Tetris

**Early Due Date:** Friday, November 20, 10:00pm  
**On Time Due Date:** Sunday, November 22, 11:59pm  
**Late Due Date:** Tuesday, November 24, 10:00pm

**Design Checks:** Monday, November 9 - Thursday, November 12  
**Design Check Signups:** You must sign up for a DC by Sunday, November 8, 11:59pm  
Refer to the [DQ Handout](#) for instructions

**Help Session:** Thursday, November 12, 6:00 - 8:00pm in MetChem Auditorium

Run demo: `cs015_runDemo Tetris`  
Run snazzy demo: `cs015_runSnazzyDemo Tetris`  
*Demos may not work well over ssh!*

To install: `cs015_install Tetris`

---

**Silly Premise**

**New Concepts Covered**  
**Assignment Specifications**  
**Important Prelude**  
**Design Considerations**  
**Bells & Whistles**  
**Handin Info**

---

**Silly Premise**

Meanwhile, on the Wildcats basketball court Tetris Board...

KeyHandler said to move right and move left  
Watch out for other blocks, and keep an eye on the edge  
Gotta run the Timeline and go  
And get the squares to stay low  
But don't be afraid  
When the piece is mislaid  

Just get full rows in the game
And keep ya head in the game

And don't be afraid
At the bugs your code made
Just keep ya head in the game
You gotta get'cha, get'cha, get'cha, get'cha
head in the game!

New Concepts Covered

- Arrays
- The Factory Pattern
- Designing a large program

Assignment Specifications

Your assignment is to write the CS15 version of the incredibly addictive game of Tetris. If you are not familiar with Tetris, you should run the demo as soon as possible in order to see how the game works. There are many renegade versions of Tetris out there, with slight differences among them; to give you a better idea of what your assignment entails, read the following description of how the CS15 version of Tetris behaves.

When the game starts, only an empty board with borders drawn around its edges should be displayed. A Tetris piece, chosen randomly from the seven possible Tetris pieces shown below, should appear at the top of the board.

This piece should fall by moving down the board, one row at a time. A piece cannot fall into a square already occupied by a previously fallen piece. When a piece can fall no further, it should stop moving. A new random piece should then appear at the top of the board and begin to fall.

As pieces fall, rows (or horizontal lines) of occupied squares spanning the board's width may form. When such a line is formed, it disappears and all the squares above it fall down one line to fill the newly empty row. This process continues until there is either a piece in the top row of the board or a new piece appears and has no room to fall because it is already resting on a previously fallen piece. The game is then over, and everything on the board should stop completely - pieces should stop being generated or moving, and the piece should not be able to be moved by pressing any keys. A message should be displayed to let the user know that the game is over.
The seven Tetris pieces: Each is an arrangement of four connected squares

While a piece is falling, the player may rotate or shift it by pressing certain keys on the keyboard. Pressing the left arrow should shift the piece one square to the left. Pressing the right arrow should shift the piece one square to the right. Pressing the up arrow should rotate the piece counter-clockwise by ninety degrees. At regular intervals, the piece should fall one row at a time. Pressing the down arrow should drop the piece an additional row each time, making the piece fall more quickly. The player should be able to drop the piece by pressing the spacebar. By dropping a piece, the player forfeits his/her chance to manipulate the piece any further and the piece simply falls as far as it can.

The player should be able to pause the game at any time by pressing ‘p’. Pressing ‘p’ again should allow the user to resume play. When the game is paused or over, text should be displayed to notify the user, and the user should no longer be able to manipulate pieces.

Important Prelude

As with DoodleJump, you should start early and code incrementally. Design the entire program before you start coding! Come to TA Hours only with bugs that you can't solve on your own (and with conceptual questions as they arise). The Eclipse Debugger and the other resources we've provided will help you fix most problems that you face.

If you go to TA hours seeking debugging help, be prepared to show the TA extensive debugging efforts. If the TA feels that you haven’t spent enough time trying to solve the bug, they have the right to refuse you. It is in your best interest to remove yourself from the list if you resolve your bug or feel as though you haven’t debugged sufficiently, if you don’t you will get turned away and have to wait an hour to sign up again (as per the Signmeup policy).

“I’ve never met a bug I couldn’t solve with a kiss...I mean... Eclipse...”

