

PA3实验报告

实验进度

我完成了所有的必做内容。

遇到的问题及思考

- 由于每一小段内容实现后都会有一个可以用于验证实验正确性的测试，因此整个PA3基本没有遇到太大的困难。
- 让ChatGPT写了份Python代码帮助解析elf文件，在其中寻找错误地址所在的函数，发现在调试SDL相关代码时发现大部分错误都在得知导致错误退出的函数后能很快解决（用修改过的ftrace能达到同样效果，但我懒得改了）

必答题

- 理解上下文结构体的前世今生

c 指向的Context结构在 abstract-machine/am/include/arch/riscv.h 中定义。这个上下文结构与所选指令集架构（本例中为RISCV32）有关，是程序进行异常处理（运行 ecall）时所保存的。

这个上下文结构的定义如下：

```
struct Context {
    uintptr_t gpr[NR_REGS], mcause, mstatus, mepc;
    void *pdir;
}
```

其中 gpr[NR_REGS], mcause, mstatus, mepc 都在 abstract-machine/am/src/riscv/nemu/trap.s 中赋值，pdir 在PA3中未出现，不知道在哪里赋值。具体地，trap.s 中定义了 STORE 等宏来批量对 gpr 中的元素进行操作，通过

```
csrr t0, mcause
csrr t1, mstatus
csrr t2, mepc

STORE t0, OFFSET_CAUSE(sp)
STORE t1, OFFSET_STATUS(sp)
STORE t2, OFFSET_EPC(sp)
```

将当前一般32个寄存器的值保存在 `gpr[NR_REGS]`，并将 mcause, mstatus, mepc 的值保存在对应处。被保存的 mcause, mstatus, mepc 的值之前在 nemu/src/isa/riscv32/system/intr.c 中被设置到 riscv32_CPU_state 结构体格式的 CPU 中。

`riscv.h` 中定义了被保存的 `Context` 结构，`trap.s` 将系统中断所要保存的信息保存到该结构体中（并跳转到异常处理函数），上述讲义文字介绍了利用 `Context` 进行异常处理的流程，我刚刚在NEMU中实现的新指令则是利用 `Context` 中信息对异常处理的实现。

- 理解穿越时空的旅程

从 `yield test` 调用 `yield()` 开始，到从 `yield()` 返回的期间，这一趟旅程具体经历了什么？软(AM, `yield test`)硬(NEMU)件是如何相互协助来完成这趟旅程的？你需要解释这一过程中的每一处细节，包括涉及的每一行汇编代码/C代码的行为，尤其是一些比较关键的指令/变量。事实上，上文的必答题“理解上下文结构体的前世今生”已经涵盖了这趟旅程中的一部分，你可以把它的回答包含进来。

软：

`am-kernels/tests/am-tests/src/tests/intr.c - yield()` 被调用，翻译为

硬：

`nemu/src/isa/riscv32/inst.c - decode_exec(Decode *s)`：中的至少两条指令：`R(17)/$a7` 的值被设为 `-1`，然后调用 `ecall`，将 `R(17)` 作为 `NO`，当前pc值作为 `epc` 调用

`nemu/src/isa/riscv32/system/intr.c - isa_raise_intr(NO, epc)`：将 `CPU.mepc` 按照异常号设为 `epc/epc + 4`，将 `CPU.mcause` 设为异常号 `NO`，（这里应该设置 `CPU.mstatus`，但我未实现）并返回异常处理函数入口地址 `CPU.mtvec`；

`nemu/src/isa/riscv32/inst.c - decode_exec(Decode *s)`：将 `isa_raise_intr` 的返回值作为跳转地址，跳转到了之前设置好的异常处理函数

`abstract-machine/am/src/riscv/npc/trap.S - __am_asm_trap`：将所有普通寄存器的值（应该只需要保存其中四个吧...）、三个m开头的特殊寄存器的值保存到栈，将下一个调用函数的第一个（唯一一个）参数 `Context *c` 保存到 `a0`，并调用

软：

`/abstract-machine/am/src/riscv/nemu/cte.c - __am_irq_handle(Context *c)`，`*c` 为 `a0`，指向之前保存到栈上的数据，根据 `c->mcause` 设置 `ev.event` 的值，并调用在 `/abstract-machine/am/src/riscv/nemu/cte.c - cte_init` 中设置的 `user_handler`

`am-kernels/tests/am-tests/src/tests/intr.c - simple_trap`，此处根据上一步设置的 `ev.event` 的值为 `EVENT_YIELD` 知发生自陷事件，函数输出 `y` 并返回。

- hello程序是什么，它从而何来，要到哪里去

hello程序一开始在电脑存储中，位置为 `~/ics2023/navy-apps/tests/hello/hello.c`，通过在 `navy-apps` 目录运行 `make ARCH=$ISA-nemu update` 在 `navy-apps/tests/dummy/` 目录下执行 `make ISA=$ISA` 并将 `navy-apps/tests/hello/build/hello-$ISA` 手动复制并重命名为 `nanos-lite/build/ramdisk.img`，可将 `hello.c` 编译为elf格式并将其放入 `nanos-lite/build/ramdisk.img` 中，程序在该文件中的偏移量存储在 `nanos-lite/src/files.h` 中为0。在编译运行nanos-lite时，Makefile根据 `nanos-lite/src/resources.s` 中的信息将 `ramdisk.img` 中的信息放入 `nanos-lite-$ISA-nemu.elf` 中的 `.data` 段，`ramdisk.img` 前后的标签为 `ramdisk_start` 和 `ramdisk_end`。在 `nanos-lite` 运行到 `init_proc` 时，`naive_upload` 被调用，该函数首先根

据文件名 `bin/hello` 找到文件描述符，在根据描述符以及 `files.h` 中的内容找到 `hello` 程序在磁盘中的偏移量，将需要加载的程序段放入内存中，最后跳转到 `ehdr.e_entry` 指示的位置运行第一条指令。

`hello` 程序首先调用 `write(1, "Hello World!\n", 13);` 语句，这会调用系统进程，与上一题“理解穿越时空的旅程”类似，但

`a7` 的值被设置为 `1`, `a0` 的值被设置为 `1` (表示是 `stdout`) , `a1` 的值被设置为字符串 `Hello World!\n` 的首地址, `a2` 的值被设置为 `13`。经历一些参数传递后运行到 `nanos-lite/src/syscall.c`，根据 `a[0]` 的值 (为 `4`) 运行 `fs_write`，三个参数为 `fd`, `buf`, `len`。在 `nanos-lite/src/fs.c` 中，由于 `file_table[1].write` 为 `serial_write`，故其会调用 `nanos-lite/src/device.c` 中的该函数，并最后用 `putch` 输出 `buf` 处长度至多为 `len` 的内容。用 `printf` 输出时会通过一系列转化转为 `write`，接下来的过程相似。

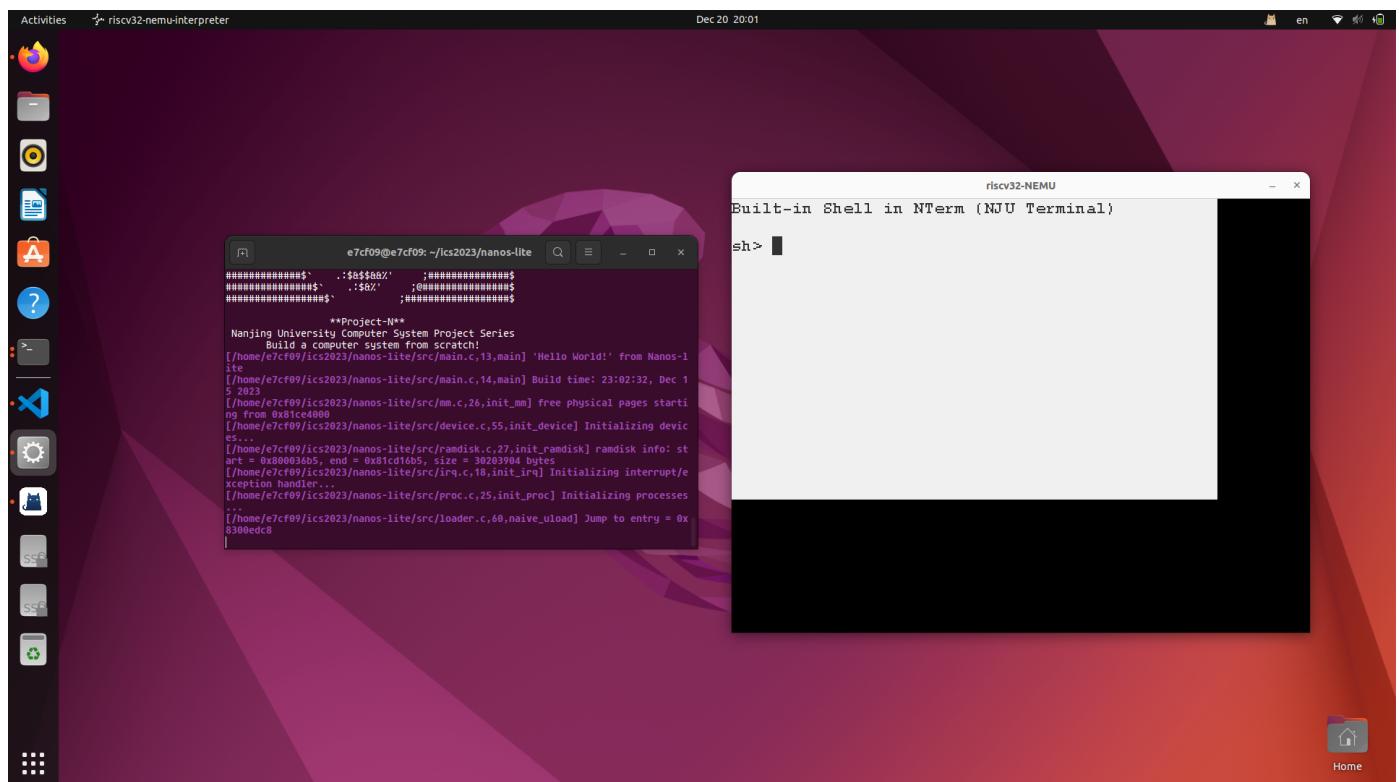
- 仙剑奇侠传究竟如何运行

读出仙鹤的像素信息时所用函数为 `PAL_MKFReadChunk`，该函数调用了库函数 `fseek` 与 `read->` 在 `libos` 中触发相应的 `syscall->Nanos-lite` 中进行实际读操作。在这个过程中，`am` 及其更底层的 `nemu` 为上述所有操作提供支撑：`nemu` 运行上述操作被编译为的 `riscv32` 代码，`am` 的 `trm` 提供运行环境（入口？）

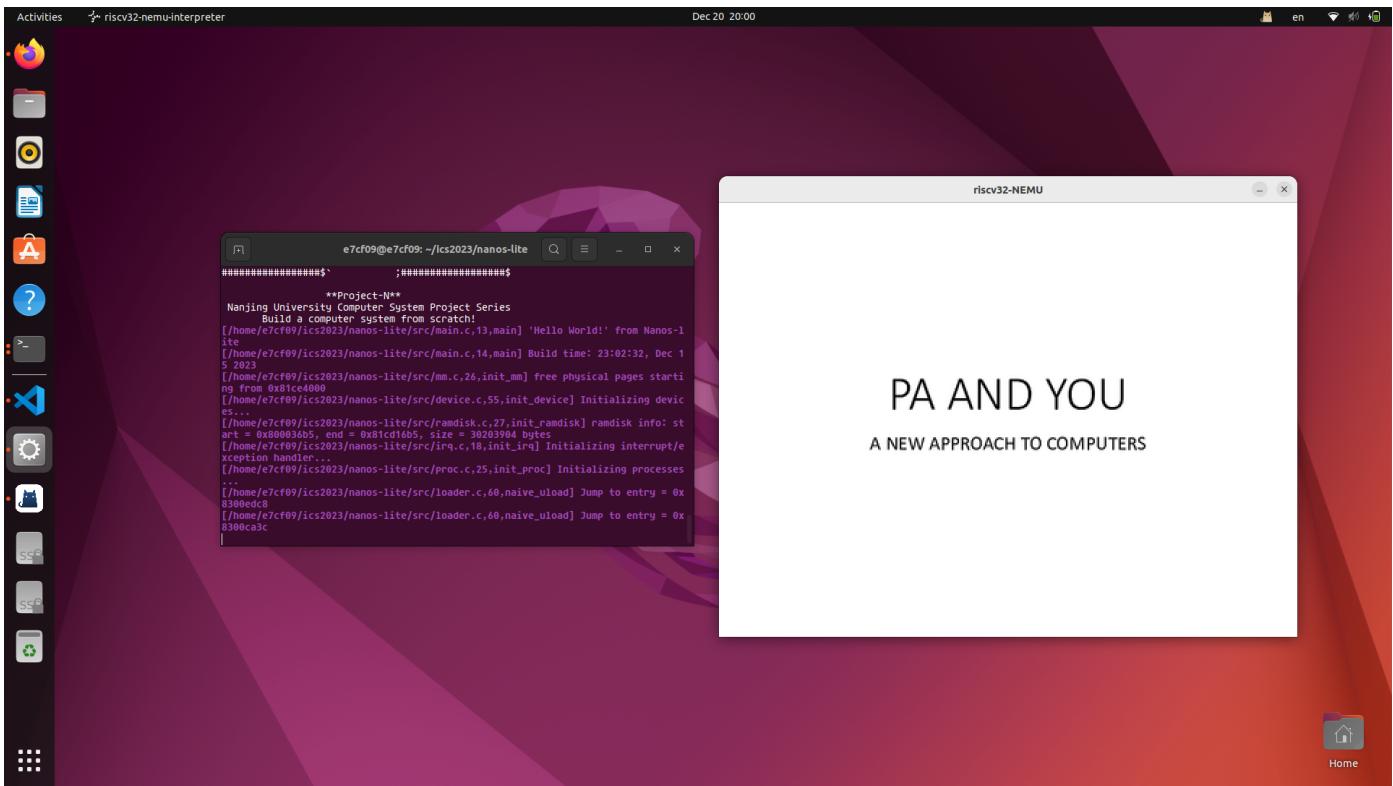
绘制仙鹤时所用函数为 `PAL_RLEblitToSurface->` 该函数调用了库函数 `write->` 在 `libos` 中触发相应的 `syscall->Nanos-lite` 中进行实际写操作。写操作还使屏幕更新，从而绘制出新的一帧。在这个过程中，`am` 及其更底层的 `nemu` 为上述所有操作提供支撑：`nemu` 运行上述操作被编译为的 `riscv32` 代码，`am` 的 `trm` 提供运行环境（入口？）

- 运行测试

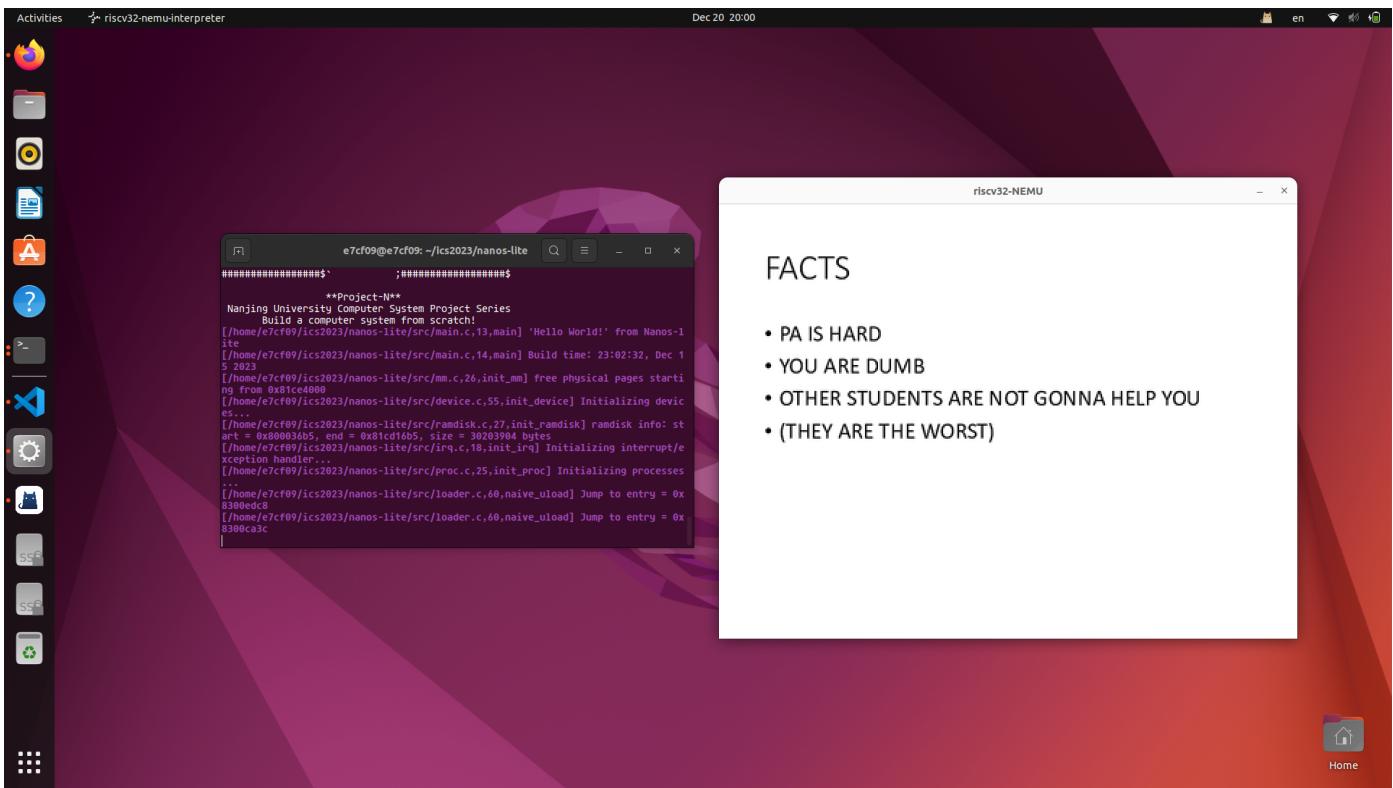
在 `nanos-lite` 目录运行 `make ARCH=$ISA-nemu update;make ARCH=$ISA-nemu run` 可得如下界面：



