume for each side in range 1 to 8
117            const rightVolume = ((nr52 >> 0) & 0b111) + 1;
118            const leftVolume = ((nr52 >> 4) & 0b111) + 1;
119
120            // Mix both sides together, by averaging (taking into account each volume)
121            const monoAudio = (rightAudio * rightVolume + leftAudio * leftVolume) / 16;
122
123            // Do some balancing so the level is correct
124            this.audioBuffer[this.sampleIndex] = monoAudio / 16;
125
126            if (++this.sampleIndex === SAMPLE_SIZE) {
127                this.sampleIndex = 0;
128                if (this.output.receiveSound) {
129                    this.output.receiveSound(this.audioBuffer);
130                }
131            }
132        }
133    }

```

```

134
135     protected addresses: Record<number, Addressable> = {
136         ...rangeObject(0xff10, 0xff14, this.channel1),
137         ...rangeObject(0xff15, 0xff19, this.channel2),
138         ...rangeObject(0xff1a, 0xff1e, this.channel3),
139         ...rangeObject(0xff1f, 0xff23, this.channel4),
140         0xff24: this.nr50,
141         0xff25: this.nr51,
142         0xff26: this.nr52,
143         ...rangeObject(0xff30, 0xff3f, this.channel3), // wave RAM
144         0xff76: this.pcm12,
145         0xff77: this.pcm34,
146     };
147
148     read(pos: number): number {
149         return this.addresses[pos].read(pos);
150     }
151
152     write(pos: number, data: number): void {
153         const component = this.addresses[pos];
154
155         if (component === this.pcm12 || component === this.pcm34) return; // read-only
156
157         // ignore writes to channel when turned off (except for NRX1 and wave RAM)
158         if (
159             !this.nr52.flag(NR52_APU_TOGGLE) &&
160             component !== this.nr52 &&
161             !(0xff30 <= pos && pos <= 0xff3f)
162         )
163             return;
164
165         if (component === this.nr52) {
166             data = data & 0xf0; // lower 4 bits (status of channels) are read-only
167             const wasOn = this.nr52.flag(NR52_APU_TOGGLE);
168             const isOn = data & NR52_APU_TOGGLE;
169
170             if (wasOn && !isOn) {
171                 // when turning off, write 0x00 to all registers
172                 for (let address = 0xff10; address <= 0xff25; address++) {
173                     // Except for NR41 for some reason
174                     if (address === 0xff20) continue;
175                     this.write(address, 0x00);
176                 }
177             }
178         }
179
180         component.write(pos, data);
181     }
182 }
183
184 export default APU;

```


SoundChannel (src/emulator/apu/SoundChannel.ts)

```
1 import { Addressable } from "../Memory";
2 import { Register } from "../Register";
3 import { Int4 } from "../util";
4
5 const FREQUENCY_SWEEP_PACE = 4;
6 const FREQUENCY_ENVELOPE = 8;
7 const FREQUENCY_LENGTH_TIMER = 2;
8
9 const NRX4_RESTART_CHANNEL = 1 << 7;
10 const NRX4_LENGTH_TIMER_FLAG = 1 << 6;
11
12 abstract class SoundChannel implements Addressable {
13     // Channel-dependent
14     protected abstract readonly NRX1_LENGTH_TIMER_BITS: number;
15
16     // Common registers
17     protected abstract nrX1: Register;
18     protected abstract nrX2: Register;
19     protected abstract nrX3: Register;
20     protected abstract nrX4: Register;
21
22     // State
23     protected enabled = false;
24     protected onStateChange: (state: boolean) => void;
25
26     // Counters
27     protected step: number = 0;
28
29     constructor(onStateChange: (state: boolean) => void) {
30         this.onStateChange = onStateChange;
31     }
32
33     /**
34      * Ticks the whole channel. This method should not be overridden by subclasses - for ticking
35      * behavior, override doTick instead.
36      * @returns The output of this channel
37      */
38     tick(divChanged: boolean): Int4 {
39         // Ticks even when disabled
40
41         if (divChanged) {
42             this.step = (this.step + 1) % 8;
43
44             if (this.step % 2 === 0) {
45                 this.tickLengthTimer();
46             }
47             if (this.step % 4 === 2) {
48                 this.tickSweep();
49             }
50             if (this.step === 7) {
51                 this.tickEnvelope();
52             }
53         }
54
55         if (!this.enabled) return 0;
56
57         this.doTick();
58
59         return this.getSample();
60     }
61
62     /** Ticks the length timer. */
63     protected tickLengthTimer(): void {
64         // Tick length timer
65         if (this.nrX4.flag(NRX4_LENGTH_TIMER_FLAG)) {
66             const timerBits = this.NRX1_LENGTH_TIMER_BITS;
67             const nrX1 = this.nrX1.get();
```

```

68         const lengthTimer = ((nrX1 & timerBits) + 1) & timerBits;
69         this.nrX1.set((nrX1 & ~timerBits) | lengthTimer);
70         // overflowed
71         if (lengthTimer === 0) {
72             this.stop();
73         }
74     }
75 }
76
77 protected tickSweep(): void {}
78
79 protected tickEnvelope(): void {}
80
81 /**
82  * Ticks the channel. This method should be overridden by subclasses for channel-specific
83  * behavior.
84  */
85 protected abstract doTick(): void;
86
87 /**
88  * @returns The current value of the channel (value between 0-F).
89  */
90 protected abstract getSample(): Int4;
91
92 /**
93  * @returns The channel's wavelength, using the NRX3 and NRX4 registers. Only relevant for
94  * channels 1, 2 and 3.
95  */
96 protected getWavelength(): number {
97     const lower8 = this.nrX3.get();
98     const higher3 = this.nrX4.get() & 0b111;
99     return (higher3 << 8) | lower8;
100 }
101
102 /**
103  * @param waveLength The new wavelength, using the NRX3 and NRX4 registers. Only relevant
104  * for channels 1, 2 and 3.
105  */
106 protected setWavelength(waveLength: number): void {
107     waveLength &= (1 << 11) - 1; // ensure it fits in 11bits
108     const lower8 = waveLength & 0xff;
109     const higher3 = (waveLength >> 8) & 0b111;
110     this.nrX3.set(lower8);
111     this.nrX4.set((this.nrX4.get() & ~0b111) | higher3);
112 }
113
114 /**
115  * Starts the channel, if it isn't started already.
116  */
117 protected start(): void {
118     if (this.enabled) return;
119     if (!this.isDACOn) return;
120
121     this.enabled = true;
122     this.onStateChange(true);
123     this.onStart();
124 }
125
126 /**
127  * Stops the channel, if it isn't stopped already.
128  */
129 protected stop(): void {
130     if (!this.enabled) return;
131
132     this.enabled = false;
133     this.onStateChange(false);
134 }
135
136 protected onStart(): void {}

```

```
137
138     protected abstract get isDACOn(): boolean;
139
140     abstract read(pos: number): number;
141     abstract write(pos: number, data: number): void;
142 }
143
144 export default SoundChannel;
145 export { NRX4_RESTART_CHANNEL, FREQUENCY_SWEEP_PACE, FREQUENCY_ENVELOPE };
```

SoundChannel1 (src/emulator/apu/SoundChannel1.ts)

```

1 import { Addressable } from "../Memory";
2 import { MaskRegister, Register } from "../Register";
3 import { Int4 } from "../util";
4 import SoundChannel2 from "../SoundChannel2";
5
6 const NRX0_SWEEP_CHANGE = 1 << 3;
7 const NRX0_MULTIPLIER = 0b0000_0111;
8
9 /**
10  * Sound channel 1 generates a pulse signal, with a wavelength sweep.
11  * @link
12  * → https://gbdev.io/pandocs/Audio\_Registers.html#sound-channel-1--pulse-with-wavelength-sweep
13  */
14 class SoundChannel1 extends SoundChannel2 {
15     protected nrX0 = new MaskRegister(0b1000_0000, 0x80);
16     protected nrX1 = new Register(0xbf);
17     protected nrX2 = new Register(0xf3);
18     protected nrX3 = new Register(0xff);
19     protected nrX4 = new Register(0xbf);
20
21     // Needed because going from negate to positive mode turns off the channel
22     protected inNegateMode: boolean = false;
23     // Wavelength sweep pace countdown
24     protected waveSweepCounter: number = 0;
25     // Whether the sweep is enabled at all
26     protected sweepEnabled: boolean = false;
27
28     protected override addresses: Record<number, Addressable> = {
29         0xff10: this.nrX0,
30         0xff11: this.nrX1,
31         0xff12: this.nrX2,
32         0xff13: this.nrX3,
33         0xff14: this.nrX4,
34     };
35
36     protected override tickSweep(): void {
37         if (this.waveSweepCounter > 1) this.waveSweepCounter--;
38         else {
39             const nextCounter = (this.nrX0.get() >> 4) & 0b111; // bits 4-6
40             if (nextCounter === 0) {
41                 this.waveSweepCounter = 8; // 0 is treated as 8
42             } else {
43                 this.waveSweepCounter = nextCounter;
44                 if (this.sweepEnabled) this.applyWavelengthSweep();
45             }
46         }
47     }
48
49     protected applyWavelengthSweep(): void {
50         const addOrSub = this.nrX0.flag(NRX0_SWEEP_CHANGE) ? -1 : 1;
51         const multiplier = this.nrX0.get() & NRX0_MULTIPLIER; // bits 0-2
52
53         const wave = this.wavelength;
54         this.inNegateMode ||= addOrSub === -1;
55         const newWave = wave + addOrSub * (wave >> multiplier);
56
57         // On overflow, stop channel
58         if (newWave > 0x7ff) this.stop();
59         else {
60             // Can't underflow, saturate at 0
61             if (newWave < 0) this.setWavelength(0);
62             // Normal case
63             else this.setWavelength(newWave);
64         }
65     }
66
67     write(pos: number, data: number): void {

```

```

67     super.write(pos, data);
68
69     if (pos === 0xff10) {
70         const wasNegating = this.inNegateMode;
71         const isNegating = this.nrX0.flag(NRX0_SWEEP_CHANGE);
72         if (wasNegating && !isNegating) {
73             this.stop();
74         }
75         this.inNegateMode = false;
76     }
77 }
78
79 override onStart(): void {
80     super.onStart();
81
82     const nrX0 = this.nrX0.get();
83     const nextCounter = (nrX0 >> 4) & 0b111; // bits 4-6
84     this.waveSweepCounter = nextCounter === 0 ? 8 : nextCounter;
85     this.sweepEnabled =
86         ((nrX0 >> 4) & 0b111) !== 0 || // sweep period != 0
87         (nrX0 & NRX0_MULTIPLIER) !== 0; // sweep shift != 0
88
89     // On start, if the shift isn't 0, a sweep overflow check is made
90     if ((nrX0 & NRX0_MULTIPLIER) > 0) {
91         this.applyWavelengthSweep();
92     }
93 }
94 }
95
96 export default SoundChannel1;

```

SoundChannel2 (src/emulator/apu/SoundChannel2.ts)

```
1 import { Addressable } from "../Memory";
2 import { RegisterFF, Register } from "../Register";
3 import { clamp, Int2, Int4 } from "../util";
4 import SoundChannel, { FREQUENCY_ENVELOPE, NRX4_RESTART_CHANNEL } from "../SoundChannel";
5
6 const NRX2_STOP_DAC = 0b1111_1000;
7 const wavePatterns: (0 | 1)[][] = [
8   [1, 1, 1, 1, 1, 1, 1, 0],
9   [0, 1, 1, 1, 1, 1, 1, 0],
10  [0, 1, 1, 1, 1, 0, 0, 0],
11  [1, 0, 0, 0, 0, 0, 0, 1],
12 ];
13
14 /**
15  * Sound channel 2 is identical to channel 1, except that it doesn't have a wavelength sweep.
16  * @link https://gbdev.io/pandocs/Audio\_Registers.html#sound-channel-2--pulse
17  */
18 class SoundChannel2 extends SoundChannel {
19   protected NRX1_LENGTH_TIMER_BITS = 0b0011_1111;
20
21   protected nrX1 = new Register(0x3f);
22   protected nrX2 = new Register(0x00);
23   protected nrX3 = new Register(0xff);
24   protected nrX4 = new Register(0xbf);
25
26   protected addresses: Record<number, Addressable> = {
27     0xff15: RegisterFF,
28     0xff16: this.nrX1,
29     0xff17: this.nrX2,
30     0xff18: this.nrX3,
31     0xff19: this.nrX4,
32   };
33
34   // Stores a private copy of wave length on trigger
35   protected waveLength: number = 0;
36
37   // For output
38   protected ticksPerWaveStep: number = 0;
39   protected waveStep: number = 0;
40   protected waveStepSubsteps: number = 0;
41
42   // NRx2 needs retriggering when changed
43   protected cachedNRX2: number = this.nrX2.get();
44
45   // Channel envelope volume
46   protected envelopeVolumeSteps: number = 0;
47   protected envelopeVolume: Int4 = 0;
48
49   protected override doTick(): void {
50     if (this.waveStepSubsteps++ >= this.ticksPerWaveStep) {
51       this.waveStepSubsteps = 0;
52       this.waveStep = (this.waveStep + 1) % 8;
53     }
54   }
55
56   protected tickEnvelope(): void {
57     if (this.envelopeVolumeSteps > 0) this.envelopeVolumeSteps--;
58     else {
59       this.envelopeVolumeSteps = this.cachedNRX2 & 0b111;
60       if ((this.cachedNRX2 & 0b111) !== 0) {
61         const direction = (this.cachedNRX2 & 0b0000_1000) === 0 ? -1 : 1;
62         this.envelopeVolume = clamp(this.envelopeVolume + direction, 0x0, 0xf) as Int4;
63       }
64     }
65   }
66
67   protected override getSample(): Int4 {
```

```

68     const dutyCycleType = ((this.nrX1.get() >> 6) & 0b11) as Int2;
69     const wavePattern = wavePatterns[dutyCycleType];
70     // if (this.constructor.name === "SoundChannel1")
71     // console.log(this.waveStep, wavePattern[this.waveStep], this.envelopeVolume);
72     return (wavePattern[this.waveStep] * this.envelopeVolume) as Int4;
73 }
74
75 protected override setWavelength(waveLength: number): void {
76     super.setWavelength(waveLength);
77     this.ticksPerWaveStep = 2048 - waveLength;
78     this.waveLength = waveLength;
79 }
80
81 /* Audio control */
82
83 get isDACOn(): boolean {
84     return (this.nrX2.get() & NRX2_STOP_DAC) !== 0;
85 }
86
87 override onStart(): void {
88     this.waveLength = this.getWavelength();
89
90     this.cachedNRX2 = this.nrX2.get();
91     this.envelopeVolume = (this.cachedNRX2 >> 4) as Int4;
92     this.ticksPerWaveStep = 2048 - this.getWavelength();
93 }
94
95 read(pos: number): number {
96     const component = this.addresses[pos];
97
98     // bits 0-5 are write only
99     if (component === this.nrX1) return component.read(pos) | 0b0011_1111;
100    // register is write only
101    if (component === this.nrX3) return 0xff;
102    // only bit 6 is readable
103    if (component === this.nrX4) return component.read(pos) | 0b1011_1111;
104
105    return component.read(pos);
106 }
107
108 write(pos: number, data: number): void {
109     const component = this.addresses[pos];
110
111     component.write(pos, data);
112
113     // Restart
114     if (component === this.nrX4 && (data & NRX4_RESTART_CHANNEL) !== 0) {
115         this.stop();
116         this.start();
117         data &= ~NRX4_RESTART_CHANNEL; // bit is write-only
118     }
119
120     // Clearing bits 3-7 of NRX2 turns the DAC (and the channel) off
121     if (component === this.nrX2 && (data & NRX2_STOP_DAC) === 0) {
122         this.stop();
123     }
124 }
125 }
126
127 export default SoundChannel2;

```

SoundChannel3 (src/emulator/apu/SoundChannel3.ts)

```
1 import { CircularRAM, Addressable } from "../Memory";
2 import { MaskRegister, Register } from "../Register";
3 import { Int2, Int4, rangeObject } from "../util";
4 import SoundChannel, { NRX4_RESTART_CHANNEL } from "./SoundChannel";
5
6 const NRX0_DAC_FLAG = 1 << 7;
7 const NRX2_OUTPUT_LEVEL = 0b0110_0000;
8
9 /** How much to shift the sound to the right */
10 const VOLUME_LEVELS = [
11     4, // = muted
12     0, // = full volume
13     1, // = 50%
14     2, // = 25%
15 ];
16
17 /**
18  * Sound channel 3 generates a wave that can be customised as needed.
19  * @link https://gbdev.io/pandocs/Audio_Registers.html#sound-channel-3--wave-output
20  */
21 class SoundChannel3 extends SoundChannel {
22     protected NRX1_LENGTH_TIMER_BITS: number = 0b1111_1111;
23
24     protected nrX0 = new MaskRegister(0b0111_1111);
25     protected nrX1 = new Register(0xbf);
26     protected nrX2 = new MaskRegister(0b1001_1111, 0xf3);
27     protected nrX3 = new Register(0xff);
28     protected nrX4 = new MaskRegister(0b0011_1000, 0xbf);
29     protected waveData = new CircularRAM(16, 0xff30);
30
31     protected addresses: Record<number, Addressable> = {
32         0xff1a: this.nrX0,
33         0xff1b: this.nrX1,
34         0xff1c: this.nrX2,
35         0xff1d: this.nrX3,
36         0xff1e: this.nrX4,
37         ...rangeObject(0xff30, 0xff3f, this.waveData),
38     };
39
40     // For output
41     protected ticksNextSample: number = 0;
42     protected waveStep: number = 0;
43     protected currentSample: Int4 = 0;
44
45     protected lastReadByte: number = 0xff;
46
47     protected override doTick() {
48         if (this.ticksNextSample-- <= 0) {
49             const frequency = (2048 - this.getWavelength()) >> 1;
50             this.ticksNextSample = frequency;
51
52             this.waveStep = (this.waveStep + 1) % 32;
53
54             const waveIndex = this.waveStep >> 1;
55             const waveByte = this.waveData.read(waveIndex);
56             const waveNibble = this.waveStep & 1 ? waveByte >> 4 : waveByte & 0b1111;
57             this.currentSample = waveNibble as Int4;
58             this.lastReadByte = waveByte;
59         } else {
60             this.lastReadByte = 0xff;
61         }
62     }
63
64     protected override getSample(): Int4 {
65         const outputLevel = ((this.nrX2.get() & NRX2_OUTPUT_LEVEL) >> 5) as Int2;
66         const volume = VOLUME_LEVELS[outputLevel];
67         return (this.currentSample >> volume) as Int4;
68     }
69 }
```



```

68     }
69
70     override onStart(): void {
71         const frequency = (2048 - this.getWavelength()) >> 1;
72         this.ticksNextSample = frequency;
73     }
74
75     get isDACOn(): boolean {
76         return this.nrX0.flag(NRX0_DAC_FLAG);
77     }
78
79     read(pos: number): number {
80         const component = this.addresses[pos];
81
82         // registers are write only
83         if (component === this.nrX1 || component === this.nrX3) return 0xff;
84         // only bit 6 is readable
85         if (component === this.nrX4) return this.nrX4.get() | 0b1011_1111;
86         // wave data is offset by 0xff30
87         if (component === this.waveData) {
88             // if (this.enabled) return this.lastReadByte;
89         }
90         return component.read(pos);
91     }
92
93     write(pos: number, data: number): void {
94         const component = this.addresses[pos];
95
96         if (component === this.nrX0) {
97             const oldDacState = this.nrX0.flag(NRX0_DAC_FLAG);
98             const newDacState = (data & NRX0_DAC_FLAG) === NRX0_DAC_FLAG;
99             if (oldDacState && !newDacState) {
100                 this.stop();
101             } else if (!oldDacState && newDacState) {
102                 this.start();
103             }
104         }
105         if (component === this.nrX4) {
106             if (data & NRX4_RESTART_CHANNEL) {
107                 this.stop();
108                 this.start();
109             }
110         }
111         component.write(pos, data);
112     }
113 }
114
115 export default SoundChannel3;

```

SoundChannel4 (src/emulator/apu/SoundChannel4.ts)

