CSE 341, Fall 2004, Assignment 7 (version 1)
Due: Friday 10 December, 9:00AM

Overview: You will write a Tetris game that has non-standard features and shapes. You will do this by subclassing a Tetris game already implemented in Squeak. Most of the work for the assignment is understanding the code already in Squeak and *using it without changing it.* The sample solution has a total of 58 lines of method bodies, and 27 of those lines are copied from code already in Squeak.

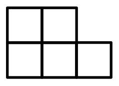
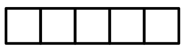
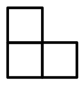
To play Tetris, you can evaluate `Tetris new initialize; openInWorld` or from the "World" menu choose "new morph" then "from alphabetical list" then "T-T" then "Tetris". Similarly, you will be able to play your game by evaluating `MyTetris new initialize; openInWorld` or from the "World" menu choosing "new morph" then "from alphabetical list" then "L-O" then "MyTetris".

Requirements:

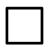
- Your game must work if a user evaluates `MyTetris new initialize; openInWorld`.
- The built-in Tetris game must work unchanged. Warning: This may be more difficult than it seems.
- Your game must have the "enhancements" described below.
- You must write a short essay as described below.
- Your game must still have all Tetris features work. Warning: An initial sample solution broke most of the buttons in the game, so test for this.
- You must write code only in category `hw7`, which you should create. You must *not* change the implementation of any existing classes.
- Your game must work in an unchanged Squeak image once the classes in `hw7` are "filed in". Warning: Test your solution in a fresh image.

Enhancements:

1. In "MyTetris", the player hits the '`u`' key to make the piece that is falling rotate 180 degrees. (It's okay if this makes the piece appear to "move left or right" for some pieces.)
2. In "MyTetris", 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:

   As in the built-in Tetris, the rotation for each piece is also chosen randomly.
3. In "MyTetris", the player hits 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).

Essay: Write a *short* essay (approximately 1–2 pages) explaining how you implemented the enhancements. Discuss how the Tetris implementation made your job more or less difficult and how your job would have been easier if you could have changed the provided code. (Even if you could change the provided code, you still need the Tetris game to *appear* unchanged.)

Assume the reader is intimately familiar with the provided Tetris code; do not explain how it works.

Extra Credit:

Add a fourth enhancement to "MyTetris" that is interesting and not similar to the enhancements already required. Amend your essay to briefly explain what your enhancement is, how you implemented it, and how the existing code base made your enhancement easy or difficult.

Do not share your enhancement with anyone unless they already have their own. We don't want to see the same thing twice except by pure coincidence.

Turn-in Instructions

- "File out" category `hw7` and rename the file `lastname_hw7.st` where `lastname` is your last name.
- You may write your essay in a text document (`lastname_hw7.txt`), a pdf document (`lastname_hw7.pdf`), or a Microsoft Word document (`lastname_hw7.doc`).
- Email your solution and essay to `daverich@cs.washington.edu`.
- Include your two files as *attachments*.

Hints:

1. First understand how the relevant parts of the Tetris game are implemented. You need to know the overall structure of the classes, how keyboard events and buttons are handled, how shapes are represented, and how shapes are chosen. Don't forget about class variables and methods.
2. Do the enhancements in order, testing as you go. Use sends to `super` to avoid doing more work than necessary, but unfortunately some code copying will be required. Keep notes for your essay.
3. For enhancement 1, you need only one class and one overloaded instance method. To rotate 180 degrees, you can rotate 90 degrees twice.
4. Enhancement 2 is the most difficult. You will need two more classes (3 total). The sample solution has one (more) overriding instance method in each, one new class variable, one new class method (which could be removed; it helped for development), and one overriding class method.
   - Avoid mutating variables used in the Tetris game; doing so will change the built-in game. Use new variables instead.
   - Remember that class variables aren't ever reset unless you explicitly do it. While developing, you will probably want to do so explicitly. Also make sure your game correctly initializes any class variables when run in a fresh image.
   - Think carefully about whether your overriding methods should send messages to `super`:
     - In two cases, it's easier not to: just duplicate the necessary functionality. Be careful: the built-in game assumes all pieces have four squares and that's no longer true.
     - In the other case, you'll need call `super` in order to get the effect of calling the super super class's method, but then you'll need to *undo* the work done in the super class's method, and replace it appropriately. (There's no way to do a resend to the super super class.) Fortunately, it's easy to remove morphs.
5. For enhancement 3, the sample solution added 1 new instance variable, 2 new instance methods, and changed 2 methods written for earlier enhancements. Here the "division of responsibility" is the most interesting part: When '`c`' is hit, if the score is high enough, subtract one hundred and "remember" that it was hit. When choosing a piece, if it's time to cheat, return the cheat-piece and "forget" that you should cheat. Remember the cheat-piece should appear *only* when the player cheats.