CSCI E-23a

Introduction to Game Development

produced by CS50

Colton Ogden
David J. Malan

Harvard Extension School
Spring 2018

Assignments

Lectures

Live Stream

Office Hours

Projects

Sections

Slack

Staff

Syllabus

Project 0: Space Invaders

Objectives

- Read and understand all of the code we've covered through Lecture 2, Breakout.
- Generate some unique sprite artwork for your game.
- Develop a simple space shooter, where your goal is to stop an alien attack by clearing waves of aliens using a small spaceship. You have a certain number of lives, and the game is over once all of them are depleted!
- Generate some unique sound effects for your game and include them to polish things up.

Getting Started

GitHub Classroom

In this course, we'll use GitHub Classroom to distribute projects and collect submissions. To begin Project 0:

1. Click here to go to the GitHub Classroom page for starting the assignment.
2. Click the green "Accept this assignment" button. This will create a GitHub repository for your project. Recall that a git repository is just a location where

your code will be stored and which can be used to keep track of changes you make to your code over time.

3. Click on the link that follows "Your assignment has been created here", which will direct you to the GitHub repository page for your project. It may take a few seconds for GitHub to finish creating your repository.
4. In the upper-right corner of the repository page, click the "Fork" button, and then (if prompted) click on your username. This will create a fork of your project repository, a version of the repository that belongs to your GitHub account.
5. Now, you should be looking at a GitHub repository titled **username/project0-username**, where **username** is your GitHub username. This will be the repository to which you will push all of your code while working on your assignment. When working on the assignment, do not directly push to the **games50/project0-username** repository: always push your code to your **username/project0-username** repository.

Setup

Time to pull down the starting code for Space Invaders, just like last week! First, on your main repository page (https://github.com/username/project0-username), click on the green "Clone or download" button. Copy the "Clone with HTTPS" link to your clipboard (if familiar with SSH, you can use that instead).

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 want to

store your project on your computer (recall that the `cd` command can change your current directory), and run

```
git clone repository_url project0
```

where `repository_url` is the link you just copied from GitHub. You will be prompted for your GitHub username and password.

Go ahead and run `cd project0` to enter your repository.

Feature Requirements

Welcome to your first project, Space Invaders! This time, we'll be implementing quite a bit of code ourselves, but we'll walk you through many of the steps needed in order to put things together!

Testing the Game

Be sure to watch Lectures 0-2 and read through their code so you have a firm understanding of how they work before diving in! Once you're ready, download and

run our staff implementation of Space Invaders here to get a sense of the gameplay. To run the `.love` application, which is effectively a fancier name for a `.zip` file, simply double-click it like a normal `.app` or `.exe` on Mac or Windows (Linux users may have to run it using `love` from the command line; see documentation here for further details), after which LÖVE2D will open it up, letting you shoot some 8-bit aliens! Once you've seen the gameplay, taking note of the title screen, the main game, and the score screen, proceed below!

Assignment Goals

1. Define all of the LÖVE2D callback functions we've described thus far: `love.load`, `love.resize`, `love.keypressed`, `love.update`, and `love.draw` in `main.lua`.
2. Define several game state classes to manage the flow of your game: `TitleScreenState` to display "Space Invaders" and the user's high score, `PlayState` to let the player shoot aliens and rack up a high score, and a `ScoreState` to show the player's score when they lose all their lives. Use a `StateMachine` object to change back and forth among these states. For reference, consult Flappy Bird's source code for an identical state setup! Note that, just like in Flappy Bird, the `PlayState` will have to share its score with the `ScoreState` in order to avoid having a global score variable in `main.lua`!
3. Define an `Alien` and `Player` class to encapsulate the rendering and basic behavior for our spaceship and for the aliens moving toward the player. They should be separate sprites; whether you grab them from the same image using

`love.graphics.newQuad` or by just using separate textures altogether is up to you!
4. Implement firing lasers from the `Player` and the `Alien`s.
5. Implement AABB collision detection between `Projectile`s, the `Player`, and `Alien`s; not only should `Projectile`s collide with both `Alien`s and the `Player`, but the `Player` should also collide with the `Alien`s!
6. Give the `Alien`s their characteristic side-to-side movement from the original game (see below and play the sample app linked above), and make them gradually make their way toward the `Player` from the top to the bottom of the screen.
7. Display our current score, level, and lives in the `PlayState`.
8. Implement speed scaling of the `Alien`s based on what level we're on; the higher the level, the faster the `Alien`s should move!
9. Persist our high score to a file that we read and write so we never lose it even after quitting.
10. Lastly, add polish to the game with some sound effects!