```
1 import { Addressable } from "../Memory";
2 import { MaskRegister, RegisterFF, Register } from "../Register";
3 import { clamp, Int4 } from "../util";
4 import SoundChannel, { FREQUENCY_ENVELOPE, NRX4_RESTART_CHANNEL } from "../SoundChannel";
5
6 const NRX2_STOP_DAC = 0b1111_1000;
7 const NRX3_CLOCK_SHIFT_OFFSET = 4;
8 const NRX3_LFSR_SHORT_MODE = 0b0000_1000;
9 const NRX3_CLOCK_DIVIDER = 0b0000_0111;
10
11 /**
12  * Sound channel 4 generates noise, that can be somewhat customised for softer/harsher noise.
13  * @link https://gbdev.io/pandocs/Audio\_Registers.html#sound-channel-4--noise
14  */
15 class SoundChannel4 extends SoundChannel {
16     protected NRX1_LENGTH_TIMER_BITS: number = 0b0011_1111;
17
18     protected nrX1 = new Register(0xff);
19     protected nrX2 = new Register(0x00);
20     protected nrX3 = new Register(0x00);
21     protected nrX4 = new MaskRegister(0b0011_1111, 0xbf);
22
23     protected addresses: Record<number, Addressable> = {
24         0xff1f: RegisterFF,
25         0xff20: this.nrX1,
26         0xff21: this.nrX2,
27         0xff22: this.nrX3,
28         0xff23: this.nrX4,
29     };
30
31     // NRx2 needs retriggering when changed
32     protected cachedNRX2: number = this.nrX2.get();
33
34     // Channel envelope volume
35     protected envelopeVolumeSteps: number = 0;
36     protected envelopeVolume: Int4 = 0;
37
38     // LFSR
39     protected ticksForLfsr: number = 0;
40     protected lfsr: number = 0x00;
41
42     protected override doTick() {
43         if (--this.ticksForLfsr <= 0) {
44             this.refreshLsfrTicks();
45             const lfsrBit = ~(this.lfsr ^ (this.lfsr >> 1)) & 1;
46             const shortMode = this.nrX3.flag(NRX3_LFSR_SHORT_MODE);
47             this.lfsr = (this.lfsr >> 1) | (lfsrBit << 15) | (shortMode ? 0 : lfsrBit << 7);
48         }
49     }
50
51     protected override tickEnvelope(): void {
52         if (this.envelopeVolumeSteps-- <= 0 && (this.cachedNRX2 & 0b111) !== 0) {
53             const direction = (this.cachedNRX2 & 0b0000_1000) === 0 ? -1 : 1;
54             this.envelopeVolume = clamp(this.envelopeVolume + direction, 0x0, 0xf) as Int4;
55             this.envelopeVolumeSteps = this.cachedNRX2 & 0b111;
56         }
57     }
58
59     protected refreshLsfrTicks(): void {
60         const clockDivider = this.nrX3.get() & NRX3_CLOCK_DIVIDER || 0.5; // 0 treated as 0.5
61         const clockShift = this.nrX3.get() >> NRX3_CLOCK_SHIFT_OFFSET;
62         this.ticksForLfsr = 4 * (clockDivider * (1 << clockShift));
63     }
64
65     protected override getSample(): Int4 {
66         return ((this.lfsr & 1) * this.envelopeVolume) as Int4;
67     }
68 }
```

```

68
69     get isDACOn(): boolean {
70         return (this.nrX2.get() & NRX2_STOP_DAC) !== 0;
71     }
72
73     override onStart(): void {
74         this.cachedNRX2 = this.nrX2.get();
75         this.envelopeVolume = (this.cachedNRX2 >> 4) as Int4;
76         this.lfsr = 0;
77         this.refreshLsfrTicks();
78     }
79
80     read(pos: number): number {
81         const component = this.addresses[pos];
82         // register is write only
83         if (component === this.nrX1) return 0xff;
84         // only bit 6 is readable
85         if (component === this.nrX4) return this.nrX4.get() | 0b1011_1111;
86         return component.read(pos);
87     }
88
89     write(pos: number, data: number): void {
90         const component = this.addresses[pos];
91
92         // Restart
93         if (component === this.nrX4 && (data & NRX4_RESTART_CHANNEL) !== 0) {
94             this.stop();
95             this.start();
96             data &= ~NRX4_RESTART_CHANNEL; // bit is write-only
97         }
98         // Turning off channel
99         if (component === this.nrX2 && (data & NRX2_STOP_DAC) === 0) {
100             this.stop();
101         }
102
103         component.write(pos, data);
104     }
105 }
106
107 export default SoundChannel4;

```

B.1.6 MBCs

MBC (src/emulator/mbc/MBC.ts)

```
1 import { Addressable, RAM, ROM } from "../Memory";
2
3 abstract class MBC implements Addressable {
4     static readonly RAM_SIZES: Partial<Record<number, number>> = {
5         0x00: 0,
6         0x01: 0, // this is unofficial, only used by homebrew roms
7         0x02: 1024 * 8,
8         0x03: 1024 * 32,
9         0x04: 1024 * 128,
10        0x05: 1024 * 64,
11    };
12
13    protected size: number;
14    protected rom: ROM;
15    protected ram: RAM | null = null;
16    protected hasSaves: boolean = false;
17
18    constructor(data: Uint8Array, hasSaves: boolean) {
19        this.size = data.length;
20        this.rom = new ROM(this.size, data);
21        this.hasSaves = hasSaves;
22    }
23
24    /**
25     * Creates a new RAM instance for this MBC. This must be called during construction. It reads
26     * the RAM size from the ROM header and creates a new RAM instance with that size.
27     * @link https://gbdev.io/pandocs/The_Cartridge_Header.html#0149--ram-size
28     */
29    protected createRAM(): RAM {
30        const ramSizeCode = this.rom.read(0x0149);
31        const ramSize = MBC.RAM_SIZES[ramSizeCode];
32        if (ramSize === undefined)
33            throw new Error(`Invalid RAM size header value: ${ramSizeCode.toString(16)}`);
34        return new RAM(ramSize);
35    }
36
37    /** Returns true if this ROM supports saves, false otherwise. */
38    supportsSaves(): boolean {
39        return this.hasSaves;
40    }
41
42    /** Returns this ROM's data if it supports saves, null otherwise. */
43    save(): Uint8Array | null {
44        return this.hasSaves && this.ram ? this.ram.rawData() : null;
45    }
46
47    /** Loads a save for this ROM. */
48    load(data: Uint8Array): void {
49        if (!this.hasSaves || !this.ram) return;
50
51        if (data.length !== this.ram.size)
52            throw new Error(
53                `[MBC] Save data is not the same size as the RAM! ` +
54                `Got ${data.length} bytes, expected ${this.ram.size} bytes.`
55            );
56
57        this.ram.rawSet(data);
58    }
59
60    abstract read(pos: number): number;
61    abstract write(pos: number, data: number): void;
62 }
63
64 export default MBC;
```

NoMBC (src/emulator/mbc/NoMBC.ts)

```
1 import MBC from "./MBC";
2
3 class NoMBC extends MBC {
4     constructor(data: Uint8Array) {
5         super(data, false);
6     }
7
8     read(pos: number): number {
9         if (0xa000 <= pos && pos <= 0xbfff) return 0xff; // eram
10        return this.rom.read(pos);
11    }
12
13    write(pos: number, data: number): void {
14        // nothing is writable by default
15    }
16 }
17
18 export default NoMBC;
```

MBC1 (src/emulator/mbc/MBC1.ts)

```

1 import { RAM } from "../Memory";
2 import { Register } from "../Register";
3 import MBC from "../MBC";
4
5 const RAM_ENABLED = 0x0a;
6
7 type MBC1Params = {
8     hasRam: boolean;
9     hasBattery: boolean;
10 };
11
12 /**
13  * Implementation of MBC1.
14  * @link https://gbdev.io/pandocs/MBC1.html
15  */
16 class MBC1 extends MBC {
17     /** @link https://gbdev.io/pandocs/MBC1.html#00001fff--ram-enable-write-only */
18     protected ramEnable = new Register(0x00);
19     /** @link https://gbdev.io/pandocs/MBC1.html#20003fff--rom-bank-number-write-only */
20     protected romBank = new Register(0x01);
21     /** @link https://gbdev.io/pandocs/MBC1.html#40005fff--ram-bank-number--or--upper-bits-of-rom-bank-number-write-only */
22     ↪
23     ↪
24     protected ramBank = new Register(0x00);
25     /** @link https://gbdev.io/pandocs/MBC1.html#60007fff--banking-mode-select-write-only */
26     protected bankingModeSelect = new Register(0x00);
27     /** The RAM contained in the ROM (ERAM). */
28     protected ram: RAM;
29     protected hasRam: boolean;
30
31     constructor(data: Uint8Array, { hasRam, hasBattery }: MBC1Params) {
32         super(data, hasBattery);
33         this.rom = this.createROM();
34         this.hasRam = hasRam && this.rom.size > 0;
35     }
36
37     /**
38      * Resolves a GameBoy address to an address in the ERAM. This uses the banking mode and
39      * the current RAM bank to determine the address.
40      */
41     protected resolveERAMAddress(pos: number): number {
42         const mode = this.bankingModeSelect.get() as 0 | 1;
43         const pos12bits = pos & ((1 << 13) - 1);
44         const ramAddressMask = this.rom.size - 1; // works for powers of 2
45         const address = pos12bits | (mode === 0 ? 0 : this.romBank.get() << 13);
46         return address & ramAddressMask;
47     }
48
49     /**
50      * Reads from the ROM, taking into account banking and the control ROMs.
51      * @link https://gbdev.io/pandocs/MBC1.html#addressing-diagrams
52      */
53     read(pos: number): number {
54         const mode = this.bankingModeSelect.get() as 0 | 1;
55         const addressMask = this.size - 1; // works for powers of 2
56         switch (pos >> 12) {
57             case 0x0:
58             case 0x1:
59             case 0x2:
60             case 0x3: {
61                 const address = mode === 0 ? pos : (this.romBank.get() << 19) | pos;
62                 return this.rom.read(address & addressMask);
63             }
64             case 0x4:
65             case 0x5:
66             case 0x6:
67             case 0x7: {

```

```

66         const address =
67             (pos & ((1 << 14) - 1)) |
68             (this.romBank.get() << 14) |
69             (this.ramBank.get() << 19);
70         return this.rom.read(address & addressMask);
71     }
72     case 0xa:
73     case 0xb: {
74         // RAM disabled
75         if (!this.hasRam || this.ramEnable.get() !== RAM_ENABLED) return 0xff;
76         const address = this.resolveERAMAddress(pos);
77         return this.ram.read(address);
78     }
79 }
80 throw new Error(`Invalid address to read from MBC1: ${pos.toString(16)}`);
81 }
82
83 write(pos: number, data: number): void {
84     switch (pos >> 12) {
85         case 0x0: // RAM enable
86         case 0x1:
87             return this.ramEnable.set(data & 0b1111); // 4 bit register
88
89         case 0x2: // ROM Bank Number
90         case 0x3:
91             const bits5 = data & 0b11111; // 5bit register
92             // Can't set ROM bank to 0
93             // In reality what happens is that a value of 0 is *interpreted* as 1. However
94             // simply overriding the write produces the same effect and is simpler.
95             return this.romBank.set(bits5 === 0 ? 1 : bits5);
96
97         case 0x4: // RAM Bank Number
98         case 0x5:
99             return this.ramBank.set(data & 0b11); // 2bit register
100
101         case 0x6: // Banking Mode Select
102         case 0x7:
103             return this.bankingModeSelect.set(data & 0b1); // 1bit register
104
105         case 0xa: // ERAM Write
106         case 0xb:
107             if (this.ramEnable.get() !== RAM_ENABLED) return; // RAM disabled
108
109             const address = this.resolveERAMAddress(pos);
110             return this.ram.write(address, data);
111     }
112
113     throw new Error(`Invalid address to write to MBC1: ${pos.toString(16)}`);
114 }
115 }
116
117 export default MBC1;

```

MBC2 (src/emulator/mbc/MBC2.ts)

```

1 import { RAM } from "../Memory";
2 import { Register } from "../Register";
3 import MBC from "../MBC";
4
5 const RAM_ENABLED = 0x0a;
6 const MBC2_ROM_BANK = 1 << 8;
7
8 type MBC2Params = {
9     hasBattery: boolean;
10 };
11
12 /**
13  * Implementation of MBC2.
14  * @link https://gbdev.io/pandocs/MBC2.html
15  */
16 class MBC2 extends MBC {
17     /** @link https://gbdev.io/pandocs/MBC2.html#00003fff--ram-enable-rom-bank-number-write-only
18     ↪ */
19     protected ramEnable = new Register(0x00);
20     /** @link https://gbdev.io/pandocs/MBC1.html#20003fff--rom-bank-number-write-only */
21     protected romBank = new Register(0x01);
22     /** The RAM contained in the ROM (ERAM). */
23     protected ram: RAM = new RAM(512);
24
25     constructor(data: Uint8Array, { hasBattery }: MBC2Params) {
26         super(data, hasBattery);
27     }
28
29     /**
30     * Reads from the ROM, taking into account banking and the control ROMs.
31     * @link https://gbdev.io/pandocs/MBC1.html#addressing-diagrams
32     */
33     read(pos: number): number {
34         switch (pos >> 12) {
35             case 0x0:
36             case 0x1:
37             case 0x2:
38             case 0x3: {
39                 return this.rom.read(pos);
40             }
41             case 0x4:
42             case 0x5:
43             case 0x6:
44             case 0x7: {
45                 const addressMask = this.size - 1; // works for powers of 2
46                 const address = (pos & ((1 << 14) - 1)) | (this.romBank.get() << 14);
47                 return this.rom.read(address & addressMask);
48             }
49             case 0xa:
50             case 0xb: {
51                 // RAM disabled
52                 if (this.ramEnable.get() !== RAM_ENABLED) return 0xff;
53                 return this.ram.read(pos & 0x1fff);
54             }
55         }
56         throw new Error(`Invalid address to read from MBC1: ${pos.toString(16)}`);
57     }
58
59     write(pos: number, data: number): void {
60         switch (pos >> 12) {
61             case 0x0: // RAM enable / ROM Bank Numbers
62             case 0x1:
63             case 0x2:
64             case 0x3:
65                 data = data & 0b1111;
66                 // bit 8 controls the register
67                 if (pos & MBC2_ROM_BANK) {

```



```

67          // Can't set ROM bank to 0
68          // In reality what happens is that a value of 0 is *interpreted* as 1.
↪ 69  However          // simply overriding the write produces the same effect and is simpler.
70          return this.romBank.set(data == 0 ? 1 : data);
71      } else {
72          return this.ramEnable.set(data);
73      }
74
75      case 0xa:
76      case 0xb: {
77          // RAM disabled
78          if (this.ramEnable.get() != RAM_ENABLED) return;
79          return this.ram.write(pos & 0x1ff, data | 0xf0);
80      }
81  }
82
83      return;
84  }
85 }
86
87 export default MBC2;

```

MBC3 (src/emulator/mbc/MBC3.ts)

```

1 import { RAM } from "../Memory";
2 import { Register } from "../Register";
3 import MBC from "../MBC";
4
5 const RAM_ENABLED = 0x0a;
6
7 type MBC3Params = {
8   hasRam: boolean;
9   hasTimer: boolean;
10  hasBattery: boolean;
11 };
12
13 /**
14  * Implementation of MBC3.
15  * @link https://gbdev.io/pandocs/MBC3.html
16  */
17 class MBC3 extends MBC {
18   /** @link https://gbdev.io/pandocs/MBC3.html#0000-1fff---ram-and-timer-enable-write-only */
19   protected ramEnable = new Register(0x00);
20   /** @link https://gbdev.io/pandocs/MBC3.html#2000-3fff---rom-bank-number-write-only */
21   protected romBank = new Register(0x01);
22   /** @link
23    ↪ https://gbdev.io/pandocs/MBC3.html#4000-5fff---ram-bank-number---or---rtc-register-select-write-only
24    ↪ */
25   protected ramBank = new Register(0x00);
26   /** The RAM contained in the ROM (ERAM). */
27   protected ram: RAM;
28
29   /**
30    * RTC registers
31    * @link https://gbdev.io/pandocs/MBC3.html#the-clock-counter-registers
32    */
33   protected rtcS = new Register(0x00);
34   protected rtcM = new Register(0x00);
35   protected rtcH = new Register(0x00);
36   protected rtcDL = new Register(0x00);
37   protected rtcDH = new Register(0x00);
38
39   protected rtcRegisters: Partial<Record<number, Register>> = {
40     0x08: this.rtcS,
41     0x09: this.rtcM,
42     0x0a: this.rtcH,
43     0x0b: this.rtcDL,
44     0x0c: this.rtcDH,
45   };
46
47   constructor(data: Uint8Array, { hasRam, hasTimer, hasBattery }: MBC3Params) {
48     super(data, hasBattery);
49     this.ram = this.createRAM();
50   }
51
52   /**
53    * Resolves a GameBoy address to an address in the ERAM. This uses the banking mode and
54    * the current RAM bank to determine the address.
55    */
56   protected resolveERAMAddress(pos: number): number {
57     const pos12bits = pos & ((1 << 13) - 1);
58     const ramAddressMask = this.ram.size - 1; // works for powers of 2
59     const address = pos12bits | (this.ramBank.get() << 13);
60     return address & ramAddressMask;
61   }
62
63   /**
64    * Reads from the ROM, taking into account banking and the control ROMs.
65    * @link https://gbdev.io/pandocs/MBC3.html#memory
66    */
67   read(pos: number): number {

```

```

66     const addressMask = this.size - 1; // works for powers of 2
67     switch (pos >> 12) {
68         case 0x0: // ROM Bank 00
69         case 0x1:
70         case 0x2:
71         case 0x3:
72             return this.rom.read(pos & addressMask);
73
74         case 0x4: // ROM Bank 1-7
75         case 0x5:
76         case 0x6:
77         case 0x7:
78             const address = (pos & ((1 << 14) - 1)) | (this.romBank.get() << 14);
79             return this.rom.read(address & addressMask);
80
81         case 0xa: // ERAM
82         case 0xb:
83             // TODO: check for RTC register ?
84             // RAM disabled
85             if (this.ramEnable.get() !== RAM_ENABLED) return 0xff;
86
87             const ramBank = this.ramBank.get();
88             if (ramBank > 0x03) {
89                 return this.rtcRegisters[ramBank]!.get();
90             }
91
92             const eramAddress = this.resolveERAMAddress(pos);
93             return this.ram.read(eramAddress);
94     }
95     throw new Error(`Invalid address to read from MBC3: ${pos.toString(16)}`);
96 }
97
98 write(pos: number, data: number): void {
99     switch (pos >> 12) {
100         case 0x0: // RAM and timer enable
101         case 0x1:
102             return this.ramEnable.set(data & 0b1111); // 4 bit register
103
104         case 0x2: // ROM Bank Number
105         case 0x3:
106             const bits = data & 0b0111_1111; // 7bit register
107             // Can't set ROM bank to 0
108             // In reality what happens is that a value of 0 is *interpreted* as 1. However
109             // simply overriding the write produces the same effect and is simpler.
110             return this.romBank.set(bits === 0 ? 1 : bits);
111
112         case 0x4: // RAM Bank Number or RTC Register Select
113         case 0x5:
114             if (data > 0x0c) {
115                 console.error(
116                     `Invalid RAM bank number in MBC3 (write ignored): ${data.toString(16)}`
117                 );
118                 return;
119             }
120             return this.ramBank.set(data);
121
122         case 0x6: // Latch clock data
123         case 0x7:
124             // TODO: check if writes 0, then 1, then latch
125             return;
126
127         case 0xa: // ERAM Write
128         case 0xb:
129             if (this.ramEnable.get() !== RAM_ENABLED) return; // RAM disabled
130             if (this.ramBank.get() > 0x03) return; // TODO: RTC registers
131             const address = this.resolveERAMAddress(pos);
132             return this.ram.write(address, data);
133     }
134 }

```

```
135         throw new Error(`Invalid address to write to MBC3: ${pos.toString(16)}`);
136     }
137 }
138
139 export default MBC3;
```

MBC5 (src/emulator/mbc/MBC5.ts)

```

1 import { RAM } from "../Memory";
2 import { Register } from "../Register";
3 import MBC from "../MBC";
4
5 const RAM_ENABLED = 0x0a;
6
7 type MBC5Params = {
8   hasRam: boolean;
9   hasRumble: boolean;
10  hasBattery: boolean;
11 };
12
13 /**
14  * Implementation of MBC5.
15  * @link https://gbdev.io/pandocs/MBC5.html
16  */
17 class MBC5 extends MBC {
18   /** @link https://gbdev.io/pandocs/MBC5.html#0000-1fff---ram-enable-write-only */
19   protected ramEnable = new Register(0x00);
20   /** @link
21    ↪ https://gbdev.io/pandocs/MBC5.html#2000-2fff---8-least-significant-bits-of-rom-bank-number-write-only
22    ↪ */
23   protected romBankLower8 = new Register(0x01);
24   /** @link https://gbdev.io/pandocs/MBC5.html#3000-3fff---9th-bit-of-rom-bank-number-write-only
25    ↪ */
26   protected romBankUpper1 = new Register(0x00);
27   /** @link https://gbdev.io/pandocs/MBC5.html#4000-5fff---ram-bank-number-write-only */
28   protected ramBank = new Register(0x00);
29   /** The RAM contained in the ROM (ERAM). */
30   protected ram: RAM;
31
32   constructor(data: Uint8Array, { hasRam, hasRumble, hasBattery }: MBC5Params) {
33     super(data, hasBattery);
34     this.romBankUpper1 = this.romBankUpper1;
35     this.romBankLower8 = this.romBankLower8;
36     this.romBank = this.romBank;
37     this.rom = this.rom;
38   }
39
40   /**
41    * Resolves a GameBoy address to an address in the ERAM. This uses the current RAM bank to
42    * determine the address.
43    */
44   protected resolveERAMAddress(pos: number): number {
45     const pos12bits = pos & ((1 << 13) - 1);
46     const ramAddressMask = this.romBank.get() << 13;
47     const address = pos12bits | (this.romBank.get() << 13);
48     return address & ramAddressMask;
49   }
50
51   /**
52    * Reads from the ROM, taking into account banking and the control ROMs.
53    * @link https://gbdev.io/pandocs/MBC5.html#memory
54    */
55   read(pos: number): number {
56     const addressMask = this.size - 1; // works for powers of 2
57     switch (pos >> 12) {
58       case 0x0: // ROM bank 00
59       case 0x1:
60       case 0x2:
61       case 0x3: {
62         return this.rom.read(pos & addressMask);
63       }
64       case 0x4:
65       case 0x5:
66       case 0x6:
67       case 0x7: {
68         const address =
69           (pos & ((1 << 14) - 1)) |
70           (this.romBankLower8.get() << 14) |

```

