Lab 10: Tetris

FAQ

Each assignment will have an FAQ linked at the top. You can also access it by adding "/faq" to the end of the URL. The FAQ for Lab 10 is located here.

Introduction

This lab will help you get started with the second phase of the project: Interactivity. You are not expected to be done with Phase 1 at the time you're working on this lab. If you have not started already though, we highly recommend that you do!

For this lab, you should consider how some ideas (or implementations!) may translate over to Project 3. It will also help you gain more familiarity with useful tools necessary for the project.

> **WARNING**
> This lab has a lot of helper methods - you do not need to know what every single method does, but please be aware that you will have to call on some of them throughout the lab. You will also end up using methods from a library. **Please make sure to read through the files to understand what you're working with!** As a reminder, the coordinates (0, 0) represent the bottom left of the board.

Tetris

In preparation for making your game, we will be constructing the game Tetris! If you're not too familiar with Tetris, it's a puzzle video game where players "complete" lines while differently shaped pieces (called tetrominoes) spawn and descend on the board. Players can move and rotate these pieces as needed to complete lines - if there are any lines completed, they disappear and the player gains points. The game ends when the uncleared lines reach the top of the board.

All of your implementation will be in `Tetris.java`. We have also provided three other files:

- `Tetromino.java`: contains the board pieces that you'll be using

- `Movement.java` : contains logic for rotating and moving the pieces
- `BagRandomizer.java` : helps randomize the pieces that are spawned

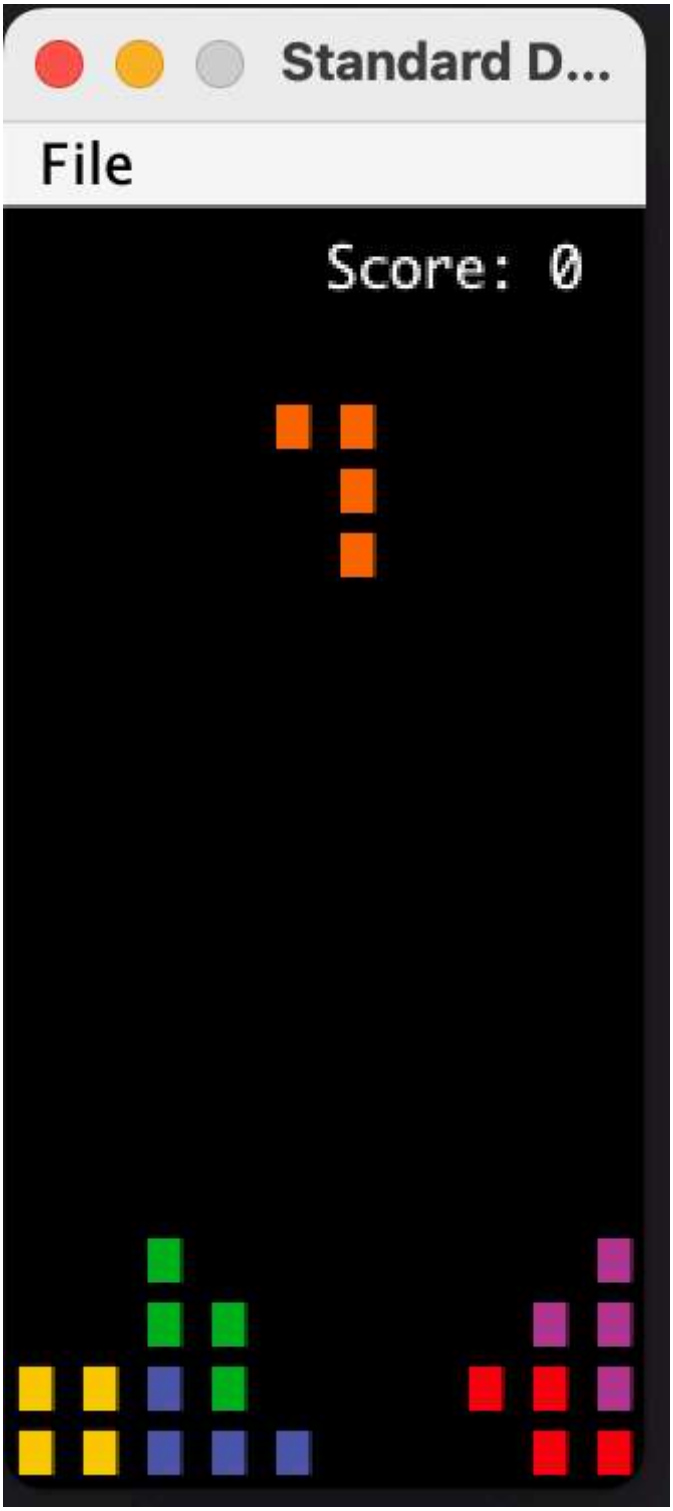
You won't be writing anything in those three files directly, but these classes are used in `Tetris.java` (or will be). We'll also be working with a library, `StdDraw` , to help implement some features, such as user input. You will find this library very useful for Project 3.