Where to Begin

But First, Random Aliens?

Wouldn't it be nice if, instead of the same aliens each time we play the game, we spawned a random assortment for some aesthetic variety?

Head here if you want to generate a sprite sheet unique to your very game that looks good to you! For ease of splitting, be sure to set it to no spacing and 1x zoom. If you'd prefer not to take this step, feel free to use the default sprite sheet that comes with the distro code!

Starting from Scratch

The first step we'll need to do is set up our LÖVE2D application from scratch, but this should be cake now! In particular, begin implementing the following methods (note we included their skeletons in the distro, in `main.lua`):

- `love.load`
- `love.resize`
- `love.keypressed`
- `love.update`
- `love.draw`

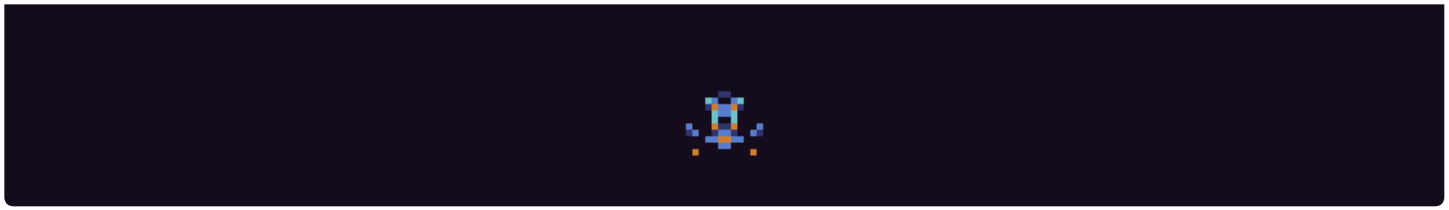
For help with these and other functions LÖVE defines, always feel free to peruse their awesome documentation!

State of Affairs

The second step to producing the next hit classic of 1978 is ensuring we have state classes to cleanly separate the logic and rendering of our game. In particular, implement the following three core states (distro skeleton code provided):

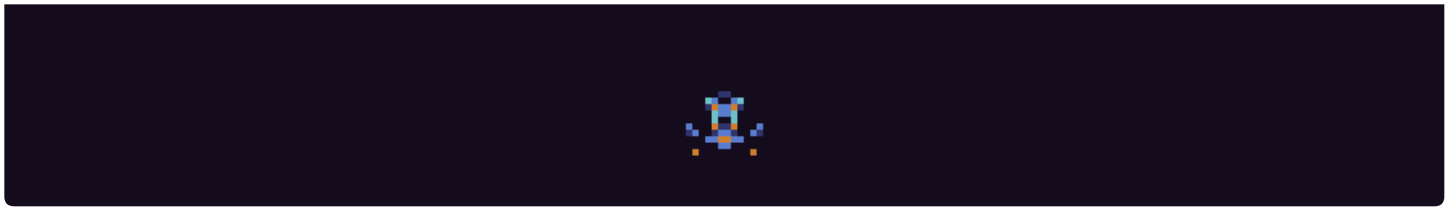
- `TitleScreenState` This is the state we begin the game in; it's a rather simple state that should just display "Space Invaders", the high score (which we'll load from a file later!), and an instruction to press Enter in order to begin the game. Upon pressing Enter, we should transition to the `PlayState` .