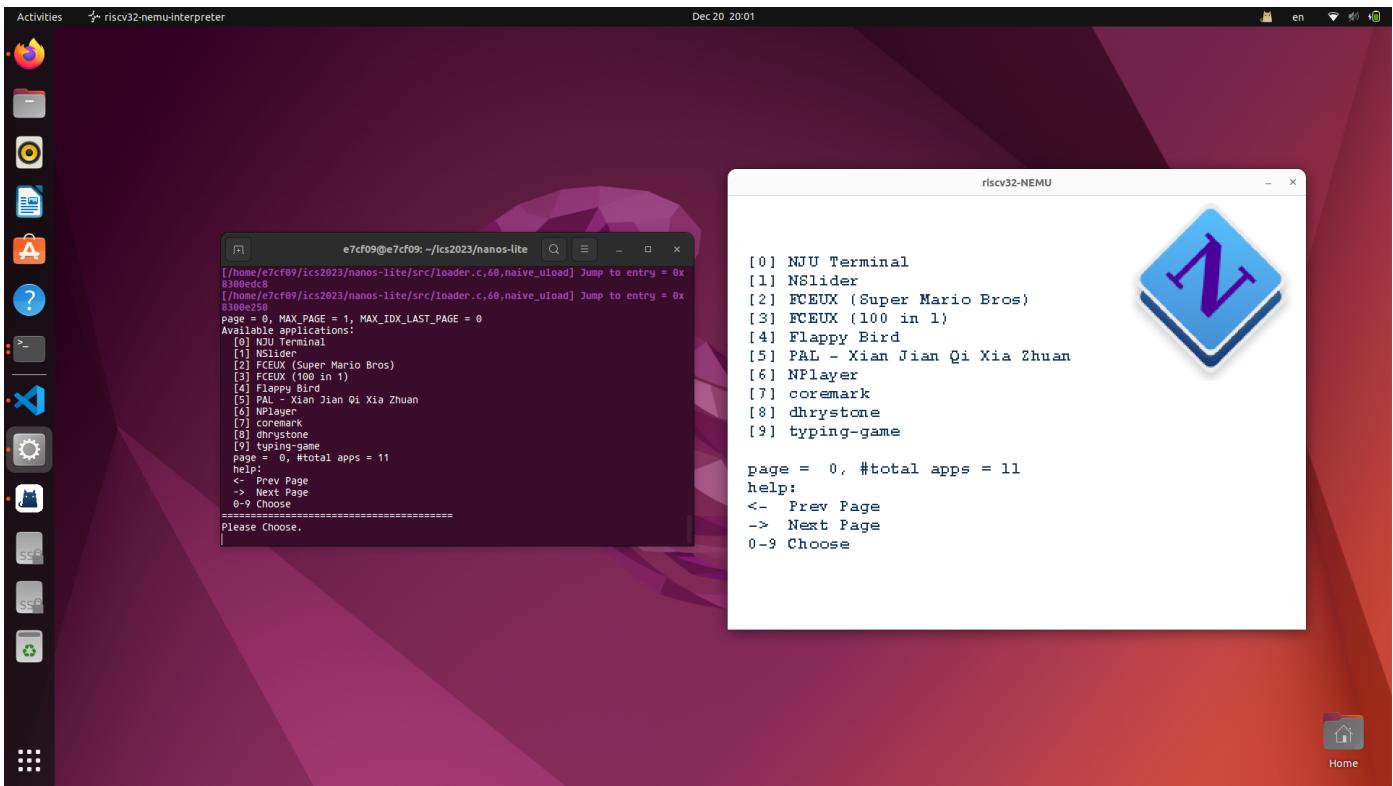
在该界面输入 `nslider` 并按下回车：



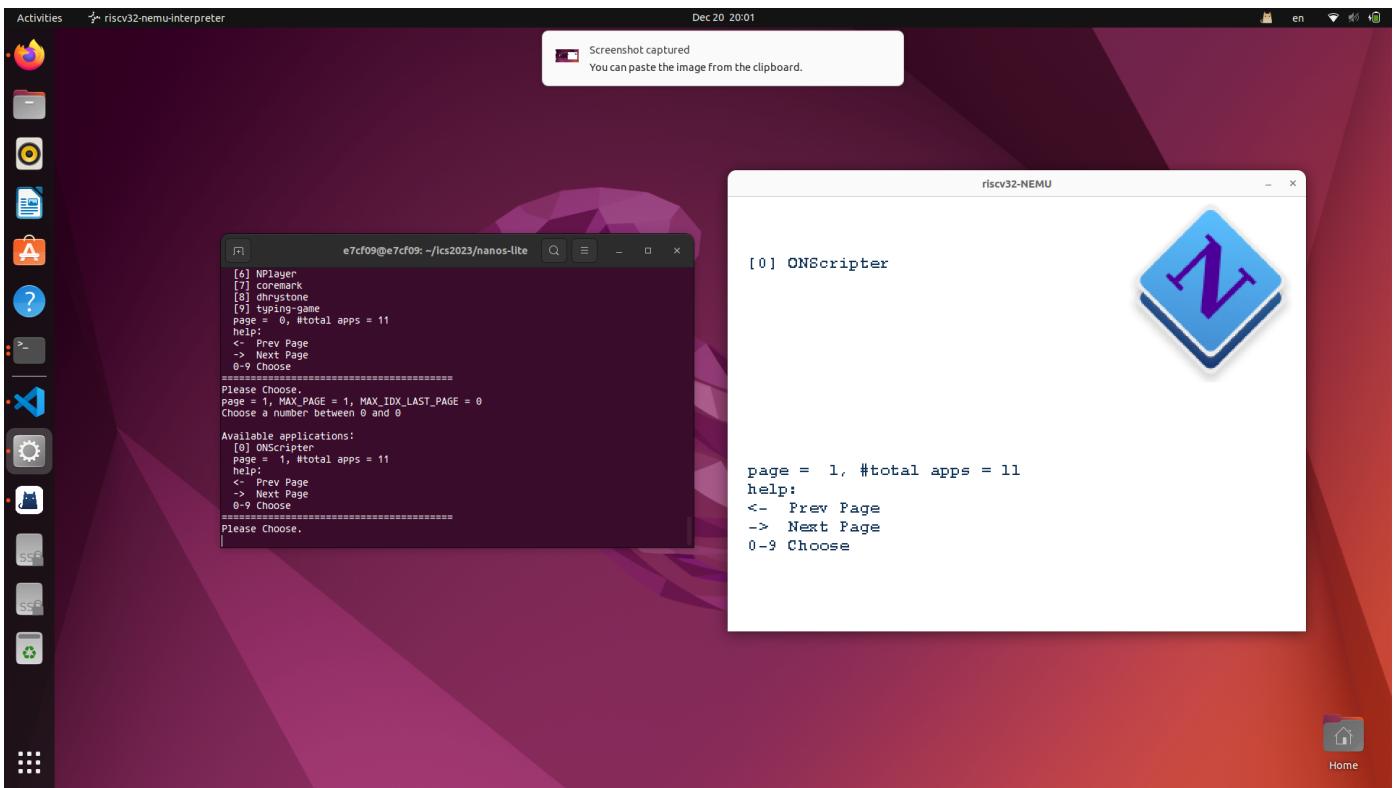
在该界面按下 DOWNKEY : (PPT内容为对2018年TGA最佳独立游戏Celeste的致敬, 可能的含义非我本意)



退出, 重新进入 nterm, 输入 menu 并按下回车:



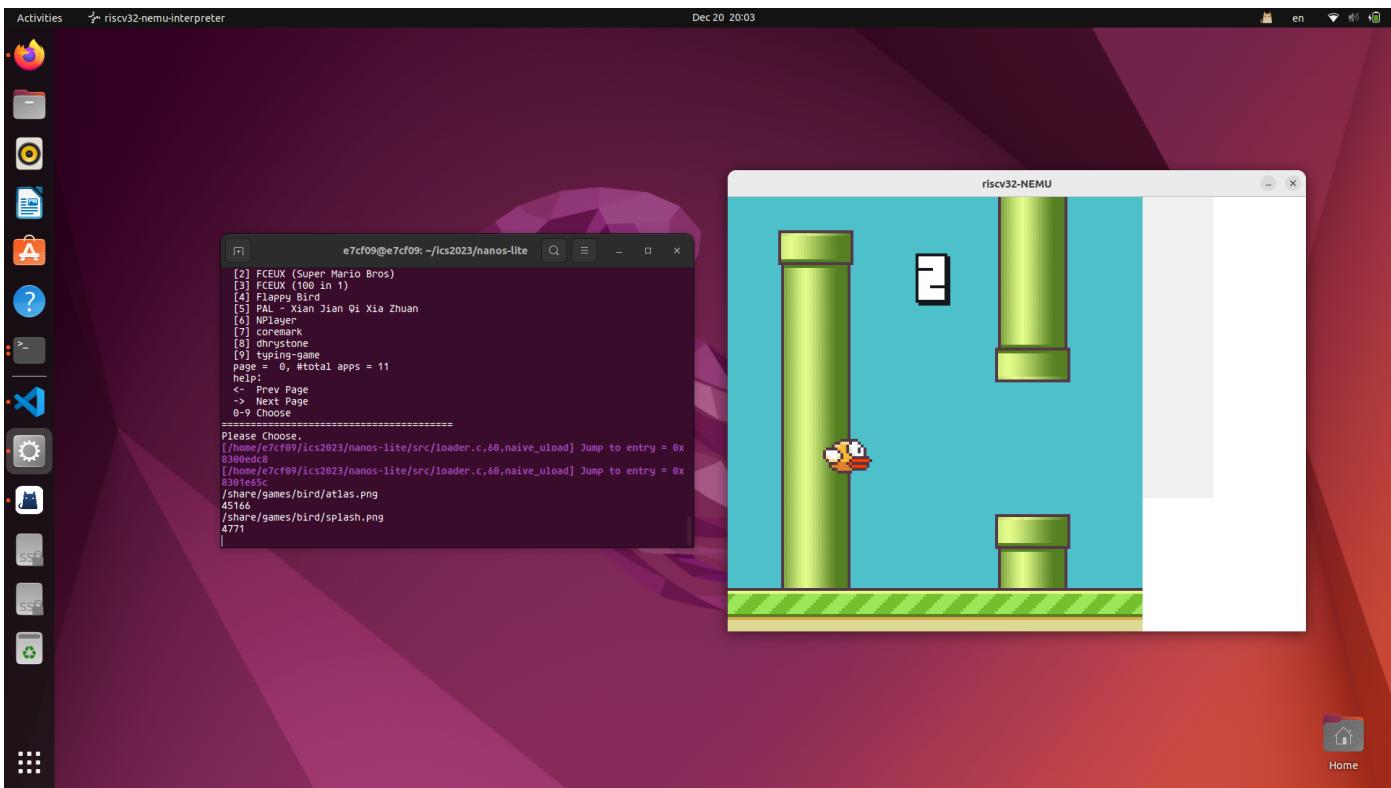
按下 RIGHTKEY :

