Tetrominos: the game

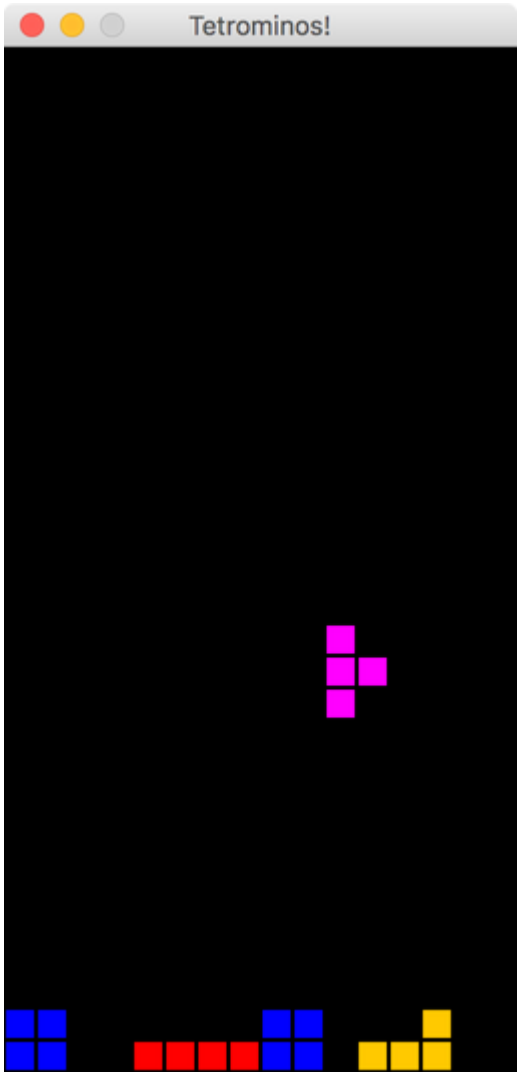
Get ready to write a falling-blocks puzzle game.

WE'LL COVER THE FOLLOWING

You now have the tools and techniques needed to write interesting and complete Java programs. In the lessons of this tutorial, we'll walk through writing a complete falling-blocks puzzle game. To do this, you'll need to install a Java development environment on your computer, but first, let's talk about the rules of the game and some elements of how we might design the code.

The basic rules of the game are the same as for the game Tetris™; you can easily find several browser-based implementations of Tetris with a web search.

A *tetromino* is a shape composed of four blocks. Here's a screenshot of the game in action, with five tetrominos on the screen; there are only a few types of tetrominos, and they are colored by type in the screenshot.



Tetrominos fall downwards, and can be rotated or moved side to side under keyboard control. I used the `s` key to move a tetromino to the left, the `f` key to move the tetromino to the right, the `d` key to rotate to the left, and the `g` key to rotate to the right.

If a horizontal row is completely filled with blocks, that row is destroyed and all blocks above that row fall downwards one row. The goal of the game is to destroy as many rows as possible before the board becomes so cluttered that a new piece cannot start falling from the top.

Graphics and user interface libraries

There are a few things you'll need to know to get started. First, you'll need to know about how to draw things on the screen. So far, you've used the `io.educative.graphics` package, but this package is only for web demonstrations. You'll also need to know how to collect keyboard input from the user in order to control the motion of the pieces.

I used the `Swing` and `awt` libraries included with Java to do the drawing, although there are other choices. These are very large libaries; we'll look at the pieces we need in a few lessons.

Code design

Before writing a program of any interesting size, you should sit down with pencil and paper and plan out the code design. With Java, this usually entails mapping out what classes you'll need and how they will work together. You should also think about what instance variables and methods each class will include.

Don't implement anything yet! Work through the design carefully, and read through the next remarks. The clearer a picture you have before you start coding, the better it will go. I usually take a day or two to think over my plans, and you should too.

Modeling blocks with the `Block` class

A good approach to design works both from the bottom up and from the top down, seeking to meet in the middle. For example, we can see in the game that there are square blocks. Those blocks are sometimes arranged into a falling tetromino, and sometimes they are just left over on the board from a previous tetromino. At the lowest level, we need to model those blocks somehow.

Since there are blocks of different colors, each block should keep track of what color it is. I created a `Block` class with an instance variable storing that color.

The game board will keep track of the location of each block somehow. Therefore, I decided that a block didn't need an instance variable to keep track of where it was.

Modeling the board with a `Board` class

In my design, most of the work for the game is handled by the `Board` class. The Board:

1. contains an array of Squares representing left-over squares on the board;

2. contains an active tetromino object that is currently falling;
3. has a method to draw the board;
4. has a method to move the active piece downwards each game turn;
5. has methods to slide the active tetromino piece left or right or rotate it;
6. has a method for testing for collisions between squares as pieces are moved;
7. has a method for checking for complete rows, and for destroying them as needed.

That's a lot, and it's too much to implement all at once. For an implementation, just putting some blocks into an array and drawing them on the screen would be a great first step.

Modeling the active tetromino with the `Piece` class

There are several types of tetromino. I used the `Piece` class to construct and store various types of tetrominos.

As usual, it's good to work out a complete design for the Piece class, but be flexible. When you implement a class to represent the Piece, start with something simple. Perhaps you might start with a falling piece that is just a single block, rather than four blocks. Then add a method to translate the piece, maybe add the color, and a `draw` method.

Next steps

Ok, with design thought through, and a tentative plan for what you might implement first, it's time to start coding. The next lessons will:

1. Introduce you to an IDE for developing code.
2. Introduce you to the AWT and Swing and graphics and user interface libraries.
3. Provide an implementation roadmap.