Space Invaders
Space Invaders
Press Enter
High Score: 160

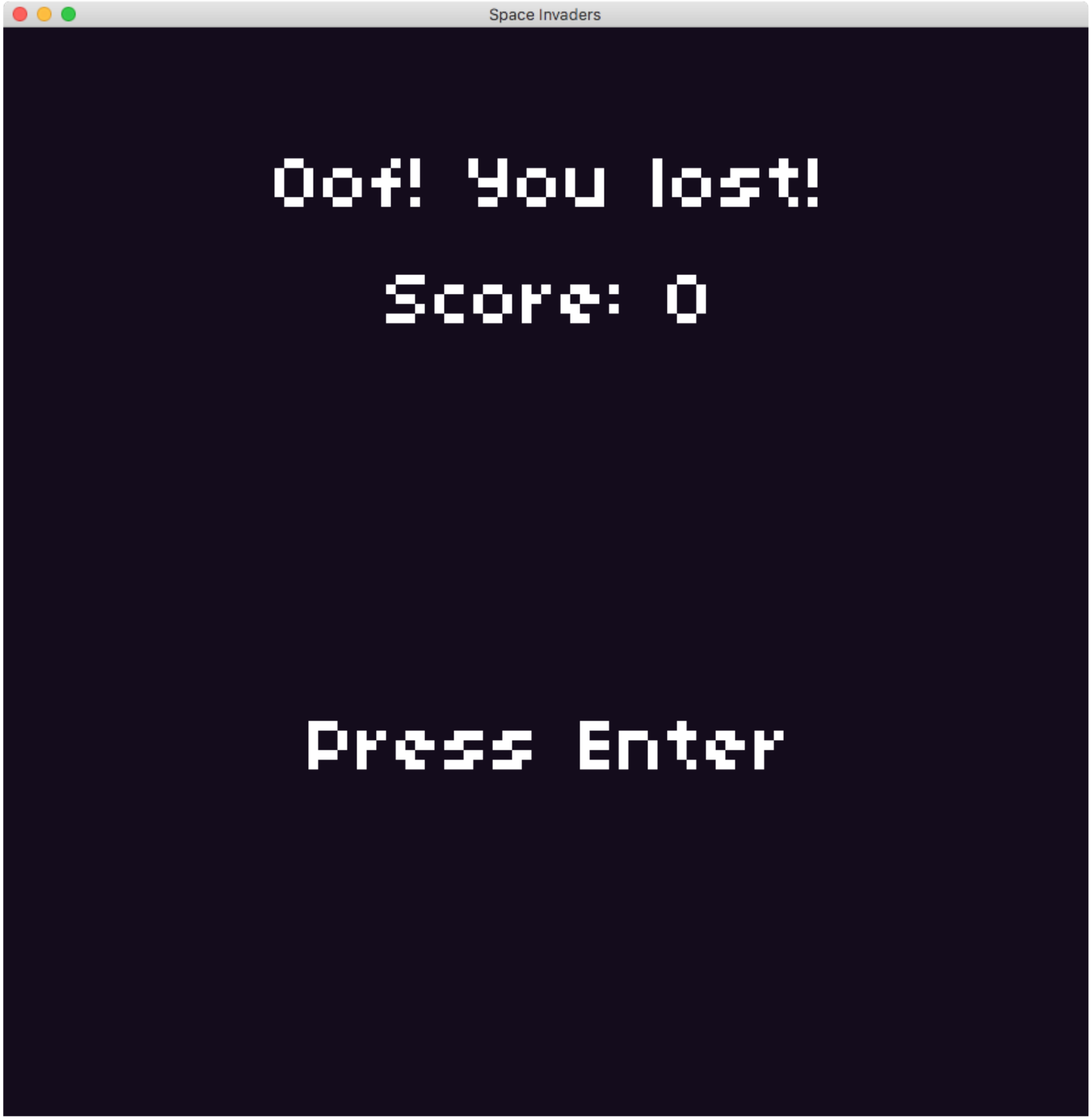


- `PlayState` This will be the state in which the action happens; we move our ship, we shoot at `Alien` s, they shoot at us, we destroy them (or they destroy us :(), and we keep track of our current level, score, and the high score for reference. When we finally lose all of our lives, we should transition to the `ScoreState` !


Space Invaders
Score
0
Level: 1
Lives: 3
High Score
160



- `ScoreState` In the `ScoreState`, we should simply display the player's score, along with the high score. If they set a new high score, they should get an extra message. Hidden from view, this state should also write a new high score to our save file before transitioning back to the `TitleScreenState`.


Space Invaders
Oof! You lost!
Score: 0
Press Enter

Players and Aliens

The primary "entities" in our game world (more on entities to come in future lectures!) are the `Player` and the `Alien`. These should be classes that you define (skeleton code in the distro). Within each of these should minimally be code to handle rendering so we can simply call `player:render` and `alien:render` (or similar) from our `PlayState` class. The rendering can either draw entire textures that each `Alien` or `Player` references that is 16x16, or however large you would like your entities to be, or you can use a sprite sheet or "atlas" of many sprites combined into one file and split it using `love.graphics.newQuad`; the choice is yours (and feel free to use the default `graphics/sprites.png` that comes with the distro)! Each should also have an `update` method to be called from the `PlayState`, and the `Player` class's should minimally have input handling; recall that in Flappy Bird, we created a global input table that would allow us to track input beyond just `main.lua`; we'll probably need a similar mechanism if we want to fire lasers with a simple press of the space bar from within our `Player` class! Just like in Flappy Bird as well, we'll likely need a new function to check the table.