```

65         (this.romBankUpper1.get() << 22);
66         return this.rom.read(address & addressMask);
67     }
68     case 0xa: // ERAM
69     case 0xb: {
70         if (this.ramEnable.get() !== RAM_ENABLED) return 0xff;
71         const address = this.resolveERAMAddress(pos);
72         return this.ram.read(address);
73     }
74 }
75
76     throw new Error(`Invalid address to read from MBC5: ${pos.toString(16)}`);
77 }
78
79 write(pos: number, data: number): void {
80     switch (pos >> 12) {
81         case 0x0: // RAM enable
82         case 0x1:
83             return this.ramEnable.set(data & 0b1111); // 4 bit register
84
85         case 0x2: // ROM Bank Number (lower 8 bits)
86             return this.romBankLower8.set(data);
87
88         case 0x3: // ROM Bank Number (upper 1 bit)
89             return this.romBankUpper1.set(data & 0b1); // 1 bit register
90
91         case 0x4: // RAM Bank Number
92         case 0x5:
93             return this.ramBank.set(data & 0b11); // 2bit register
94
95         case 0x6: // Nothing here
96         case 0x7:
97             return;
98
99         case 0xa: // ERAM Write
100        case 0xb:
101            if (this.ramEnable.get() !== RAM_ENABLED) return; // RAM disabled
102            const address = this.resolveERAMAddress(pos);
103            return this.ram.write(address, data);
104    }
105    throw new Error(`Invalid address to write to MBC5: ${pos.toString(16)}`);
106 }
107 }
108
109 export default MBC5;

```

B.1.7 Other Components

GameCartridge (src/emulator/GameCartridge.ts)

```
1 import MBC from "../mbc/MBC";
2 import MBC1 from "../mbc/MBC1";
3 import MBC2 from "../mbc/MBC2";
4 import MBC3 from "../mbc/MBC3";
5 import MBC5 from "../mbc/MBC5";
6 import NoMBC from "../mbc/NoMBC";
7 import { Addressable } from "../Memory";
8
9 const TITLE_START = 0x134;
10 const TITLE_END = 0x143;
11
12 const IDENTIFIER_START = 0x134;
13 const IDENTIFIER_END = 0x14f;
14
15 const CARTRIDGE_TYPE = 0x147;
16
17 /**
18  * The game cartridge of the game boy. This class is a wrapper around the
19  * different Memory Bank Controllers (MBCs), and is responsible for choosing the right MBC and
20  * parsing some of the meta data from the cartridge header.
21  * @link https://gbdev.io/pandocs/MBCs.html
22  * @link https://gbdev.io/pandocs/The_Cartridge_Header.html
23  */
24 class GameCartridge implements Addressable {
25     /** The MBC to handle addressing etc. */
26     protected mbc: MBC;
27
28     /** The ROM's title (included from 0x134 to 0x143), in ASCII uppercase. */
29     protected title: string;
30
31     /**
32      * The identifier includes the title and extra header data, to properly identify the ROM.
33      * The title can't be used alone, as multiple ROMs can have the same title.
34      */
35     protected identifier: string;
36
37     constructor(data: Uint8Array) {
38         this.title = [...new Array(TITLE_END - TITLE_START)]
39             .map((_, i) => String.fromCharCode(data[TITLE_START + i]))
40             .reduce((prev, x) => prev + x, "")
41             .replaceAll("\u0000", "");
42
43         this.identifier = [...new Array(IDENTIFIER_END - IDENTIFIER_START)]
44             .map((_, i) => String.fromCharCode(data[IDENTIFIER_START + i]))
45             .reduce((prev, x) => prev + x, "");
46
47         const mbcType = data[CARTRIDGE_TYPE];
48         const mbcInstance = {
49             // No MBC
50             0x00: () => new NoMBC(data),
51             // MBC1
52             0x01: () => new MBC1(data, { hasRam: false, hasBattery: false }),
53             0x02: () => new MBC1(data, { hasRam: true, hasBattery: false }),
54             0x03: () => new MBC1(data, { hasRam: true, hasBattery: true }),
55             // MBC2
56             0x05: () => new MBC2(data, { hasBattery: false }),
57             0x06: () => new MBC2(data, { hasBattery: true }),
58             // MBC3
59             0x0f: () => new MBC3(data, { hasTimer: true, hasRam: false, hasBattery: true }),
60             0x10: () => new MBC3(data, { hasTimer: true, hasRam: true, hasBattery: true }),
61             0x11: () => new MBC3(data, { hasTimer: false, hasRam: false, hasBattery: false }),
62             0x12: () => new MBC3(data, { hasTimer: false, hasRam: true, hasBattery: false }),
63             0x13: () => new MBC3(data, { hasTimer: false, hasRam: true, hasBattery: true }),
64             // MBC5
```

```

65         0x19: () => new MBC5(data, { hasRam: false, hasRumble: false, hasBattery: false }),
66         0x1a: () => new MBC5(data, { hasRam: true, hasRumble: false, hasBattery: false }),
67         0x1b: () => new MBC5(data, { hasRam: true, hasRumble: false, hasBattery: true }),
68         0x1c: () => new MBC5(data, { hasRam: false, hasRumble: true, hasBattery: false }),
69         0x1d: () => new MBC5(data, { hasRam: true, hasRumble: true, hasBattery: false }),
70         0x1e: () => new MBC5(data, { hasRam: true, hasRumble: true, hasBattery: true }),
71     }[mbcType];
72     if (mbcInstance === undefined)
73         throw new Error(`Invalid cartridge type: ${mbcType.toString(16)}`);
74     this.mbc = mbcInstance();
75     console.debug(`Saved data for game "${this.title}": `, data);
76 }
77
78 write(pos: number, data: number): void {
79     this.mbc.write(pos, data);
80 }
81
82 read(pos: number): number {
83     return this.mbc.read(pos);
84 }
85
86 getTitle(): string {
87     return this.title;
88 }
89
90 getIdentifier(): string {
91     return this.identifier;
92 }
93
94 supportsSaves(): boolean {
95     return this.mbc.supportsSaves();
96 }
97
98 save(): Uint8Array | null {
99     return this.mbc.save();
100 }
101
102 load(data: Uint8Array): void {
103     this.mbc.load(data);
104 }
105 }
106
107 export default GameCartridge;

```


JoypadInput (src/emulator/JoypadInput.ts)

```
1 import GameBoyInput from "../GameBoyInput";
2 import { Addressable } from "../Memory";
3
4 // Inputs
5 const BUTTON_A = 1 << 0;
6 const BUTTON_B = 1 << 1;
7 const BUTTON_SELECT = 1 << 2;
8 const BUTTON_START = 1 << 3;
9 const ARROW_RIGHT = 1 << 0;
10 const ARROW_LEFT = 1 << 1;
11 const ARROW_UP = 1 << 2;
12 const ARROW_DOWN = 1 << 3;
13
14 const READ_ARROWS_BIT = 1 << 4;
15 const READ_BUTTON_BIT = 1 << 5;
16 const CONTROL_BITS = READ_ARROWS_BIT & READ_BUTTON_BIT;
17
18 enum JoypadMode {
19     BUTTONS,
20     ARROWS,
21 }
22
23 /**
24  * The joypad input, that takes care of receiving inputs for the buttons and directional arrows.
25  * @see https://gbdev.io/pandocs/Joypad\_Input.html
26  */
27 class JoypadInput implements Addressable {
28     protected input: GameBoyInput;
29
30     // bits 0-3 are state (button or arrow)
31     // bit 4 is to read arrow data
32     // bit 5 is to read button data
33     protected register: number = 0;
34
35     protected buttonData: number = 0;
36     protected arrowsData: number = 0;
37
38     protected currentlyReading: JoypadMode = JoypadMode.BUTTONS;
39
40     constructor(input: GameBoyInput) {
41         this.input = input;
42     }
43
44     readInput(): void {
45         const data = this.input.read();
46         this.buttonData =
47             (data.a ? 0 : BUTTON_A) |
48             (data.b ? 0 : BUTTON_B) |
49             (data.start ? 0 : BUTTON_START) |
50             (data.select ? 0 : BUTTON_SELECT);
51         this.arrowsData =
52             (data.up ? 0 : ARROW_UP) |
53             (data.down ? 0 : ARROW_DOWN) |
54             (data.left ? 0 : ARROW_LEFT) |
55             (data.right ? 0 : ARROW_RIGHT);
56     }
57
58     read(): number {
59         const data =
60             this.currentlyReading === JoypadMode.BUTTONS ? this.buttonData : this.arrowsData;
61         return 0b100_0000 | (CONTROL_BITS & this.register) | (~CONTROL_BITS & data);
62     }
63
64     write(_ : number, data: number): void {
65         this.register = data & 0b0011_0000;
66
67         // the switch is done when the bit moves to a LOW state.
```

```
68         if ((this.register & READ_ARROWS_BIT) === 0) this.currentlyReading = JoypadMode.ARROWS;
69         if ((this.register & READ_BUTTON_BIT) === 0) this.currentlyReading = JoypadMode.BUTTONS;
70     }
71 }
72
73 export default JoypadInput;
```

Timer (src/emulator/Timer.ts)

```
1 import { IFLAG_TIMER } from "./constants";
2 import Interrupts from "./Interrupts";
3 import { Addressable } from "./Memory";
4 import { MaskRegister, DoubleRegister, Register } from "./Register";
5 import { Int2, wrap16 } from "./util";
6
7 /**
8  * Represents the division applied to the clock speed depending on the value of the
9  * first two bits of the TAC.
10  * @link https://gbdev.io/pandocs/Timer\_and\_Divider\_Registers.html#ff07--tac-timer-control
11  */
12 const TIMER_CONTROLS = [9, 3, 5, 7];
13
14 /**
15  * The TIMA counter only runs if this flag is true in the TAC.
16  */
17 const TIMER_ENABLE_FLAG = 1 << 2;
18
19 class Timer implements Addressable {
20     // DIV - divider register
21     protected divider = new DoubleRegister(0xab00);
22     // TIMA - timer counter
23     protected timerCounter = new Register();
24     // TMA - timer modulo
25     protected timerModulo = new Register();
26     // TAC - timer control
27     protected timerControl = new MaskRegister(0b1111_1000);
28
29     protected previousBitState: number = 0;
30     protected timerOverflowed: boolean = false;
31     protected previousTimerOverflowed: boolean = false;
32
33     /**
34      * Ticks the timer system in M-cycles
35      * @link https://gbdev.io/pandocs/Timer\_and\_Divider\_Registers.html
36      */
37     tick(interrupts: Interrupts) {
38         // Increase internal counter, update DIV
39         const newDivider = wrap16(this.divider.get() + 4);
40         this.divider.set(newDivider);
41
42         // Check overflow + interrupt
43         this.previousTimerOverflowed = false;
44         if (this.timerOverflowed) {
45             const modulo = this.timerModulo.get();
46             this.timerCounter.set(modulo);
47             interrupts.requestInterrupt(IFLAG_TIMER);
48             this.timerOverflowed = false;
49             this.previousTimerOverflowed = true;
50         }
51
52         // Increase TIMA
53         // Store bit for TIMA edge-detection
54         const timerControl = this.timerControl.get();
55         const timerIsEnabled = timerControl & TIMER_ENABLE_FLAG;
56         const speedMode = (timerControl & 0b11) as Int2;
57         const checkedBit = TIMER_CONTROLS[speedMode];
58
59         // Bit is read if enabled, otherwise it's 0
60         const currentBitState = timerIsEnabled && (newDivider >> checkedBit) & 1;
61
62         // Timer increases when *read bit* goes from 1 to 0
63         // This meaning turning off the timer when the bit was 1 triggers an increase
64         if (this.previousBitState && !currentBitState) {
65             const result = (this.timerCounter.get() + 1) & 0xff;
66             this.timerCounter.set(result);
67         }
68     }
69 }
```

```

68         // overflow, need to warn for reset + interrupt
69         if (result === 0) {
70             this.timerOverflowed = true;
71         }
72     }
73
74     this.previousBitState = currentBitState;
75 }
76
77 protected addresses: Record<number, Register> = {
78     0xff04: this.divider.h, // we only ever read the upper 8 bits of the divider
79     0xff05: this.timerCounter,
80     0xff06: this.timerModulo,
81     0xff07: this.timerControl,
82 };
83
84 read(pos: number): number {
85     return this.addresses[pos].get();
86 }
87
88 write(pos: number, data: number): void {
89     // Trying to write anything to DIV clears it.
90     if (pos === 0xff04) {
91         this.divider.set(0);
92         return;
93     }
94
95     const register = this.addresses[pos];
96
97     if (register === this.timerCounter) {
98         // If overflow (reload) occurred, writes are ignored
99         if (this.previousTimerOverflowed) return;
100         // Otherwise it negates the overflow flag
101         this.timerOverflowed = false;
102     }
103     // If an overflow (reload) just happened, we update the value to the new modulo
104     else if (register === this.timerModulo && this.previousTimerOverflowed) {
105         this.timerCounter.set(data);
106     }
107
108     register.set(data);
109 }
110 }
111
112 export default Timer;

```

Interrupts (src/emulator/Interrupts.ts)

```

1 import { IFLAG_JOYPAD, IFLAG_LCDC, IFLAG_SERIAL, IFLAG_TIMER, IFLAG_VBLANK } from "./constants";
2 import { Addressable } from "./Memory";
3 import { MaskRegister, Register } from "./Register";
4
5 /** The state of the IME. */
6 enum IMEType {
7     DISABLED,
8     WILL_ENABLE,
9     WILL_ENABLE2,
10    ENABLED,
11 }
12
13 /**
14  * Transition for the IME. We need two intermediate, because the system ticks right after the CPU,
15  * ↪ so if we go
16  * straight from WILL_ENABLE to ENABLED the CPU will never tick during a non-enabled state.
17  */
18 const IMETransitions: Record<IMEType, IMEType> = {
19     [IMEType.DISABLED]: IMEType.DISABLED,
20     [IMEType.WILL_ENABLE]: IMEType.WILL_ENABLE2,
21     [IMEType.WILL_ENABLE2]: IMEType.ENABLED,
22     [IMEType.ENABLED]: IMEType.ENABLED,
23 };
24
25 const IFLAGS = [IFLAG_VBLANK, IFLAG_LCDC, IFLAG_TIMER, IFLAG_SERIAL, IFLAG_JOYPAD];
26
27 /**
28  * Mapping of interrupt flags to their corresponding interrupt handler address.
29  */
30 const INTERRUPT_CALLS: Record<number, number> = {
31     [IFLAG_VBLANK]: 0x0040,
32     [IFLAG_LCDC]: 0x0048,
33     [IFLAG_TIMER]: 0x0050,
34     [IFLAG_SERIAL]: 0x0058,
35     [IFLAG_JOYPAD]: 0x0060,
36 };
37
38 class Interrupts implements Addressable {
39     // Interrupts
40     protected intMasterEnable: IMEType = IMEType.DISABLED; // IME - master enable flag
41     protected intEnable = new Register(0x00); // IE - interrupt enable (handler)
42     protected intFlag = new MaskRegister(0b1110_0000, 0xe1); // IF - interrupt flag (requests)
43
44     protected addresses: Record<number, Register> = {
45         0xffff: this.intEnable,
46         0xff0f: this.intFlag,
47     };
48
49     tick(): void {
50         // Tick IME
51         this.intMasterEnable = IMETransitions[this.intMasterEnable];
52     }
53
54     read(address: number): number {
55         return this.addresses[address].get();
56     }
57
58     write(address: number, value: number): void {
59         this.addresses[address].set(value);
60     }
61
62     get interruptsEnabled(): boolean {
63         return this.intMasterEnable === IMEType.ENABLED;
64     }
65
66     /** Enables the master interrupt toggle. */
67     enableInterrupts() {

```

```

67         if (this.intMasterEnable === IMType.DISABLED)
68             this.intMasterEnable = IMType.WILL_ENABLE;
69     }
70     /** Disables the master interrupt toggle. */
71     disableInterrupts() {
72         this.intMasterEnable = IMType.DISABLED;
73     }
74
75     /**
76      * Forces the transition to IME = enabled (needed when halting).
77      * Returns the state of the IME: enabled (true) or disabled (false)
78      */
79     fastEnableInterrupts(): boolean {
80         // forces transition
81         if (this.intMasterEnable !== IMType.DISABLED) {
82             this.intMasterEnable = IMType.ENABLED;
83         }
84         return this.intMasterEnable === IMType.ENABLED;
85     }
86
87     /** Requests an interrupt for the given flag type. */
88     requestInterrupt(flag: number) {
89         this.intFlag.sflag(flag, true);
90     }
91
92     /** Returns whether there are any interrupts to handle. (IE & IF) */
93     get hasPendingInterrupt(): boolean {
94         return !(this.intEnable.get() & this.intFlag.get() & 0b11111);
95     }
96
97     /**
98      * Returns the address for the current interrupt handler. This also disables interrupts, and
99      * clears the interrupt flag.
100     */
101    handleNextInterrupt(): number {
102        for (const flag of IFLAGS) {
103            if (this.intEnable.flag(flag) && this.intFlag.flag(flag)) {
104                this.intFlag.sflag(flag, false);
105                this.disableInterrupts();
106                return INTERRUPT_CALLS[flag];
107            }
108        }
109        throw new Error("Cleared interrupt but nothing was called");
110    }
111 }
112
113 export default Interrupts;

```

WRAM (src/emulator/WRAM.ts)

```
1 import { Addressable, CircularRAM } from "./Memory";
2 import { MaskRegister } from "./Register";
3
4 const WRAM_SIZE = 0x2000;
5 const WRAM_BANK_INDEX = 0b111;
6 const WRAM_BANK_SIZE = 0x1000;
7
8 class DMGWRAM extends CircularRAM {
9     constructor() {
10         super(WRAM_SIZE, 0xc000);
11     }
12 }
13
14 class GBCWRAM implements Addressable {
15     protected bank0 = new CircularRAM(WRAM_BANK_SIZE, 0x0000);
16     // For convenience, we create the 7 banks and map them to the 8 possible bank indices, since
17     // a bank index of 0 is mapped to bank 1.
18     protected banks1To7 = (() => {
19         const banks = [...new Array(7)].map(() => new CircularRAM(WRAM_BANK_SIZE, 0x1000));
20         return [banks[0], banks[0], banks[1], banks[2], banks[3], banks[4], banks[5], banks[6]];
21     })();
22     protected wramBank = new MaskRegister(0b1111_1000);
23
24     protected address(address: number): Addressable {
25         if (address === 0xff70) return this.wramBank;
26         // we need to do modulo, since in reality WRAM is a circular buffer
27         return (address - 0xc000) % WRAM_SIZE < WRAM_BANK_SIZE
28             ? this.bank0
29             : this.banks1To7[this.wramBank.get() & WRAM_BANK_INDEX];
30     }
31
32     read(address: number): number {
33         return this.address(address).read(address);
34     }
35
36     write(address: number, value: number): void {
37         this.address(address).write(address, value);
38     }
39 }
40
41 export { DMGWRAM, GBCWRAM };
```

B.1.8 Helpers

Memory (src/emulator/Memory.ts)

```
1  /**
2   * A basic interface for all addressable objects.
3   */
4  interface Addressable {
5      read(pos: number): number;
6      write(pos: number, data: number): void;
7  }
8
9  /**
10   * Simple read only memory object.
11   */
12  class ROM implements Addressable {
13      size: number;
14      protected data: Uint8Array;
15
16      constructor(size: number, data?: Uint8Array) {
17          this.size = size;
18          this.data = data ?? new Uint8Array(size);
19      }
20
21      read(pos: number) {
22          return this.data[pos];
23      }
24
25      write(pos: number, data: number): void {
26          throw new Error("Can't write to ROM.");
27      }
28
29      /**
30       * Returns the raw data of this memory. Shouldn't be used for regular access - useful for
31       * backups and save states.
32       */
33      rawData(): Uint8Array {
34          return this.data;
35      }
36
37      /**
38       * Sets the raw data of this memory. Shouldn't be used for regular access - useful for
39 ↪ backups
40       * and save states.
41       */
42      rawSet(data: Uint8Array): void {
43          this.data.set(data);
44      }
45
46      /**
47       * Live memory, that can be read from and written to.
48       */
49      class RAM extends ROM {
50          write(pos: number, data: number) {
51              this.data[pos] = data;
52          }
53      }
54
55      /**
56       * Circular RAM is similar to RAM, but it also stores an offset. Any access on the memory (both
57       * reads and writes) occur at the position  $(X - 0) \% S$ , where  $S$  is the size of the memory,  $0$  the
58       * offset, and  $X$  the accessed address. This means out of bound exceptions cannot happen.
59       */
60      class CircularRAM extends RAM {
61          protected offset: number;
62
63          constructor(size: number, offset: number, data?: Uint8Array) {
```



```

64         super(size, data);
65         this.offset = offset;
66     }
67
68     override read(pos: number): number {
69         return super.read((pos - this.offset) % this.size);
70     }
71
72     override write(pos: number, data: number): void {
73         super.write((pos - this.offset) % this.size, data);
74     }
75 }
76
77 export type { Addressable };
78 export { ROM, RAM, CircularRAM };

```

Register (src/emulator/Register.ts)

