CSE 11 **START EARLY!**
Spring 2015
Programming/Homework Assignment #7

Due: 5 June 2015, 11:59pm (electronic turnin) 200 points

This is a programming assignment and will be turned in electronically. You may develop your code on your own machines, however, the code *must* run on lab machines and *must* be turned in from a lab machine.

The Program – Tetris
This is a double programming assignment. It is intended to NOT be doable in a single night. It's long with a non-optimized solution taking about 700-800 lines of code and white space (not including comments). It's much more complex than any of your other programs but at the time of the assignment we have already covered the critical material. This assignment synthesizes nearly everything covered so far in CSE11.

The goal is to create a Tetris game program that you play on your computer. To achieve this goal you will be using Arrays (or ArrayLists), rectangular arrays, Threads, graphical user interfaces with java Swing objects, for loops, while loops, boolean expressions, and good program design.

This is a much more substantial program than your previous assignments and you should expect it take you much more time. Begin early. Ask questions. Be Patient and develop in stages. You have two weeks. Use them!.

If you do nothing on this program the first week, you will find it very difficult and frustrating. Start working the first week. Read the suggested development process. The program is quite doable, but NOT if you try to solve the entire problem at once, or in a single week.

What the Final Game looks like when initialized and after being played for awhile with default the default blocksize (20 pixels)

