CSE 341, Spring 2023, Assignment 6
Due: Friday May 26, 5:00PM

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 Racket. There are four Racket files involved, available from the course website:

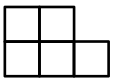
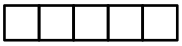
1. The Racket code in `hw6provided.rkt` implements a simple but fully functioning Tetris game (see, for example, `http://en.wikipedia.org/wiki/Tetris#Gameplay`).
2. In `hw6assignment.rkt`, you will create a second game that is Tetris with some *enhancements*, described below. This is the only file you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

You will turn in only `hw6assignment.rkt`, 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 try to minimize it.

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.

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 `my-tetris%`. We have provided empty class definitions for you to complete. You do not need any other classes.
- All your new pieces, including your cheat piece, must use the same format as the provided pieces.
- **Do not use the Racket graphics library directly in any way.** The only use of `racket/gui` should occur indirectly by using instances of classes defined in `hw6graphics.rkt` as needed (only a little is needed). Do not have `(require racket/gui)` in your `hw6assignment.rkt` file.
- Make it so subclasses of `my-tetris%` could add also add additional keybindings to keys not used by `my-tettris%`.

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 55 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 exactly the way you would want, and you are not allowed to change the provided code. This is a fairly realistic situation.

Challenge Problem

First, in `hw6assignment.rkt`, create classes `my-tetris-challenge%` and `my-board-challenge%` that subclass the classes you modified to complete your enhancements. Second, modify `hw6runner.rkt` so that running `racket hw6runner.rkt --challenge` uses these new classes. Third, in these classes add a fourth enhancement to your game that is interesting and not similar to the enhancements already required. Fourth, in a short text file `challenge.txt`, explain what your enhancement is and how you implemented it. Enhancements that are particularly easy will not necessarily receive credit; aim for something at least as substantial as the required enhancements and perhaps affecting a different part of the game. Fifth, make sure your main solution (what runs with `racket hw6runner.rkt`) is not affected — your fourth enhancement should be only in your new subclasses.

Your challenge problem may use the `racket/gui` graphics library directly if you wish — you are not limited to the functionality in `hw6graphics.rkt`.

Turn-in Instructions

- Put your solutions in `hw6assignment.rkt` and tests in `hw6tests.rkt`. Focus on testing small pieces of your code; we do not expect you to test the GUI automatically.
- If you do the challenge problem, also turn in `hw6runner.rkt` and `challenge.txt`.
- Upload your files to Gradescope.