```
1 import { Addressable } from "../Memory";
2 import { wrap16 } from "../util";
3
4 /**
5  * A register, containing an 8bit value.
6  * For convenience's sake, a register implements `Addressable`. Calling `read` will simply
7  * return the value (ignoring the position), and calling `write` will simply set the value,
8  * ignoring the position too.
9  */
10 class Register implements Addressable {
11     protected value: number;
12
13     /** Initialises a register with either 0 or a value */
14     constructor(value: number = 0) {
15         this.value = value;
16     }
17
18     /** Sets this register byte to the given value */
19     set(value: number) {
20         this.value = value;
21     }
22     /** The value of this register byte */
23     get() {
24         return this.value;
25     }
26     /** Sets the given flag to 0/1 according to the boolean */
27     sflag(flag: number, state: boolean) {
28         this.set(state ? this.value | flag : this.value & ~flag);
29     }
30     /** Returns if the given flag is set */
31     flag(flag: number) {
32         return (this.get() & flag) === flag;
33     }
34
35     /** Read this register. */
36     read(): number {
37         return this.get();
38     }
39     /** Write to this register. Position is ignored. */
40     write(_: number, data: number): void {
41         this.set(data);
42     }
43 }
44
45 /**
46  * A MaskRegister is similar to a Register but it only uses a set given of bits. All
47  * other bits are hard-wired to 1, and can't be changed.
48  */
49 class MaskRegister extends Register {
50     protected mask: number;
51
52     constructor(mask: number, value: number = 0) {
53         mask &= 0xff;
54         super(value | mask);
55         this.mask = mask;
56     }
57
58     override set(value: number): void {
59         super.set(value | this.mask);
60     }
61 }
62
63 /**
64  * A double register, containing two 8 bit values.
65  */
66 class DoubleRegister {
67     // Most significant byte (0xFF00)
```

```

68     public h = new Register();
69     // Least significant byte (0x00FF)
70     public l = new Register();
71
72     constructor(value: number = 0) {
73         this.set(value);
74     }
75
76     /** Sets this register to the given 16bit value. */
77     set(value: number) {
78         this.h.set((value >> 8) & 0xff);
79         this.l.set(value & 0xff);
80     }
81
82     /** Returns the 16bit value in this register */
83     get() {
84         return (this.h.get() << 8) | this.l.get();
85     }
86
87     /** Increments this register's value and returns the previous value (equivalent to r++) */
88     inc() {
89         const temp = this.get();
90         this.set(wrap16(temp + 1));
91         return temp;
92     }
93
94     /** Decrements this register's value and returns the previous value (equivalent to r--) */
95     dec() {
96         const temp = this.get();
97         this.set(wrap16(temp - 1));
98         return temp;
99     }
100 }
101
102 const Register00: Addressable = { read: () => 0x00, write: () => {} };
103 const RegisterFF: Addressable = { read: () => 0xff, write: () => {} };
104
105 export { DoubleRegister, Register, MaskRegister, Register00, RegisterFF };

```

constants (src/emulator/constants.ts)

```
1 // CPU
2 export const CLOCK_SPEED = 4194304; // 2~22Hz
3 export const FRAME_RATE = 60;
4 export const CYCLES_PER_FRAME = Math.floor(CLOCK_SPEED / FRAME_RATE);
5
6 // Screen
7 export const SCREEN_WIDTH = 160;
8 export const SCREEN_HEIGHT = 144;
9
10 // Memory
11 export const HRAM_SIZE = 352;
12
13 // Interrupt flags
14 export const IFLAG_VBLANK = 1 << 0;
15 export const IFLAG_LCDC = 1 << 1;
16 export const IFLAG_TIMER = 1 << 2;
17 export const IFLAG_SERIAL = 1 << 3;
18 export const IFLAG_JOYPAD = 1 << 4;
19
20 // Types
21 export enum ConsoleType {
22     DMG,
23     CGB,
24 }
25 export enum SpeedMode {
26     Normal,
27     Double,
28 }
29 export enum CGBMode {
30     CGB,
31     DMG,
32     DMGExtended,
33 }
```

util (src/emulator/util.ts)

```
1 export const high = (x: number) => (x >> 8) & 0xff;
2 export const low = (x: number) => x & 0xff;
3 export const combine = (high: number, low: number) => (high << 8) | low;
4 export const wrap8 = (x: number) => x & 0xff;
5 export const wrap16 = (x: number) => x & 0xffff;
6 export const asSignedInt8 = (x: number) => (x > 127 ? x - 256 : x);
7 export const clamp = (x: number, min: number, max: number) => Math.min(Math.max(x, min), max);
8 export const range = (from: number, to: number) =>
9     [...new Array(to - from + 1)].map((_, i) => i + from);
10 export const rangeObject = <T>(from: number, to: number, obj: T) =>
11     Object.fromEntries(range(from, to).map((i) => [i, obj]));
12
13 export type Int1 = 0 | 1;
14 export type Int2 = 0 | 1 | 2 | 3;
15 export type Int3 = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7;
16 export type Int4 = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15;
```

B.2 Frontend Code

B.2.1 Main App

```
main (src/frontend/main.tsx)

1 import { render } from "preact";
2
3 import App from "@frontend/app";
4 import { ConfigProvider } from "@helpers/ConfigContext";
5
6 import "@frontend/index.css";
7 import "@frontend/mobile.css";
8
9 render(
10   <ConfigProvider>
11     <App />
12   </ConfigProvider>,
13   document.getElementById("app") as HTMLElement
14 );
```

app (src/frontend/app.tsx)

```
1 import { useSignal } from "@preact/signals";
2 import localforage from "localforage";
3 import { FastForward, Pause, Play, Redo, Save, Volume2, VolumeX } from "lucide-preact";
4 import { FunctionalComponent } from "preact";
5 import { useCallback, useEffect, useRef, useState } from "preact/hooks";
6
7 import AudioPlayer from "@helpers/AudioPlayer";
8 import { useConfig } from "@helpers/ConfigContext";
9
10 import { addAlert, AlertManager } from "@components/Alerts";
11 import Drawer from "@components/Drawer/Drawer";
12 import GameInput from "@components/GameInput";
13 import IconButton from "@components/IconButton";
14 import RomInput from "@components/RomInput";
15 import Screen, { VideoReceiver } from "@components/Screen";
16
17 import GameBoyColor from "@emulator/GameBoyColor";
18 import GameBoyInput from "@emulator/GameBoyInput";
19 import GameBoyOutput from "@emulator/GameBoyOutput";
20
21 const CACHE_KEY = "rom";
22 const SAVE_CACHE_KEY = "save_";
23
24 const App: FunctionalComponent = () => {
25   // Interaction
26   const tripleSpeed = useSignal(false);
27   const emulatorRunning = useSignal(true);
28   const canStep = useSignal(true);
29   const [config, setConfig] = useConfig();
30   const configPtr = useSignal(config);
31   useEffect(() => {
32     configPtr.value = config;
33   }, [config]);
34   const soundOutput = useSignal<AudioPlayer | undefined>(undefined);
35   const joystickInput = useSignal<undefined | GameBoyInput>(undefined);
36
37   // DOM Refs
38   const emulatorFrameIn = useRef<VideoReceiver | undefined>(undefined);
39   const bgDebugger = useRef<VideoReceiver | undefined>(undefined);
40   const tilesetDebugger = useRef<VideoReceiver | undefined>(undefined);
41
42   // Emulator data
43   const [loadedGame, setLoadedGame] = useState<Uint8Array>();
44   const [gameboy, setGameboy] = useState<GameBoyColor>();
45
46   const effectivePalette = gameboy?.getMode() === "DMG" ? config.gbPalette : undefined;
47
48   // Debug state
49   const stepCount = useRef<HTMLDivElement>(null);
50   const millisPerFrame = useRef<HTMLDivElement>(null);
51
52   const toggleHasSound = () => {
53     const audioEnabled = !config.audioEnabled;
54     setConfig({ audioEnabled });
55     if (audioEnabled) {
56       soundOutput.value = new AudioPlayer({
57         get value() {
58           return configPtr.value.volume;
59         },
60       });
61     } else {
62       soundOutput.value?.delete();
63       delete soundOutput.value;
64     }
65   };
66
67   const saveGame = useCallback(() => {
```

```

68     if (gameboy && gameboy.supportsSaves()) {
69         const save = gameboy.save();
70         if (!save) return;
71         localforage.setItem(SAVE_CACHE_KEY + gameboy.getIdentifier(), save, (err) => {
72             if (err)
73                 console.error(
74                     `Could not save game ${gameboy.getTitle()}`
75 ↪     ` (${gameboy.getIdentifier()})`,
76                     err
77                 );
78             else {
79                 console.log(
80                     `Saved game ${gameboy.getTitle()} (${gameboy.getIdentifier()})`
81                 );
82                 addAlert(`Saved game '${gameboy.getTitle()}'`);
83             }
84         });
85     }, [gameboy]);
86
87     /** Backup */
88     useEffect(() => {
89         window.addEventListener("beforeunload", saveGame);
90         return () => window.removeEventListener("beforeunload", saveGame);
91     }, [saveGame]);
92
93     /**
94     * Loads a ROM into the gameboy, instantiating it. Also creates the 2nd emulator if needed
95     */
96     const loadGame = useCallback(
97         (rom: Uint8Array) => {
98             /** Save previous state, clear variables */
99             saveGame();
100
101             /** Setup input (we can't pass the value directly, because the object might change)
102 ↪     */
103             const gameIn: GameBoyInput = {
104                 read: () => joypadInput.value!.read(),
105             };
106
107             /** Setup output (relies on most things not changing) */
108             let serialOutTxt = "";
109             const gbOut: GameBoyOutput = {
110                 get receiveGraphics() {
111                     return emulatorFrameIn.current;
112                 },
113                 receiveSound: (d) => soundOutput.value?.enqueue(d),
114                 serialOut: (d) => {
115                     serialOutTxt += String.fromCharCode(d);
116                     console.log("Serial out > ", serialOutTxt);
117                 },
118                 get debugBackground() {
119                     return bgDebugger.current;
120                 },
121                 get debugTileset() {
122                     return tilesetDebugger.current;
123                 },
124             };
125
126             let bootRom: Uint8Array | null = null;
127             if (config.bootRom === "real") {
128                 if (config.console === "dmg") bootRom = config.bootRomDmg;
129                 else bootRom = config.bootRomCgb;
130             }
131
132             /** Create the emulator (ensure it loads correctly.) */
133             let gbc: GameBoyColor;
134             try {
135                 gbc = new GameBoyColor(

```



```

135         config.console === "dmg" ? "DMG" : "CGB",
136         rom,
137         gameIn,
138         gbOut,
139         {
140             bootRom,
141         }
142     );
143 } catch (e) {
144     alert("Could not load ROM: " + e);
145     return;
146 }
147 addAlert(`Loaded game '${gbc.getTitle()}'`);
148
149 /** Load a save (if one exists) */
150 if (gbc.supportsSaves()) {
151     localStorage.getItem<Uint8Array>(
152         SAVE_CACHE_KEY + gbc.getIdentifier(),
153         (err, save) => {
154             if (!save) return;
155             try {
156                 gbc.load(save);
157                 console.log(
158                     `Loaded save for ${gbc.getTitle()} (${gbc.getIdentifier()})`
159                 );
160                 addAlert(`Loaded save for '${gbc.getTitle()}'`);
161             } catch (e) {
162                 console.error(
163                     `Could not load save for ${gbc.getTitle()}
↪      (${gbc.getIdentifier()})`,
164                     e
165                 );
166             }
167         }
168     );
169 }
170
171 setGameboy(gbc);
172
173 let lastFrame = Date.now();
174
175 /** Run the emulator (this is the "main loop") */
176 const runEmulator = () => {
177     /**
178      * This is a bit of a hack to ensure that the emulator doesn't run if the
179      * instance has changed. It doesn't update the state, because the instance
180      * remains the same.
181      * This relies on the fact the state setter (setGameboy) doesn't change and is
182      * synchronous.
183      */
184     const expectedInstance = gbc;
185     let currentInstance: GameBoyColor | undefined = undefined;
186     setGameboy((g) => (currentInstance = g));
187
188     // if the instance has changed, stop this loop.
189     if (currentInstance !== expectedInstance) return;
190
191     /** Run the emulator */
192     const now = Date.now();
193     const delta = now - lastFrame;
194     lastFrame = now;
195     const framesToRun = Math.min(3, delta / (1000 / 60)); // can't catch up more than
↪    3 frames
196     const speed = tripleSpeed.value ? 3 : 1;
197     const frames = framesToRun * speed;
198
199     const before = configPtr.value.showStats ? performance.now() : 0;
200     gbc.drawFrame(frames, !emulatorRunning.value);
201

```

```

202         if (configPtr.value.showStats && stepCount.current && millisPerFrame.current) {
203             const millis = performance.now() - before;
204             const cpuSteps = gbc["cpu"]["stepCounter"];
205
206             stepCount.current.innerHTML = `${cpuSteps.toLocaleString()} steps`;
207             millisPerFrame.current.innerText = `${millis.toLocaleString()} ms/frame`;
208         }
209
210         /** Need to handle wait for a step to be made. */
211         if (!emulatorRunning.value) {
212             const waitForStep = () => {
213                 if (canStep.value || emulatorRunning.value) {
214                     canStep.value = false;
215                     lastFrame = Date.now();
216                     requestAnimationFrame(runEmulator);
217                 } else {
218                     requestAnimationFrame(waitForStep);
219                 }
220             };
221             requestAnimationFrame(waitForStep);
222         } else {
223             requestAnimationFrame(runEmulator);
224         }
225     };
226
227     requestAnimationFrame(runEmulator);
228
229     // @ts-ignore helpful for debugging :)
230     window.gbc = gbc;
231     return gbc;
232 },
233 [gameboy, config]
234 );
235
236 /**
237  * Utility refresh: gets the caches ROM and plays it.
238  */
239 useEffect(() => {
240     const listener = (e: KeyboardEvent) => {
241         if (e.key === "r" && loadedGame) {
242             var target = (e.target || e.srcElement) as HTMLElement;
243             var targetTagName = target === null ? "null" : target.nodeName.toUpperCase();
244             if (/INPUT|SELECT|TEXTAREA/.test(targetTagName)) {
245                 return;
246             }
247             loadGame(loadedGame);
248         }
249     };
250     document.addEventListener("keydown", listener);
251     return () => document.removeEventListener("keydown", listener);
252 }, [loadGame, setLoadedGame]);
253
254 /**
255  * Cache the loaded ROM, and load it in.
256  */
257 const loadRom = useCallback(
258     (rom: ArrayBuffer) => {
259         const romArray = new Uint8Array(rom);
260         // try caching the rom for reloads / refreshes
261         localforage.setItem(
262             CACHE_KEY,
263             romArray,
264             (err) => err && console.warn("Error caching ROM: ", err)
265         );
266         setLoadedGame(romArray);
267         loadGame(romArray);
268     },
269     [loadGame, setLoadedGame]
270 );

```

```

271
272 /**
273  * Hot load: if a ROM is cached, instantly loads it on startup.
274  */
275 useEffect(() => {
276     localStorage.getItem<Uint8Array>(CACHE_KEY, (err, value) => {
277         if (!value) return;
278         setLoadedGame(value);
279         loadGame(value);
280     });
281 }, []);
282
283 return (
284     <>
285     <AlertManager />
286     <Drawer loadRom={loadRom} />
287
288     <div id="emulator">
289         <h1>Emmy</h1>
290         <h2>The GBC Browser Emulator</h2>
291
292         <RomInput onLoad={loadRom} />
293
294         <div id="emu-options">
295             <IconButton
296                 title="Play/Pause"
297                 Icon={emulatorRunning.value ? Pause : Play}
298                 onClick={() =>
299                     (emulatorRunning.value = canStep.value = !emulatorRunning.value)
300                 }
301                 showTooltip
302             />
303
304             <IconButton
305                 title="Step"
306                 Icon={Redo}
307                 onClick={() => (canStep.value = true)}
308                 disabled={emulatorRunning.value}
309                 showTooltip
310             />
311
312             <IconButton
313                 title="Sound Enabled"
314                 onClick={toggleHasSound}
315                 Icon={config.audioEnabled ? Volume2 : VolumeX}
316                 showTooltip
317             />
318
319             <IconButton
320                 id="emu-speed"
321                 title="Triple Speed"
322                 onClick={() => (tripleSpeed.value = !tripleSpeed.value)}
323                 Icon={FastForward}
324                 toggled={tripleSpeed.value}
325                 showTooltip
326             />
327
328             <IconButton
329                 title={
330                     gameboy?.supportsSaves() === false
331                     ? "Game doesn't support saves "
332                     : "Save Game"
333                 }
334                 onClick={() => saveGame()}
335                 Icon={Save}
336                 showTooltip
337                 disabled={!gameboy || !gameboy.supportsSaves()}
338             />
339         </div>

```

```

340
341     {gameboy && (
342         <div id="emu-stack">
343             {config.showStats && (
344                 <div id="emu-stats">
345                     <div ref={stepCount} />
346                     <div ref={millisPerFrame} />
347                 </div>
348             )}
349             <Screen
350                 inputRef={emulatorFrameIn}
351                 scale={1 << config.scale}
352                 Filter={config.filter}
353                 blending={config.frameBlending}
354                 palette={effectivePalette}
355                 id="emulator-frame"
356             />
357         </div>
358     )}
359
360     <GameInput inputHandler={(x) => (joypadInput.value = x)} />
361
362     {gameboy && config.showDebugScreens && (
363         <div id="emu-screens">
364             <Screen width={256} height={256} inputRef={bgDebugger} />
365             <Screen width={256} height={192} inputRef={tilesetDebugger} />
366         </div>
367     )}
368 </div>
369 </>
370 );
371 };
372
373 export default App;

```

B.2.2 Main Styling

index (src/frontend/index.css)

```
1 :root {
2   font-family: Inter, Avenir, Helvetica, Arial, sans-serif;
3   font-size: 16px;
4   line-height: 24px;
5   font-weight: 400;
6
7   color-scheme: light dark;
8   color: rgba(255, 255, 255, 0.87);
9   background-color: #242424;
10
11  font-synthesis: none;
12  text-rendering: optimizeLegibility;
13  -webkit-font-smoothing: antialiased;
14  -moz-osx-font-smoothing: grayscale;
15  -webkit-text-size-adjust: 100%;
16 }
17
18 #app {
19   width: 100%;
20   height: 100vh;
21   display: flex;
22   flex-direction: row;
23 }
24
25 #emulator {
26   text-align: center;
27   padding: 2rem;
28   width: 100%;
29   overflow-x: auto;
30 }
31
32 a {
33   font-weight: 500;
34   color: #646cff;
35   text-decoration: inherit;
36 }
37 a:hover {
38   color: #535bf2;
39 }
40
41 body {
42   margin: 0;
43   display: flex;
44   min-width: 320px;
45   min-height: 100vh;
46 }
47
48 h1 {
49   font-size: 3.2em;
50   line-height: 1.1;
51 }
52
53 canvas {
54   image-rendering: pixelated;
55   max-width: 100%;
56 }
57
58 #emu-stack {
59   display: flex;
60   flex-direction: column;
61   align-items: center;
62 }
63
64 #emu-options {
```

```

65     display: flex;
66     flex-direction: row;
67     justify-content: center;
68     margin: 16px 0;
69 }
70
71 #emu-options > * {
72     margin: 0 8px;
73 }
74
75 #emu-screens {
76     display: flex;
77     flex-direction: row;
78     align-items: center;
79     justify-content: center;
80     flex-wrap: wrap;
81 }
82
83 .screen-container {
84     margin: 4px;
85     margin-bottom: -3px;
86 }
87
88 #emu-stack code {
89     padding: 2em;
90     white-space: break-spaces;
91     overflow-wrap: anywhere;
92 }
93
94 code {
95     font-family: "Fira Code", monospace;
96     transition: color 0.5s ease;
97 }
98
99 button {
100     position: relative;
101     border-radius: 8px;
102     border: 1px solid transparent;
103     padding: 0.6em 1.2em;
104     font-size: 1em;
105     font-weight: 500;
106     font-family: inherit;
107     background-color: #1a1a1a;
108     transition: border-color 0.25s ease, background-color 0.25s ease;
109 }
110 button:not(:disabled) {
111     cursor: pointer;
112 }
113 button:hover:not(:disabled) {
114     border-color: #646cff;
115 }
116 button:focus,
117 button:focus-visible {
118     outline: 0;
119     background-color: #202020;
120     border-color: mediumslateblue;
121 }
122
123 button.icon-button {
124     border-radius: 4px;
125     padding: 0.2em 0.4em;
126     font-size: 0.5em;
127     aspect-ratio: 1/1;
128 }
129
130 button.icon-button svg {
131     transform: scale(0.5);
132 }
133

```

