Tetris

Help Slides Released: Monday, November 7th, 2pm

Design Discussion Mini-Assignment Due: Monday, November 7th, 2pm
Design Discussions: Monday, November 7th - Wednesday, November 9th

Early Handin: Friday, November 18th, 10:00pm
On-Time Handin: Sunday, November 20th, 11:59pm
Late Handin: Tuesday, November 22nd, 11:59pm

To run the demo: cs015_runDemo Tetris
To run the snazzy demos: cs015_runSnazzyDemo Tetris
   Demos may not work well over ssh! Try FastX or the Sunlab.

To install: cs015_install Tetris
To hand in: cs015_handin Tetris

Silly Premise

Congratulations!
Today is your day.
It’s down to the wire,
No time for delay!
With your AndyBot solved,
And your LitePegs all set,
The CeilingLight’s flashing,
‘Cause it’s not over yet!

So put down your Cartoon,
And let your Doodle fall through.
Now before final projects,
We’ve got one more task for you...

Boo hoo hoo! The poor humor TAs are having trouble coming up with a water skit. In the spirit of CS15, they decide to delegate their skit duties to Dr. Seuss himself. But in order for Dr. Seuss to come up with a great skit (and still be in the running towards becoming America’s Next Top Bottle), he must bootstrap from the greats and code up a storm to make everything fall into place. Join Dr. Seuss in completing the oldest project to date—a rite of passage that all CS15 alums have completed. Get ready, it’s time for Tetris!

New Concepts Covered

- Arrays
- The Factory Pattern
- Designing a large program

[Action Required] New Collaboration Policy

Important: In order to receive a grade for DoodleJump, you will have to complete the new collaboration policy quiz/contract.

From the collaboration policy addendum:

You may collaborate on the mini-assignment for the project and all material covered in the mini-assignment over the course of the entire project, to the extent of the mini-assignment. Thus, if the mini-assignment asks for a containment diagram and two algorithms in pseudocode, you may collaborate on what the containment diagram will look like, and pseudocode (though not actual Java code) for those algorithms. You can not discuss portions of the assignment not covered in the mini-assignment, nor can you go further than the scope of the mini-assignment questions—if the question requires pseudocode, then you may discuss pseudocode, but you may not discuss how to write it in Java code, i.e., how to translate the pseudocode to Java. You should never look at another student’s Java code, and all written work turned in must still be your own.
If you work with other students on your project, you must include the logins of anyone you worked with in the `logins.txt` file that is included in the project stencil. If you do not put down logins of other students, you are claiming that you did not discuss the project with anyone and all thought process behind your work is exclusively your own (with help from the course staff).

As always, you are encouraged to discuss any course material that is not assignment-specific.

**Mini Assignment**

Before starting to code your program, you should think through your design carefully by completing the design discussion mini assignment! This will help you think through your containment relationships as well as some of the larger algorithms that you will code in this project. A PDF with your mini assignment solutions is due by the deadline at the top of the handout via [cs015_handin TetrisDiscussion](#).

**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.

To reiterate, your program should have the following functionality (all of which can be seen in the demo):

- The seven pieces pictured above should be randomly generated
- At the start of the game, a random piece should appear and move down the screen, one square at a time
- Pieces should stop moving when they can fall no further; a new randomly generated piece should then appear at the top and fall down the screen
- When a row is full, it should be cleared and every row above it should move down
- User input summary:
  - Pressing the left and right arrow keys should move the current piece left and right if this would not move the piece into an occupied square or offscreen
  - Pressing the up arrow key should rotate the current piece 90 degrees counterclockwise
    - Pressing up should not affect the square piece visually
  - Pressing the down arrow should move the piece down one row
  - Pressing the spacebar should drop the piece as far as it can go, maintaining current rotation and left/right position
  - Pressing ‘p’ should pause the game if it is running, unpause it if it is paused - when paused, text should be displayed to alert the user, and the piece should not be able to be manipulated with the arrow keys
• When a piece is in the top row of the board or a new piece appears and has no room to fall, the game is over. Text should be displayed to alert the user, and the piece should no longer move either due to a timeline or due to key input.

• A quit button

**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, printlines 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...”

– Princess 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
• Make it easy to add new types of pieces (extensibility)
• 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
• Check if a desired move is legal - i.e., do 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

**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 discussion and in your header comments!

**The Pieces:**

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

You will be using random numbers to decide which one of the seven possible Tetris piece shapes 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**. **Another hint:** Check out the `Arrays.asList(...) method` for a quick way to make a list out of several objects!

**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, so here’s some of the math. To move a point 90 degrees counter-clockwise in a circle around another point, you can use these formulas:

\[
\text{newXLocation} = \text{centerOfRotationX} - \text{centerOfRotationY} + \text{oldYLocation} \\
\text{newYLocation} = \text{centerOfRotationY} + \text{centerOfRotationX} - \text{oldXLocation}
\]

where \text{newXLocation} and \text{newYLocation} are the new coordinates of the point being moved, \text{centerOfRotationX} and \text{centerOfRotationY} are the coordinates of the \textit{fixed} point around which this point is moving, and \text{oldXLocation} and \text{oldYLocation} are the original coordinates of the point being moved. Note that this assumes that the positive y-axis points down (as Java does).

**Animation and Timelines:**

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

**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 whole 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 \texttt{Node}, you need to add or remove that \texttt{Node} to/from the Scene Graph in order to see a visible change.

- 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 in the same way as checking occupied squares within the board.

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

**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 \texttt{Constants} class and
define public static final constants within it. Take into consideration both numbers and objects that never change. **Hint:** does the arrangement of each of the 7 pieces ever change? Can this be reflected in constants? A constant does not have to be an integer!

**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 Javadocs, [JavaFX Guide Keyboard Interaction section](#), or [DoodleJump Handout](#) for pointers.

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

**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 as far as it can go
- **‘P’ Key:** Pauses/unpauses the game

**Coding Incrementally**

We suggest coming up with an incremental plan to approach your project before you begin coding. Unlike DoodleJump, we will not be including this in the handout--it’s now your turn to come up with a plan!

**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.
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.
- Give the user the ability to ‘store’ a piece they don’t want to use right now for later use
- 2-Player Tetris
- 2-Player Tetris against AI

Anything you can think of! (except music, since department machines often don’t support sound)

---

**Minimum Functionality Requirements**

MF Policy Summary: *In order to pass CS15, you will have to meet minimum functionality requirements for all projects. If you don’t meet them the first time around, you may hand the project in again until you succeed, but you will keep your original grade. MF requirements are not the same as the requirements for full credit on the project. You should attempt the full requirements on every project to keep pace with the course material. An ‘A’ project would meet all of the requirements enumerated in the assignment specification section of the handout and have good design and code style.*

To meet minimum functionality for Tetris:

- There are at least 6 kinds of randomly generated pieces that fall down the screen using a Timeline.
- User can move pieces left and right with the arrow keys.
- Must be able to rotate pieces to 4 rotate positions in “open space”, preserving the layout/shape of the Piece.
- Pieces cannot move through/overlap other pieces when falling or moving.
- Lines clear when full, and board updates graphically and logically
  - When a line is full, all blocks in it should disappear and all pieces above should move down
  - Subsequent pieces should not “float on air” after a line has been cleared
- If a newly generated piece cannot move, Pieces should stop automatically moving and being generated (game over).
Start Early... Start Today... Start Yesterday!!!

Good Luck! Joyous Coding!