Programming Languages (Coursera / University of Washington) Assignment 6

Note: Playing Tetris is fun, but playing too much can make the assignment take longer than necessary.

Overview

This assignment is about a Tetris game written in Ruby. There are four Ruby files involved:

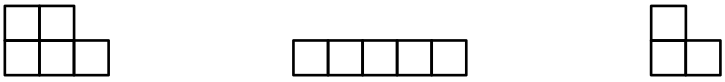
1. The Ruby code in `hw6provided.rb` implements a simple but fully functioning Tetris game (see, for example, `http://en.wikipedia.org/wiki/Tetris#Gameplay`).
2. In `hw6assignment.rb`, you will create a second game that is Tetris with some *enhancements*, described below. This is the only file you will submit (other than, as usual, a file you used for testing).
3. The Ruby code in `hw6runner.rb` is the main starting point. From the command-line (**not** within `irb`), you can run `ruby hw6runner.rb` to play a Tetris game that includes your enhancements. To use the original game rather than your code in `hw6assignment.rb`, run `ruby hw6runner.rb original`.
4. The Ruby code in `hw6graphics.rb` provides a simple graphics library, tailored to Tetris. It is used by the Tetris game in `hw6provided.rb`. Your code can also use the classes and methods in `hw6graphics.rb` (as well as the classes and methods in `hw6provided.rb`), except for the methods marked with comments as not to be called by student code. Your code *cannot* use the Tk graphics library directly.

You will turn in only `hw6assignment.rb`, so your solution cannot involve modifying any of the provided code. Instead, use subclassing (since, after all, this is the OOP portion of the course) to nonetheless *reuse* as much of the code as possible. Some code-copying will still be necessary, but you should try to minimize it. Your solution should also *not* change any of the provided classes in any way. You will define subclasses that behave differently. As a result, the provided Tetris game will run without change.

Much of the work in this assignment is understanding the provided code. There are parts you will need to understand very well and other parts you will not need to understand. Part of the challenge is figuring out which parts are which. This experience is fairly realistic.

Enhancements

1. In your game, the player can press the '`u`' key to make the piece that is falling rotate 180 degrees. (Note it is normal for this to make some pieces appear to move slightly.)
2. In your game, instead of the pieces being randomly (and uniformly) chosen from the 7 classic pieces, the pieces are randomly (and uniformly) chosen from 10 pieces. They are the classic 7 and these 3:



   The initial rotation for each piece is also chosen randomly.
3. In your game, the player can press the '`c`' key to *cheat*: If the score is less than 100, nothing happens. Else the player loses 100 points (cheating costs you) and the next piece that appears will be:

   The piece after is again chosen randomly from the 10 above (unless, of course, the player hits '`c`' while the "cheat piece" is falling and still has a large enough score). Hitting 'c' multiple times while a single piece is falling should behave no differently than hitting it once.

How To Run The Game

We recommend you *not* use `irb` to load `hw6runner.rb` and start a game. We have not had success getting the graphics library to restart properly more than once inside `irb`, so you would likely have to exit the REPL after playing each game anyway. You can use the REPL for testing individual methods and exploring the program, but to launch a game, go outside `irb` and run `ruby hw6runner.rb` (or to make sure the original unenhanced game still works correctly, run `ruby hw6runner.rb original`). Make sure the 4 Ruby files are in the same directory and run the `ruby` command (or `irb` when exploring) from that directory.

Requirements

For our testing scripts to work, you **must** follow these guidelines.

- Your game should have all the features of the original Tetris game, as well as the enhancements.
- The subclasses you create must start with `My` followed by the name of the original class. For example, your Tetris class should be called `MyTetris`. We have provided empty class definitions for you to complete. You do not need any other classes.
- Do not add to or modify any classes defined in other files or the standard library.
- It is required that your board `MyBoard` has a `next_piece` method that provides the same functionality that `Board`'s `next_piece` provides, which is that it sets `@current_block` to the next piece that will fall, which might or might not be the cheat piece.
- You must have a `MyPiece` class and it must define a class constant `All_My_Pieces` that contains exactly the ten "normal" pieces (i.e., the initial seven plus the three additional pieces from enhancement two). It must not contain your cheat piece. It must be in the same format as the `All_Pieces` array in the provided code.
- All your new pieces, including your cheat piece, must use the same format as the provided pieces. Hint: Be particularly careful to have enough nesting in your arrays or your game may be subtly, *almost* imperceptibly, incorrect.
- **Do not use the Tk library directly in any way.** The only use of Tk should occur indirectly by using instances of classes defined in `hw6graphics.rb` as needed (only a little is needed). Do not have `require 'tk'` in your `hw6assignment.rb` file (or any other use of `require` for that matter).

Advice and Guidance

- For the piece you are adding that has 5 squares in it and is not a line, be sure to add the piece as pictured and not its mirror image.
- It takes time to understand code you are given. Be patient and work methodically. You might try changing things to see what happens, but be sure you undo any changes you make to the provided code.
- The sample solution is about 85 lines of code. As always, that is just a rough guideline; your solution might be longer or shorter.
- While you should not copy code unless necessary, some copying will be necessary. After all, the original game may not have functionality broken down into overridable methods the way you would want and you are not allowed to change the provided code. This is a fairly realistic situation.

(Lack Of) Challenge Problem

The best thing to do to continue the assignment is to implement an additional enhancement (or more) to your game, but we do not have a reasonable way to grade this and additional enhancements could possibly mess up auto-grading. So:

- This assignment will not have a graded challenge problem.
- Be sure to turn in a solution that adds *only* the three asked-for enhancements.
- Then enjoy trying additional enhancements. Be creative.

We encourage you to share on the discussion forum *what you did* but **not** *how you did it* as that would take away the fun from others who might try to repeat your accomplishment.

Turn-in Instructions (same as usual except as described on the website to support multiple Ruby versions): First, follow the instructions on the course website to submit your solution file (not your testing file) for auto-grading. Do not proceed to the peer-assessment submission until you receive a high-enough grade from the auto-grader: Doing peer assessment requires instructions that include a sample solution, so these instructions will be "locked" until you receive high-enough auto-grader score. Then submit your same solution file again for peer assessment and follow the peer-assessment instructions.