– Prince Charming TA
Design Considerations

There are several new concepts used in this assignment. Before you start designing and coding Tetris, you should make sure that you completely understand all of the concepts listed in this handout. Be sure to review the lecture slides, read Piazza, and visit TA hours if you need further clarification. Tetris will be much easier to design and to code if you thoroughly understand these concepts before you start.

Run the Tetris demo. Once you are familiar with how the CS15 version of the game behaves, you should think critically about your design. You will want to think about how to:

- Randomly create the different pieces
- Generate new pieces modularly (easy extensibility to add new types of pieces)
- Make new pieces appear at the top of the board
- Keep track of each piece's current location
- Make pieces fall
- Shift pieces to the left or to the right
- Rotate pieces
- Handle user input
- Check if a desired move is legal - i.e., does not move a piece into a square that's already occupied or outside the edges of the board
- Keep track of where the pieces have fallen
- Check for horizontal lines
- Update the board after a horizontal line disappears
- Stop the piece when the game is paused
- Check for the end of the game
- Actually stop playing the game and start coding

Also, you will want to give special consideration to how the different pieces will rotate in a polymorphic way.

**Note:** You'll notice that in many cases, there are several different objects that could handle a desired functionality. For example, either the piece or the board could be responsible for checking move legality. When making design decisions, think carefully about the tradeoffs between designs. All designs have their pros and cons, make sure your decisions are well justified for your design check!

**Keyboard Interaction:**

It's that time again! You will once again be implementing keyboard interaction to allow the user to shift, rotate, and drop pieces, as well as pause the game.
As with before, the method we recommend involves using the interface `javafx.event.EventHandler<KeyEvent>`. You are welcome to use other ways of implementing keyboard interaction, of course, as long as they are readable and function well (but be aware that the TAs may not be familiar with a different implementation).

If you need to brush up on your keyboard-wizarding skills, refer to the Javads, [JavaFX Guide Keyboard Interaction section](#), or [DoodleJump Handout](#) for pointers.

For a complete list of the different `KeyCode`s, go to: [Javads](#).

A few important notes to keep in mind regarding keyboard interaction:

JavaFX has some built-in functionality for certain `KeyEvent`s. For example, the down arrow key moves the input focus to another node in the Scene Graph. Depending on your Tetris implementation, this may mean that pressing down would move the focus out of the Pane that holds your game and into another, which can cause all sorts of issues! There are two ways to fix this:

1. **Call the `consume()` method**
   One way to make sure the program only executes what you indicate in the `EventHandler` is to “consume” the event, which you can think of as throwing away the event after it has done everything you need it to do. To do this, call the `consume()` method on the `KeyEvent` at the end of the method you use it in. You can read more about this [here](#) (the section under “Consuming of an Event” at the bottom of the page.)

2. **Call the `requestFocus()` and `setFocusTraversable(true)` methods**
   We can also explicitly manage where focus is set. To do this, you’ll need to make sure to call `requestFocus()` and `setFocusTraversable(true)` on the Pane that listens to your `KeyEvent`s. To make sure no other node (in this case, the quit button) grabs focus inadvertently, you can call `setFocusTraversable(false)` on each one. You can read about this method [here](#).

**User Input Summary:**
Here’s a handy summary of all the required user inputs and what they should do:

- **Left Arrow Key:** Moves currently falling piece one space to the left
- **Right Arrow Key:** Moves currently falling piece one space to the right
- **Up Arrow Key:** Rotates currently falling piece 90° counter-clockwise
- **Down Arrow Key:** Moves the currently falling piece one space down
- **Spacebar:** Drops the current piece
• ‘P’ Key: Pauses/Unpauses the game

Animation and Timelines:

Like in previous projects, you’ll find a Timeline useful in controlling your pieces. Take a look at the Graphics Pt. I lecture if you need to review tips and tricks for Timelines.

Moving and Rotating Pieces:

When the user tries to move a piece a new location, the piece should move only if the new location is not already occupied by a previously fallen piece and that it is not beyond the edges of the board.

In other words, if you are keeping track of already occupied squares within a board, then as the current piece falls, you simply have to check with the board to see if the squares where it wants to move are already occupied. This same sort of checking should be used for rotations. If all squares of the Tetris piece can make a valid move, then the piece can move/rotate; if any one of these squares is not free, then the piece cannot move/rotate.