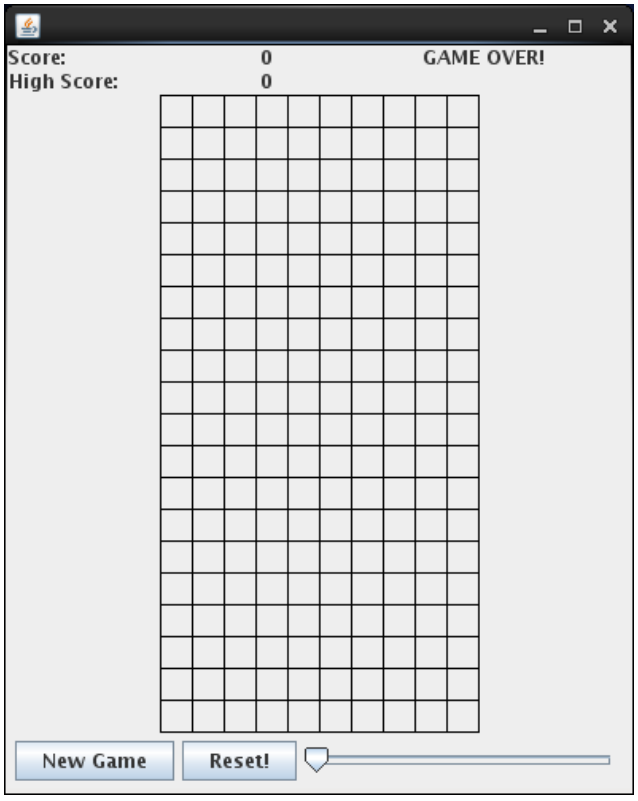


Illustration 1: When Game First Begins

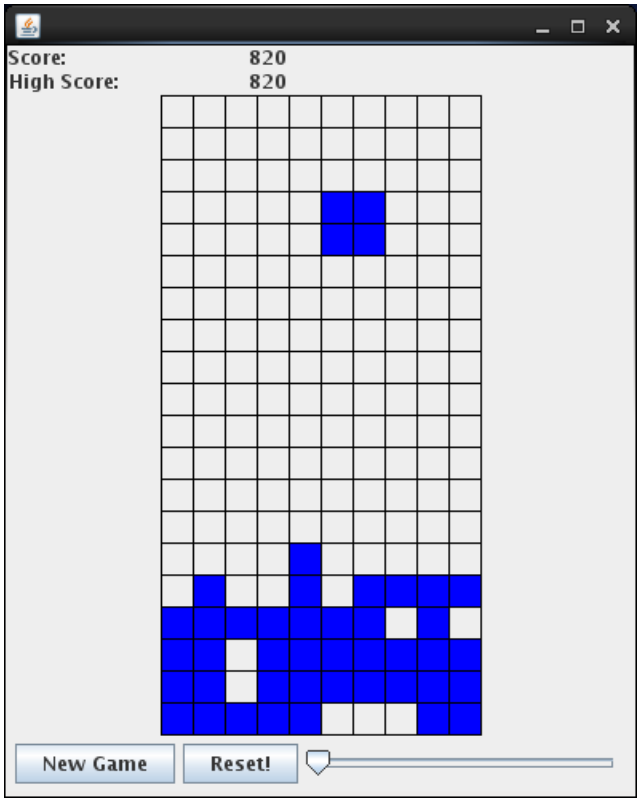
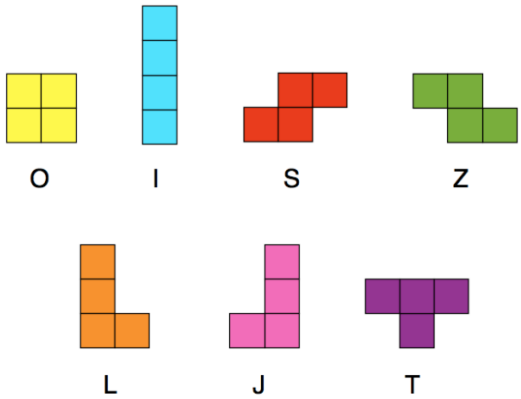


Illustration 2: After playing for awhile

Basic rules/operation of the game

There are seven Tetris shapes (each happens to occupy four grid squares). These shapes are, visually:



When a new game is begun and **empty field/board that is 10 blocks wide, 20 blocks high is created**. Then a Randomly-selected shape is created and begins moving downwards toward the bottom of the grid. The shape starts roughly at the center top row. A shape may be rotated clockwise or counter-clockwise by the user. A shape may be moved left or right. And the user may drop the shape into place.

A shape may not be moved out of bounds. It may not be rotated if the rotation would make it run into an existing block on the board. When shape can no longer move downwards (hits the bottom, or runs into existing blocks, a new shape is automatically created.

The game is over when a shape cannot fit onto the grid.

Movement:
The player has five keys at his/her disposal to move the current Tetris shape

1. h – moves the Tetris shape left
2. l – moves the Tetris shape right
3. j – rotates the Tetris shape counter clockwise
4. k – rotates the Tetris shape clockwise
5. spacebar – drops the shape downwards from its current position until it would stop

Scoring

1. Every time a shape moves downwards, score should be incremented by 10.
2. Rows completed when a single shape stops movement
    a. 1 row – 100
    b. 2 rows – 400
    c. 3 rows – 800
    d. 4 rows – 1600

Automated Speedup
blocks should begin movement at 1 row/second. maximum speed should be 20 rows/second. Block movement should speed up every 2000 points, until 40000 points are reached, and then maximum speed should be maintained

