CS50's Introduction to Game Development

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

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://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

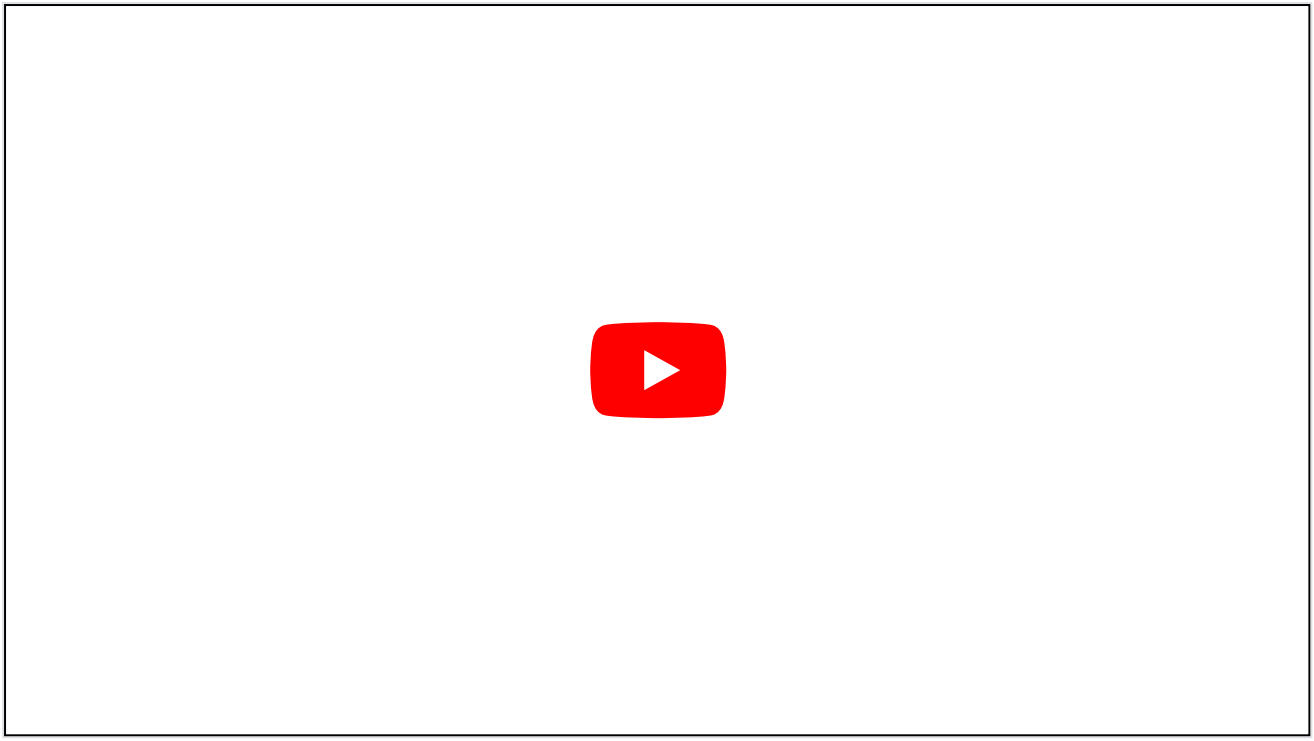
Pong

Objectives

- Read and understand all of the Pong source code from Lecture 0.
- Implement a basic AI for either Player 1 or 2 (or both!).

Demo

by Edward Kang



Your First Game

Download the distribution code for your game from cdn.cs50.net/games/2018/x/projects/0/pong.zip (https://cdn.cs50.net/games/2018/x/projects/0/pong.zip) and unzip `pong.zip`, which should yield a directory called `pong`.

Then, in a terminal window (located in `/Applications/Utilities` on Mac or by typing `cmd` in the Windows task bar), move to the directory where you extracted `pong` (recall that the `cd` command can change your current directory), and run

```
cd pong
```

It's Game Time

Your first assignment in the course will be a fairly easy one, since the dive into game programming can be deep enough as it is without having to implement an entire code base from scratch! Instead, we'll take the Pong example we covered in class and extend it in a small but fun way by giving one of the paddles (or perhaps both) logic for playing the game so that you don't always need a buddy to play the game with you! We'll approach problem sets in the course this way generally, taking the full code bases we've used in lecture and extending them so that you'll get plenty of experience interacting with fully implemented games. You can even use these projects as templates and jumping boards for your own games!

Of course, the code won't run if you don't have LÖVE installed, so we'll have to tackle that in addition to grabbing the code, so do just choose the appropriate distribution of that version for

your system here:

https://love2d.org (https://love2d.org)

If using macOS and unable to run LÖVE after installing it because it's "from an unidentified developer, see support.apple.com/guide/mac-help/open-a-mac-app-from-an-unidentified-developer-mh40616/mac (https://support.apple.com/guide/mac-help/open-a-mac-app-from-an-unidentified-developer-mh40616/mac).

For further information on how to actually run games, do just visit the following page:

https://love2d.org/wiki/Getting_Started (https://love2d.org/wiki/Getting_Started)

Once the code and LÖVE have been downloaded and installed, the actual change you'll be making to the code base is small, but it will require you to understand what many of the pieces do, so be sure to watch Lecture 0 and read through the code so you have a firm understanding of how it works before diving in! In particular, take note of how paddle movement works, reading both the `Paddle` class as well as the code in `main.lua` that actually drives the movement, located in the `update` function (currently done using keyboard input for each). If our agent's goal is just to deflect the ball back toward the player, what needs to drive its movement?

Specification

- Implement an AI-controlled paddle (either the left or the right will do) such that it will try to deflect the ball at all times. Since the paddle can move on only one axis (the Y axis), you will need to determine how to keep the paddle moving in relation to the ball. Currently, each paddle has its own chunk of code where input is detected by the keyboard; this feels like an excellent place to put the code we need! Once either the left or right paddle (or both, if desired) try to deflect the paddle on their own, you've done it!

Errata

At the time of the course's lectures being filmed, we were using a now-outdated version of LÖVE (0.10.2); most of everything has remained the same into the newer version 11 series of the framework, but one core change is worth noting: `love.graphics.clear` and functions like it which take four arguments representing the red, green, blue, and alpha components of a color (or RGBA) formerly took integer values between `0` and `255`:

```
love.graphics.clear(255, 0, 0, 255)
```

However, the API has now changed to where all functions that formerly took integer values for color components now take a floating-point value between 0 and 1; for example, the above API call would translate to the following:

```
love.graphics.clear(1, 0, 0, 1)
```

An easy way to normalize the old style to the new style is simply to divide components by 255. If, for example, we had a color we liked with the components 120, 30, 70, and 255, we could write it like the below:

```
love.graphics.clear(120/255, 30/255, 70/255, 255/255)
```

CS50 Games exists only in archive form, as of 1 July 2024. While you cannot submit this project for credit any longer, it is a great exercise to test your understanding of the course material.