```

134 button.icon-button.toggled {
135     border-color: greenyellow;
136 }
137
138 button > .tooltip {
139     position: absolute;
140     top: 0;
141     left: 50%;
142     transform: translate(-50%, -50%);
143     text-align: center;
144     opacity: 0;
145     margin-top: -1.5em;
146     width: max-content;
147     background-color: #1a1a1a77;
148     font-size: 1.3em;
149     padding: 2px;
150     border-radius: 2px;
151     transition: opacity 0.1s ease;
152     pointer-events: none;
153 }
154
155 button:hover > .tooltip {
156     display: block;
157     opacity: 1;
158 }
159
160 @media (prefers-color-scheme: light) {
161     :root {
162         color: #213547;
163         background-color: #ffffff;
164     }
165     a:hover {
166         color: #747bff;
167     }
168     button {
169         background-color: #f9f9f9;
170     }
171 }
172
173 @media (any-pointer: fine) {
174     .mobile-only {
175         display: none;
176     }
177 }

```

mobile (src/frontend/mobile.css)

```
1 @media (max-width: 700px) {
2   #drawer {
3     order: 2;
4     width: calc(100vw - 2em) !important;
5     max-width: calc(100vw - 2em) !important;
6     min-width: calc(100vw - 2em) !important;
7     overflow-x: hidden;
8     overflow-y: unset;
9   }
10
11   #drawer .resizer {
12     display: none;
13   }
14
15   #emulator {
16     width: unset;
17     padding: 0.5rem;
18   }
19
20   #app {
21     height: unset;
22     flex-direction: column;
23   }
24 }
```


B.2.3 Components

Alerts (src/frontend/components/Alerts.tsx)

```
1 import { FunctionalComponent } from "preact";
2 import { useEffect, useState } from "preact/hooks";
3
4 import "./Alerts.css";
5
6 type Alert = {
7   id: number;
8   text: string;
9   endTime: Date;
10  state: "new" | "visible" | "fading";
11  fade: number;
12 };
13
14 const alerts: Alert[] = [];
15 let keyId: number = 0;
16
17 function addAlert(text: string, duration: number = 3) {
18   alerts.push({
19     id: keyId++,
20     text,
21     endTime: new Date(Date.now() + duration * 1000),
22     state: "new",
23     fade: 0,
24   });
25 }
26
27 const AlertManager: FunctionalComponent = () => {
28   const [refresh, setRefresh] = useState<number>(0);
29
30   useEffect(() => {
31     const interval = setInterval(() => {
32       let changed = false;
33       const now = Date.now();
34       for (const alert of alerts) {
35         if (alert.state === "new") {
36           alert.state = "visible";
37           changed = true;
38         }
39         if (alert.endTime.getTime() < now) {
40           if (alert.state === "visible") {
41             alert.state = "fading";
42             changed = true;
43           } else {
44             alert.fade++;
45             if (alert.fade > 5) {
46               alerts.splice(alerts.indexOf(alert), 1);
47               changed = true;
48             }
49           }
50         }
51       }
52       if (changed) setRefresh((r) => r + 1);
53     }, 100);
54     return () => clearInterval(interval);
55   }, [setRefresh]);
56
57   return (
58     <div id="alert-box">
59       {alerts.map((alert) => (
60         <div key={alert.id} className={`alert ${alert.state} ${alert.id}`}>
61           {alert.text}
62         </div>
63       ))}
64     </div>
```

```
65     );  
66 };  
67  
68 export { addAlert, AlertManager };
```

Alerts (src/frontend/components/Alerts.css)

```
1 #alert-box {
2     position: fixed;
3     top: 0;
4     left: 0;
5     width: 100%;
6     height: 100%;
7     z-index: 100;
8     display: flex;
9     flex-direction: column;
10    justify-content: end;
11    align-items: end;
12    pointer-events: none;
13 }
14
15 #alert-box .alert {
16     padding: 5px;
17     background-color: rgba(0, 0, 0, 0.5);
18     color: white;
19     border-radius: 5px;
20     margin: 5px;
21     transition: opacity 0.2s ease;
22 }
23
24 #alert-box .alert.visible {
25     opacity: 1;
26 }
27
28 #alert-box .alert.fading {
29     opacity: 0;
30 }
```

GameInput (src/frontend/components/GameInput.tsx)

```
1 import { FunctionalComponent } from "preact";
2 import { useCallback, useEffect, useMemo } from "preact/hooks";
3
4 import GameBoyInput from "@emulator/GameBoyInput";
5 import { useConfig } from "@helpers/ConfigContext";
6 import useKeys from "@helpers/useKeys";
7
8 import "./GameInput.css";
9
10 type GameBoyInputObj = ReturnType<GameBoyInput["read"]>;
11
12 type JoypadButtonProps = {
13   name: string;
14   objKey: keyof GameBoyInputObj;
15   obj: GameBoyInputObj;
16 };
17
18 const JoypadButton: FunctionalComponent<JoypadButtonProps> = ({ name, objKey, obj }) => {
19   return (
20     <button
21       className={`control-button btn-${objKey}`}
22       onTouchStart={(e) => {
23         obj[objKey] = true;
24         if (e.cancelable) e.preventDefault();
25       }}
26       onTouchEnd={(e) => {
27         obj[objKey] = false;
28         if (e.cancelable) e.preventDefault();
29       }}
30     >
31       {name}
32     </button>
33   );
34 };
35
36 const NO_INPUT: GameBoyInputObj = {
37   a: false,
38   b: false,
39   start: false,
40   select: false,
41   up: false,
42   down: false,
43   left: false,
44   right: false,
45 };
46
47 type GameInputProps = {
48   inputHandler: (input: GameBoyInput) => void;
49 };
50
51 const GameInput: FunctionalComponent<GameInputProps> = ({ inputHandler }) => {
52   const [
53     {
54       controlA,
55       controlB,
56       controlStart,
57       controlSelect,
58       controlArrowUp,
59       controlArrowDown,
60       controlArrowLeft,
61       controlArrowRight,
62     },
63     ] = useConfig();
64
65   const pressedKeys = useKeys([
66     controlA,
67     controlB,
```

```

68     controlStart,
69     controlSelect,
70     controlArrowUp,
71     controlArrowDown,
72     controlArrowLeft,
73     controlArrowRight,
74   ]);
75
76   const touchControlStatus = useMemo(() => ({ ...NO_INPUT }), []);
77
78   const inputFn = useMemo(() => {
79     const obj = { ...NO_INPUT };
80     return () => {
81       obj.a = touchControlStatus.a || pressedKeys.includes(controlA);
82       obj.b = touchControlStatus.b || pressedKeys.includes(controlB);
83       obj.start = touchControlStatus.start || pressedKeys.includes(controlStart);
84       obj.select = touchControlStatus.select || pressedKeys.includes(controlSelect);
85       obj.up = touchControlStatus.up || pressedKeys.includes(controlArrowUp);
86       obj.down = touchControlStatus.down || pressedKeys.includes(controlArrowDown);
87       obj.left = touchControlStatus.left || pressedKeys.includes(controlArrowLeft);
88       obj.right = touchControlStatus.right || pressedKeys.includes(controlArrowRight);
89       return obj;
90     };
91   }, [
92     touchControlStatus,
93     pressedKeys,
94     controlA,
95     controlB,
96     controlStart,
97     controlSelect,
98     controlArrowUp,
99     controlArrowDown,
100    controlArrowLeft,
101    controlArrowRight,
102  ]);
103
104  useEffect(() => {
105    inputHandler({ read: inputFn });
106  }, [inputFn]);
107
108  return (
109    <div className="mobile-only joypad-input">
110      <div className="arrow-buttons">
111        <JoypadButton name="↑" objKey="up" obj={touchControlStatus} />
112        <JoypadButton name="←" objKey="left" obj={touchControlStatus} />
113        <JoypadButton name="→" objKey="right" obj={touchControlStatus} />
114        <JoypadButton name="↓" objKey="down" obj={touchControlStatus} />
115      </div>
116      <div className="main-buttons">
117        <JoypadButton name="A" objKey="a" obj={touchControlStatus} />
118        <JoypadButton name="B" objKey="b" obj={touchControlStatus} />
119        <JoypadButton name="START" objKey="start" obj={touchControlStatus} />
120        <JoypadButton name="SELECT" objKey="select" obj={touchControlStatus} />
121      </div>
122    </div>
123  );
124 };
125
126 export default GameInput;

```

GameInput (src/frontend/components/GameInput.css)

```
1 .joypad-input {
2     display: flex;
3     padding: 5px;
4     justify-content: space-between;
5 }
6
7 .joypad-input > * {
8     width: 45%;
9     position: relative;
10    aspect-ratio: 1;
11 }
12
13 .joypad-input .arrow-buttons > * {
14     position: absolute;
15     border-radius: 100%;
16     width: 38%;
17     inset: 0;
18     aspect-ratio: 1;
19     margin: auto;
20 }
21
22 .joypad-input .arrow-buttons .btn-up {
23     bottom: unset;
24 }
25
26 .joypad-input .arrow-buttons .btn-left {
27     right: unset;
28 }
29
30 .joypad-input .arrow-buttons .btn-right {
31     left: unset;
32 }
33
34 .joypad-input .arrow-buttons .btn-down {
35     top: unset;
36 }
37
38 .joypad-input .main-buttons > * {
39     position: absolute;
40     width: 55%;
41     height: 23%;
42 }
43
44 .joypad-input .main-buttons .btn-a {
45     bottom: 0;
46     left: 0;
47 }
48
49 .joypad-input .main-buttons .btn-b {
50     bottom: 25%;
51     right: 0;
52 }
53
54 .joypad-input .main-buttons .btn-start {
55     top: 25%;
56     left: 0;
57 }
58
59 .joypad-input .main-buttons .btn-select {
60     top: 0;
61     right: 0;
62 }
```

IconButton (src/frontend/components/IconButton.tsx)

```
1 import { LucideProps } from "lucide-preact";
2 import { FunctionalComponent, JSX } from "preact";
3
4 type IconButtonProps = {
5   id?: string;
6   title: string;
7   Icon: (props: LucideProps) => JSX.Element;
8   toggled?: boolean;
9   disabled?: boolean;
10  onClick?: () => void;
11  showTooltip?: boolean;
12 };
13
14 const IconButton: FunctionalComponent<IconButtonProps> = ({
15   id,
16   title,
17   Icon,
18   toggled,
19   disabled,
20   onClick,
21   showTooltip,
22 }) => {
23   return (
24     <button
25       title={title}
26       className={`icon-button ${toggled ? "toggled" : ""}`}
27       onClick={onClick}
28       disabled={disabled}
29       id={id}
30     >
31       {showTooltip && <div className="tooltip">{title}</div>}
32       <Icon />
33     </button>
34   );
35 };
36
37 export default IconButton;
```

KeybindingInput (src/frontend/components/KeybindingInput.tsx)

```
1 import { FunctionalComponent } from "preact";
2 import { useCallback, useEffect, useRef, useState } from "preact/hooks";
3
4 import "./KeybindingInput.css";
5
6 type KeybindingInputProps = {
7   value: string;
8   onChange: (value: string) => void;
9 };
10
11 const KEY_DISPLAY: Record<string, string | undefined> = {
12   ArrowUp: "↑",
13   ArrowDown: "↓",
14   ArrowLeft: "←",
15   ArrowRight: "→",
16   " ": "Space",
17 };
18
19 const KeybindingInput: FunctionalComponent<KeybindingInputProps> = ({ value, onChange }) => {
20   const [isEditing, setIsEditing] = useState<boolean>(false);
21   const [key, setKey] = useState<string | undefined>(undefined);
22
23   const ref = useRef<HTMLButtonElement>(null);
24
25   const onKeyDown = useCallback(
26     (e: KeyboardEvent) => {
27       if (e.key === "Escape") {
28         setIsEditing(false);
29         ref.current?.blur();
30         return;
31       }
32       if (e.key === "Enter" && key) {
33         setIsEditing(false);
34         onChange(key);
35         ref.current?.blur();
36         return;
37       }
38
39       setKey(e.key);
40       e.stopPropagation();
41       e.preventDefault();
42     },
43     [setIsEditing, onChange, setKey, key]
44   );
45
46   useEffect(() => {
47     if (isEditing) {
48       document.addEventListener("keydown", onKeyDown);
49     }
50     return () => {
51       document.removeEventListener("keydown", onKeyDown);
52     };
53   }, [isEditing, onKeyDown]);
54
55   return (
56     <button
57       ref={ref}
58       className={`keybinding-input ${isEditing ? "editing" : ""}`}
59       onClick={() => {
60         setIsEditing(true);
61         setKey(undefined);
62       }}
63       onBlur={() => {
64         setIsEditing(false);
65       }}
66     >
67     {isEditing
```



```

68         ? key
69         ? KEY_DISPLAY[key] ?? key
70         : "Press a key"
71         : KEY_DISPLAY[value] ?? value}
72     </button>
73 );
74 };
75
76 export default KeybindingInput;

```

KeybindingInput (src/frontend/components/KeybindingInput.css)

```
1 .keybinding-input {
2     text-align: center;
3     font-family: "Fira Code", monospace;
4     font-size: 12px;
5     background: #1e1e1e;
6     margin: 1px;
7     padding: 1px;
8     border-radius: 4px;
9 }
10
11 .keybinding-input:focus {
12     background: #2d2d2d;
13 }
14
15 /* Animate text pulsating */
16 .keybinding-input.editing {
17     font-weight: bolder;
18     animation: pulse 1s infinite;
19 }
20
21 @keyframes pulse {
22     0% {
23         color: #eee;
24     }
25     50% {
26         color: #888;
27     }
28     100% {
29         color: #eee;
30     }
31 }
```

Resizable (src/frontend/components/Resizable.tsx)

```
1 import { useSignal } from "@preact/signals";
2 import { FunctionalComponent, JSX } from "preact";
3
4 import "./Resizable.css";
5
6 export type ResizableProps = JSX.HTMLAttributes<HTMLDivElement> & { initialWidth: number };
7
8 export const Resizable: FunctionalComponent<ResizableProps> = ({
9   initialWidth,
10  children,
11  ...rest
12 }) => {
13   const width = useSignal(initialWidth);
14
15   const mouseDownHandler = (e: MouseEvent) => {
16     const startX = e.clientX;
17     const startWidth = width.value;
18
19     const onMouseMove = (e: MouseEvent) => {
20       width.value = startWidth + (e.clientX - startX);
21       e.stopPropagation();
22     };
23
24     const onMouseUp = () => {
25       document.removeEventListener("mousemove", onMouseMove);
26       document.removeEventListener("mouseup", onMouseUp);
27     };
28
29     document.addEventListener("mousemove", onMouseMove);
30     document.addEventListener("mouseup", onMouseUp);
31   };
32
33   return (
34     <div {...rest} style={{ minWidth: width.value, maxWidth: width.value }}>
35       {children}
36       <div className="resizer" onMouseDown={mouseDownHandler} />
37     </div>
38   );
39 };
```

Resizable (src/frontend/components/Resizable.css)

```
1 .resizer {  
2     position: absolute;  
3     width: 10px;  
4     height: 100%;  
5     top: 0;  
6     right: 0;  
7     transform: translateX(50%);  
8     z-index: 1;  
9     touch-action: none;  
10    cursor: ew-resize;  
11 }
```

RomInput (src/frontend/components/RomInput.tsx)

```
1 import { FunctionalComponent } from "preact";
2 import { useRef } from "preact/hooks";
3 import { JSXInternal } from "preact/src/jsx";
4
5 type RomInputProps = {
6   onLoad: (data: ArrayBuffer) => void;
7 };
8
9 const RomInput: FunctionalComponent<RomInputProps> = ({ onLoad }) => {
10   const fileInput = useRef<HTMLInputElement>(null);
11
12   const romClick = () => {
13     fileInput.current?.click();
14   };
15
16   const romChange: JSXInternal.GenericEventHandler<HTMLInputElement> = (e) => {
17     if (!e.currentTarget.files?.length) return;
18     const uploadedRom = e.currentTarget.files[0];
19     const reader = new FileReader();
20     reader.readAsArrayBuffer(uploadedRom);
21     reader.onload = (e) => {
22       if (e.target?.result && typeof e.target.result === "object") {
23         onLoad(e.target.result);
24       }
25     };
26     console.log("Uploaded rom", uploadedRom);
27   };
28
29   return (
30     <>
31       <button onClick={romClick}>Import a ROM</button>
32
33       <input
34         type="file"
35         id="rom-input"
36         accept=".gb,.gbc"
37         ref={fileInput}
38         onChange={romChange}
39         style={{ display: "none" }}
40       />
41     </>
42   );
43 };
44
45 export default RomInput;
```

Screen (src/frontend/components/Screen.tsx)

```
1 import { FunctionalComponent } from "preact";
2 import { MutableRef, useEffect, useMemo, useRef, useState } from "preact/hooks";
3
4 import { SCREEN_HEIGHT, SCREEN_WIDTH } from "@emulator/constants";
5 import { Identity, ImageFilter } from "@helpers/ImageFilter";
6
7 type ScreenProps = {
8   width?: number;
9   height?: number;
10  scale?: number;
11  inputRef: MutableRef<VideoReceiver || undefined>;
12  Filter?: ImageFilter;
13  blending?: boolean;
14  id?: string;
15  palette?: Partial<Record<number, number>>;
16 };
17
18 export type VideoReceiver = (data: Uint32Array) => void;
19
20 function applyPalette(frame: Uint32Array, palette: Partial<Record<number, number>>) {
21   for (let index = 0; index < frame.length; index++) {
22     frame[index] = palette[frame[index]] ?? frame[index];
23   }
24 }
25
26 function mixImages(frame1: Uint32Array, frame2: Uint32Array, target: Uint32Array) {
27   for (let index = 0; index < frame1.length; index++) {
28     const r1 = (frame1[index] >> 16) & 0xff;
29     const g1 = (frame1[index] >> 8) & 0xff;
30     const b1 = (frame1[index] >> 0) & 0xff;
31
32     const r2 = (frame2[index] >> 16) & 0xff;
33     const g2 = (frame2[index] >> 8) & 0xff;
34     const b2 = (frame2[index] >> 0) & 0xff;
35
36     const r = Math.floor((r1 + r2) / 2);
37     const g = Math.floor((g1 + g2) / 2);
38     const b = Math.floor((b1 + b2) / 2);
39
40     target[index] = (0xff << 24) | (r << 16) | (g << 8) | b;
41   }
42 }
43
44 const Screen: FunctionalComponent<ScreenProps> = ({
45   inputRef,
46   width = SCREEN_WIDTH,
47   height = SCREEN_HEIGHT,
48   scale = 1,
49   Filter = Identity,
50   blending = false,
51   id = undefined,
52   palette = undefined,
53 }) => {
54   const [stateRefresh, setStateRefresh] = useState(0);
55   const canvasRef = useRef<HTMLCanvasElement>(null);
56
57   const newFrame = useMemo(() => {
58     const canvas = canvasRef.current;
59     if (!canvas) {
60       setStateRefresh((state) => state + 1);
61       return;
62     }
63
64     const currentFrame = new Uint32Array(width * height);
65     const previousFrame = new Uint32Array(width * height);
66
67     const filterInstance = new Filter(width, height);
```

```

68     const dataAsUint8 = new Uint8ClampedArray(filterInstance.output.buffer);
69     const imageData = new ImageData(dataAsUint8, ...filterInstance.outputSize);
70
71     const targetWidth = width * scale * window.devicePixelRatio;
72     const targetHeight = height * scale * window.devicePixelRatio;
73
74     let firstFrame = true;
75
76     return (data: Uint32Array) => {
77         const context = canvas.getContext("2d", { alpha: false });
78         if (!context) return;
79
80         context.imageSmoothingEnabled = false;
81
82         previousFrame.set(currentFrame);
83         currentFrame.set(data);
84
85         if (palette) {
86             // Apply the color palette if needed
87             applyPalette(currentFrame, palette);
88         }
89
90         if (firstFrame) {
91             firstFrame = false;
92             // We copy the current frame into the previous one as it is entirely black
93             // This avoid a black frame at the beginning
94             previousFrame.set(currentFrame);
95         }
96
97         if (blending) {
98             // We mix both frames into the previous one. This is needed because some games
99             ↪ actually // flicker entities to display more sprites and have a darker color
100                 // (example: Link's Awakening chains)
101                 mixImages(currentFrame, previousFrame, previousFrame);
102         } else {
103             // We copy the current frame into the previous one to avoid a one frame delay
104             previousFrame.set(currentFrame);
105         }
106
107         // We apply the filter to the frame
108         filterInstance.apply(previousFrame);
109
110         // Actual drawing to the canvas - we scale the image to fit the canvas
111         createImageBitmap(imageData).then((bitmap) => {
112             context.drawImage(bitmap, 0, 0, targetWidth, targetHeight);
113             bitmap.close();
114         });
115     };
116 }, [stateRefresh, canvasRef.current, width, height, scale, Filter, blending, palette]);
117
118 useEffect(() => {
119     inputRef.current = newFrame;
120     return () => (inputRef.current = undefined);
121 }, [inputRef, newFrame]);
122
123 useEffect(() => {
124     const canvas = canvasRef.current;
125     if (!canvas) return;
126
127     let currentImage: ImageData | null = null;
128
129     const oldContext = canvas.getContext("2d", { alpha: false });
130     if (oldContext) {
131         currentImage = oldContext.getImageData(0, 0, canvas.width, canvas.height);
132     }
133
134     canvas.width = width * scale * window.devicePixelRatio;
135     canvas.height = height * scale * window.devicePixelRatio;

```

