Experimental – Hacking games

Introduction

This is not a Game Boy feature itself, but something that could make AI/bots work easier. In some of the game wrappers, we have implemented overrides of ROM data to alter the code it executes. In Mario, we can manipulate the world/level selection, while in Tetris, we can force the shape of the next Tetromino.

You can find disassemblies online, as well as tools to assist the work[1][2][3].

To make it easier to identify important RAM addresses, it might also be interesting to make a memory scanner, which can help identify changed values or exact searches[4].

The details of the Game Boy are described in the Pan Docs[5]

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Have a look at the game wrappers in PyBoy and see if it has the needed functionality. If it doesn't, it is easy to extend it.

Locate functionality to exploit Find a game to exploit, find a feature to implement or change and locate the sections in the ROM.

Create prototypes When you've found an appropriate method, try to implement it, and see if it can work.

Final polishing and testing Perform some tests to verify the solution. This includes test ROMs, but also practical game tests.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[1] `https://github.com/mattcurrie/mgbdis`
[2] `https://github.com/pret/pokered`
[3] `https://github.com/kaspermeerts/supermarioland`
[4] `https://github.com/Baekalfen/PyBoy/issues/72`
[5] `http://bgb.bircd.org/pandocs.htm`

Emulation – Improve Sub-frame Input Timing

Introduction

The current implementation puts all user input at the beginning of each frame. This is not always enough. Some games depend on more precise input timing[6].

The hardware details are described in the Pan Docs[7]

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Current input is performed in SDL2 on all platforms, and SDL2 should provide the needed support for this task. Look through the current implementation and familiarize yourself with the code. Especially the contents of `pyboy/core/interaction.py` and `pyboy/plugins/window`$_{sdl2.py}$.

Record input with timing data Record when inputs were provided in the WindowEvent objects. SDL2 might have functionality for this.

Delay and inject user input at the correct timing Modify the input handler to inject the user input at the correct timing offset.

Final polishing and testing Perform some tests to verify the solution. This includes test ROMs, which verify the interaction with the new registers, but also practical game tests. If possible, compare the video output produced to a physical Game Boy.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[6] `https://github.com/Baekalfen/PyBoy/issues/116`
[7] `http://bgb.bircd.org/pandocs.htm`

Emulation – Improve Screen Rendering

Introduction

The current implementation of PyBoy has support for rendering graphics, but it is quite limited in it's accuracy. The task will be to pass as much of the DMG Acid2[8] test as possible.

The hardware details are described in the Pan Docs[9]

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Current video rendering is performed in SDL2 on all platforms, and SDL2 should provide the needed support for this task. Look through the current implementation and familiarize yourself with the code. Especially the contents of `pyboy/core/lcd.py`.

Identify issues Try starting the test ROM with the current implementation and identify any issues.

Fix the video renderer Modify the renderer to output the correct result. Go back and identify the remaining issues, and keep iterating.

Final polishing and testing Perform some tests to verify the solution. This includes test ROMs, which verify the interaction with the new registers, but also practical game tests. If possible, compare the video output produced to a physical Game Boy.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[8] https://github.com/mattcurrie/dmg-acid2

[9] `http://bgb.bircd.org/pandocs.htm#videodisplay`

Emulation – Color

Introduction

The current implementation of PyBoy only supports the original black-and-white Game Boy from 1989. The Game Boy Color shares almost all of the same architecture of the original Game Boy, but adds a little more memory, a modified video controller and double the CPU frequency. The current implementation trivially supports doubling the CPU speed, so this won't be an issue.

The details are described in the Pan Docs[10]

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Current video rendering is performed in SDL2 on all platforms, and SDL2 should provide the needed support for color. Look through the current implementation and familiarize yourself with the code.

Change PyBoy to make it start a GBC ROM The Game Boy ROMs do a few checks to determine it is on Game Boy Color (GBC) hardware. If these checks fail, it shows a warning and stops. Find these checks in the Pac Docs, and modify PyBoy to make it past the non-color warning message.

Add the extra memory The Game Boy Color had more memory than the original Game Boy. Add these extra memory banks to the system, and the control register (see FF4F in Pan Docs). Memory banks are present multiple places already, so it won't be hard to implement.