While we want to have a working game of `Tetris.java` , we should break it down into smaller steps instead of tackling it all at once. The game can be loosely broken down into the following steps:

1. Create the game window.
2. Randomly spawn a piece for the player to control and keep a display of the current score.
3. Update the movements of the piece based on the player's input.
4. Once the piece can no longer move, check if any lines need to be cleared, update the score and respawn a new piece.
5. Repeat steps 2-4 until the game is over (when the uncleared lines reach the top).

In general, good coding practice is to first build small procedures with explicit purposes and then compose more complex methods using the basic ones. If you take a look at `Tetris.java` , you'll note that it contains many helper methods to help build a more complex game mechanism - breaking the game logic into individual methods is highly recommended for Project 3. It will give you a clear path forward in development and will be easier to break down your logic for unit testing.

By the end of the lab, you'll have something that functions like the below:



If you would like to play the game yourself, you can find it here.

StdDraw

As mentioned, we'll be working with the library `StdDraw`. `StdDraw` is a provided library that gives basic capabilities to create drawings in your program as well as grab user input. Please take a look at the API before getting started as you will find some of the methods useful not only for this lab, but for Project 3.

Running the Game

To run the game, run the main method in `Tetris.java`. Currently, this should output nothing but a black box. As you implement more methods, you can use this method to verify the correctness of your game logic.

Methods Overview

Since all of your implementation will be in `Tetris.java`, there are a couple of methods you'll have to fill out to get your game working.

> **INFO**
> As mentioned earlier, please make sure to read through `Tetris.java`, `Tetromino.java`, and `Movement.java` to gain familiarity with the helper methods in this lab. While you don't necessarily need to understand how every helper method works (abstraction!), it is likely that some of them will be helpful for your implementation, so please read through them carefully so that you are aware of what is available to you.

`updateBoard`

This method updates the board based on user input. The first step is to check if the user has typed in anything and grab the input if the user has. There will be some methods from the `StdDraw` API that will be useful for implementing this part of the lab. Consider looking at `hasNextKeyTyped` and `nextKeyTyped`.

The next step is to implement the actions that are taken from specific keys. The user is able to input 5 keys:

- `a`: move the current piece towards the left by one tile
- `s`: move the current piece downwards by one tile
- `d`: move the current piece towards the right by one tile
- `q`: rotate the current piece to the left 90 degrees
- `w`: rotate the current piece to the right 90 degrees

We recommend that you look at some of the provided helper methods in `Movement.java` to see which ones you can call on to move a piece or rotate a piece (**you should not have to implement any of the movement logic yourself, but instead, rely on understanding what the helper methods do**).

A `Movement` instance has also been provided for you to use.

> **TASK**
> Fill out the method according to the description above. As a reminder, make sure to read through `Movement.java` and the `StdDraw` API!

`incrementScore`

This is a helper method to help update your score. The player's score increases based on the number of lines that have been cleared from a single move, if any are cleared. There are four cases that this can be broken down into, as listed below:

- 1: 100 points
- 2: 300 points
- 3: 500 points
- 4: 800 points

> **TASK**
> Fill out `incrementScore` so the player's score increases as described above.