```

136     canvas.style.width = `${width * scale}px`;
137     canvas.style.aspectRatio = `${width} / ${height}`;
138
139     if (currentImage) {
140         const newContext = canvas.getContext("2d", { alpha: false });
141         if (!newContext) return;
142         newContext.imageSmoothingEnabled = false;
143         // Actual drawing to the canvas - we scale the image to fit the canvas
144         createImageBitmap(currentImage).then((bitmap) => {
145             newContext.drawImage(bitmap, 0, 0, canvas.width, canvas.height);
146             bitmap.close();
147         });
148     }
149     }, [width, height, scale, canvasRef.current]);
150
151     return (
152         <div className="screen-container">
153             <canvas id={id} ref={canvasRef} />
154         </div>
155     );
156 };
157
158 export default Screen;

```


B.2.4 Drawer Components

Drawer (src/frontend/components/Drawer/Drawer.tsx)

```
1 import { FunctionalComponent } from "preact";
2
3 import { Resizable } from "@components/Resizable";
4
5 import DrawerSection from "./DrawerSection";
6 import ExpressionDrawer from "./ExpressionDrawer";
7 import KeysDrawer from "./KeysDrawer";
8 import MemoryDrawer from "./MemoryDrawer";
9 import SettingsDrawer from "./SettingsDrawer";
10 import TestDrawer from "./TestDrawer";
11
12 import "./Drawer.css";
13
14 type DrawerProps = {
15   loadRom: (rom: Uint8Array) => void;
16 };
17
18 const Drawer: FunctionalComponent<DrawerProps> = ({ loadRom }) => (
19   <Resizable initialWidth={300} id="drawer">
20     <DrawerSection title="settings">
21       <SettingsDrawer />
22     </DrawerSection>
23     <DrawerSection title="watch expressions">
24       <ExpressionDrawer />
25     </DrawerSection>
26     <DrawerSection title="test ROMs">
27       <TestDrawer loadRom={loadRom} />
28     </DrawerSection>
29     <DrawerSection title="memory">
30       <MemoryDrawer />
31     </DrawerSection>
32     <DrawerSection title="keybindings">
33       <KeysDrawer />
34     </DrawerSection>
35   </Resizable>
36 );
37
38 export default Drawer;
```

Drawer (src/frontend/components/Drawer/Drawer.css)

```
1  /* General drawer */
2
3  #drawer {
4      position: relative;
5      max-height: calc(100% - 2em);
6      background-color: #303030;
7      padding: 1em;
8      overflow-y: auto;
9      overflow-x: hidden;
10 }
11
12 .drawer-section {
13     border-bottom: solid gray 1px;
14     padding: 2px 0;
15 }
16
17 .drawer-title {
18     display: flex;
19     flex-direction: row;
20     margin: 4px 0;
21     align-items: center;
22 }
23
24 .drawer-title > h3 {
25     font-weight: bolder;
26     text-transform: capitalize;
27     width: 100%;
28     text-align: start;
29     margin: 0;
30 }
31
32 .drawer-section > .drawer-content > div {
33     background-color: #242424;
34     border-radius: 8px;
35     padding: 4px;
36     margin-bottom: 2px;
37 }
38
39 .drawer-section-title {
40     display: flex;
41     align-items: center;
42     margin: 2px 0;
43     flex-wrap: wrap;
44 }
45
46 .drawer-section-title > div:first-child {
47     margin-right: auto;
48     text-align: start;
49     font-weight: bold;
50 }
51
52 .drawer-section-title > button {
53     margin-left: 2px;
54 }
55
56 .drawer-section-description {
57     font-size: 14px;
58     padding: 4px;
59 }
60
61 /* Test Drawer */
62
63 .test-drawer label {
64     width: 100%;
65     display: flex;
66     flex-direction: row;
67     justify-content: space-between;
```

```

68 }
69
70 .group-label > strong {
71     margin-left: 8px;
72 }
73
74 .test-result {
75     all: inherit;
76     width: 100%;
77     margin: -4px 0;
78     display: flex;
79     flex-direction: row;
80     justify-content: space-between;
81 }
82
83 .test-result .test-name {
84     font-family: "Fira Code", monospace;
85     font-size: 12px;
86 }
87
88 .test-result .test-state {
89     font-size: 12px;
90     margin-right: 8px;
91 }
92
93 /* Expression drawer */
94
95 .exp-watch {
96     display: flex;
97     flex-direction: column;
98     text-align: left;
99 }
100
101 .exp-drawer-control {
102     display: flex;
103     width: 100%;
104     justify-content: flex-end;
105 }
106
107 .exp-drawer-control > button {
108     margin-right: 8px;
109 }
110
111 .exp-watch > * {
112     font-family: "Fira Code", monospace;
113 }
114
115 .exp-watch > textarea {
116     resize: vertical;
117 }
118
119 .exp-watch > div {
120     transition: color 0.5s ease;
121     margin-bottom: 12px;
122 }
123
124 .exp-watch > div > .label {
125     width: 30%;
126     color: white;
127     margin-right: 8px;
128     background: none;
129     border: none;
130     font-size: inherit;
131     font-family: Inter, Avenir, Helvetica, Arial, sans-serif;
132 }
133
134 .exp-watch > div.error {
135     color: orangered;
136 }

```

```

137
138 .exp-watch-output {
139     display: inline-block;
140     white-space: pre-wrap;
141     word-break: break-word;
142 }
143
144 /* Memory drawer */
145
146 .memory-output {
147     font-family: "Fira Code", monospace;
148     font-size: 10px;
149     white-space: pre-wrap;
150     text-align: start;
151     line-height: 110%;
152 }
153
154 /* Keys drawer */
155
156 .keys-drawer {
157     display: flex;
158     flex-direction: row;
159     flex-wrap: wrap;
160     justify-content: space-between;
161 }
162
163 .keys-drawer > div {
164     width: 20%;
165     text-align: center;
166 }
167
168 .keys-drawer > button {
169     width: 25%;
170 }

```

DrawerSection (src/frontend/components/Drawer/DrawerSection.tsx)

```
1 import { useSignal } from "@preact/signals";
2 import { ChevronDown, ChevronUp } from "lucide-preact";
3 import { ComponentChildren, FunctionalComponent } from "preact";
4 import { useEffect, useRef } from "preact/hooks";
5
6 import IconButton from "@components/IconButton";
7
8 type DrawerSectionProps = {
9   title: string;
10  children: ComponentChildren;
11 };
12
13 const localStorageKey = "drawer-section-status";
14
15 const DrawerSection: FunctionalComponent<DrawerSectionProps> = ({ title, children }) => {
16   const contentRef = useRef<HTMLDivElement>(null);
17   const isOpen = useSignal<boolean>(false);
18
19   const id = `drawer-section-${title.toLowerCase().replace(/ /g, "-")}`;
20
21   useEffect(() => {
22     isOpen.value = localStorage.getItem(`${localStorageKey}-${title}`) === "1";
23   }, []);
24   useEffect(() => {
25     localStorage.setItem(`${localStorageKey}-${title}`, isOpen.value ? "1" : "0");
26   }, [isOpen.value]);
27
28   return (
29     <div className="drawer-section">
30       <div className="drawer-title">
31         <h3>{title}</h3>
32         <IconButton
33           id={id}
34           title="Open/Close Tab"
35           onClick={() => (isOpen.value = !isOpen.value)}
36           icon={isOpen.value ? ChevronUp : ChevronDown}
37         />
38       </div>
39       <div className="drawer-content">
40         {isOpen.value && <div ref={contentRef}>{children}</div>}
41       </div>
42     </div>
43   );
44 };
45
46 export default DrawerSection;
```

ExpressionDrawer (src/frontend/components/Drawer/ExpressionDrawer.tsx)

```
1 import { useSignal } from "@preact/signals";
2 import { Plus, Trash } from "lucide-preact";
3 import { FunctionalComponent } from "preact";
4 import { useEffect, useState } from "preact/hooks";
5
6 import IconButton from "@components/IconButton";
7 import ExpressionWatch from "../ExpressionWatch";
8
9 const localStorageKey = "exp-drawer-list";
10
11 const ExpressionDrawer: FunctionalComponent = () => {
12   const refresh = useState(0)[1];
13   const expressionList = useSignal<[string, string][]>([]);
14   useEffect(
15     () =>
16       (expressionList.value = JSON.parse(localStorage.getItem(localStorageKey) ?? "[]")),
17     []
18   );
19   const saveToLocalStorage = () =>
20     localStorage.setItem(localStorageKey, JSON.stringify(expressionList.value));
21   useEffect(saveToLocalStorage, [expressionList.value]);
22
23   return (
24     <div className="exp-drawer">
25       {expressionList.value.map(([exp, label], i) => (
26         <ExpressionWatch
27           key={i}
28           expression={exp}
29           onChange={e => {
30             expressionList.value[i][0] = e;
31             saveToLocalStorage();
32             refresh((r) => r + 1);
33           }}
34           label={label}
35           onLabelChange={l => {
36             expressionList.value[i][1] = l;
37             saveToLocalStorage();
38             refresh((r) => r + 1);
39           }}
40         />
41       ))}
42       <div className="drawer-section-title">
43         <div>Add/Remove:</div>
44         <IconButton
45           title="Add"
46           Icon={Plus}
47           onClick={() => (expressionList.value = [...expressionList.value, ["", ""]])}
48         />
49         {expressionList.value.length > 0 && (
50           <IconButton
51             title="Delete"
52             Icon={Trash}
53             onClick={() =>
54               (expressionList.value = expressionList.value.slice(0, -1))
55             }
56           />
57         )}
58       </div>
59     </div>
60   );
61 };
62
63 export default ExpressionDrawer;
```

ExpressionWatch (src/frontend/components/Drawer/ExpressionWatch.tsx)

```

1 import { FunctionalComponent } from "preact";
2 import { useEffect, useState } from "preact/hooks";
3
4 type ExpressionWatchProps = {
5   expression: string;
6   onChange: (expression: string) => void;
7   label: string;
8   onLabelChange: (label: string) => void;
9 };
10
11 const handleValue = (
12   value: Object | { get: () => number } | number | string | null
13 ): string => {
14   if (value === null) return "null";
15   if (value === undefined) return "undefined";
16
17   if (typeof value === "object" && "get" in value) value = value.get.call(value);
18
19   if (value === null) return "null";
20   if (value === undefined) return "undefined";
21   if (typeof value === "number") return `0x${value.toString(16).padStart(4, "0")}`;
22   return value.toString();
23 };
24
25 const ExpressionWatch: FunctionalComponent<ExpressionWatchProps> = ({
26   expression,
27   onChange,
28   label,
29   onLabelChange,
30 }) => {
31   const [func, setFunction] = useState<Function | null>(null);
32   const [output, setOutput] = useState<string | null>(null);
33
34   useEffect(() => {
35     try {
36       setFunction(() => new Function(`return ${expression}`));
37     } catch {
38       setFunction(null);
39     }
40   }, [expression]);
41
42   useEffect(() => {
43     const callbackId = setInterval(() => {
44       let value: typeof output;
45       try {
46         if (func === null) value = null;
47         else value = handleValue(func());
48       } catch {
49         value = null;
50       }
51       setOutput(value);
52     }, 100);
53     return () => clearInterval(callbackId);
54   }, [func]);
55
56   return (
57     <div className="exp-watch">
58       <textarea
59         placeholder="gbc.cpu.regAF"
60         spellCheck={false}
61         value={expression}
62         onInput={(e) => onChange(e.currentTarget.value)}
63       />
64       <div className={output === null ? "error" : undefined}>
65         <input
66           placeholder="label"
67           type="text"

```

```

68         className="label"
69         spellcheck={false}
70         value={label}
71         onInput={(e) => onLabelChange(e.currentTarget.value)}
72     />
73     <span className="exp-watch-output">{output === null ? "Error" : output}</span>
74 </div>
75 </div>
76 );
77 };
78
79 export default ExpressionWatch;

```


Grid2x (src/frontend/components/Drawer/Grid2x.tsx)

```

1 import { LucideProps } from "lucide-preact";
2 import { ComponentType, h, toChildArray } from "preact";
3 import { JSX } from "preact/jsx-runtime";
4
5 const defaultAttributes = {
6   xmlns: "http://www.w3.org/2000/svg",
7   width: 24,
8   height: 24,
9   viewBox: "0 0 24 24",
10  fill: "none",
11  stroke: "currentColor",
12  "stroke-width": 2,
13  "stroke-linecap": "round",
14  "stroke-linejoin": "round",
15 };
16
17 type IconNode = [elementName: keyof JSX.IntrinsicElements, attrs: Record<string, string>] [];
18
19 export const toKebabCase = (string: string) =>
20   string.replace(/([a-z0-9])([A-Z])/g, "$1-$2").toLowerCase();
21
22 /**
23  * Taken from Preact library directly:
24  * @link
25  * ↪ https://github.com/lucide-icons/lucide/blob/main/packages/lucide-preact/src/createPreactComponent.ts
26  */
27 const createPreactComponent = (
28   iconName: string,
29   iconNode: IconNode
30 ): ((props: LucideProps) => JSX.Element) => {
31   const Component = ({
32     color = "currentColor",
33     size = 24,
34     strokeWidth = 2,
35     children,
36     ref, // ignore ref
37     ...rest
38   }: LucideProps) =>
39     h(
40       "svg" as unknown as ComponentType<
41         Partial<JSX.SVGAttributes<SVGSVGElement> & { "stroke-width": number | string }>
42       >,
43       {
44         ...defaultAttributes,
45         width: String(size),
46         height: size,
47         stroke: color,
48         "stroke-width": strokeWidth,
49         class: `lucide lucide-${toKebabCase(iconName)}-`,
50         ...rest,
51       },
52       [...iconNode.map(([tag, attrs]) => h(tag, attrs)), ...toChildArray(children)]
53     );
54
55   Component.displayName = `${iconName}`;
56
57   return Component;
58 };
59
60 const Grid2x = createPreactComponent("Grid2x", [
61   [
62     "rect",
63     {
64       x: "3",
65       y: "3",
66       width: "18",
67       height: "18",
68       rx: "2",

```

```

67         ry: "2",
68         key: "maln0c",
69     },
70 ],
71 [
72     "line",
73     {
74         x1: "3",
75         y1: "12",
76         x2: "21",
77         y2: "12",
78         key: "1uch6j",
79     },
80 ],
81 [
82     "line",
83     {
84         x1: "12",
85         y1: "3",
86         x2: "12",
87         y2: "21",
88         key: "nvcl17",
89     },
90 ],
91 ]);
92
93 export default Grid2x;

```

KeysDrawer (src/frontend/components/Drawer/KeysDrawer.tsx)

```
1 import { FunctionalComponent } from "preact";
2
3 import KeybindingInput from "@components/KeybindingInput";
4 import { useConfig } from "@helpers/ConfigContext";
5
6 const KeysDrawer: FunctionalComponent = () => {
7   const [
8     {
9       controlArrowUp,
10      controlArrowDown,
11      controlArrowLeft,
12      controlArrowRight,
13      controlA,
14      controlB,
15      controlStart,
16      controlSelect,
17    },
18    setConfig,
19  ] = useConfig();
20  return (
21    <div>
22      <div className="drawer-section-description">
23        Click to edit, press enter to save, press escape to cancel.
24      </div>
25      <div className="keys-drawer">
26        <div>Up</div>
27        <KeybindingInput
28          value={controlArrowUp}
29          onChange={(v) => setConfig({ controlArrowUp: v })}
30        />
31        <div>A</div>
32        <KeybindingInput
33          value={controlA}
34          onChange={(v) => setConfig({ controlA: v })}
35        />
36        <div>Down</div>
37        <KeybindingInput
38          value={controlArrowDown}
39          onChange={(v) => setConfig({ controlArrowDown: v })}
40        />
41        <div>B</div>
42        <KeybindingInput
43          value={controlB}
44          onChange={(v) => setConfig({ controlB: v })}
45        />
46        <div>Left</div>
47        <KeybindingInput
48          value={controlArrowLeft}
49          onChange={(v) => setConfig({ controlArrowLeft: v })}
50        />
51        <div>Start</div>
52        <KeybindingInput
53          value={controlStart}
54          onChange={(v) => setConfig({ controlStart: v })}
55        />
56        <div>Right</div>
57        <KeybindingInput
58          value={controlArrowRight}
59          onChange={(v) => setConfig({ controlArrowRight: v })}
60        />
61
62        <div>Select</div>
63        <KeybindingInput
64          value={controlSelect}
65          onChange={(v) => setConfig({ controlSelect: v })}
66        />
67      </div>
68    </div>
69  );
70}
```

```
68         </div>
69     );
70 };
71
72 export default KeysDrawer;
```

MemoryDrawer (src/frontend/components/Drawer/MemoryDrawer.tsx)

```

1 import { useSignal } from "@preact/signals";
2 import { FunctionalComponent } from "preact";
3 import { useEffect } from "preact/hooks";
4
5 import GameBoyColor from "@emulator/GameBoyColor";
6
7 const refreshMemory = (offset: number) => {
8   // @ts-ignore
9   const gbc: GameBoyColor = window.gbc;
10  if (!gbc) return "Something went wrong";
11
12  let memory = "";
13  for (let address = offset; address < 0x10000; address += 16) {
14    memory += address.toString(16).padStart(4, "0") + ": ";
15    for (let i = 0; i < 16; i++) {
16      memory +=
17        gbc["system"]
18          .read(address + i)
19          .toString(16)
20          .padStart(2, "0") + "\u2009"; // thin space character
21    }
22    memory += "\n";
23  }
24  return memory;
25 };
26
27 const MEMORY_DRAWER_LOCAL_STORAGE_KEY = "memory-drawer-offset";
28
29 const MemoryDrawer: FunctionalComponent = () => {
30   const memory = useSignal<string>("");
31   const offset = useSignal<number>(0);
32
33   useEffect(() => {
34     const value = localStorage.getItem(MEMORY_DRAWER_LOCAL_STORAGE_KEY);
35     if (value) offset.value = +value;
36     memory.value = refreshMemory(offset.value);
37   }, []);
38
39   useEffect(() => {
40     const callbackId = setInterval(() => {
41       memory.value = refreshMemory(offset.value);
42     }, 500);
43     return () => clearInterval(callbackId);
44   }, [offset]);
45
46   return (
47     <div>
48       <div className="drawer-section-title">
49         <div>Offset:</div>
50         <input
51           type="text"
52           value={offset.value.toString(16).padStart(4, "0")}
53           onChange={e => {
54             offset.value = Number(`0x${e.currentTarget.value}`);
55             localStorage.setItem(
56               MEMORY_DRAWER_LOCAL_STORAGE_KEY,
57               offset.value.toString()
58             );
59           }}
60         />
61       </div>
62       <div className="memory-output">{memory}</div>
63     </div>
64   );
65 };
66
67 export default MemoryDrawer;

```

SettingsDrawer (src/frontend/components/Drawer/SettingsDrawer.tsx)

```

1  import {
2    Bug,
3    Circle,
4    Dice1,
5    Dice2,
6    Dice4,
7    FileDigit,
8    FileX2,
9    Flame,
10   Flower,
11   Gamepad,
12   Grid,
13   Image,
14   ImageOff,
15   LineChart,
16   Palette,
17   Square,
18   Waves,
19 } from "lucide-preact";
20 import { FunctionalComponent } from "preact";
21
22 import IconButton from "@components/IconButton";
23 import { useConfig } from "@helpers/ConfigContext";
24 import { Identity, Scale2x, Scale4x } from "@helpers/ImageFilter";
25
26 import Grid2x from "../Grid2x";
27
28 const availableFilters = [
29   {
30     name: "identity",
31     filter: Identity,
32     icon: Square,
33   },
34   {
35     name: "scale2x",
36     filter: Scale2x,
37     icon: Grid2x,
38   },
39   {
40     name: "scale4x",
41     filter: Scale4x,
42     icon: Grid,
43   },
44 ];
45
46 type DMGPalette = Partial<Record<number, number>> | undefined;
47
48 const palette = (white: number, lightGray: number, darkGray: number, black: number) => ({
49   0xffffffff: white | 0xff000000,
50   0xffaaaaaa: lightGray | 0xff000000,
51   0xff555555: darkGray | 0xff000000,
52   0xff000000: black | 0xff000000,
53 });
54
55 const isPaletteEquivalent = (palette1: DMGPalette, palette2: DMGPalette) => {
56   if (!palette1 && !palette2) return true; // both undefined
57   if (!palette1 || !palette2) return false; // only one undefined
58   if (Object.keys(palette1).length !== Object.keys(palette2).length) return false;
59   for (const key of Object.keys(palette1)) {
60     // @ts-ignore
61     if (palette1[key] !== palette2[key]) return false;
62   }
63   return true;
64 };
65
66 function loadFileWithInput(id: string, callback: (data: Uint8Array) => void): void {
67   const input = document.getElementById(id) as HTMLInputElement;

```