Screenshots of the Game in Progress

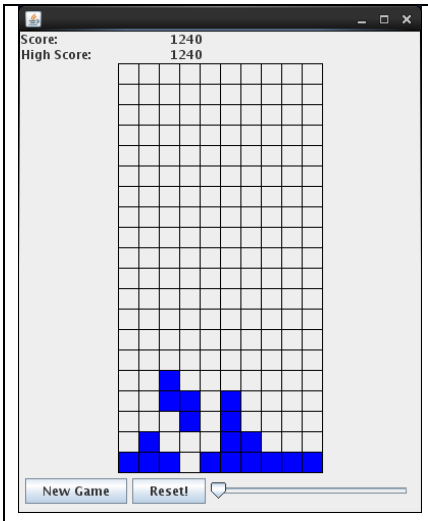


Figure 1 - Just prior to shape going to final position

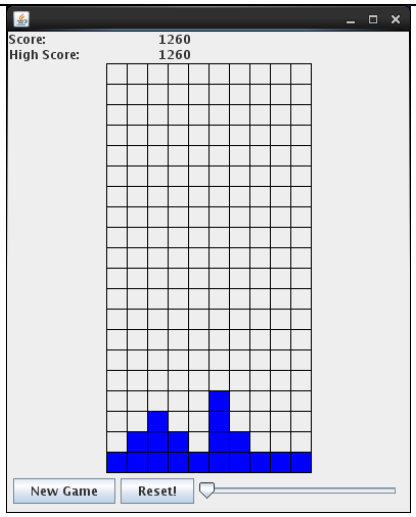


Figure 2 - Shape completing a row, before the row is cleared

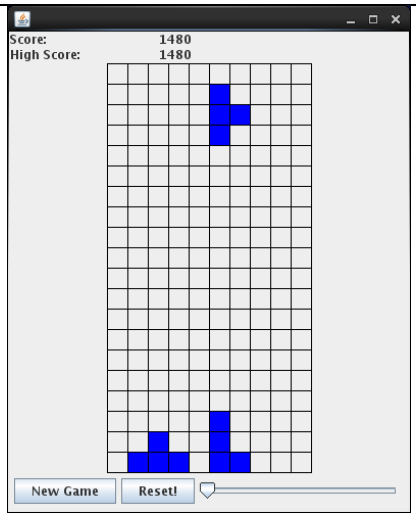


Figure 3 -single row cleared. Notice score change.

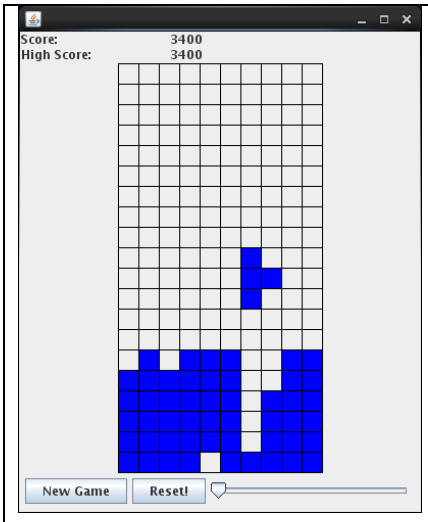


Figure 4 - shape dropping prior to completing 2 rows

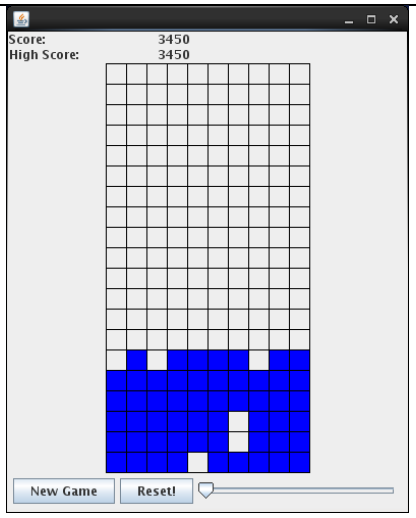


Figure 5 - shape in place, prior to two rows being cleared

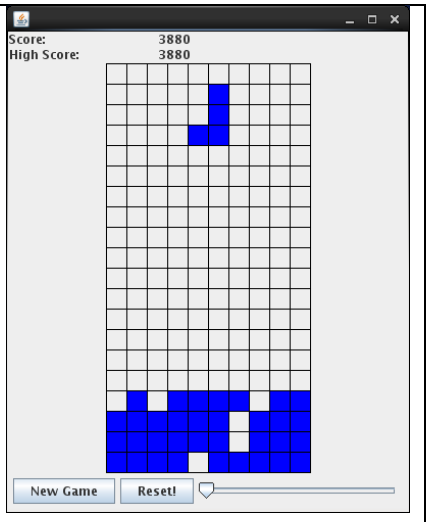


Figure 6- 2 rows cleared. Notice score change of 400 points for completing 2 rows at once

The Game is over when the next random piece will not fit. You may draw the final piece or not draw it (not drawn in the is diagram)

