Assignment 3: “Match-3, The Shiny Update”

Objectives

- Read and understand all of the Match-3 source code from Lecture 3.
- Implement time addition on matches, such that scoring a match extends the timer by 1 second per tile in a match.
- Ensure Level 1 starts just with simple flat blocks (the first of each color in the sprite sheet), with later levels generating the blocks with patterns on them (like the triangle, cross, etc.). These should be worth more points, at your discretion.
- Creat random shiny versions of blocks that will destroy an entire row on match, granting points for each block in the row.
- Only allow swapping when it results in a match. If there are no matches available to perform, reset the board.
- (*Optional*) Implement matching using the mouse. (Hint: you’ll need `push:toGame(x,y)`; see the `push` library’s documentation here for details!

Getting Started

Download the distro code for your first game from https://cdn.cs50.net/games/2019/x/assignments/3/assignment3.zip and unzip `assignment3.zip`, which should yield a directory called `assignment3`.

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 `assignment3` (recall that the `cd` command can change your current directory), and run

```
cd assignment3
```

A Match (3) Made in Heaven

Welcome to your fourth assignment! There was a lot to learn with timers, tweens, and more in this lecture, but unfortunately, our game is still lacking in a few areas. By extending its functionality, we’ll have something even closer to famous titles such as *Bejeweled* and *Candy Crush Saga*!

Your goals this assignment:

- *Implement time addition on matches, such that scoring a match extends the timer by 1 second per tile in a match.* This one will probably be the easiest! Currently, there’s code that

calculates the amount of points you'll want to award the player when it calculates any matches in `PlayState:calculateMatches`, so start there!

- *Ensure Level 1 starts just with simple flat blocks (the first of each color in the sprite sheet), with later levels generating the blocks with patterns on them (like the triangle, cross, etc.). These should be worth more points, at your discretion.* This one will be a little trickier than the last step (but only slightly); right now, random colors and varieties are chosen in `Board:initializeTiles`, but perhaps we could pass in the `level` variable from the `PlayState` when a `Board` is created (specifically in `PlayState:enter`), and then let that influence what `variety` is chosen?
- *Create random shiny versions of blocks that will destroy an entire row on match, granting points for each block in the row.* This one will require a little more work! We'll need to modify the `Tile` class most likely to hold some kind of flag to let us know whether it's shiny and then test for its presence in `Board:calculateMatches`!
- *Only allow swapping when it results in a match. If there are no matches available to perform, reset the board.* There are multiple ways to try and tackle this problem; choose whatever way you think is best! The simplest is probably just to try and test for `Board:calculateMatches` after a swap and just revert back if there is no match! The harder part is ensuring that potential matches exist; for this, the simplest way is most likely to pretend swap everything left, right, up, and down, using essentially the same reverting code as just above! However, be mindful that the current implementation uses all of the blocks in the sprite sheet, which mathematically makes it highly unlikely we'll get a board with any viable matches in the first place; in order to fix this, be sure to instead only choose a subset of tile colors to spawn in the `Board` (8 seems like a good number, though tweak to taste!) before implementing this algorithm!
- *(*Optional*) Implement matching using the mouse. (Hint: you'll need `push:toGame(x,y)`; see the `push` library's documentation here for details!* This one's fairly self-explanatory; feel free to implement click-based, drag-based, or both for your application! This one's only if you're feeling up for a bonus challenge :) Have fun!

How to Submit

1. If you haven't done so already, visit this link, 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**.

2. Using Git, push your work to `https://github.com/me50/USERNAME.git`, where `USERNAME` is your GitHub username, on a branch called `games50/assignments/2019/x/3` or, if you've installed `submit50`, execute

   ```
   submit50 games50/assignments/2019/x/3
   ```

   instead.

3. Record a 1- to 5-minute screencast in which you demonstrate your app’s functionality and/or walk viewers through your code. Upload that video to YouTube (as unlisted or public, but not private) or somewhere else.
4. Submit this form.

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