Modify the video renderer Add the needed functionality for the emulator to read the color data when rendering. Also add the functionality of the special registers, which controls the color palettes.

Final polishing and testing Perform some tests to verify the solution. This includes test ROMs, which verify the interaction with the new registers, but also practical game tests. If possible, compare the video output produced to a physical Game Boy Color.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[10] `http://bgb.bircd.org/pandocs.htm#videodisplay`

Emulation – Link Cable

Introduction

The current implementation of PyBoy lacks support for the Link Cable. The Link Cable was a primitive serial connection between two Game Boys. It was used to play against others etc.

The details are described in the Pan Docs[11]

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation The current code simply prints out the data being pushed to the serial port. But look at the code for the motherboard, and look how special registers are implemented.

Determine what is possible The way data is transferred, presents some problems for TCP/UDP Link Cable emulation. Investigate if it is possible to do this over a network, under which circumstances it might be, or if both emulators need to be on the same computer.

Implement registers and internal clock Two registers are used for the Link Cable. One for control and one for data. Implement both, and the accompanied "Internal Clock", which defines the connection speed.

Implement emulation layer Emulate the Link Cable in the way you found appropriate earlier. This might include one or more implementations, for games which uses the Link Cable differently.

Final polishing and testing Perform some tests to verify the solution. This might include test ROMs, which verify the interaction with the new registers, but also practical game tests.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[11] `http://bgb.bircd.org/pandocs.htm#serialdatatransferlinkcable`

Experimental – AI

Introduction

This is not a Game Boy feature itself, but something that is easier to approach through emulation. What I imagine, is a neural network – or maybe a simple, handwritten bot – which can play the Game Boy autonomously.

PyBoy already supports external controls to report objects on the screen and sending input to the Game Boy[12][13]. It supports running at unlimited speed and without rendering the display to speed up the learning process.

The details of the Game Boy are described in the Pan Docs[14]

There are already examples of other people implementing AIs for console games[15] [16].

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Have a look at the 'botsupport' module and the "game wrappers" in PyBoy and see if it has the needed functionality. If it doesn't, it is easy to extend it. There is also a crude Tetris and Mario example in the repo to get inspiration from.

Figure out a method If you decide to solve the task using machine learning, search online for a fitting software package to help with it – maybe Tensorflow or PyTorch will work. Create a prototype to find out what is needed to connect the machine learning framework and PyBoy.

Implement the algorithm When you've found an appropriate method, try to implement it, and see if it can work.

Final polishing and testing Perform some tests to verify the solution. This might include test ROMs, which verify the interaction with the new registers, but also practical game tests.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[12] `https://docs.pyboy.dk/botsupport/index.html`
[13] `https://docs.pyboy.dk/plugins/index.html`
[14] `http://bgb.bircd.org/pandocs.htm`
[15] `https://www.youtube.com/watch?v=qv6UVOQ0F44`
[16] `https://www.youtube.com/watch?v=iakFfOmanJU`

Experimental – Debugger

Introduction

To improve functionality on the emulator, we will need good debugging tools. The current code does not have a debugger. Some of the functionality could be a plain memory view, a view with disassembled machine code or improved tools to see call stacks, interrupts, graphics settings, memory scanner[17] and so on.

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Have a look at the `debug` module in PyBoy and see if it has the needed functionality. If it doesn't, it is easy to extend it. Familiarize yourself with the CPU architecture and instruction set – it's quite simplistic.

Locate the needed data in the emulator Make prototypes to see if you can find the needed information you want to present to the user. This could be memory values, opcode-to-name translations, CPU registers, special purpose registers and so on.

Find the data Make prototypes to see if you can find the needed information you want to present to the user.

Present the data Either make a simple presentation of the data in the terminal, or a more advanced graphical interface.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[17] `https://github.com/Baekalfen/PyBoy/issues/72`

Experimental – New Game Wrappers

Introduction

This is not a Game Boy feature itself, but something that will make AI/bots work easier. We have already implemented a set of wrappers[18] which present an easy interface to some popular games – Tetris, Super Mario Land and Kirby. But we could always use more. You can choose exactly which game you want to support – or improve one of the existing wrappers.

