Check-Ins

To keep your progress on-track, you will have 2 Check-Ins before the final report is due. Check-Ins must be in-person and can be completed during section or office hours, by a TA or Instructor, but not by a tutor. All group-members must be present at your Check-Ins.

In the "rom_archive", you will see exactly what is required of you for each Check-In. Check-Ins can be completed at any time, as long as they are before the following deadlines:

- Check-In #1 by 5/10 11:59PM (CPU and Memory)
- Check-In #2 by 5/31 11:59PM (GPU and Input)

Note that you may be asked to do code refactoring before passing each Check-In. Leave enough time to implement requested changes. Each Check-In is worth 20% of your final grade.

Your final project report is due by 6/7 11:59PM. See the Final Project PDF on Canvas for more information.

Check-In 1

For Check-In 1, you will need to have your CPU and Address Space completed. To demonstrate this, you shall run the three provided "Hello, World" programs in `"hws/"`.

You can run the tests with the `"tests/run_test.sh"` script.

Check-In 2

For Check-In 2, you will need to have your GPU and Controller Input completed. To demonstrate this, you shall run the three provided programs in `"gpu/"`.

- `"input.slug"` prints out the controller value to `STDOUT` on every `loop()` call.
- `"image.slug"` fills the VRAM in setup so that the generated image matches `"image.png"`.

- `"box.slug"` tests both your GPU and Controller Input implementations. You should be able to move and scale the box that appears on the screen.

In addition to the slug files, you must have the following completed:

- Set up GitHub Actions to test that your emulator is building and works for the “Hello, World” programs in `"hws/"`.
- Set up GitHub Actions to verify the formatting of all your C++ files.
- Create a `".gitignore"` file to prevent build files from being pushed.
- Your `main()` function should be nearly empty. It should just call your emulator top-level.
- All your variable names should be clear and informative.
- Replace magic numbers with enums, macros, `constexpr` variables, or similar.
- Comments should be minimal. Explanations of code functionality should ideally be placed entirely in variable names.
- Source files should be in a subdirectory with a meaningful name (bad names include, “build” and “starter”).
- Break apart your design into separate classes where it makes sense.
- Ensure your code is consistent and easy to understand.
- Add `const` everywhere that makes sense.