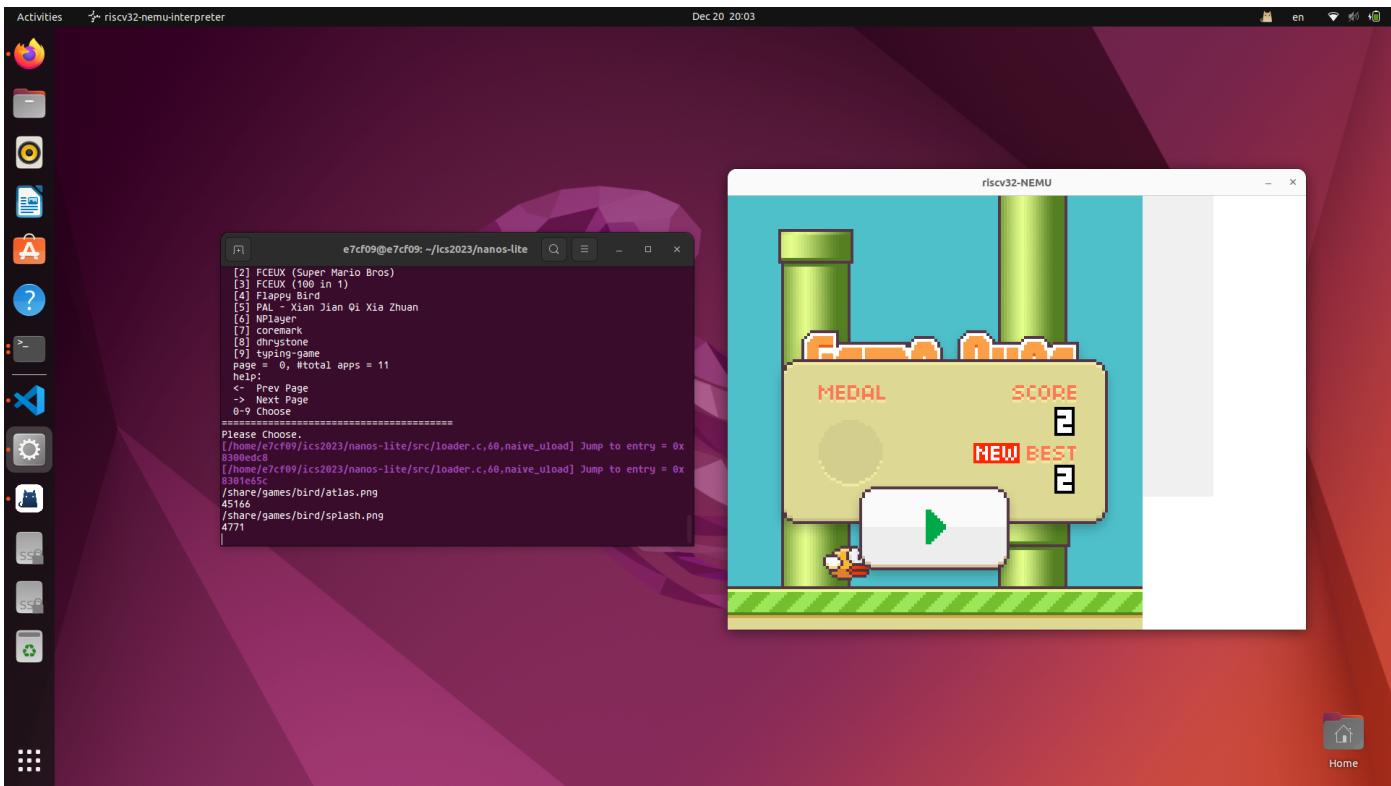


按下 LEFTKEY , 并按下 0 回到 nterm ; 输入 bird 并按下回车：

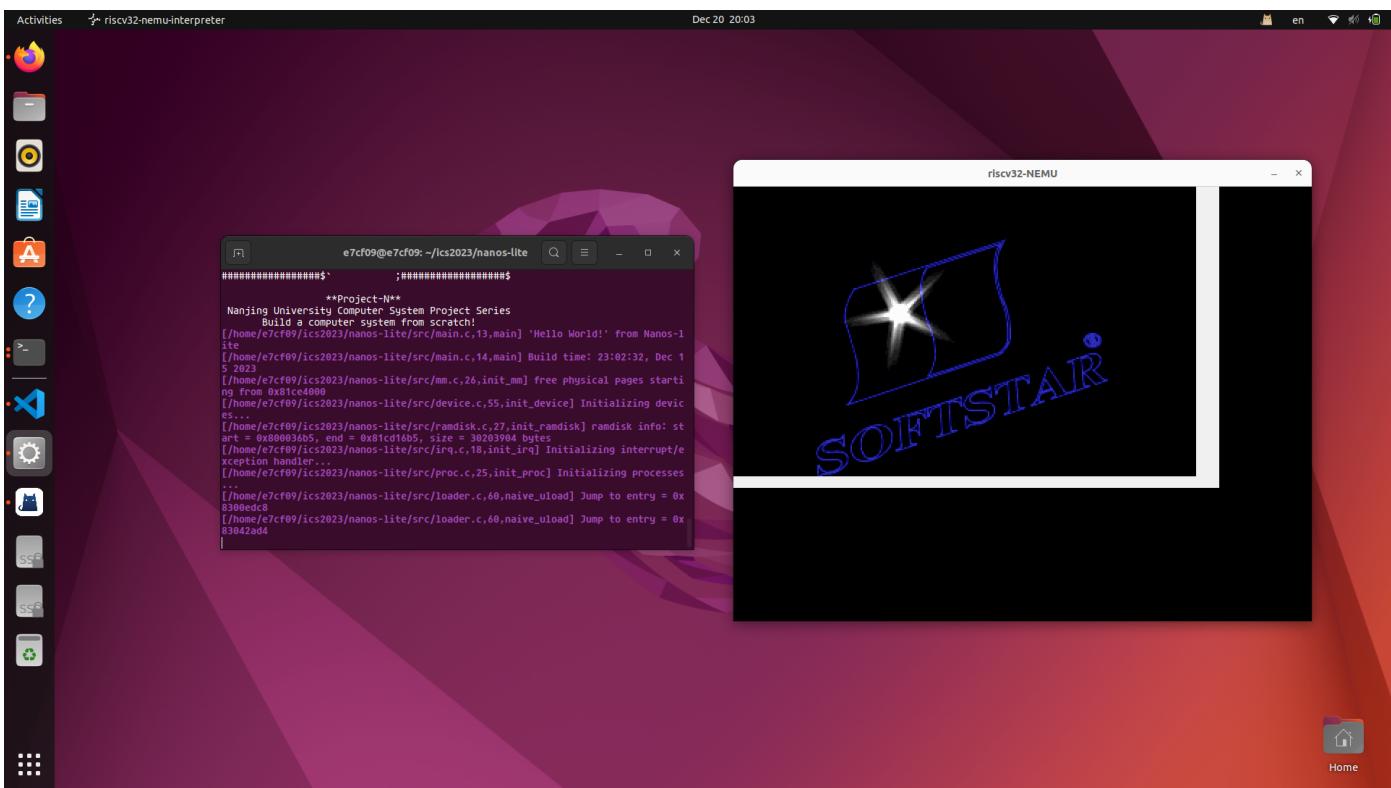


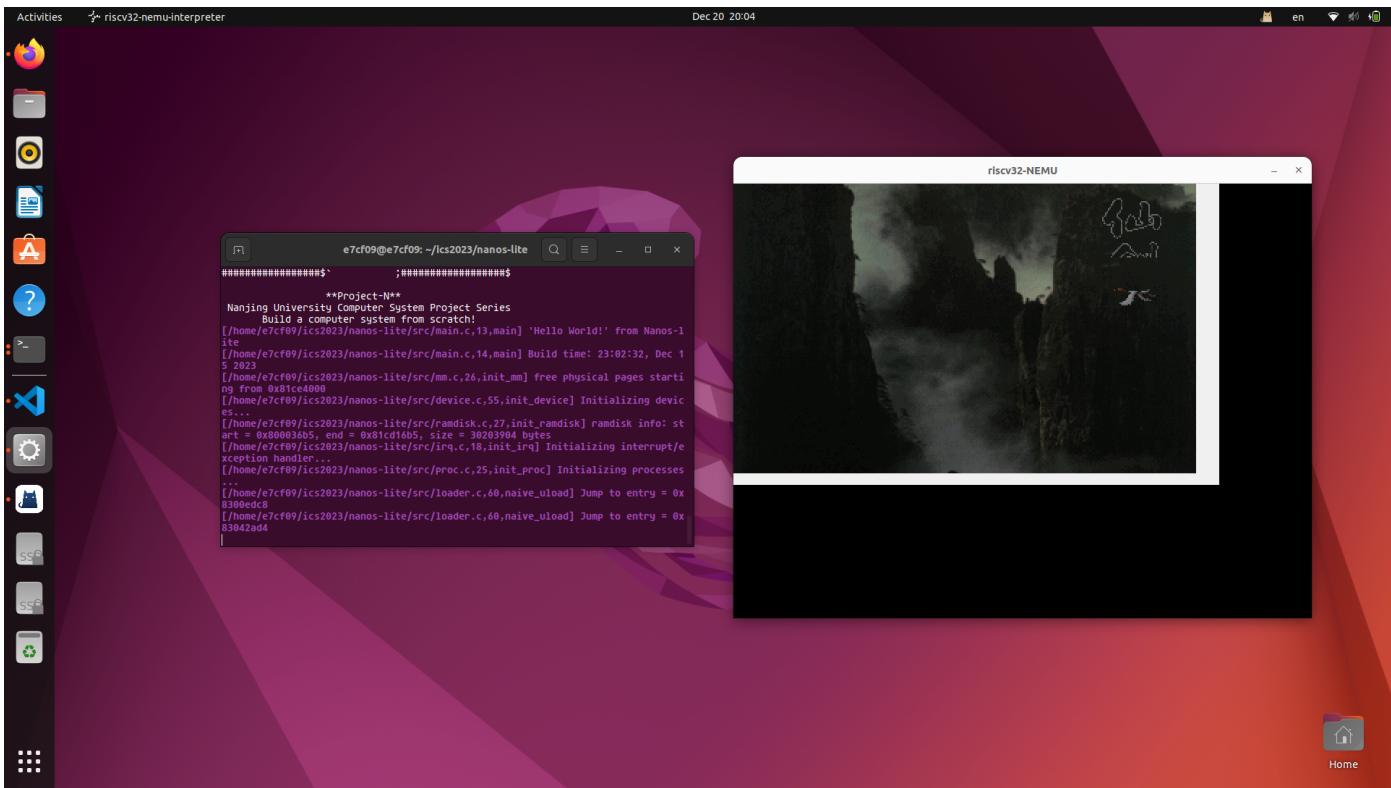
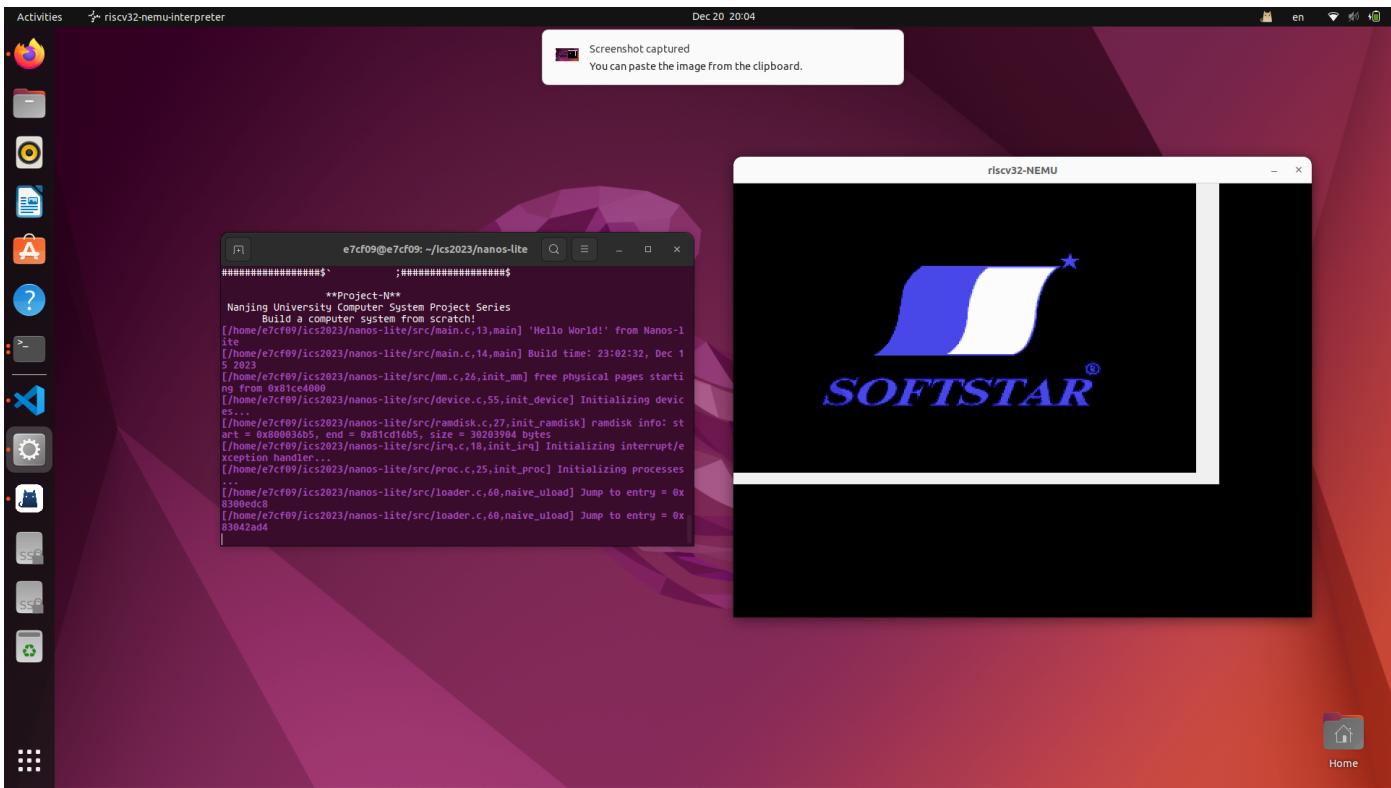
试玩结果：

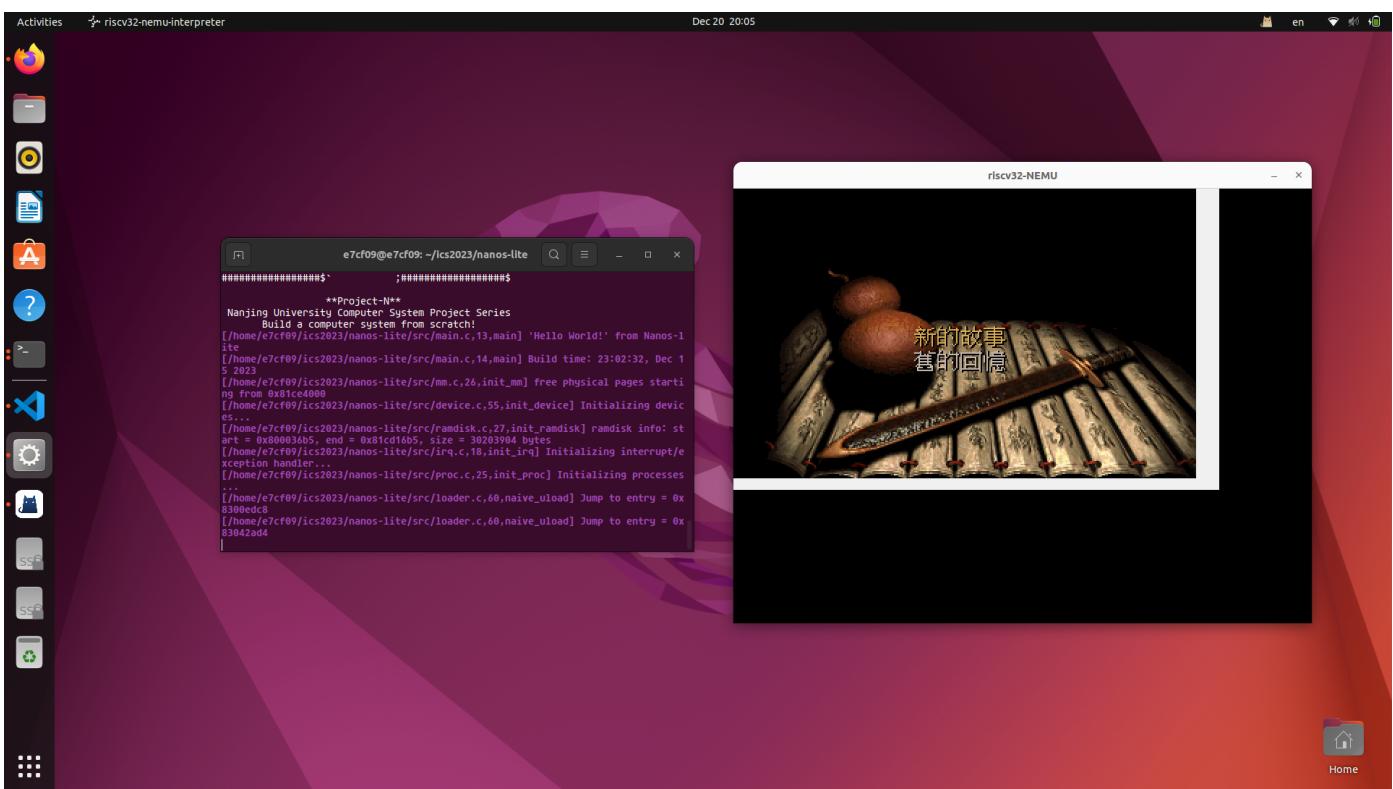
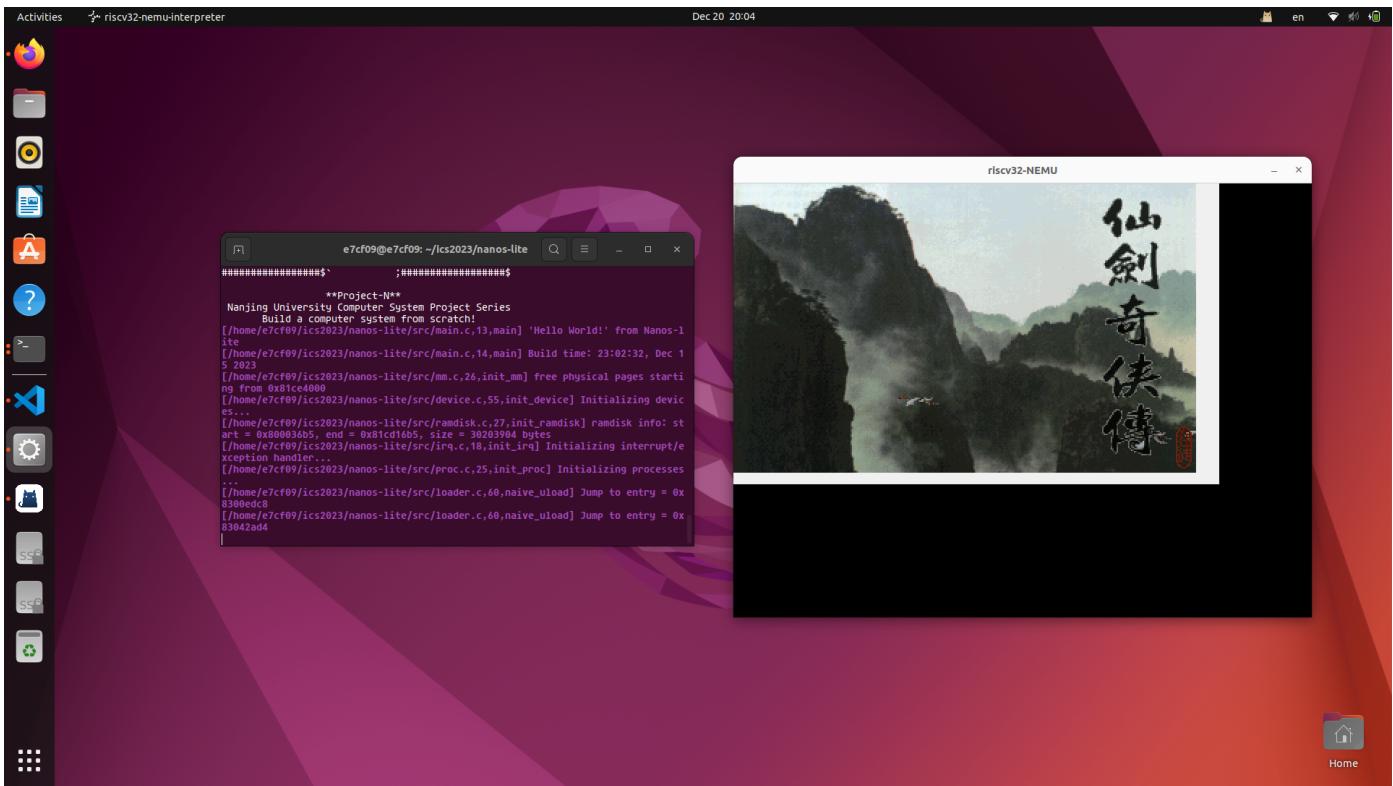


