CS50’s
Introduction to Game Development

OpenCourseWare

Colton Ogden (https://www.linkedin.com/in/colton-ogden-0514029b/)
cogden@cs50.harvard.edu

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/) (https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan) (https://www.tiktok.com/@davidjmalan) (https://davidjmalan.t.me/) (https://twitter.com/davidjmalan)

Final Project

Objectives

- Create a full game from scratch using either LÖVE or Unity.

From Start to Finish

It’s time to begin the course’s culmination: your final project! While most of the course thus far has been a series of assignments centered around adding features to existing code bases, this project will take place from the first line of code to the last, a true end-to-end experience to help tie everything we’ve learned together thus far.

Specification

- Your game must be in either LÖVE or Unity.

- Your game must be a cohesive start-to-finish experience for the user; the game should boot up, allow the user to play toward some end goal, and feature a means of quitting the game.
- Your game should have at least three `GameState`s to separate the flow of your game's user experience, even if it's as simple as a `StartState`, a `PlayState`, and an `EndState`, though you're encouraged to implement more as needed to suit a more robust game experience (e.g., a fantasy game with a `MenuState` or even a separate `CombatState`).
- Your game can be most any genre you'd like, though there needs to be a definitive way of winning (or at least scoring indefinitely) and losing the game, be it against the computer or another player. This can take many forms; some loss conditions could be running out of time in a puzzle game, being slain by monsters in an RPG, and so on, while some winning conditions may be defeating a final boss in an RPG, making it to the end of a series of levels in a platformer, and tallying score in a puzzle game until it becomes impossible to do more.
- You are allowed to use libraries and assets in either game development environment, but the bulk of your game's logic must be handwritten (i.e., putting together an RPG in Unity while using a UI library would be considered valid, assuming the bulk of the game logic is also not implemented in a library, but recycling a near-complete game prototype from Unity's asset store with slightly changed labels, materials, etc. would not be acceptable).

The most common cause for failure of the final project is not spending enough effort on this next instruction. Your README.md file should be minimally multiple paragraphs in length, and should provide a relatively comprehensive documentation of what you did and, if applicable, why you did it. Ensure you allocate sufficient time and energy to writing a README.md that you are proud of and that documents your project thoroughly, and that distinguishes this project from others in the course and defends its complexity!

- Your project must be at least as complex as (and distinct from!) the games you've implemented in this course, and should really be moreso. Submissions of low complexity may be rejected! You must explain your game in detail and why you believe it to meet the complexity and distinctiveness requirement in a `README.md` file at the root of your project.

How to Submit

When you submit your project, the contents of your `games50/projects/2018/x/final` branch must generally match the file structure of the other projects in this course. Your branch should also not contain any code from any other projects, only this one. Failure to adhere to this file structure will likely result in your submission being rejected.

By way of a simple example, for this project that means that if the grading staff visits `https://github.com/me50/USERNAME/blob/games50/projects/2018/x/final/README.md` (where `USERNAME` is your own GitHub username as provided in the form, below) we should be brought to the `README.md` file you need to write as part of Step 6 of the specification. If that's not how your code is organized when you check (e.g., you get a 404 error or don't see your file), reorganize your repository as needed to match this paradigm.

1. If you haven't done so already, visit this link (https://submit.cs50.io/invites/46e6f2ea29954ce9bb1bdc478a440055), log in with your GitHub account, and click **Authorize cs50**. Then, check the box indicating that you'd like to grant course staff access to your submissions, and click **Join course**.

The change to `/projects/2018` below is intentional, as CS50 courses have changed to a scheme that reflects when the project was initially released. So the 2018 here is correct, even though it's no longer 2018!

2. Using Git (https://git-scm.com/downloads), push your work to `https://github.com/me50/USERNAME.git`, where `USERNAME` is your GitHub username, on a branch called `games50/projects/2018/x/final` or, if you've installed `submit50` (https://cs50.readthedocs.io/submit50/), execute

```
submit50 games50/projects/2018/x/final
```

   instead.
3. Record a screencast (https://www.howtogeek.com/205742/how-to-record-your-windows-mac-linux-android-or-ios-screen/), not to exceed 5 minutes in length (and not uploaded more than one month prior to your submission of this project) in which you demonstrate your app's functionality. Upload that video to YouTube (https://www.youtube.com/upload) (as unlisted or public, but not private) or somewhere else.
4. Submit this form (https://forms.cs50.io/6d9af4c5-d037-4536-ba17-61a171f72e98).

You can then go to https://cs50.me/cs50g (https://cs50.me/cs50g) to view your current progress!