Pew Pew

We can't do a whole lot against the `Alien` invasion if we don't have a means of defending ourselves, so let's shoot some `Projectile`s! You'll find skeleton code for a `Projectile` class tucked away in the distro code. `Projectile`s are simple but the backbone of our gameplay. Note that they can be fired in two directions: upwards if we fire them from the `Player` and downwards if we fire them from the `Alien`s. Note also if watching gameplay for Space Invaders that only the bottommost aliens (the lowest row) in the formation can fire at the `Player`, so let's adopt that gameplay trait as well. You can choose to implement two separate classes for the `Projectile` if you'd like, but it's quite easy to do it with just one class, since the `Projectile`s are functionally similar except for which direction they're traveling in, which sounds like it would make a great class field we can adjust and update based on! We can think of `Projectiles` in terms of how we handled `Pipe`s in Flappy Bird; objects that spawn given certain criteria and then despawn given certain other criteria. In this case, the criterion for firing a laser can be random for the `Alien`s but dependent on pressing Space for the Player. Similarly, the criterion for despawning a shot from the laser (let's call it a `Projectile`) can be that it goes beyond the top or bottom edge of the screen, parallel to what we saw with Flappy Bird!

I'm Hit!

Tying into the point about `Projectile`s, they're not much use if they can't collide with `Alien`s and `Player`s to influence the battlefield. Using what we've learned in prior lectures about simple AABB collision detection, ensure that on hit, `Alien`s disappear and the `Player` not only disappears but loses a life. When all `Alien`s

are cleared, the map should reset with aliens and the level counter should increment, thereby also increasing the speed of the `Alien` s (since their speed should ultimately scale with the current level). If the `Player` loses all of his lives, on the other hand, this should trigger a "game over" and the subsequent transition to the `ScoreState` , where we can see again our high score and whether we just beat it!

Zigzag

A characteristic of the `Alien` s in Space Invaders is their movement style: side-to-side, slowly descending toward the `Player` each time they hit a left or right wall. Implement this style of movement scaled by our current level (which should start at 1), with the level to increment each time we clear a full formation of `Alien` s. They should start centered along the X axis and immediately begin moving to either the right or the left, your choice; as soon as they reach the wall, however, they should move down by a small amount and then begin the exact same process, only in the opposite X direction. Here is some pseudocode to help illustrate (note the nature of our LÖVE app precludes literally using a `while` loop; pretend this is the LÖVE2D flow at large):

```
while aliens are alive:
    if current_direction is left:
        move all aliens to the left by some step * level
        if leftmost alien is touching or past the left wall:
            move all aliens down slightly
            shift direction to the right
```

```
else:
    move all aliens to the right by some step * level
    if rightmost alien is touching or past the right wall:
        move all aliens down slightly
        shift direction to the left
if an alien collides with the player:
    decrement the player's lives by 1
    reset the alien formation
```

Score!

It's of course important that we keep track of just how awesome we've gotten at playing Space Invaders, so we need a means of keeping track of our score; a simple variable we can pass between our states will suffice, and we should get some amount of points each time we kill an `Alien`. Make sure to display this variable in your `PlayState`, as well as the current high score for reference! Additionally, we have to know how many lives we have at all times to ramp up the tension, so make sure to display this on the screen as well; the top is the natural choice for both of these, above where `Alien`s can spawn.

Persistence is Key

An important touch for our game will be to make sure our high scores that we work so hard for are saved game to game, to a file; the important functions we'll need here are in `love.filesystem`, which you can read about here; in particular, we'll need `love.fileystem.read` and `love.filesystem.write`, which will read and write a particular file from and to our file system and give us the persistence we're

looking for. We'll just need to store a string containing our score ultimately, but make sure to cast it appropriately when reading, particularly, since by default the data will be in the form of a `string` and not a `number` !

What's That Sound?

The final piece to our retro puzzle is adding sound effects, which the distro includes for you (but feel free to generate some using bfxr!). Recall that we just need to use `love.audio.newSource` for this, as well as the `play` method on each audio object; aside from that, it's as easy as calling the function wherever you want a sound to play! In particular, make sure you add sound effects for firing lasers (both from the `Alien` and the `Player`) and when you or an `Alien` is defeated. Have fun testing your game with a new layer of feedback!

How to Submit

Step 1 of 1

1. Go to the GitHub page for your **username/project0-username** repository (note: this is different from the **games50/project0-username** repository).
2. On the right side of the screen, click the Pull request button.
3. Make sure that the "base fork" is `games50/project0-username` , and the "head fork" is `username/project0-username` .
4. Click "Create pull request".
5. On the next page, click the "Create pull request" button again.

Congratulations! You've completed Project 0.