`clearLines`

Whenever a line is completed in Tetris, we want to update our score and clear the line. This method will help check if a row or multiple rows have been horizontally filled after a piece is placed. Consider the following:

- Since you don't know exactly which row is completed, if there are any, we'll want to check for the entire board.
  - How do you know when a row is complete? Specifically, when do we know when it's *not complete*?
- Once you find a row that is completed, that row needs to be cleared.
  - When you clear a row, all of the rows above it need to be shifted down.
  - For each row that is cleared, keep track of it in a variable (`linesCleared`).
- At the very end, we want to update our score based on the number of lines cleared. Consider using a helper method that you've implemented to do this.

> **TASK**
> Fill out `clearLines` to check for the amount of lines that are cleared and update the board accordingly. **The board should be passed in as an argument, so please make sure to use the**

> **argument `tiles` (it might affect the autograder if you don't!).**

`runGame`

This is where the main game logic takes place. Comments have been left in the skeleton code to help you get started. A couple of things to note:

- You'll need to ensure that the game does not exit or stop until the game is over (hint: how do you make sure this happens continuously?).
- If the current tetromino is unable to move down or can no longer move from its current position, **it is set to `null`.** The logic for setting it to `null` has been taken care of for you and you do not need to work with it.
  - Once a piece has been placed and can no longer move, make sure to check if any lines need to be cleared and spawn another piece.
- Make sure to update the board based on the user input and then render those changes.

Here are some relevant helper methods you may use, alongside the ones you've already implemented:

- `spawnPiece`: spawns a piece on the board and sets the current Tetromino piece to a randomly chosen piece
- `isGameOver`: checks for if the game is over
- `clearLines`: checks for any lines that need to be cleared and updates the score based on the number of lines cleared
  - Make sure to pass the board into `clearLines`
- `updateBoard`: checks for the player movement and updates the board based on the user's input
- `renderBoard`: renders the state of the board (called on after user input and clearing lines)

> **TASK**
> Fill out `runGame`.

`renderScore`

At this point, if you run the game, you'll notice that something is missing. The score! That's because we haven't yet displayed it. Fill out the method `renderScore` so that the score displays, and you can use this to verify if your score is updated correctly when lines are completed in the game.

Here are the steps:

- Make sure to set the color of the text to white (rgb value of (255, 255, 255)) so the score is visible against a black background.
- The score should appear at position x = 7, y = 19.
- Make sure to render the score once it's drawn!

Here are some useful methods from the `StdDraw` library that you might find helpful (you might not use all of them, but they are here for reference):

- `StdDraw.setFont`
- `StdDraw.clear`
- `StdDraw.text`
- `StdDraw.setPenColor`
- `StdDraw.show`
- `StdDraw.pause`

> **TASK**
> Fill out `renderScore` and run your game to check that the score shows up!

Submission and Grading

This lab is worth 5 points. The completion of this lab will be partly based on an autograder submission *and* checkoff. You may come to labs or office hours to get checked off in-person, or you may get checked off asynchronously (through Ed).

For the checkoff, we'll be looking for the following:

- Pieces can be moved left, down, and right with the `a`, `s` and `d` keys respectively and rotated left and right with `q` and `w` respectively.
- Other keys should not do anything (i.e. if we press on `h`, that should not affect the game board)
- The score renders throughout the game and does not disappear at any point.
- The pieces do not fall through when they are stacked on top of each other or reach the ground/bottom.

We will also be taking a look at your implementation, so make sure you're able to explain your thought process and how some of what was completed in this lab can be translated to implementing interactivity in Project 3! Once checked off, a magic word will be given for you to fill in your `magic_word.txt`, where you can then submit to Gradescope. You'll need to submit the magic word and pass the test on Gradescope to receive full score.

To reiterate, here is a brief summary of the methods to implement:

- `updateBoard` : updates the board based on user input
- `incrementScore` : updates the score based on number of lines cleared
- `clearLines` : clears the rows if they're completed
- `runGame` : where the game logic takes place to run the game
- `renderScore` : displays the score

Asynchronous Checkoffs

If you're getting checked off asynchronously through Ed, you will need to provide a screen recording. We suggest using Zoom screen recordings or a screen recording software. During this recording, we will need to see an **on screen keyboard**. Here are a few links for an on-screen keyboard:

- Mac
- Windows

Please make sure to answer the questions that are provided on the template and please plan ahead of time as it will take time to make the recording and to get checked off.

Credits

4 TAs were harmed in the creation of this lab (Noah Adhikari, Erik Nelson, Omar Yu and Jasmine Lin).