```

68   input.click();
69   input.onChange = () => {
70     if (input.files) {
71       const file = input.files[0];
72       const reader = new FileReader();
73       reader.onload = () => {
74         console.log("reader", reader);
75         if (reader.result) {
76           const buffer = reader.result as ArrayBuffer;
77           callback(new Uint8Array(buffer));
78         }
79       };
80       reader.readAsArrayBuffer(file);
81     }
82   };
83 }
84
85 const availablePalettes = [
86   {
87     name: "monochrome",
88     values: undefined, // no transform
89     icon: Circle,
90   },
91   {
92     name: "classic",
93     values: palette(0x95ddca, 0x6aa48b, 0x3d6042, 0x11180c),
94     icon: Gamepad,
95   },
96   {
97     name: "ocean",
98     values: palette(0xace2b9, 0x8a9965, 0x67582c, 0x35250c),
99     icon: Waves,
100  },
101  {
102    name: "magma",
103    values: palette(0x9ed4e5, 0x645ab0, 0x451f7b, 0x3c0112),
104    icon: Flame,
105  },
106  {
107    name: "sakura",
108    values: palette(0xe1dee9, 0x9377cd, 0x623cb5, 0x2b1449),
109    icon: Flower,
110  },
111 ];
112
113 const SettingsDrawer: FunctionalComponent = () => {
114   const [
115     {
116       filter: currentFilter,
117       frameBlending,
118       scale,
119       bootRom,
120       console,
121       gbPalette,
122       volume,
123       showStats,
124       showDebugScreens,
125       bootRomDmg,
126       bootRomCgb,
127     },
128     setConfig,
129   ] = useConfig();
130
131   return (
132     <>
133     <div className="drawer-section-title">
134       <div>Console:</div>
135       <IconButton
136         id="dmg-mode"

```

```

137         title="Classic (DMG)"
138         showTooltip
139         toggled={console === "dmg"}
140         Icon={Gamepad}
141         onClick={() => setConfig({ console: "dmg" })}
142     />
143     <IconButton
144         id="cgb-mode"
145         title="Color (CGB)"
146         showTooltip
147         toggled={console === "cgb"}
148         Icon={Palette}
149         onClick={() => setConfig({ console: "cgb" })}
150     />
151 </div>
152
153 <div className="drawer-section-title">
154   <div>Filter:</div>
155   {availableFilters.map(({ name, filter, icon }) => (
156     <IconButton
157       title={name}
158       showTooltip
159       toggled={filter === currentFilter}
160       Icon={icon}
161       onClick={() => setConfig({ filter })}
162     />
163   ))}
164 </div>
165
166 <div className="drawer-section-title">
167   <div>Scale:</div>
168   <IconButton
169     title="100%"
170     showTooltip
171     toggled={scale === 0}
172     Icon={Dice1}
173     onClick={() => setConfig({ scale: 0 })}
174   />
175   <IconButton
176     title="200%"
177     showTooltip
178     toggled={scale === 1}
179     Icon={Dice2}
180     onClick={() => setConfig({ scale: 1 })}
181   />
182   <IconButton
183     title="400%"
184     showTooltip
185     toggled={scale === 2}
186     Icon={Dice4}
187     onClick={() => setConfig({ scale: 2 })}
188   />
189 </div>
190
191 <div className="drawer-section-title">
192   <div>Volume:</div>
193
194   <input
195     type="range"
196     min="0"
197     max="2"
198     step="0.02"
199     value={volume}
200     onChange={(e) => setConfig({ volume: +e.currentTarget.value })}
201   />
202 </div>
203
204 <div className="drawer-section-title">
205   <div>GB Palette:</div>

```



```

206         {availablePalettes.map(({ name, icon, values }) => (
207             <IconButton
208                 title={name}
209                 Icon={icon}
210                 onClick={() => setConfig({ gbPalette: values })}
211                 toggled={isPaletteEquivalent(gbPalette, values)}
212                 showTooltip
213             />
214         ))}
215     </div>
216
217     <div className="drawer-section-title">
218         <div>Boot ROM Upload:</div>
219
220         <input type="file" id="upload-boot-rom" style="display: none" />
221         <IconButton
222             title="DMG"
223             Icon={Gamepad}
224             onClick={() =>
225                 loadFileWithInput("upload-boot-rom", (bootRomDmg) => {
226                     if (bootRomDmg.length !== 0x0100)
227                         alert(`Boot ROM of the DMG must be ${0x100} bytes long!`);
228                     else setConfig({ bootRomDmg });
229                 })
230             }
231             toggled={bootRomDmg !== null}
232             showTooltip
233         />
234         <IconButton
235             title="CGB"
236             Icon={Palette}
237             onClick={() =>
238                 loadFileWithInput("upload-boot-rom", (bootRomCgb) => {
239                     window.console.log(bootRomCgb);
240                     if (bootRomCgb.length !== 0x0900)
241                         alert(`Boot ROM of the DMG must be ${0x900} bytes long!`);
242                     else setConfig({ bootRomCgb });
243                 })
244             }
245             toggled={bootRomCgb !== null}
246             showTooltip
247         />
248     </div>
249
250     <div className="drawer-section-title">
251         <div>Other:</div>
252         <IconButton
253             title="Boot ROM"
254             Icon={bootRom === "real" ? FileDigit : FileX2}
255             onClick={() => setConfig({ bootRom: bootRom === "none" ? "real" : "none" })}
256             toggled={bootRom === "real"}
257             showTooltip
258         />
259         <IconButton
260             title="Toggle blending"
261             Icon={frameBlending ? Image : ImageOff}
262             onClick={() => setConfig({ frameBlending: !frameBlending })}
263             toggled={frameBlending}
264             showTooltip
265         />
266         <IconButton
267             title="Show Stats"
268             Icon={LineChart}
269             onClick={() => setConfig({ showStats: !showStats })}
270             toggled={showStats}
271             showTooltip
272         />
273     </div>
274

```

```

275         />
276
277         <IconButton
278             title="Debug"
279             Icon={Bug}
280             onClick={() => setConfig({ showDebugScreens: !showDebugScreens })}
281             toggled={showDebugScreens}
282             showTooltip
283         />
284     </div>
285 </>
286 );
287 };
288
289 export default SettingsDrawer;

```

TestDrawer (src/frontend/components/Drawer/TestDrawer.tsx)

```

1 import { useSignal } from "@preact/signals";
2 import { BoxSelect, FileQuestion } from "lucide-preact";
3 import { Fragment, FunctionalComponent } from "preact";
4 import { useEffect } from "preact/hooks";
5
6 import GameBoyColor from "@emulator/GameBoyColor";
7 import GameBoyInput from "@emulator/GameBoyInput";
8 import GameBoyOutput from "@emulator/GameBoyOutput";
9
10 import IconButton from "@components/IconButton";
11 import tests, { Test } from "@frontend/testConfig";
12
13 const TEST_PASS = "\u2705"; // tick emoji
14 const TEST_FAIL = "\u274C"; // cross emoji
15 const TEST_TIMEOUT = "\u231B"; // hourglass emoji
16 const TEST_CRASH = "\ud83e\udea6"; // skull emoji
17
18 type TestOutput = typeof TEST_PASS | typeof TEST_FAIL | typeof TEST_TIMEOUT | typeof TEST_CRASH;
19
20 const makeGameboy = (
21   type: "DMG" | "CGB",
22   rom: Uint8Array,
23   videoOut: (d: Uint32Array) => void,
24   serialOut: (s: string) => void
25 ) => {
26   const gameIn: GameBoyInput = {
27     read: () => ({
28       up: false,
29       down: false,
30       left: false,
31       right: false,
32       a: false,
33       b: false,
34       start: false,
35       select: false,
36     })),
37   };
38
39   const gbOut: GameBoyOutput = {
40     receiveGraphics: videoOut,
41     serialOut: (d) => serialOut(String.fromCharCode(d)),
42   };
43
44   return new GameBoyColor(type, rom, gameIn, gbOut);
45 };
46
47 type TestResult = [Test, TestOutput][];
48
49 const loadTestRom = async (testType: string, fileName: string) => {
50   const romResponse = await fetch(`/tests/${testType}/${fileName}.gb`);
51   const romBlob = await romResponse.blob();
52   return new Uint8Array(await romBlob.arrayBuffer());
53 };
54
55 const runTests = async (validGroups: string[] = [], results: (r: TestResult) => void) => {
56   const localResults: TestResult = [];
57
58   for (const test of tests) {
59     const { testType, subTestType, file, consoleType, check } = test;
60     if (!validGroups.includes(`${testType}/${subTestType}`)) continue;
61
62     console.log(`Running test ${testType}/${subTestType} -> ${file}`);
63
64     const romArray = await loadTestRom(testType, file);
65
66     let videoOut: Uint32Array = new Uint32Array(160 * 144);
67     let serialOut: string = "";

```

```

68     let state: TestOutput;
69
70     try {
71         const gbc = makeGameboy(
72             consoleType,
73             romArray,
74             (v) => videoOut.set(v),
75             (s) => (serialOut += s)
76         );
77
78         let prevSteps = 0;
79         while (true) {
80             gbc.drawFrame();
81
82             const newState = await check(gbc, serialOut, videoOut, file);
83             if (newState !== null) {
84                 state = newState === "failure" ? TEST_FAIL : TEST_PASS;
85                 break;
86             }
87
88             const steps = gbc["cpu"]["stepCounter"];
89             if (steps > 10_000_000 || steps === prevSteps) {
90                 state = TEST_TIMEOUT;
91                 break;
92             }
93             prevSteps = steps;
94         }
95     } catch (e) {
96         console.error("Caught error, skipping test", e);
97         state = TEST_CRASH;
98     }
99
100     localResults.push([test, state]);
101     results(localResults);
102 }
103
104 const passedTests = localResults.filter((t) => t[1] === TEST_PASS).length;
105 const totalTests = localResults.length;
106 console.log(`Finished running tests! Passed ${passedTests}/${totalTests} tests.`);
107 };
108
109 const testGroups = tests
110     .map((t) => `${t.testType}/${t.subTestType}`)
111     .filter((v, i, a) => a.indexOf(v) === i); // Unique
112
113 const localStorageKey = "test-drawer-groups";
114
115 type TestDrawerProps = {
116     loadRom: (rom: Uint8Array) => void;
117 };
118
119 const TestDrawer: FunctionalComponent<TestDrawerProps> = ({ loadRom }) => {
120     const testsRunning = useSignal<boolean>(false);
121     const testResults = useSignal<TestResult>([]);
122     const keptTests = useSignal<string[]>(testGroups);
123
124     // Loading
125     useEffect(
126         () => (keptTests.value = JSON.parse(localStorage.getItem(localStorageKey) ?? "[]")),
127         []
128     );
129     // Saving
130     useEffect(
131         () => localStorage.setItem(localStorageKey, JSON.stringify(keptTests.value)),
132         [keptTests.value]
133     );
134
135     return (
136         <div className="test-drawer">

```

```

137 <div className="drawer-section-title">
138 <div>Settings:</div>
139 <IconButton
140   title="Test"
141   showTooltip
142   Icon={FileQuestion}
143   disabled={testsRunning.value}
144   onClick={() => {
145     testResults.value = [];
146     testsRunning.value = true;
147     runTests(keptTests.value, (r) => (testResults.value = [...r])).then(
148       () => (testsRunning.value = false)
149     );
150   }}
151 />
152 <IconButton
153   title="Select/Unselect All"
154   showTooltip
155   Icon={BoxSelect}
156   disabled={testsRunning.value}
157   onClick={() =>
158     (keptTests.value = keptTests.value.length === 0 ? testGroups : [])
159   }
160 />
161 </div>
162 <div
163   style={{
164     display: "flex",
165     flexDirection: "column",
166     alignItems: "flex-start",
167   }}
168 >
169   {testGroups.map((group) => {
170     const selected = keptTests.value.includes(group);
171     const matchingTests = tests
172       .filter((t) => `${t.testType}/${t.subTestType}` === group)
173       .map(
174         (t) => [t, testResults.value.find((r) => r[0] === t)?.[1]] as const
175       );
176
177     const passedTests = matchingTests.filter((v) => v[1] === TEST_PASS).length;
178     const totalTests = matchingTests.length;
179
180     return (
181       <Fragment key={group}>
182         <label>
183           <span className="group-label">
184             {group}
185             {selected && matchingTests.length > 0 && (
186               <strong>
187                 {passedTests}/{totalTests}
188               </strong>
189             )}
190           </span>
191           <input
192             type="checkbox"
193             checked={selected}
194             disabled={testsRunning.value}
195             onChange={(e) =>
196               (keptTests.value = e.currentTarget.checked
197                 ? [...keptTests.value, group]
198                 : keptTests.value.filter((v) => v !== group))
199             }
200           />
201         </label>
202         {selected &&
203           matchingTests.map(([test, state]) => (
204             <button
205               key={test.file}

```

```

206         className="test-result"
207         onClick={() =>
208             loadTestRom(test.testType, test.file).then(loadRom)
209         }
210     >
211         <span className="test-name">{test.file}</span>
212         <span className="test-state">{state}</span>
213     </button>
214   </Fragment>
215 </div>
216 </div>
217 </div>
218 </div>
219 </div>
220 </div>
221 </div>
222 </div>
223 export default TestDrawer;

```

B.2.5 Helpers

AudioPlayer (src/frontend/helpers/AudioPlayer.ts)

```
1  /**
2   * A non-gameboy related class, that handles playing the received sound data.
3   * Largely inspired by:
4   * @link https://github.com/denislins/gameboy/blob/master/emulator/apu/Player.js
5   */
6  class AudioPlayer {
7      protected volume: { value: number };
8      protected context: AudioContext | undefined;
9      protected lastPlayEnd: number | undefined;
10     protected enqued: number = 0;
11
12     protected maxQueueSize: number;
13
14     protected windowBlurListener = () => this.context?.suspend();
15     protected windowFocusListener = () => this.context?.resume();
16
17     constructor(volume: { value: number } = { value: 1 }, maxQueueSize = 8) {
18         this.maxQueueSize = maxQueueSize;
19         this.volume = volume;
20         this.context = new AudioContext();
21         this.context.resume();
22         window.addEventListener("blur", this.windowBlurListener, false);
23         window.addEventListener("focus", this.windowFocusListener, false);
24     }
25
26     delete() {
27         this.context?.close();
28         delete this.context;
29         window.removeEventListener("blur", this.windowBlurListener, false);
30         window.removeEventListener("focus", this.windowFocusListener, false);
31     }
32
33     enqueue(sample: Float32Array) {
34         // Not allowed to have more than 8 samples in the queue, to avoid delay
35         if (this.enqued > this.maxQueueSize || !this.context) {
36             return;
37         }
38
39         this.enqued++;
40
41         const sampleDuration = sample.length / 44100;
42         const startTime =
43             this.lastPlayEnd && this.lastPlayEnd >= this.context.currentTime
44                 ? this.lastPlayEnd + sampleDuration
45                 : this.context.currentTime + sampleDuration; // add a delay to start
46         this.lastPlayEnd = startTime;
47
48         // Create the buffer
49         const buffer = this.context.createBuffer(1, sample.length, 44100);
50         const bufferContent = buffer.getChannelData(0);
51         for (let i = 0; i < sample.length; i++)
52             bufferContent[i] = sample[i] * this.volume.value;
53
54         const source = this.context.createBufferSource();
55         source.buffer = buffer;
56         source.onended = () => this.enqued--;
57
58         source.connect(this.context.destination);
59         source.start(startTime);
60     }
61 }
62
63 export default AudioPlayer;
```

ConfigContext (src/frontend/helpers/ConfigContext.tsx)

```
1 import { ComponentChildren, createContext, FunctionalComponent } from "preact";
2 import { useCallback, useContext, useState } from "preact/hooks";
3
4 import { filterByName, Identity, ImageFilter } from "@helpers/ImageFilter";
5
6 export type Configuration = {
7   scale: 0 | 1 | 2;
8   filter: ImageFilter;
9   audioEnabled: boolean;
10  frameBlending: boolean;
11  bootRom: "none" | "real";
12  console: "dmg" | "cgb";
13  gbPalette: undefined | Partial<Record<number, number>>;
14  volume: number;
15  showStats: boolean;
16  showDebugScreens: boolean;
17
18  bootRomDmg: Uint8Array | null;
19  bootRomCgb: Uint8Array | null;
20
21  controlArrowUp: string;
22  controlArrowDown: string;
23  controlArrowLeft: string;
24  controlArrowRight: string;
25  controlA: string;
26  controlB: string;
27  controlStart: string;
28  controlSelect: string;
29 };
30
31 type ConfigLoader<T> = {
32   to: (v: T) => string;
33   from: (v: string) => T;
34 };
35
36 const IdentitySave: ConfigLoader<any> = { to: (v) => v, from: (v) => v };
37
38 /** Not very memory efficient, but the sizes of the boot ROMs are small enough that it's
39 ↳ acceptable */
40 const Uint8ArrayStringSave: ConfigLoader<Uint8Array | null> = {
41   to: (v) =>
42     v === null
43       ? "0"
44       : Array.from(v)
45         .map((x) => x.toString(16).padStart(2, "0"))
46         .join(""),
47   from: (v) =>
48     v === "0"
49       ? null
50       : new Uint8Array(v.length / 2).map((_, i) =>
51         Number.parseInt(v.substring(i * 2, i * 2 + 2), 16)
52       ),
53 };
54
55 const configLoaders: {
56   [k in keyof Configuration]: null | ConfigLoader<Configuration[k]>;
57 } = {
58   scale: IdentitySave,
59   filter: {
60     to: (v: ImageFilter) => v.name,
61     from: (v: string) => filterByName(v) ?? Identity,
62   },
63   audioEnabled: null,
64   frameBlending: IdentitySave,
65   bootRom: IdentitySave,
66   console: IdentitySave,
67   gbPalette: IdentitySave,
```



```

67     volume: IdentitySave,
68     showStats: IdentitySave,
69     showDebugScreens: IdentitySave,
70
71     bootRomDmg: Uint8ArrayStringSave,
72     bootRomCgb: Uint8ArrayStringSave,
73
74     controlArrowUp: IdentitySave,
75     controlArrowDown: IdentitySave,
76     controlArrowLeft: IdentitySave,
77     controlArrowRight: IdentitySave,
78     controlA: IdentitySave,
79     controlB: IdentitySave,
80     controlStart: IdentitySave,
81     controlSelect: IdentitySave,
82   };
83
84   const defaultConfig: Configuration = {
85     scale: 1,
86     filter: Identity,
87     audioEnabled: false,
88     frameBlending: true,
89     bootRom: "none",
90     console: "dmg",
91     gbPalette: undefined,
92     volume: 0.5,
93     showStats: false,
94     showDebugScreens: false,
95
96     bootRomDmg: null,
97     bootRomCgb: null,
98
99     controlArrowUp: "ArrowUp",
100    controlArrowDown: "ArrowDown",
101    controlArrowLeft: "ArrowLeft",
102    controlArrowRight: "ArrowRight",
103    controlA: "z",
104    controlB: "x",
105    controlStart: "Enter",
106    controlSelect: "Backspace",
107  };
108
109  const configToString = (config: Configuration): string =>
110    JSON.stringify(
111      Object.fromEntries(
112        Object.entries(config) // @ts-ignore
113          .filter(([k, v]) => configLoaders[k] !== null) // @ts-ignore
114          .map(([k, v]) => [k, configLoaders[k].to(v)])
115      )
116    );
117
118  const configFromString = (configString: string): Configuration => {
119    const rawConfig = JSON.parse(configString);
120    // Create a partial config with only objects that are defined and part of the default config
121    const loadedConfig: Partial<Configuration> = Object.fromEntries(
122      Object.entries(rawConfig)
123        .filter(([k, v]) => k in configLoaders && v !== undefined) // @ts-ignore
124        .map(([k, v]) => [k, configLoaders[k].from(v)])
125    );
126    return { ...defaultConfig, ...loadedConfig };
127  };
128
129  const ConfigContext = createContext<
130    [Configuration, (newConfig: Partial<Configuration>) => void]
131  >([defaultConfig, () => {}]);
132
133  export const useConfig = () => useContext(ConfigContext);
134
135  const localStorageKey = "config";

```

```

136
137 export const ConfigProvider: FunctionalComponent<ComponentChildren> = ({ children }) => {
138   const [config, setConfig] = useState<Configuration>(() => {
139     const savedConfig = localStorage.getItem(localStorageKey);
140     if (savedConfig) {
141       const config = configFromString(savedConfig);
142       return { ...defaultConfig, ...config };
143     }
144     return defaultConfig;
145   });
146   const configUpdater = useCallback(
147     (newConfig: Partial<Configuration>) => {
148       let fullNewConfig: Configuration;
149       setConfig((c) => (fullNewConfig = { ...c, ...newConfig }));
150       localStorage.setItem(localStorageKey, configToString(fullNewConfig!));
151     },
152     [setConfig]
153   );
154
155   return (
156     <ConfigContext.Provider value={{[config, configUpdater]}}>
157       {children}
158     </ConfigContext.Provider>
159   );
160 };

```

useKeys (src/frontend/helpers/useKeys.ts)

```
1 import { useEffect, useRef } from "preact/hooks";
2
3 const useKeys = (codes: string[] = []) => {
4   const pressedKeys = useRef<string[]>([]);
5
6   useEffect(() => {
7     const codesLower = codes.map((code) => code.toLowerCase());
8
9     const keyDownListener = (e: KeyboardEvent) => {
10       if (!codesLower.includes(e.key.toLowerCase())) return;
11
12       var target = (e.target || e.srcElement) as HTMLElement;
13       var targetTagName = target === null ? "null" : target.nodeName.toUpperCase();
14       if (/INPUT|SELECT|TEXTAREA/.test(targetTagName)) {
15         return;
16       }
17
18       e.preventDefault();
19       const index = pressedKeys.current.indexOf(e.key);
20       if (index === -1) {
21         pressedKeys.current.push(e.key);
22       }
23     };
24
25     const keyUpListener = (e: KeyboardEvent) => {
26       if (!codesLower.includes(e.key.toLowerCase())) return;
27
28       const index = pressedKeys.current.indexOf(e.key);
29       if (index !== -1) {
30         pressedKeys.current.splice(index, 1);
31       }
32     };
33
34     document.addEventListener("keydown", keyDownListener);
35     document.addEventListener("keyup", keyUpListener);
36
37     return () => {
38       document.removeEventListener("keydown", keyDownListener);
39       document.removeEventListener("keyup", keyUpListener);
40     };
41   }, [codes]);
42
43   return pressedKeys.current;
44 }
45
46 export default useKeys;
```

B.2.6 Image Filters

index (src/frontend/helpers/ImageFilter/index.ts)

