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)

Mario

Objectives

- Read and understand all of the Super Mario Bros. source code from Lecture 4.
- Program it such that when the player is dropped into the level, they're always done so above solid ground.
- In `LevelMaker.lua`, generate a random-colored key and lock block (taken from `keys_and_locks.png` in the `graphics` folder of the distro). The key should unlock the block when the player collides with it, triggering the block to disappear.
- Once the lock has disappeared, trigger a goal post to spawn at the end of the level. Goal posts can be found in `flags.png`; feel free to use whichever one you'd like! Note that the flag and the pole are separated, so you'll have to spawn a `GameObject` for each segment of the flag and one for the flag itself.
- When the player touches this goal post, we should regenerate the level, spawn the player at the beginning of it again (this can all be done via just reloading `PlayState`), and make

it a little longer than it was before. You'll need to introduce `params` to the `PlayState:enter` function that keeps track of the current level and persists the player's score for this to work properly.

Demo

by Edward Kang

Getting Started

Download the distro code for your game from cdn.cs50.net/games/2018/x/projects/4/mario.zip (https://cdn.cs50.net/games/2018/x/projects/4/mario.zip) and unzip `mario.zip`, which should yield a directory called `mario`.

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 `mario` (recall that the `cd` command can change your current directory), and run

```
cd mario
```

It's-a Key!

Welcome to your fifth assignment! So far, we have a fair foundation for a platforming game present in the distro.

Specification

- *Program it such that when the player is dropped into the level, they're always done so above solid ground*. Just like we generate the level column by column (as can be seen in `LevelMaker.lua`), we can check the game's map column by column and simply ensure that the player isn't placed above a column that just spawned a chasm by looking at all of the tiles along the Y-axis, going from left to right, until we've come across a column where we encounter a solid tile (as by checking whether the id is equal to `TILE_ID_GROUND`).
- *In `LevelMaker.lua`, generate a random-colored key and lock block (taken from `keys_and_locks.png` in the `graphics` folder of the distro). The key should unlock the block when the player collides with it, triggering the block to disappear.* This is something you'll introduce into `LevelMaker.generate` while it's actively generating the level; simply maintaining a flag for whether the key and lock have been spawned and placed and randomly choosing to place them down could do (or you could simply do it after the whole rest of the level is generated). The former will likely be easier so you can conditionally do it when you're not already spawning a block, since otherwise you'll have to iterate over all of the blocks you've already generated throughout the level and compare their positions with that of where you'd potentially like to generate a key or lock. See how the code for spawning gems works (particularly with the `onConsume` callback) for how you might implement picking up the key, and see the code for spawning blocks and the `onCollide` function for how you might implement the key blocks!
- *Once the lock has disappeared, trigger a goal post to spawn at the end of the level. Goal posts can be found in `flags.png`; feel free to use whichever one you'd like! Note that the flag and the pole are separated, so you'll have to spawn a `GameObject` for each segment of the flag and one for the flag itself.* This is code we can likely add to the `onCollide` function of our lock blocks, once we've collided with them and have the key they need to unlock. Just like gems spawn when we collide with some overhead blocks, you'll simply need to add new `GameObject`s to the scene that comprise a flag pole. Note that the pole and flag are separate objects, but they should be placed in such a way that makes them look like one unit! (See the scene mockup in `full_sheet.png` for some inspiration).
- *When the player touches this goal post, we should regenerate the level, spawn the player at the beginning of it again (this can all be done via just reloading `PlayState`), and make it a little longer than it was before. You'll need to introduce `params` to the `PlayState:enter` function that keeps track of the current level and persists the player's score for this to work properly.* The easiest way to do this is to just add an `onConsume` callback to each flag

piece when we instantiate them in the last goal; this `onConsume` method should then just restart our `PlayState`, only now we'll need to ensure we pass in our `score` and `width` of our game map so that we can generate a map larger than the one before it. For this, you'll need to implement a `PlayState:enter` method accordingly; see prior assignments for plenty of examples on how we can achieve this! And don't forget to edit the default `gStateMachine:change('play')` call to take in some default score and level width!

How to Submit

When you submit your project, the contents of your `games50/projects/2018/x/mario` branch must match the file structure of the unzipped distribution code exactly as originally received. That is to say, your files should not be nested inside of any other directories of your own creation or otherwise deviate from the file structure we gave you. 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/mario/src/LevelMaker.lua` (where `USERNAME` is your own GitHub username as provided in the form, below) we should be brought to the `LevelMaker.lua` file for Super 50 Bros. If that's not how your code is organized when you check (e.g., you get a 404 error or don't see your edits), 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/mario` or, if you've installed `submit50` (https://cs50.readthedocs.io/submit50/), execute

```
submit50 games50/projects/2018/x/mario
```

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.
    - To aid in the staff's review, in your video's description on YouTube, you should timestamp where each of the following occurs *in your gameplay demonstration*. This is **not optional**; failure to do this will result in your submission being rejected:
    - Flag not spawned without key block having been unlocked
    - Key obtained
    - Key block unlocked
    - Flag spawned only after key block has been unlocked
4. Submit this form (https://forms.cs50.io/5a18355c-09b3-4559-8952-4231bcadb33f).

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