Properly rotating a piece can be tricky, but the math is quite straightforward. To move a point 90 degrees counter-clockwise in a circle around another point, you can use these formulas:

\[
\begin{align*}
\text{newXLocation} &= \text{centerOfRotationX} - \text{centerOfRotationY} + \text{oldYLocation} \\
\text{newYLocation} &= \text{centerOfRotationY} + \text{centerOfRotationX} - \text{oldXLocation}
\end{align*}
\]

where newXLocation and newYLocation are the new coordinates of the point being moved, centerOfRotationX and centerOfRotationY are the coordinates of the fixed point around which this point is moving, and oldXLocation and oldYLocation are the original coordinates of the point being moved. Note that this assumes that the positive y-axis points down (as Java does). We will cover this formula in more detail at the help session.

Piece-Board Interaction:

The relationship between a piece and the board is as follows:

- Once a piece has fallen, the squares from that piece should (1) remain on the screen in their original color, (2) block other pieces’ motion, and (3) be able to be removed from the board in rows.
  - When clearing lines:
    - You may only be removing part of what used to be a single Tetris piece.
    - Once a row has been removed, all the rows above it should move down one row.
**Remember:** Whenever you want to add or remove a Node, you need to add or remove that Node to/from the Scene Graph in order to see a visible change. Look at the DoodleJump Handout to refresh.

- Prevent pieces from moving or rotating off the edge of the board. You could do this by checking the boundaries of your board's array, but it might be easier to come up with a design that allows edge-checking to occur simultaneously when checking occupied squares within the board.

  **Hint:** Take a look at the demo and think about how you might accomplish this.

**The Pieces:**

One major design consideration for the Pieces is how to create seven different types of Piece while factoring out as much code as possible.

You will be using random numbers to decide which one of the seven possible Tetris pieces will be the next to appear (Remember Math.random() from DoodleJump?). How can you employ the *factory pattern* to your advantage? Take a look at the Making Decisions lecture for more information.

You’ll also need to consider how you want to add the individual squares in your Pieces to the Scene Graph so that they are displayed on the screen. **Hint:** Look at the JavaFX Lab for a detailed discussion of (and our recommendation for) different ways of adding shapes that are contained in a separate class to a Pane.

**Constants Class:**

Tetris is a great program in which to use constants, which you are familiar with from DoodleJump and Cartoon. For Tetris, you’ll need to make your own Constants class and define public static final constants within it.

**Bells & Whistles**

There is plenty of room for creativity in this assignment.

**Remember:** First get your program to meet the specs; then, if time permits, go ahead and add Bells & Whistles! *Late projects get no points for extra credit.*

**The Proxy Pattern:**
For this assignment, most of the KeyEvents will operate on the current falling piece. Thus your EventHandlers (and possibly other classes) will all need a way to communicate with the current piece. Luckily, for a basic Tetris, it is really only necessary to access the current piece from one class (and possibly that class's private inner classes). But what if multiple classes needed a reference to the current piece, and had to constantly update this reference every time the current piece changed? One option would be to use the Proxy pattern.

A Proxy is not strictly necessary for this project, but we are giving you the option to implement the Proxy pattern for extra credit. Your Proxy must communicate with the current piece. See the Design Patterns lecture for more information on proxies.

Other Extra Credit:
Play with the Snazzy Demos for good ideas! Here are some additional suggestions for Bells & Whistles:

- Keep score and line count (remember: completing 4 lines at once is a Tetris!).
- Make the game progressively harder (e.g., make the pieces drop progressively faster) as the player completes more and more lines.
- Allow the player to restart the game at any time (i.e., aborting the current game and starting a new game immediately).
- Allow the user to see the next piece which will fall down the screen. (this is harder than it sounds)
- 2-Player Tetris
- 2-Player Tetris against AI
- Tetris music

Handin Info

Your design will be graded interactively in a Design Check. Design Checks are from Monday, November 9 to Thursday, November 12. Tetris is due at 11:59pm on Sunday, November 22. The early deadline for an extra 4% is November 20 at 10:00pm, and the late deadline is November 24 at 10:00pm. Type cs015_handin Tetris in Terminal to hand in Tetris.

Start Early... Start Now... Start Yesterday!!!
Good Luck! Joyous Coding!