```
1 import ImageFilterConcrete from "./Base";
2 import Identity from "./Identity";
3 import Scale2x from "./Scale2x";
4 import Scale4x from "./Scale4x";
5
6 export type ImageFilter = typeof ImageFilterConcrete;
7 export const filterByName = (name: string): ImageFilter | undefined =>
8     [Identity, Scale2x, Scale4x].find((f) => f.name === name);
9 export { Identity, Scale2x, Scale4x };
```

Base (src/frontend/helpers/ImageFilter/Base.ts)

```
1 class ImageFilter {
2   protected readonly width: number;
3   protected readonly height: number;
4
5   readonly output: Uint32Array;
6
7   constructor(width: number, height: number) {
8     this.width = width;
9     this.height = height;
10
11     const outSize = this.outputSize;
12     this.output = new Uint32Array(outSize[0] * outSize[1]);
13   }
14
15   get outputSize(): [number, number] {
16     throw new Error("Not implemented");
17   }
18
19   apply(image: Uint32Array): void {
20     throw new Error("Not implemented");
21   }
22 }
23
24 export default ImageFilter;
```

Identity (src/frontend/helpers/ImageFilter/Identity.ts)

```
1 import ImageFilter from "./Base";
2
3 /**
4  * An identity filter that doesn't altered the image.
5  */
6 class Identity extends ImageFilter {
7     override get outputSize(): [number, number] {
8         return [this.width, this.height];
9     }
10    override apply(image: Uint32Array): void {
11        this.output.set(image);
12    }
13 }
14
15 export default Identity;
```

Scale2x (src/frontend/helpers/ImageFilter/Scale2x.ts)

```
1 import ImageFilter from "./Base";
2
3 /**
4  * Filter that uses the Scale2x algorithm
5  * @link https://www.scale2x.it/algorithm
6  */
7 class Scale2x extends ImageFilter {
8   override get outputSize(): [number, number] {
9     return [this.width * 2, this.height * 2];
10  }
11
12  override apply(image: Uint32Array): void {
13    for (let y = 0; y < this.height; y++) {
14      let value = image[y * this.width];
15      let w = value;
16      let e = image[y * this.width + 1];
17      for (let x = 0; x < this.width; x++) {
18        const n = y - 1 < 0 ? value : image[x + (y - 1) * this.width];
19        const s = y + 1 >= this.height ? value : image[x + (y + 1) * this.width];
20        if (n !== s && w !== e) {
21          this.output[x * 2 + y * 2 * this.width * 2] = w === n ? w : value;
22          this.output[x * 2 + 1 + y * 2 * this.width * 2] = n === e ? e : value;
23          this.output[x * 2 + (y * 2 + 1) * this.width * 2] = w === s ? w : value;
24          this.output[x * 2 + 1 + (y * 2 + 1) * this.width * 2] = s === e ? e : value;
25        } else {
26          this.output[x * 2 + y * 2 * this.width * 2] = value;
27          this.output[x * 2 + 1 + y * 2 * this.width * 2] = value;
28          this.output[x * 2 + (y * 2 + 1) * this.width * 2] = value;
29          this.output[x * 2 + 1 + (y * 2 + 1) * this.width * 2] = value;
30        }
31
32        w = value;
33        value = e;
34        e = x + 2 >= this.width ? value : image[x + 2 + y * this.width];
35      }
36    }
37  }
38 }
39
40 export default Scale2x;
```

Scale4x (src/frontend/helpers/ImageFilter/Scale4x.ts)

```
1 import Scale2x from "./Scale2x";
2
3 class Scale4x extends Scale2x {
4     protected subFilter: Scale2x;
5
6     constructor(width: number, height: number) {
7         super(width * 2, height * 2);
8         this.subFilter = new Scale2x(width, height);
9     }
10
11     override apply(image: Uint32Array): void {
12         this.subFilter.apply(image);
13         super.apply(this.subFilter.output);
14     }
15 }
16
17 export default Scale4x;
```


B.2.7 Automated Test Config

testConfig (src/frontend/testConfig.ts)

```
1 import GameBoyColor from "@emulator/GameBoyColor";
2
3 type MiniTest = {
4   file: string;
5   consoleType: "DMG" | "CGB";
6 };
7
8 const mkTests: (consoleType: "DMG" | "CGB", ...names: string[]) => MiniTest[] = (
9   consoleType,
10  ...names
11 ) =>
12   names.map((n) => ({
13     file: n,
14     consoleType,
15   }));
16
17 const dmgTests: (...names: string[]) => MiniTest[] = (...names) => mkTests("DMG", ...names);
18 const cgbTests: (...names: string[]) => MiniTest[] = (...names) => mkTests("CGB", ...names);
19
20 /** The list of files to run for tests, per "test group" */
21 const rawTestFiles = {
22   blaarg: {
23     cpu: dmgTests(
24       "cpu-01-special",
25       "cpu-02-interrupts",
26       "cpu-03-op sp,hl",
27       "cpu-04-op r,imm",
28       "cpu-05-op rp",
29       "cpu-06-ld r,r",
30       "cpu-07-jr,jp,call,ret,rst",
31       "cpu-08-misc instrs",
32       "cpu-09-op r,r",
33       "cpu-10-bit ops",
34       "cpu-11-op a,(hl)",
35       "instr_timing",
36       "mem_timing"
37     ),
38     apu: dmgTests(
39       "apu-01-registers",
40       "apu-02-len ctr",
41       "apu-03-trigger",
42       "apu-04-sweep",
43       "apu-05-sweep details",
44       "apu-06-overflow on trigger",
45       "apu-07-len sweep period sync",
46       "apu-08-len ctr during power",
47       "apu-09-wave read while on",
48       "apu-10-wave trigger while on",
49       "apu-11-regs after power",
50       "apu-12-wave write while on"
51     ),
52     other: dmgTests("halt_bug", "oam_bug"),
53   },
54   mooneye: {
55     itrAndCpu: dmgTests(
56       "daa",
57       "ei_sequence",
58       "halt_ime0_ei",
59       "ie_push",
60       "if_ie_registers",
61       "rapid_di_ei",
62       "reg_f",
63       "unused_hwio-GS"
64     ),
```

```

65     ppu: dmgTests(
66         "ppu_hblank_ly_scx_timing-GS",
67         "ppu_intr_1_2_timing-GS",
68         "ppu_intr_2_0_timing",
69         "ppu_intr_2_mode0_timing",
70         "ppu_intr_2_mode0_timing_sprites",
71         "ppu_intr_2_mode3_timing",
72         "ppu_intr_2_oam_ok_timing",
73         "ppu_lcdon_timing-GS",
74         "ppu_lcdon_write_timing-GS",
75         "ppu_stat_irq_blocking",
76         "ppu_stat_lyc_onoff",
77         "ppu_vblank_stat_intr-GS"
78     ),
79     cpuTiming: dmgTests(
80         "add_sp_e_timing",
81         "call_cc_timing",
82         "call_cc_timing2",
83         "call_timing",
84         "call_timing2",
85         "di_timing-GS",
86         "div_timing",
87         "ei_timing",
88         "halt_ime0_nointr_timing",
89         "halt_ime1_timing",
90         "halt_ime1_timing2-GS",
91         "intr_timing",
92         "jp_cc_timing",
93         "jp_timing",
94         "ld_hl_sp_e_timing",
95         "pop_timing",
96         "push_timing",
97         "ret_cc_timing",
98         "reti_intr_timing",
99         "reti_timing",
100        "rst_timing"
101    ),
102    timer: dmgTests(
103        "timer_div_write",
104        "timer_rapid_toggle",
105        "timer_tim00",
106        "timer_tim00_div_trigger",
107        "timer_tim01",
108        "timer_tim01_div_trigger",
109        "timer_tim10",
110        "timer_tim10_div_trigger",
111        "timer_tim11",
112        "timer_tim11_div_trigger",
113        "timer_tima_reload",
114        "timer_tima_write_reloading",
115        "timer_tma_write_reloading"
116    ),
117    mbc1: dmgTests(
118        "mbc1_bits_bank1",
119        "mbc1_bits_bank2",
120        "mbc1_bits_mode",
121        "mbc1_bits_ramg",
122        "mbc1_ram_256kb",
123        "mbc1_ram_64kb",
124        "mbc1_rom_16Mb",
125        "mbc1_rom_1Mb",
126        "mbc1_rom_2Mb",
127        "mbc1_rom_4Mb",
128        "mbc1_rom_512kb",
129        "mbc1_rom_8Mb"
130    ),
131    mbc2: dmgTests(
132        "mbc2_bits_ramg",
133        "mbc2_bits_romb",

```

```

134         "mbc2_bits_unused",
135         "mbc2_ram",
136         "mbc2_rom_1Mb",
137         "mbc2_rom_2Mb",
138         "mbc2_rom_512kb"
139     ),
140     mbc5: dmgTests(
141         "mbc5_rom_16Mb",
142         "mbc5_rom_1Mb",
143         "mbc5_rom_2Mb",
144         "mbc5_rom_32Mb",
145         "mbc5_rom_4Mb",
146         "mbc5_rom_512kb",
147         "mbc5_rom_64Mb",
148         "mbc5_rom_8Mb"
149     ),
150     oam: dmgTests(
151         "mem_oam",
152         "oam_dma_restart",
153         "oam_dma_start",
154         "oam_dma_timing",
155         "oam_dma_basic",
156         "oam_dma_reg_read",
157         "oam_dma_sources-GS"
158     ),
159 },
160 acid: {
161     acid: [...dmgTests("dmg-acid2"), ...cgbTests("cgb-acid2")],
162 },
163 samesuite: {
164     dma: cgbTests("gbc_dma_cont", "gdma_addr_mask", "hdma_lcd_off", "hdma_mode0"),
165     apu: cgbTests(
166         "div_trigger_volume_10",
167         "div_write_trigger",
168         "div_write_trigger_10",
169         "div_write_trigger_volume",
170         "div_write_trigger_volume_10"
171     ),
172     channel1: cgbTests(
173         "channel_1_align",
174         "channel_1_align_cpu",
175         "channel_1_delay",
176         "channel_1_duty",
177         "channel_1_duty_delay",
178         "channel_1_extra_length_clocking-cgb0B",
179         "channel_1_freq_change",
180         "channel_1_freq_change_timing-A",
181         "channel_1_freq_change_timing-cgb0BC",
182         "channel_1_freq_change_timing-cgbDE",
183         "channel_1_nrx2_glitch",
184         "channel_1_nrx2_speed_change",
185         "channel_1_restart",
186         "channel_1_restart_nrx2_glitch",
187         "channel_1_stop_div",
188         "channel_1_stop_restart",
189         "channel_1_sweep",
190         "channel_1_sweep_restart",
191         "channel_1_sweep_restart_2",
192         "channel_1_volume",
193         "channel_1_volume_div"
194     ),
195     channel2: cgbTests(
196         "channel_2_align",
197         "channel_2_align_cpu",
198         "channel_2_delay",
199         "channel_2_duty",
200         "channel_2_duty_delay",
201         "channel_2_extra_length_clocking-cgb0B",
202         "channel_2_freq_change",

```

```

203         "channel_2_nrx2_glitch",
204         "channel_2_nrx2_speed_change",
205         "channel_2_restart",
206         "channel_2_restart_nrx2_glitch",
207         "channel_2_stop_div",
208         "channel_2_stop_restart",
209         "channel_2_volume",
210         "channel_2_volume_div"
211     ),
212     channel3: cgbTests(
213         "channel_3_and_glitch",
214         "channel_3_delay",
215         "channel_3_extra_length_clocking-cgb0",
216         "channel_3_extra_length_clocking-cgbB",
217         "channel_3_first_sample",
218         "channel_3_freq_change_delay",
219         "channel_3_restart_delay",
220         "channel_3_restart_during_delay",
221         "channel_3_restart_stop_delay",
222         "channel_3_shift_delay",
223         "channel_3_shift_skip_delay",
224         "channel_3_stop_delay",
225         "channel_3_stop_div",
226         "channel_3_wave_ram_dac_on_rw",
227         "channel_3_wave_ram_locked_write",
228         "channel_3_wave_ram_sync"
229     ),
230     channel4: cgbTests(
231         "channel_4_align",
232         "channel_4_delay",
233         "channel_4_equivalent_frequencies",
234         "channel_4_extra_length_clocking-cgb0B",
235         "channel_4_freq_change",
236         "channel_4_frequency_alignment",
237         "channel_4_lfsr",
238         "channel_4_lfsr15",
239         "channel_4_lfsr_15_7",
240         "channel_4_lfsr_7_15",
241         "channel_4_lfsr_restart",
242         "channel_4_lfsr_restart_fast",
243         "channel_4_volume_div"
244     ),
245 },
246 };
247
248 type TestType = keyof typeof rawTestFiles;
249 type SubTestType = { [k in TestType]: keyof typeof rawTestFiles[k] }[TestType];
250 type TestChecker = (
251     gbc: GameBoyColor,
252     out: string,
253     vid: Uint32Array,
254     testName: string
255 ) => Promise<null | "success" | "failure">;
256
257 export type Test = MiniTest & {
258     testType: TestType;
259     subTestType: SubTestType;
260     check: TestChecker;
261 };
262
263 const loadImageData = async (fileName: string): Promise<Uint32Array> => {
264     let promiseResolve: (value: Uint32Array) => void;
265     let endPromise = new Promise<Uint32Array>((r) => (promiseResolve = r));
266
267     let img = new Image();
268     img.onload = function () {
269         var canvas = document.createElement("canvas");
270         var ctx = canvas.getContext("2d");
271         ctx.drawImage(img, 0, 0);

```

```

272     const imageData = ctx.getImageData(0, 0, 160, 144);
273     const imageDataAsUint32 = new Uint32Array(imageData.data.buffer);
274     promiseResolve(imageDataAsUint32);
275 };
276 img.src = `/tests/${fileName}.png`;
277 return endPromise;
278 };
279
280 const compareImages = (imgA: Uint32Array, imgB: Uint32Array) => {
281     if (imgA.length !== imgB.length) return "failure";
282     for (let i = 0; i < imgA.length; i++) {
283         if ((imgA[i] & 0xf8f8f8) !== (imgB[i] & 0xf8f8f8)) return "failure";
284     }
285     return "success";
286 };
287
288 const blaargTestCheck: TestChecker = async (gbc, serialOut, vid, testName) => {
289     if (serialOut.toLowerCase().includes("pass")) return "success";
290     if (serialOut.toLowerCase().includes("fail")) return "failure";
291
292     // Some blaarg tests 'sign' $a001-$a003 with the string "de b0 61", to then
293     // write the status to $a000
294     if (gbc["system"].inspect(0xa001, 3) === "de b0 61") {
295         const status = gbc["system"].read(0xa000);
296         if (status === 0x80) return null;
297         return status === 0x00 ? "success" : "failure";
298     }
299
300     if (testName === "halt_bug" && gbc["cpu"].getStepCounts() >= 700_000) {
301         const expected = await loadImageData("blaarg/reference-halt_bug");
302         return compareImages(expected, vid);
303     }
304     return null;
305 };
306
307 const mooneyeAndSamesuiteTestCheck: TestChecker = async (gbc) => {
308     if (
309         gbc["cpu"]["srB"].get() === 3 &&
310         gbc["cpu"]["srC"].get() === 5 &&
311         gbc["cpu"]["srD"].get() === 8 &&
312         gbc["cpu"]["srE"].get() === 13 &&
313         gbc["cpu"]["srH"].get() === 21 &&
314         gbc["cpu"]["srL"].get() === 34
315     )
316         return "success";
317     if (
318         gbc["cpu"]["srB"].get() === 0x42 &&
319         gbc["cpu"]["srC"].get() === 0x42 &&
320         gbc["cpu"]["srD"].get() === 0x42 &&
321         gbc["cpu"]["srE"].get() === 0x42 &&
322         gbc["cpu"]["srH"].get() === 0x42 &&
323         gbc["cpu"]["srL"].get() === 0x42
324     )
325         return "failure";
326     return null;
327 };
328
329 const acidTestCheck: TestChecker = async (gbc, _, vid, test) => {
330     if (test === "dmg-acid2" && gbc["cpu"]["stepCounter"] >= 85_000) {
331         const imageData = await loadImageData("acid/reference-dmg");
332         return compareImages(imageData, vid);
333     } else if (test === "cgb-acid2" && gbc["cpu"]["stepCounter"] >= 140_000) {
334         const imageData = await loadImageData("acid/reference-cgb");
335         console.log("comparing ", { imageData, vid });
336         return compareImages(imageData, vid);
337     }
338
339     return null;
340 };

```

```

341
342 /** The list of test categories, with a runnable that says the status of the test */
343 const testConfig: Record<TestType, TestChecker> = {
344     blaarg: blaargTestCheck,
345     mooneye: mooneyeAndSamesuiteTestCheck,
346     acid: acidTestCheck,
347     samesuite: mooneyeAndSamesuiteTestCheck,
348 };
349
350 const tests: Test[] = (
351     Object.entries(rawTestFiles) as [TestType, Record<SubTestType, MiniTest[]>][]
352 ).flatMap(([testType, subTests]) =>
353     (Object.entries(subTests) as [SubTestType, MiniTest[]][]).flatMap(
354         ([subTestType, miniTests]) =>
355             miniTests.map((test) => ({
356                 ...test,
357                 testType,
358                 subTestType,
359                 check: testConfig[testType],
360             })))
361     )
362 );
363
364 export default tests;

```

Appendix C

GBEmulatorShootout Contribution

As part of the evaluation of this project, some code was contributed to the open source project GBEmulatorShootout, available at <https://github.com/daid/GBEmulatorShootout/>. The pull request in which the changes were contributed can be found at <https://github.com/daid/GBEmulatorShootout/pull/19>.

The deployed results of the tool can be found at <https://nlark.github.io/GBEmulatorShootout/>. These test results are hosted on my fork of the original project, because an error occurred during the build of the original repository, and I do not have the permissions to re-run the script on the original repository. My fork has the same code as the original repository.

Although multiple files were modified to accomodate the addition of Emmy to the tested emulators, the main addition to the source is the file `emulators.emmy.py`. The other files will not be included, as I am not their sole author. These can be found on the repository previously mentioned.

C.1 Emmy Testing Code

```
1 from util import *
2 from emulator import Emulator
3 from test import *
4 from selenium import webdriver
5 import PIL.Image
6
7 class Emmy(Emulator):
8     def __init__(self):
9         super().__init__("Emmy", "https://emmy-gbc.vercel.app/", startup_time=0.5)
10
```

```

11 def setup(self):
12     # download("http://problemkaputt.de/nogmb.zip", "downloads/nogmb.zip",
13     ↪ fake_headers=True)
14     # extract("downloads/nogmb.zip", "emu/nogmb")
15     # https://chromedriver.storage.googleapis.com/109.0.5414.74/chromedriver_win32.zip
16
17     ↪ download("https://chromedriver.storage.googleapis.com/109.0.5414.74/chromedriver_win32.zip",
18     ↪ "downloads/chromedriver_win32.zip")
19     extract("downloads/chromedriver_win32.zip", "emu/chromedriver_win32")
20     self.driver = webdriver.Chrome("emu/chromedriver_win32/chromedriver.exe")
21     self.driver.get("https://emmy-gbc.vercel.app/")
22     self.driver.find_element(value="emu-speed").click()
23     self.driver.find_element(value="drawer-section-settings").click()
24
25 def isWindowOpen(self):
26     return self.driver is not None
27
28 def isProcessAlive(self, p):
29     return True
30
31 def processOutput(self, p):
32     return None
33
34 def endProcess(self, p):
35     pass
36
37 def returncode(self, p):
38     return 0
39
40 def undoSetup(self):
41     self.driver.quit()
42
43 def startProcess(self, rom, *, model, required_features):
44     systemmode = {DMG: "dmg-mode", CGB: "cgb-mode"}.get(model)
45     if systemmode is None:
46         return None
47     self.driver.find_element(value=systemmode).click()
48     rom_path = os.path.abspath(rom)
49     try:
50         self.driver.find_element(value="rom-input").send_keys(rom_path)
51         try:
52             # if an alert appeared, it means the rom is incompatible
53             self.driver.switch_to.alert.accept()
54             return None
55         except:
56             # no alert, so error thrown, so the rom is compatible
57             return self.driver
58     except Exception as e:
59         return None
60
61 # must return a pillow image object
62 def getScreenshot(self):
63     canvas = self.driver.find_element(value="emulator-frame")
64     canvas_base64 = self.driver.execute_script("return
65     ↪ arguments[0].toDataURL('image/png').substring(21);", canvas)
66
67     # decode
68     canvas_png = base64.b64decode(canvas_base64)
69     # by default, 4 canvas pixels = 1 screen pixel
70     large_image = PIL.Image.open(io.BytesIO(canvas_png))
71     # resize to 1:1
72     small_image = large_image.resize((160, 144), PIL.Image.NEAREST)
73     return small_image

```