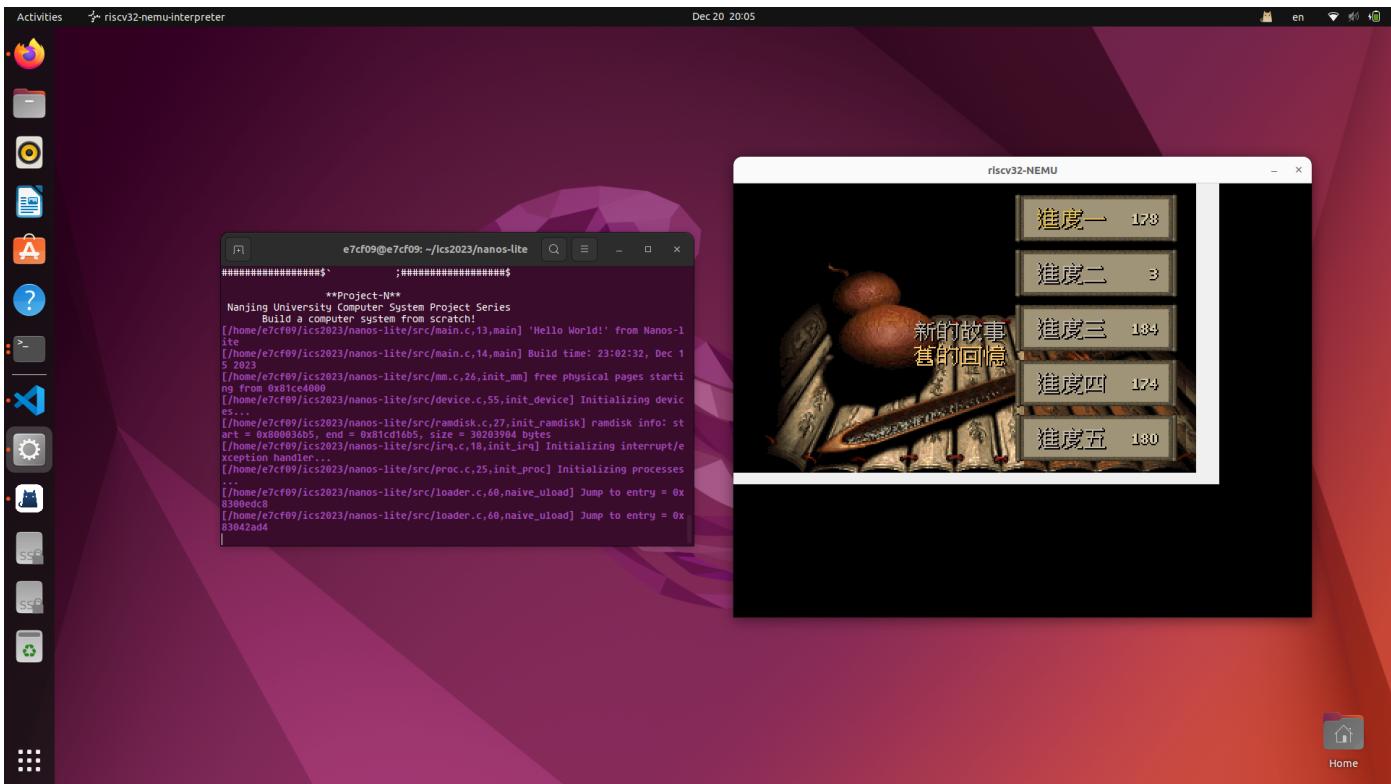


退出，重新进入 nterm，输入 pal 并按下回车：

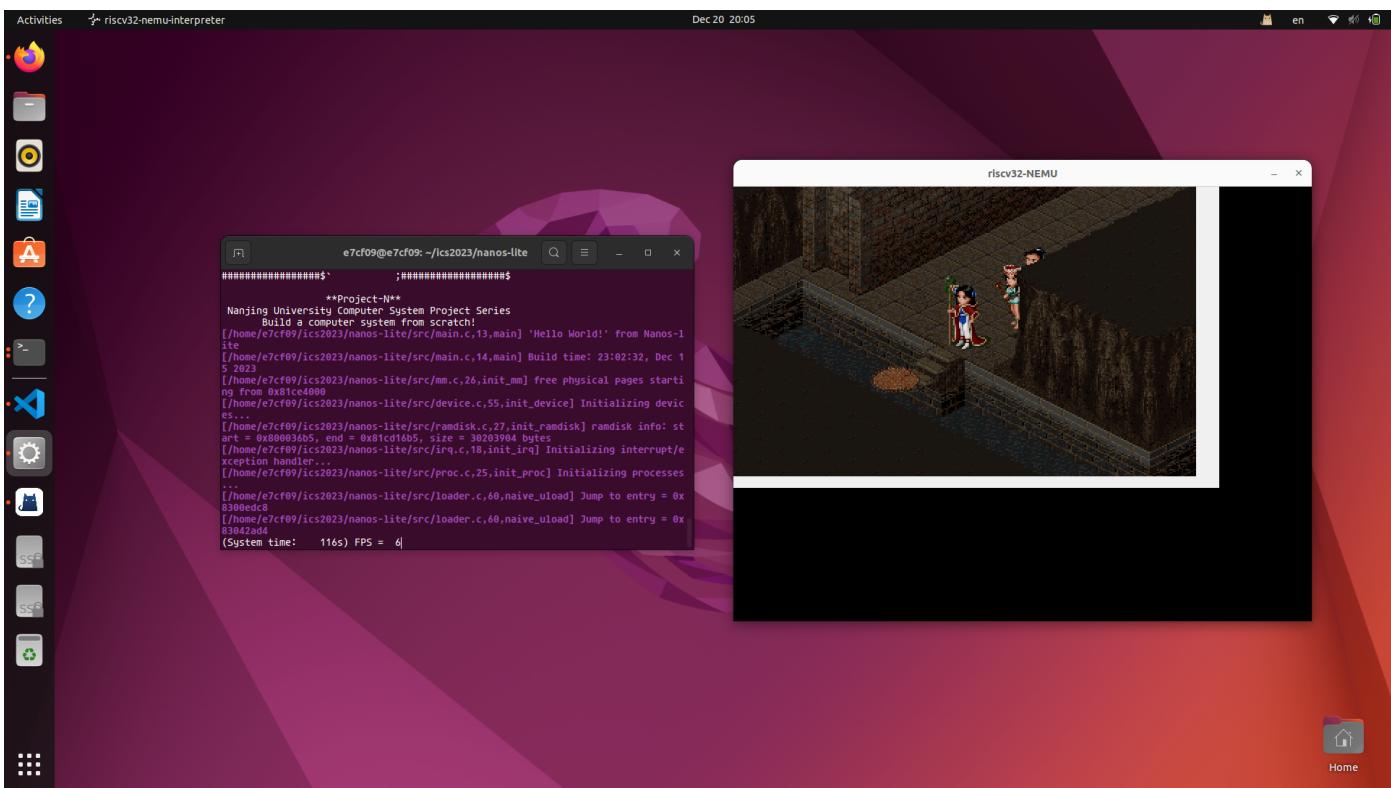


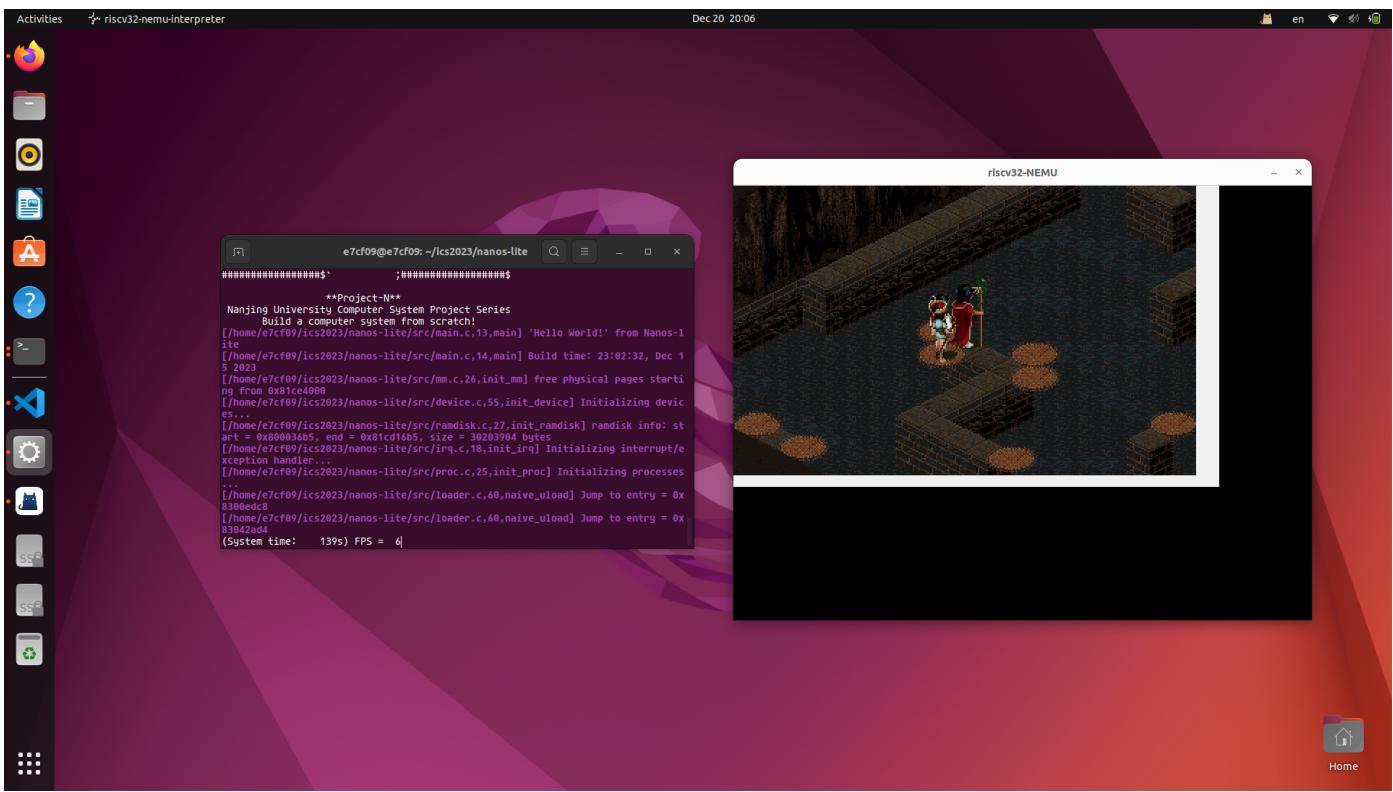
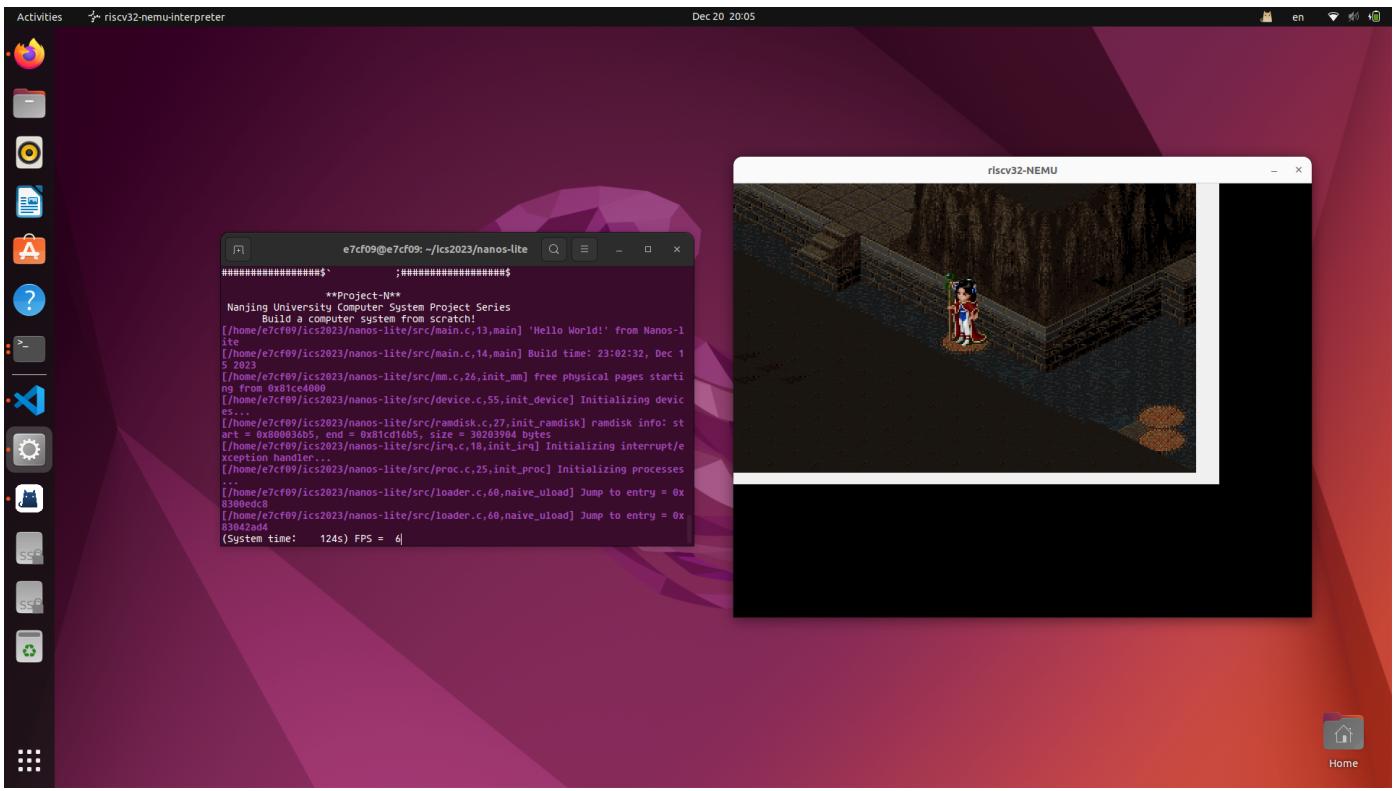