In the Super Mario Land wrapper, it can quickly identify enemies, Mario, boundaries, score, time left and so on. This is an immense help for developers of the AIs or bots.

To make it easier to identify important RAM addresses, it might also be interesting to make a memory scanner, which can help identify changed values or exact searches[19].

The details of the Game Boy are described in the Pan Docs[20]

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Have a look at the 'botsupport' module in PyBoy and see if it has the needed functionality. If it doesn't, it is easy to extend it. There is also a crude Tetris bot to get inspiration from.

Figure out a method Figure out how to best represent the data on the screen for the AI/bot developer. Find the best performing way to extract the necessary information from PyBoy – use the BotSupport module as much as possible. If PyBoy lacks the necessary features, figure out how to build an extension.

Implement the algorithm When you've found an appropriate method, try to implement it, and see if it can work.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[18] `https://docs.pyboy.dk/plugins/index.html`
[19] `https://github.com/Baekalfen/PyBoy/issues/72`
[20] `http://bgb.bircd.org/pandocs.htm`

Completed Projects

Experimental – Rewind Time

Introduction

This is not a Game Boy feature itself, but something we can do when emulating. I imagine a feature, where it is possible to go back in time and redo something that went wrong in the game. It could be fine-grained at CPU-cycle level, less fine at each produced frame (60 FPS), or coarse at larger intervals. It depending on what is possible to implement.

The details of the Game Boy are described in the Pan Docs[21]

There are already examples of other people implementing a rewind feature[22].

Tasks

Read and find documentation The Pan Docs should provide all the needed information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Look at the current implementation Familiarize yourself with the PyBoy code base and look for a good place to implement the feature, and at which granularity it is possible.

Figure out an algorithm How is the data of each level of granularity going to be stored, and how is it going to be loaded? Will you track changes on a byte-level, or simply save large chunks of memory at an interval? How much space will it use, and how will it be managed? Is compression needed?

Implement the algorithm When you've found an appropriate algorithm, try to implement it in PyBoy. See if there are any unexpected issues.

Final polishing and testing Perform some tests to verify the solution. This might include test ROMs, which tests the system integrity in general, while the time is rewinded, but also practical game tests.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[21] `http://bgb.bircd.org/pandocs.htm`

[22] `https://binji.github.io/2017/12/31/binjgb-rewind.html`

Emulation – Sound

Introduction

The current implementation of PyBoy does not emulate the built-in sound controller.

The sound controller supports 4 primitive types of tone and noise generators. These are described in detail in the Pan Docs[23].

Tasks

Read and find documentation The Pan Docs do provide a lot of information about the internals of the Game Boy, but sometimes, more information is needed. Read the Pan Docs, and look for other possible sources online. Maybe even look at other emulators for inspiration.

Find library, which supports the requirements Current video rendering is performed in SDL2 on all platforms, and SDL2 should provide support for sound as well. See if SDL2 has the required functionality to implement the tone generators. Alternatively, find a way for Python to generate the sound as needed.

Implement prototype in standalone Python script It might be hard to implement the emulated sound controller directly into PyBoy. I recommend to implement it separately, if it is too big of a task to do both at once.

Find test ROMs or games for reference To consistently debug the sound controller, you should find some dedicated test ROMs or a game which has simple sound usage. This will make it easier to debug possible issues. We will later test on games with complex sounds.

Modify PyBoy to support the sound control registers The Game Boy interfaces with the sound controller through some dedicated registers in memory (see Pan Docs). These will need to be implemented and hooked up to a placeholder for the sound controller code.

Move the prototype into PyBoy When the registers are in place, move the prototype into the Game Boy and see if we can produce sound. Test if there are any oversights in the implementation, and keep improving the code until it works.

Final polishing and testing Perform some tests to verify the solution. This includes test ROMs, which verify the interaction with the sound control registers, but also listening tests. If possible, compare the sounds produced to a physical Game Boy.

Conclude results Sum up what works, and what didn't work, and what should be done by the developer, who takes over.

[23] `http://bgb.bircd.org/pandocs.htm#soundcontroller`