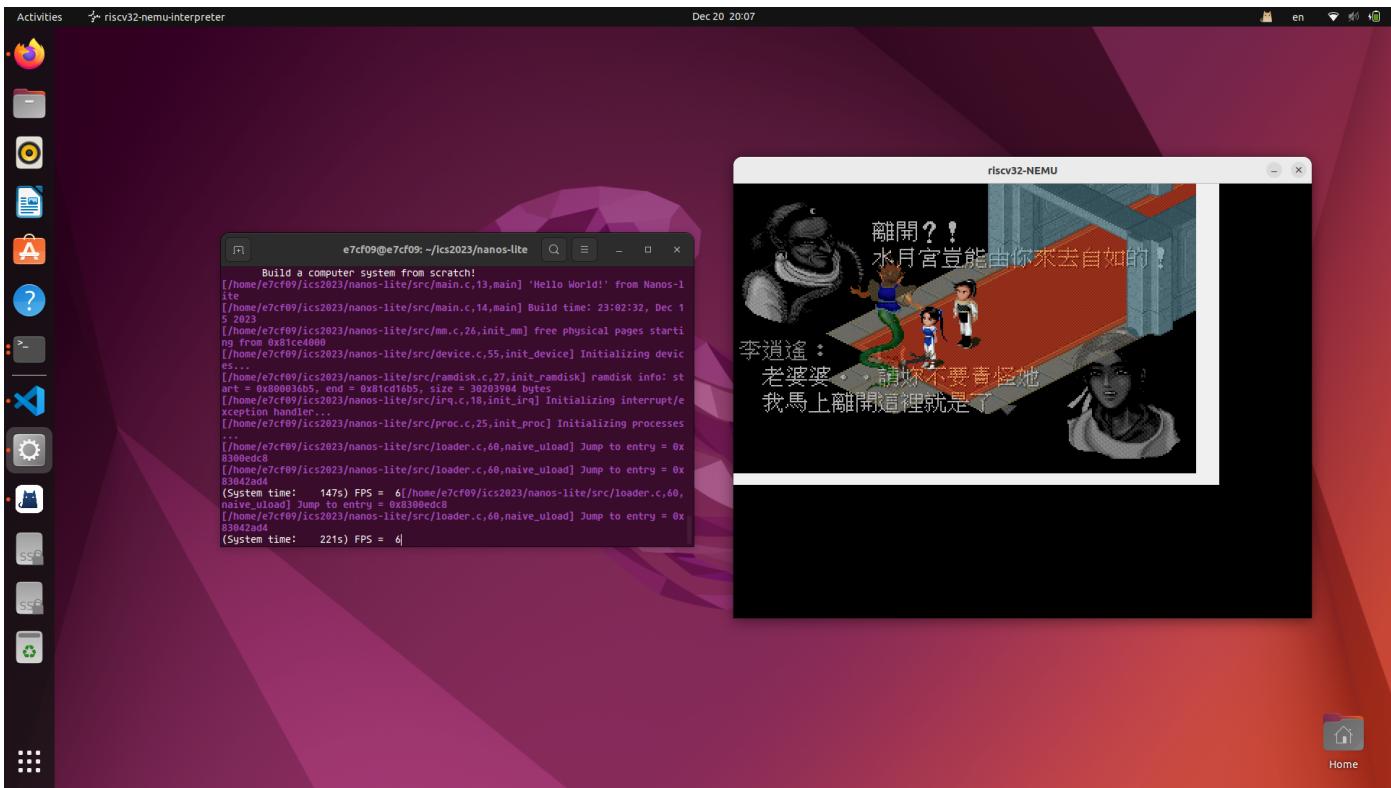
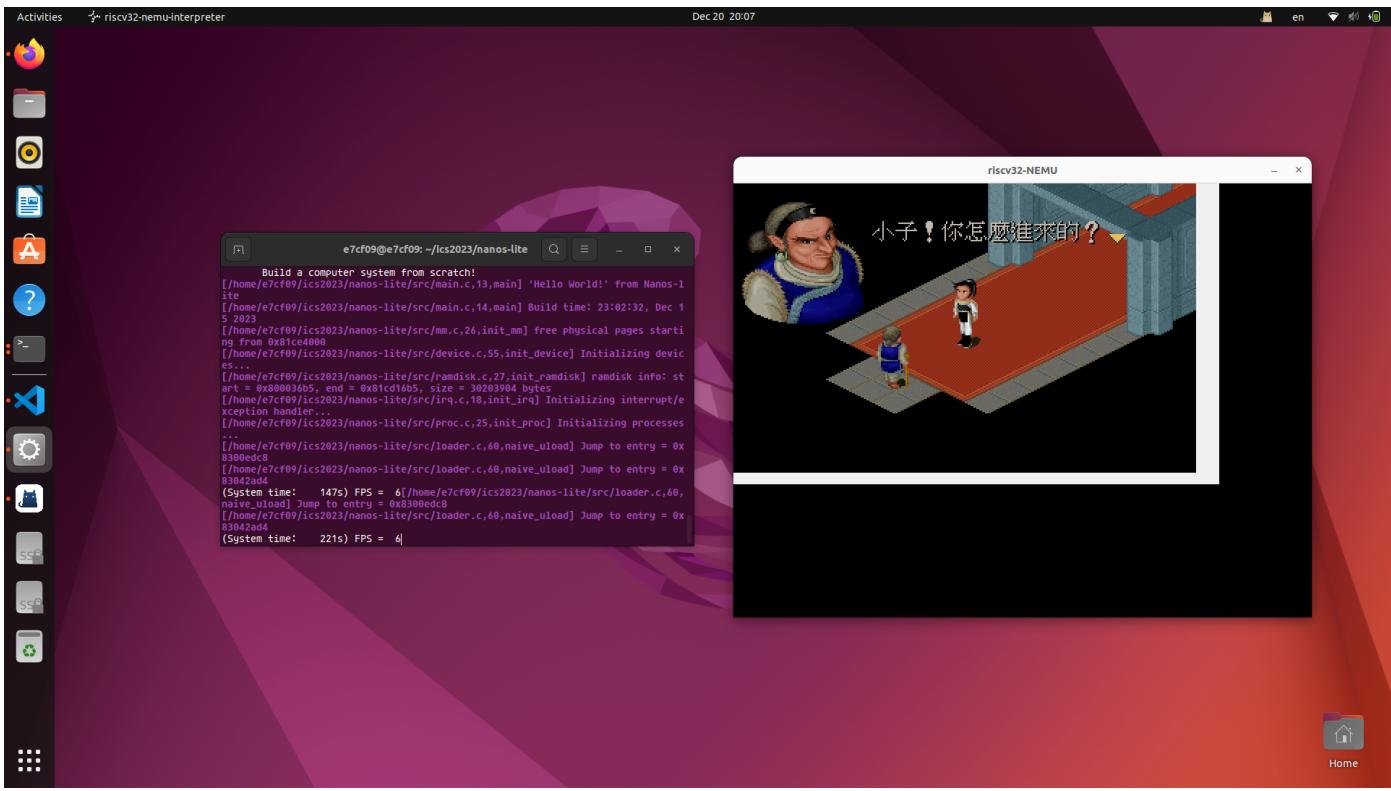


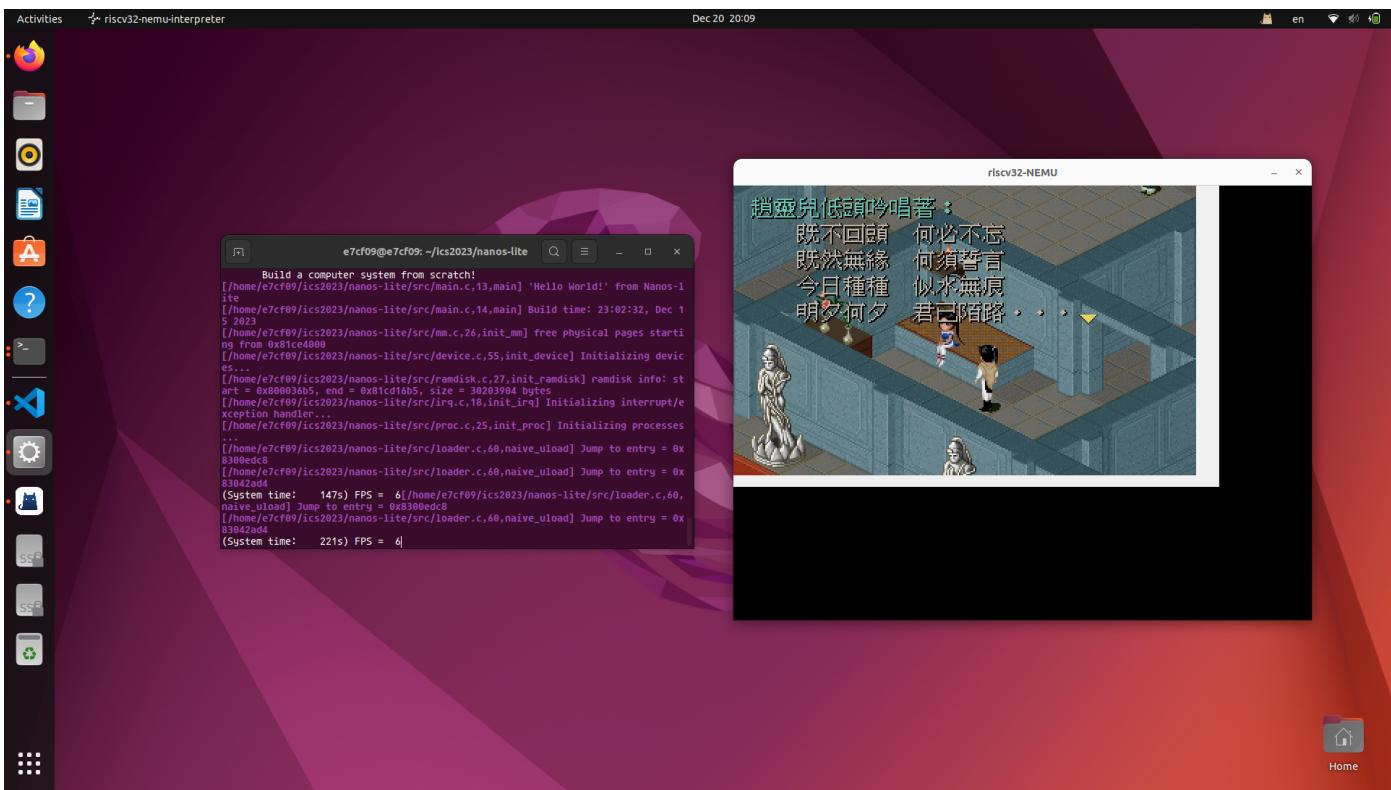
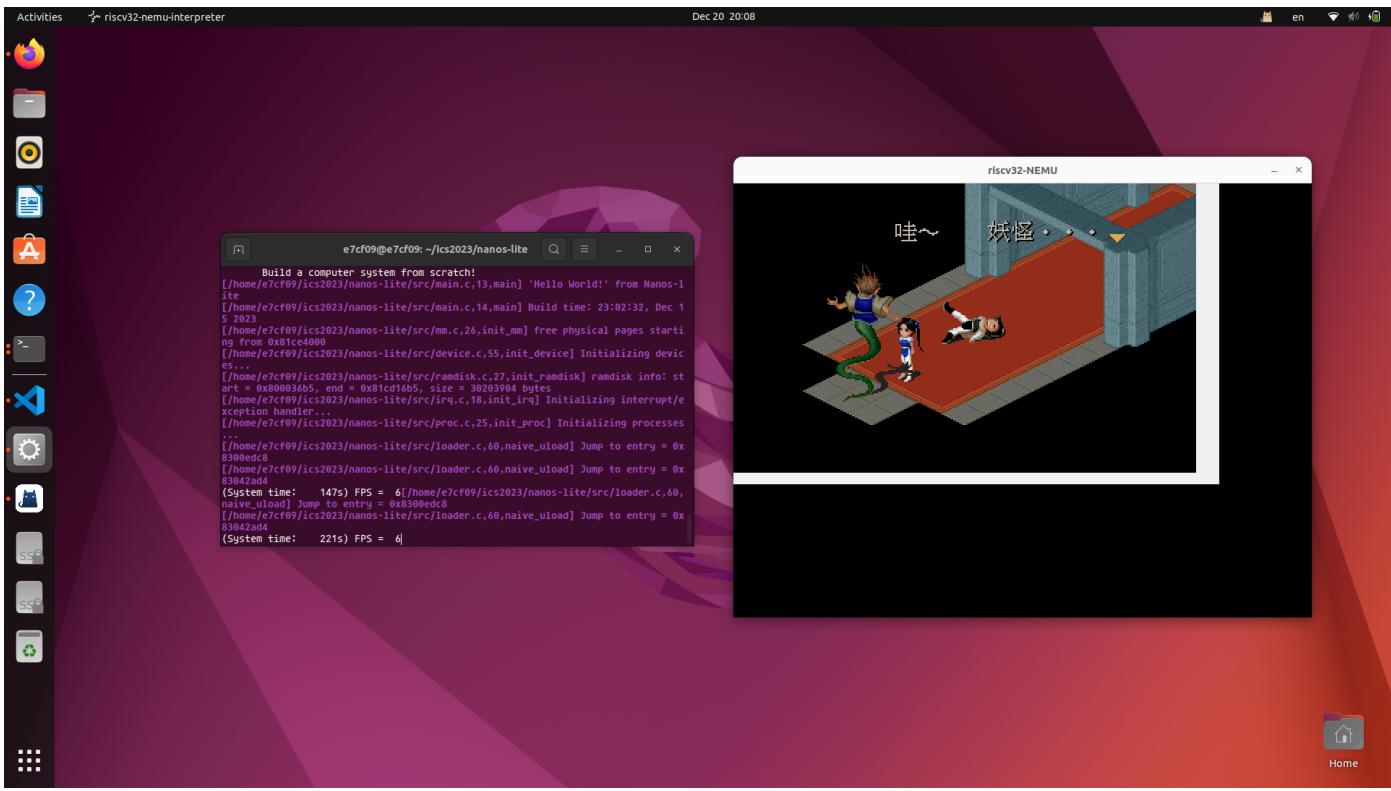
对第一个存档的试玩：

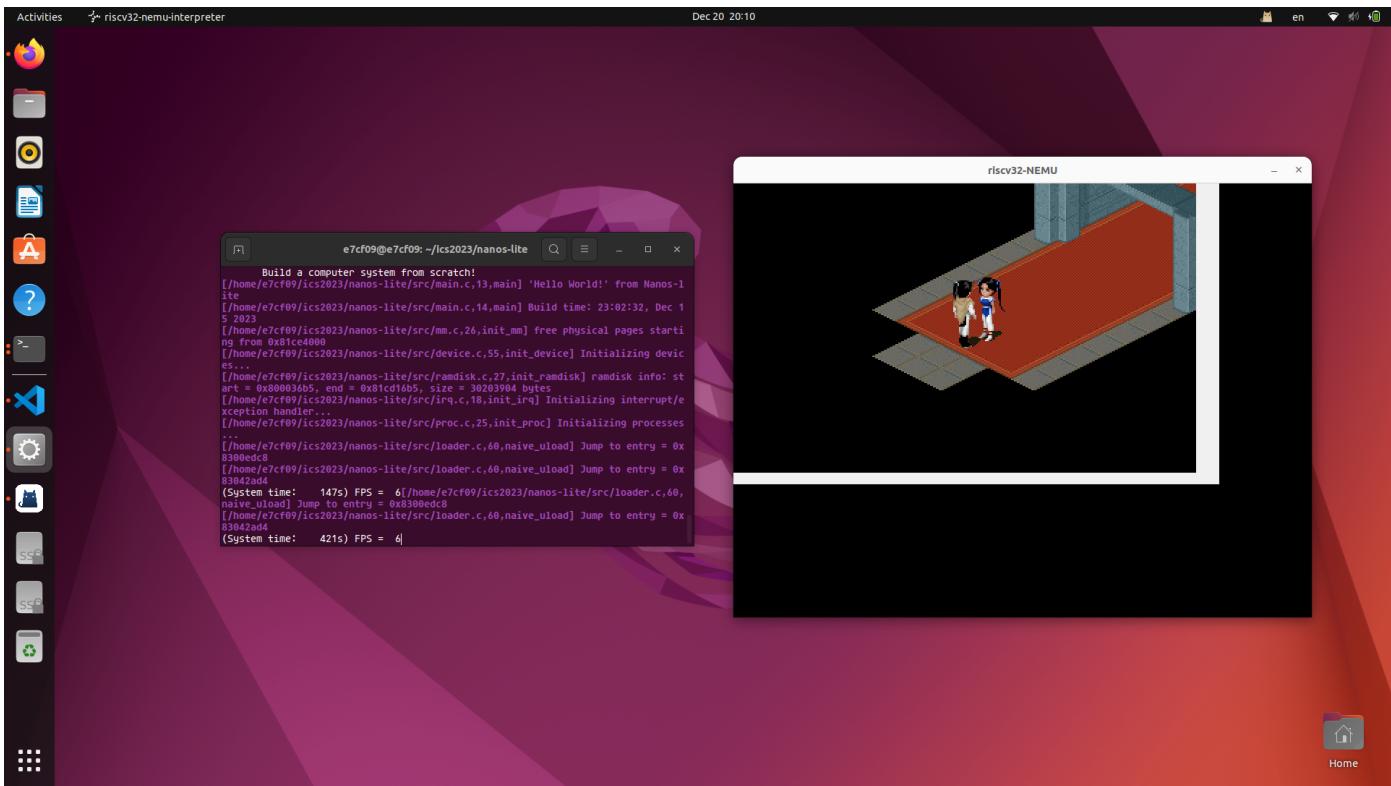




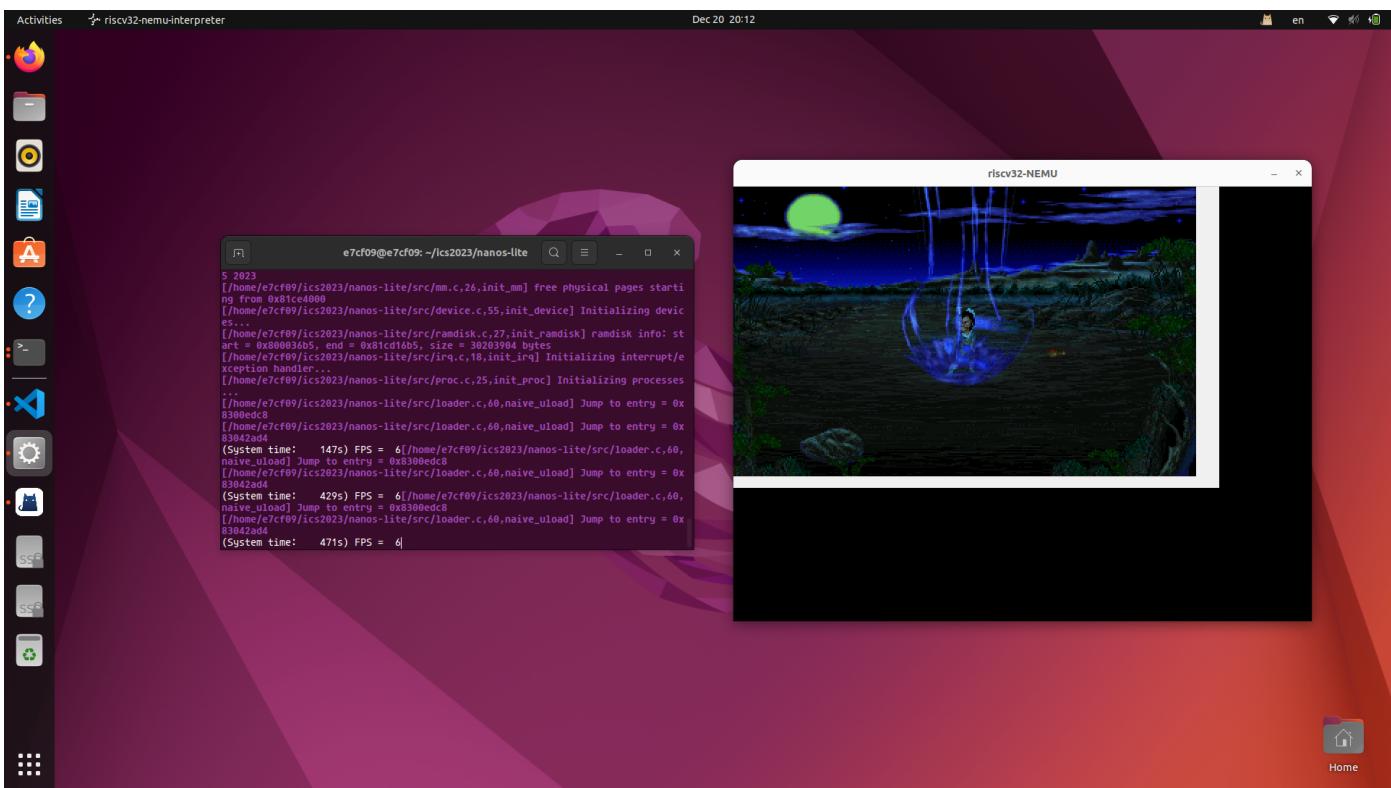
对第二个存档的试玩：

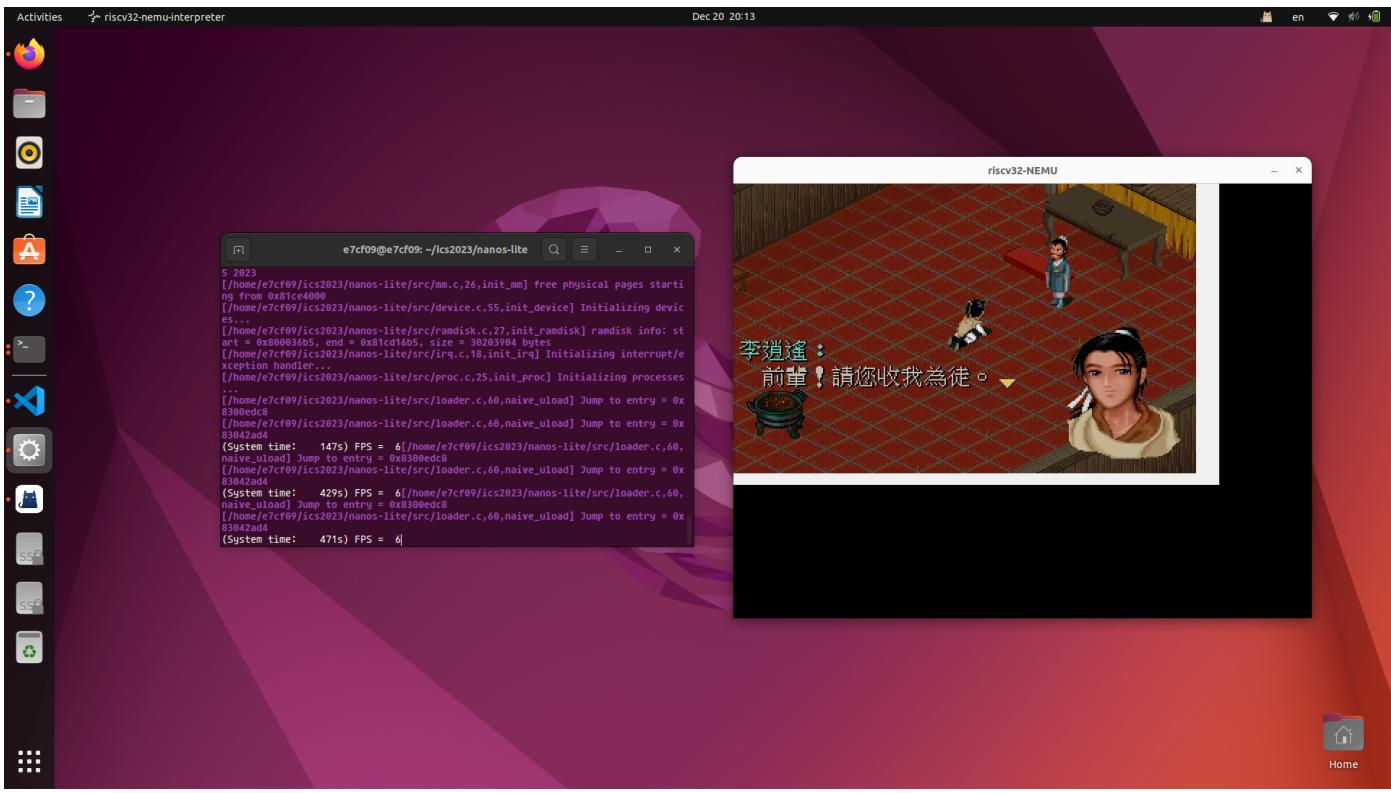
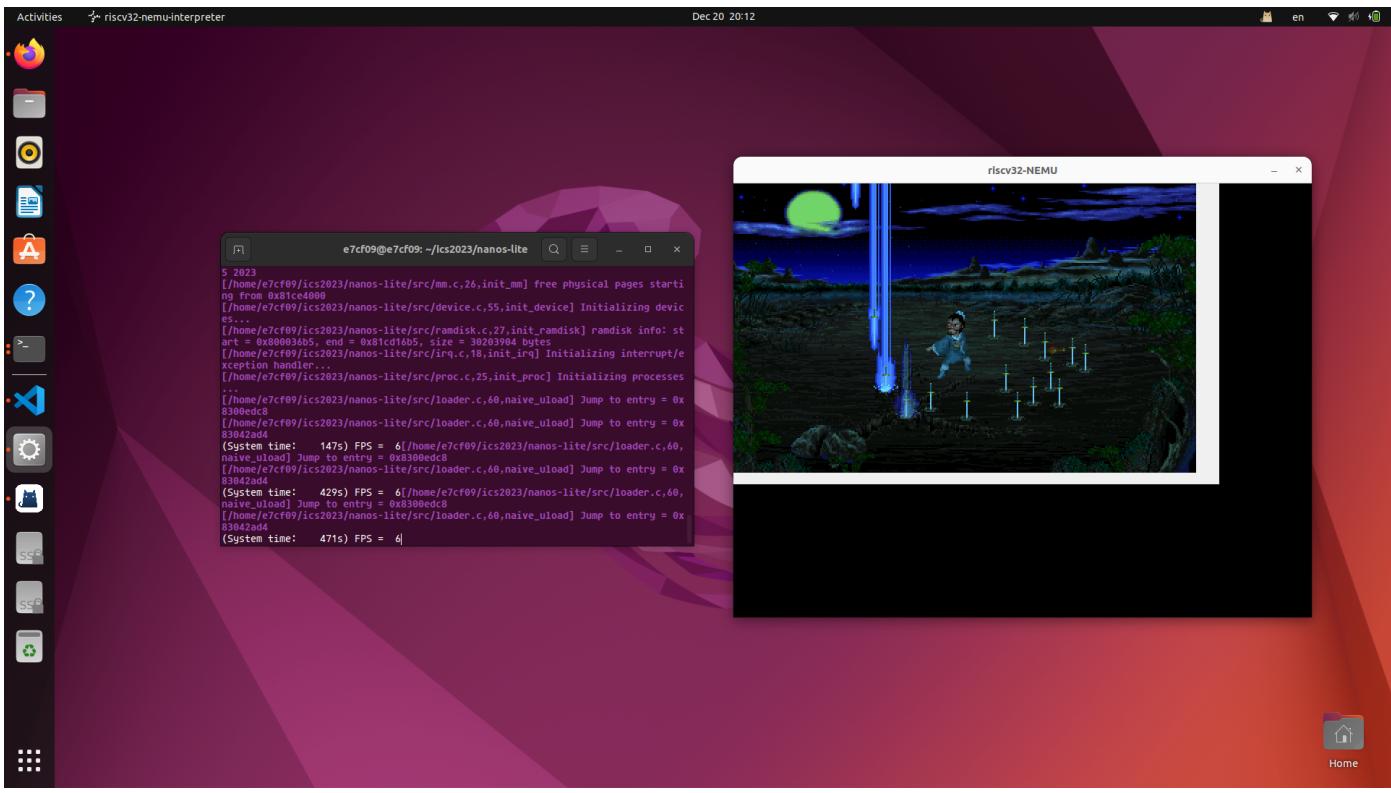


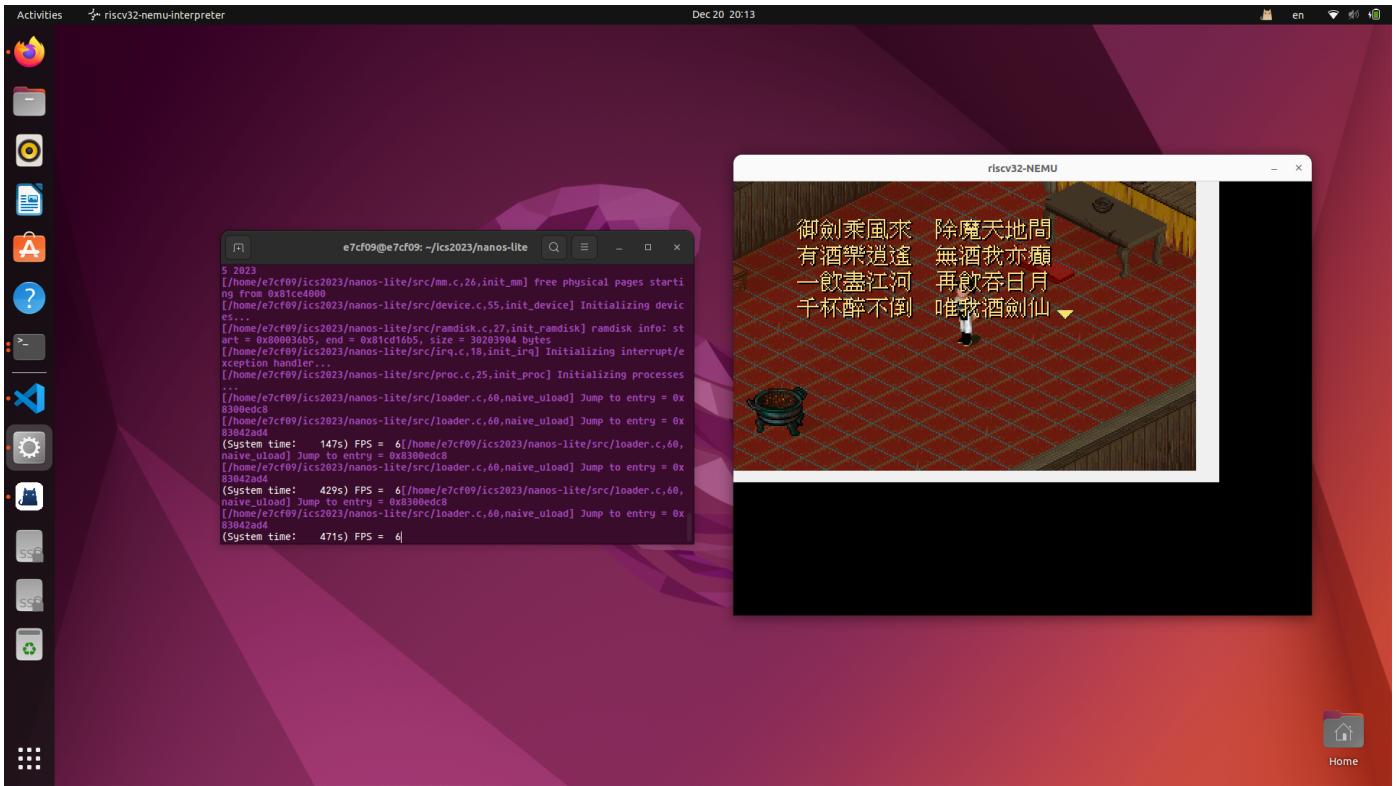




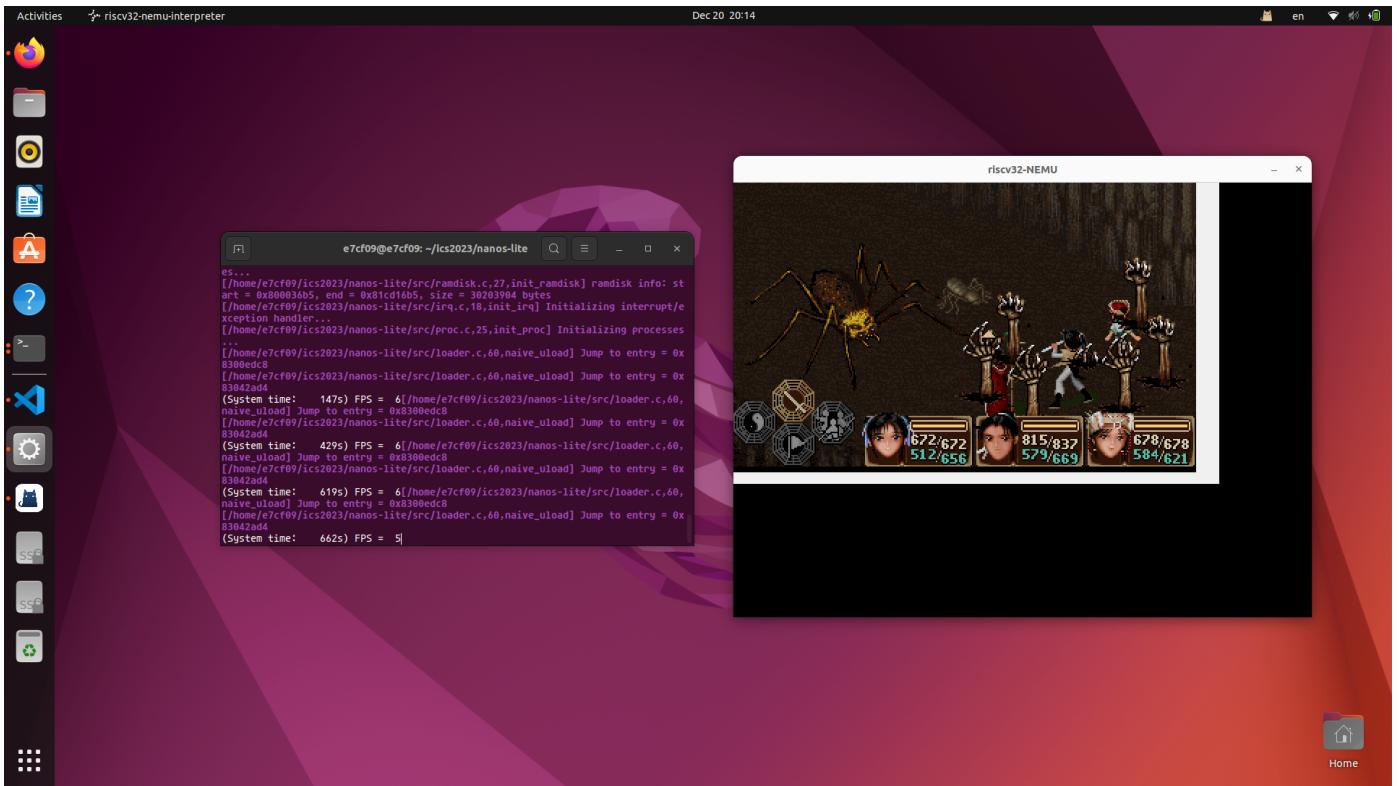
对第三个存档的试玩：

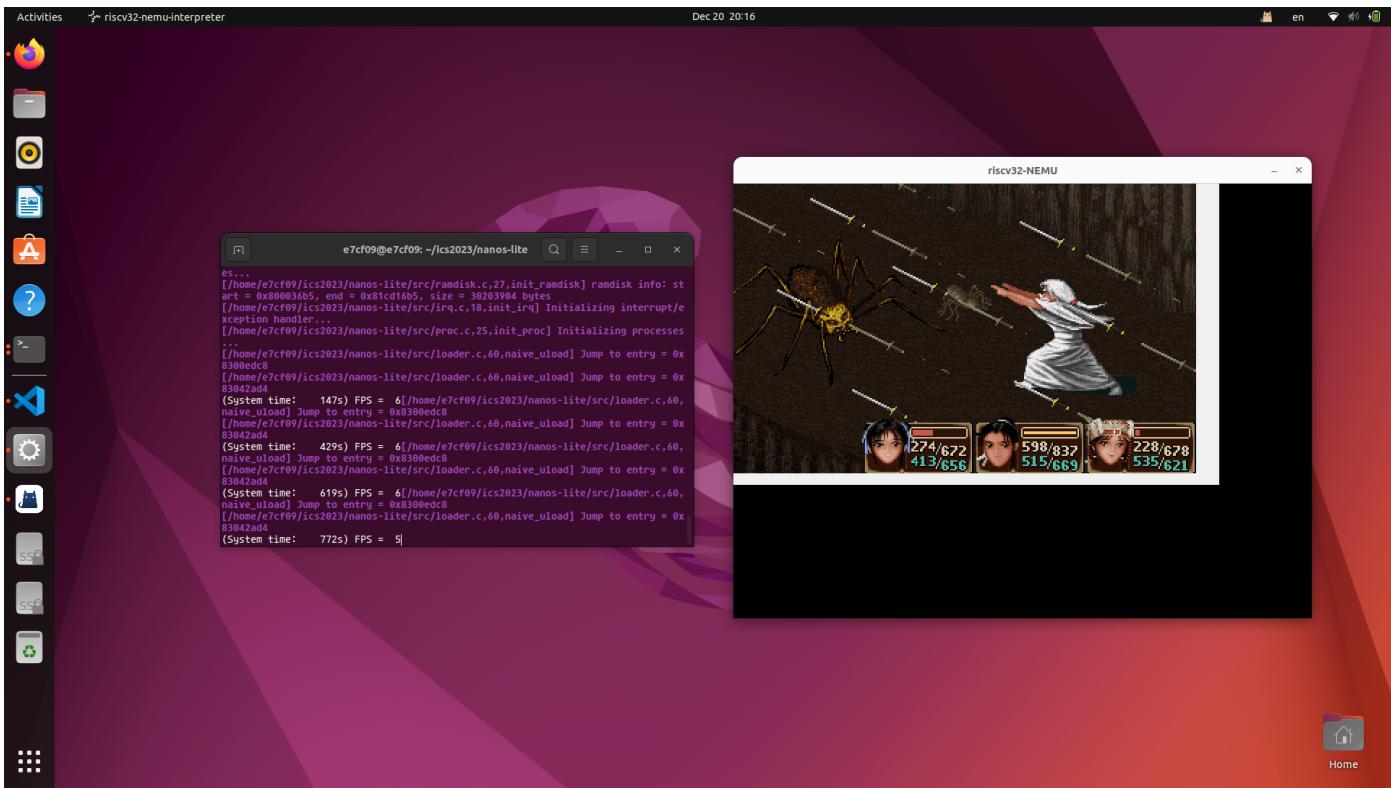
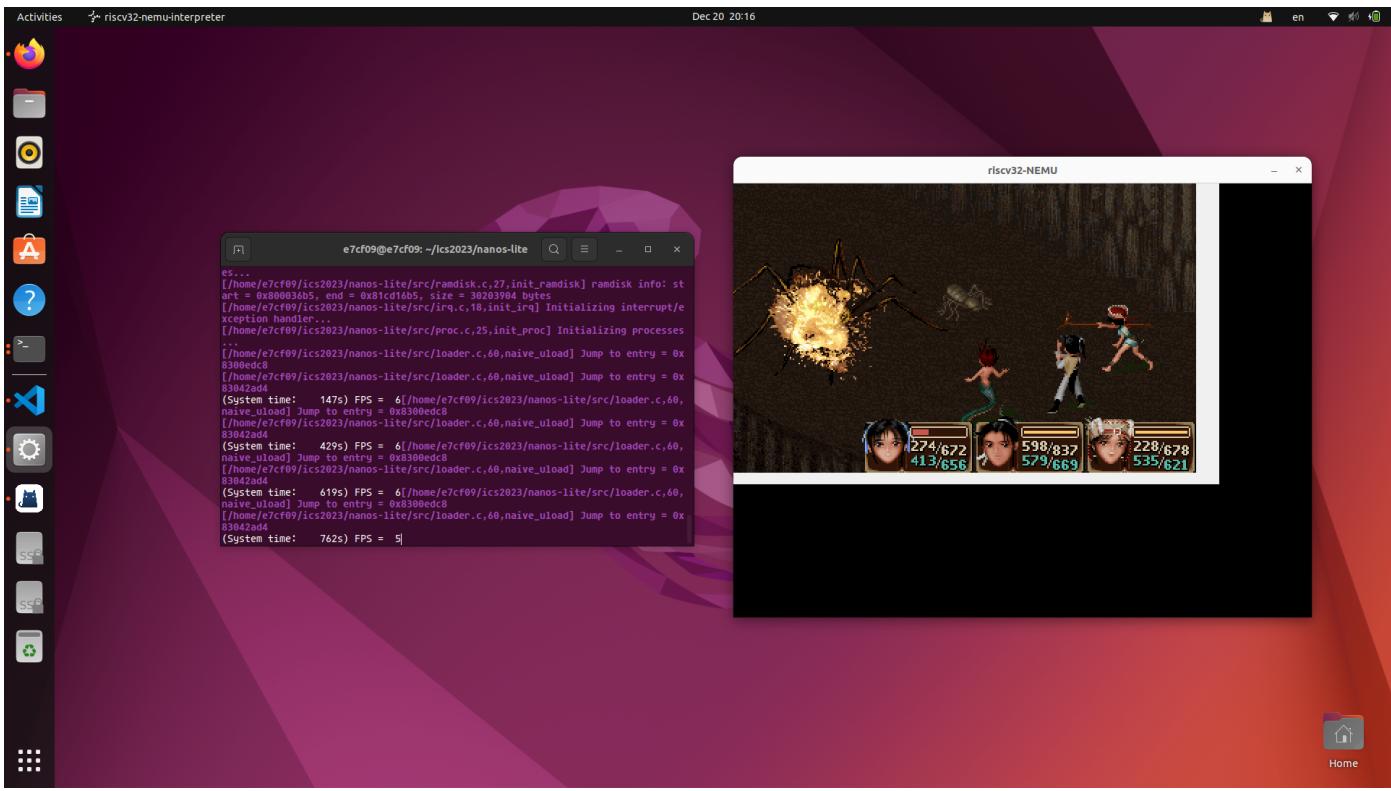


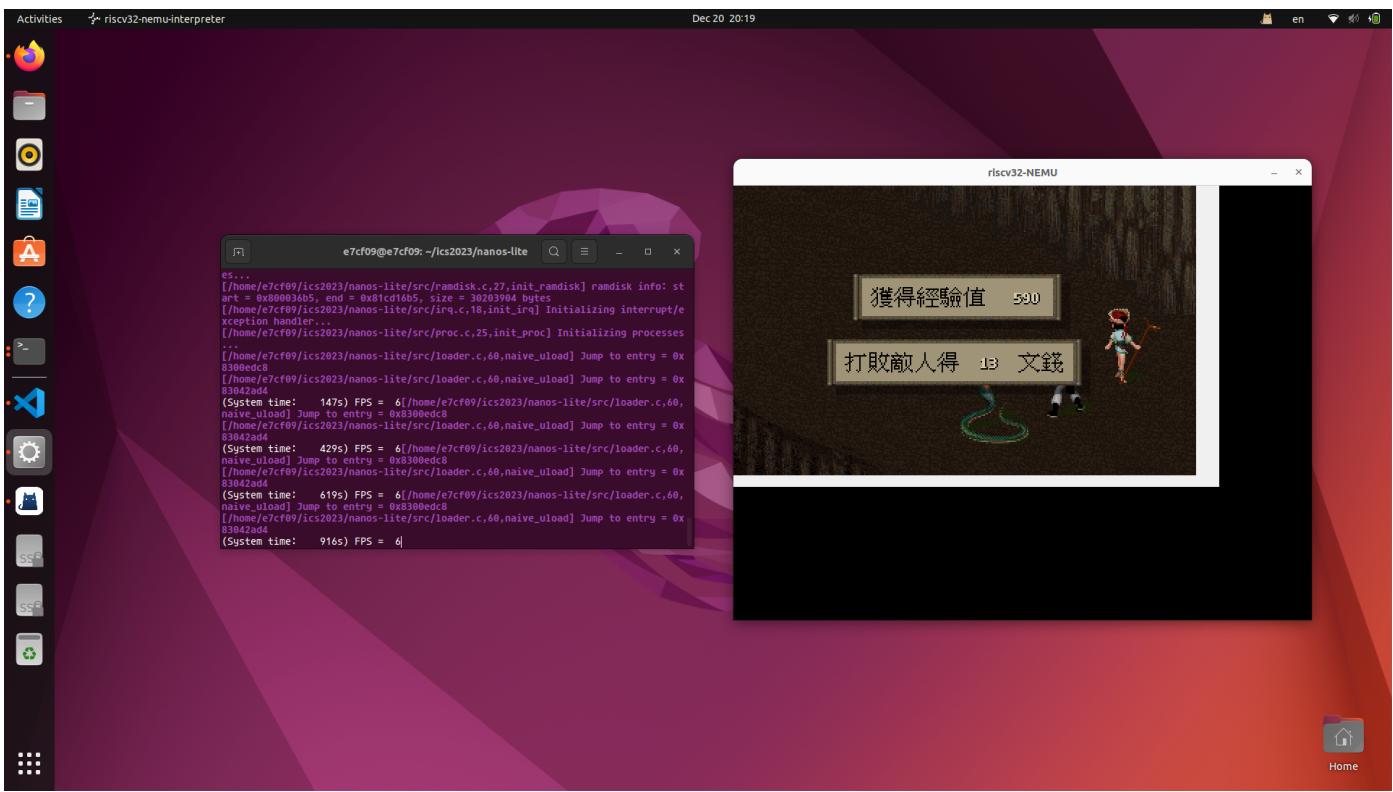
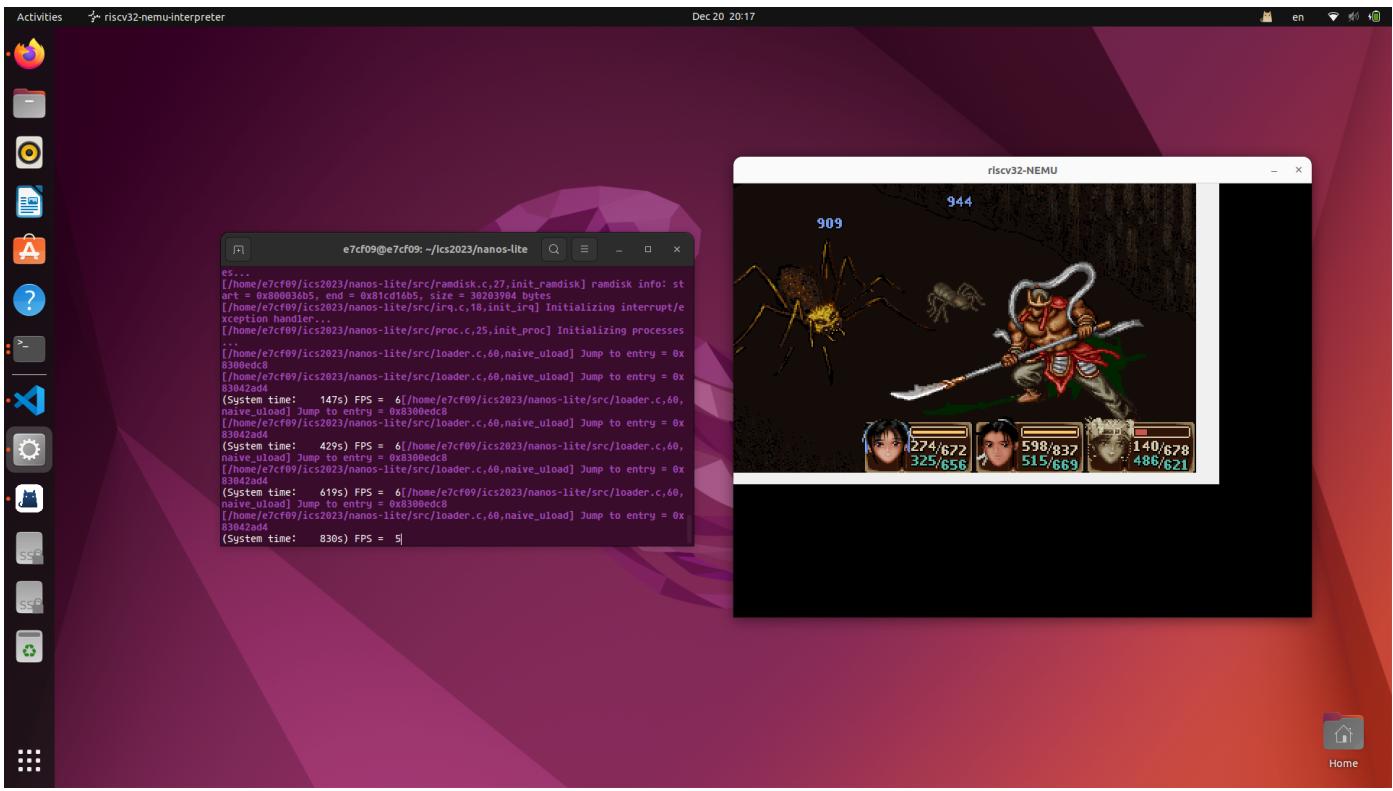




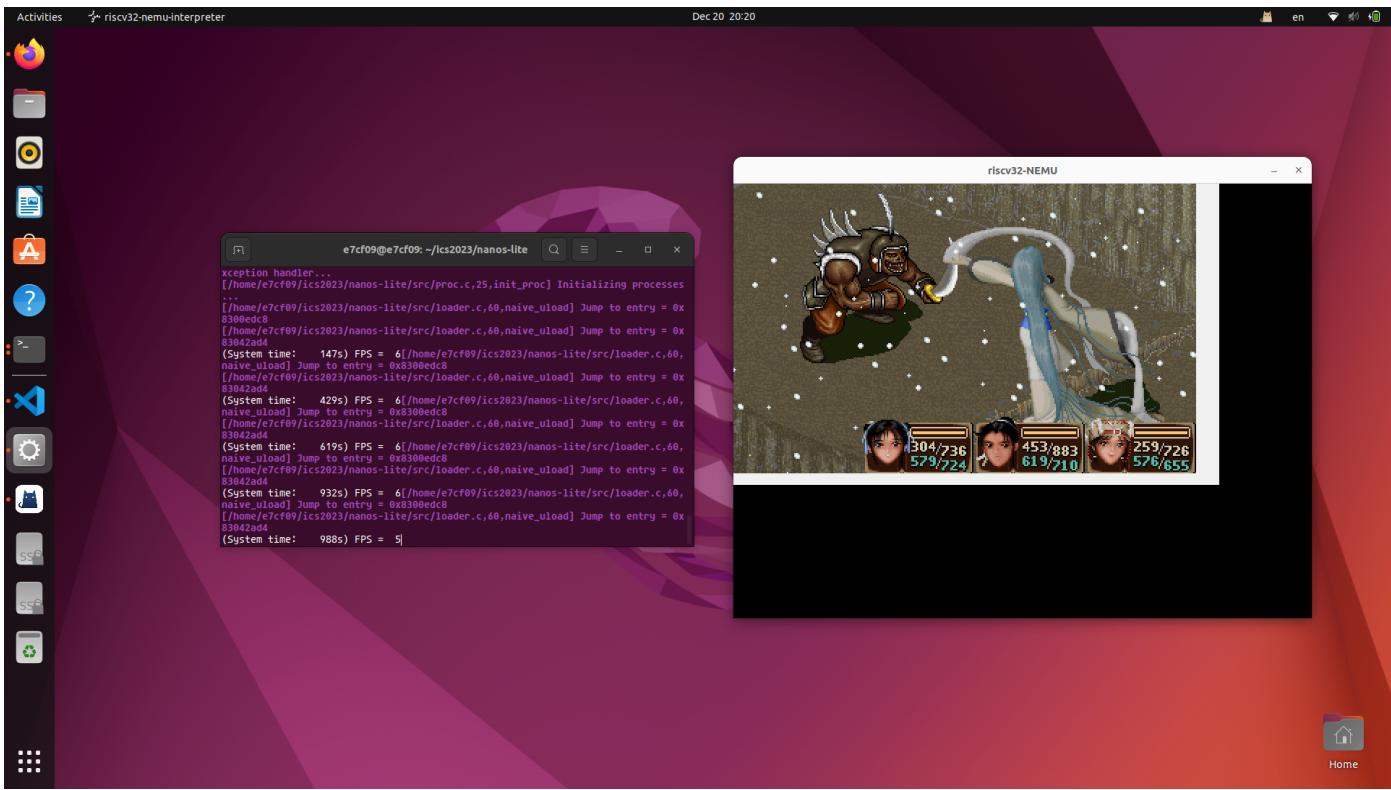
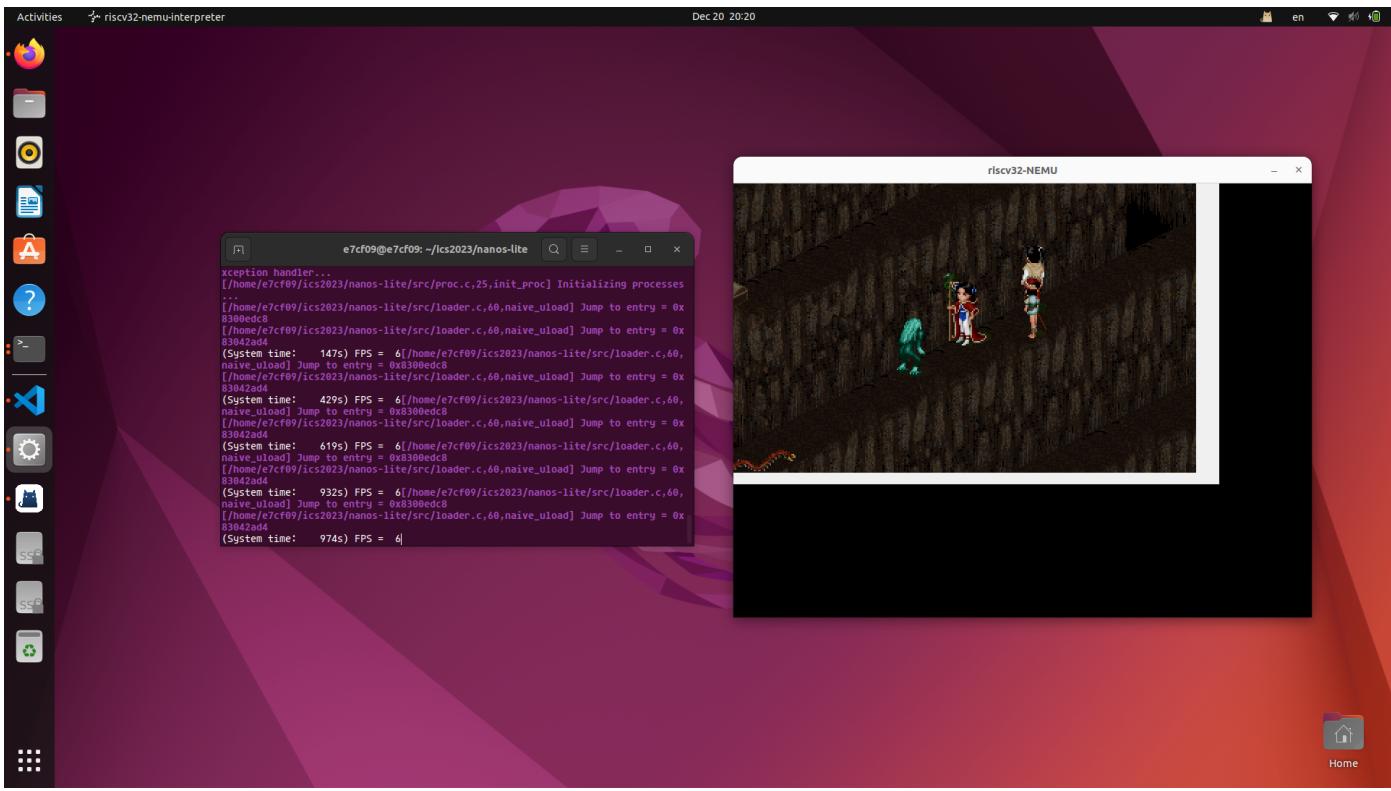
对第四个存档的试玩：

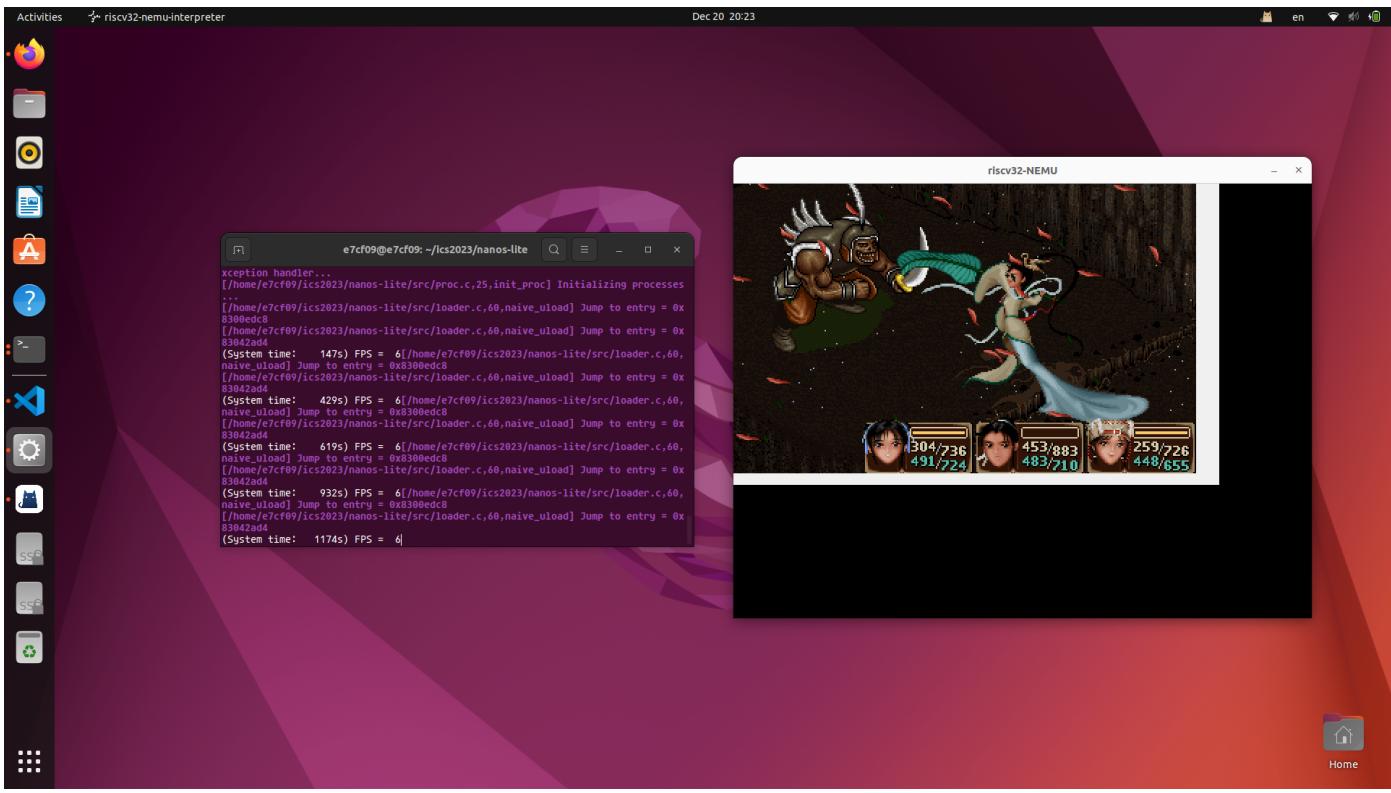
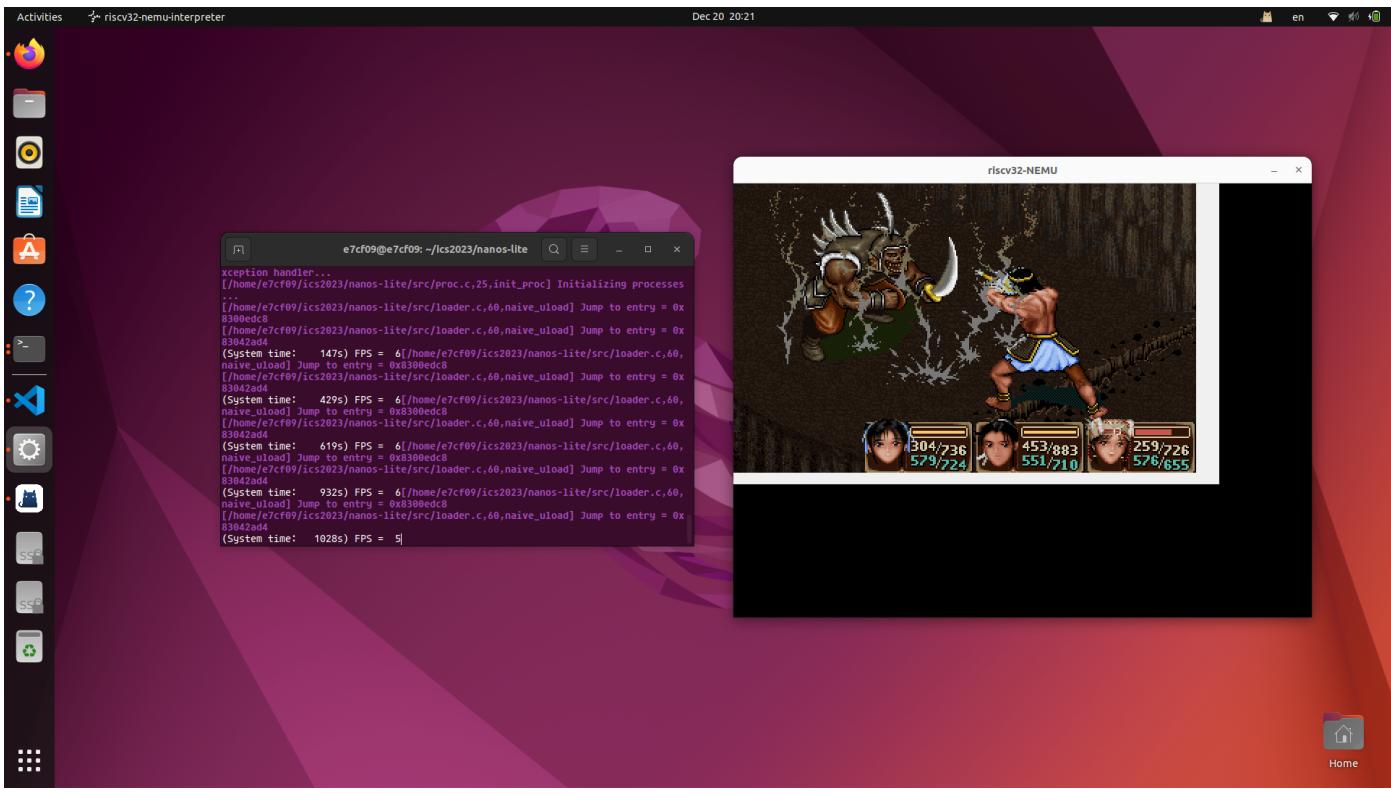


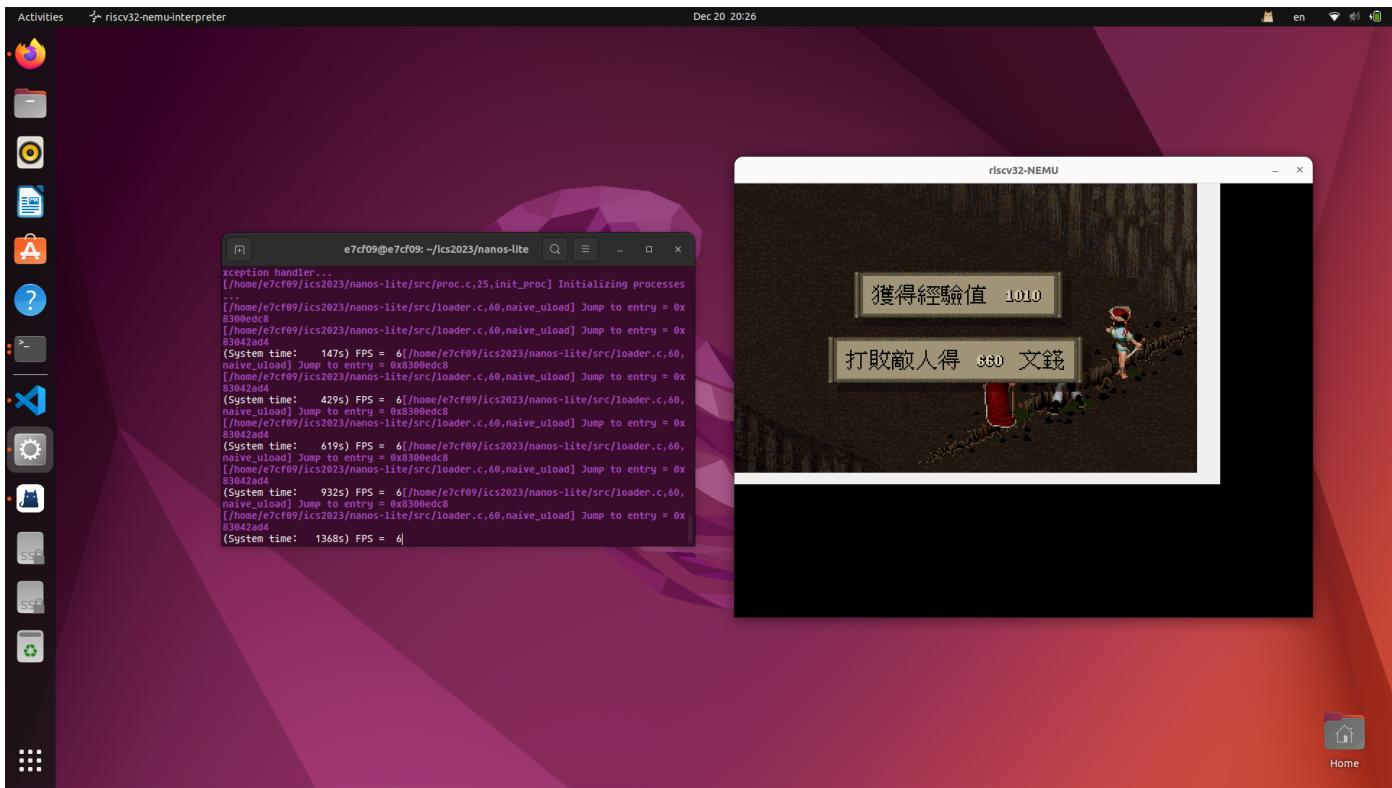




对第五个存档的试玩：







全程未观察到明显画面错误。