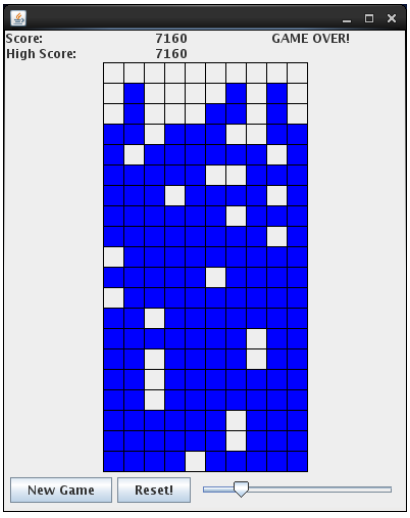


Figure 7 - Game Over

The Major Classes

When creating a larger program, one has to figure out how to break up the problem into smaller components. Since, this is likely the first big program you've ever created, this assignment guides you through some of this process. Let's begin by looking at the most likely classes, and then expanding

- Tetris Shapes
- the grid (or field) over in which shapes are being placed
- The graphical interface

There are two less clear classes at this stage.

- A ShapeMover that causes the current shape to move under program control

- a coordinate class (basically row,column coordinates)

The TetrisShape Class (`TetrisShape.java`)

There can be several approaches to designing a Tetris Shape.

- Must be able to create one of 7 specific shapes
- Must be able to create a random shape
- must be able to rotate clockwise
- must be able to rotate counterclockwise
- should be able to move it left/right (we'll describe why this class is NOT the best place for this functionality.

What about displaying the Shape on the canvas? That's a good design question. This assignment recommends that you do NOT make the TetrisShape class a graphics object, but instead represent shape as an array of four coordinates (See the Coord class below). Does a shape need to know the boundaries of the field? Perhaps, but is a simpler design if the Tetris does not understand where it is on the field.

The Shape as an array (row,column) coordinates.

A good approach is to think about a shape as a set of coordinates, with one of the blocks being at coordinate location (0,0). This make rotation, much simpler. All shapes can be rotated, except for the 2x2 square shape.

The T-shape could be represented as coordinates [(-1,0),(0,0),(0,1),(1,0)]
One of the Ell shapes could be represented as coordinates [(-2,0),(-1,0),(0,0),(0,1)]
and the other Ell shape as coordinates [(-2,0),(-1,0),(0,0),(0,-1)]

How to do rotation: You can look up Rotation matrices for graphics on the web, if you want the full background. But our rotation is special (90 degrees).
If a coordinate is (r,c)

- rotation in one direction makes the rotated coordinate (-c,r),
- rotation in the other direction makes the rotated coordinate (c,-r)

If you want to rotate a shape, you apply the formula for all coordinates of the shape. For example, rotating one of Ells [(-2,0),(-1,0),(0,0),(0,1)] using the first rotation above yields new coordinates of [(0,-2),(0,-1),(0,0),(-1,0)] (draw dots to see that this should be counter clockwise rotation)

One of the reasons we don't put where the shape is located on the board, is that rotation is MUCH easier.

I suggest that you create two "getters" for this class. One getter, returns the array of coordinates as they are stored. The other returns an array of coordinates with a given offset. For example for the first Ell shape, a getter with no arguments would return

[(-2,0),(-1,0),(0,0),(0,1)]

but a getter with an offset of (3,4), would add three to the first dimension and four to the second and return

[(1,4),(2,4),(3,4),(3,5)]

You should find the second form very convenient for placing a shape at a particular location on the board. External to this class, you keep track of where on the board to place the shape. The shape itself keeps track of its orientation (how it has been rotated).

Notice that Tetris shape is NOT a graphics object, it is simply a logical object.

Before going on, let's talk about the Coord class

The Coord Class (`Coord.java`)

It is highly recommended that you create a Coord class where an instance of the class is an integer pair `(r,c)`. You probably want to write an equals() method and a toString() method, but this is not essential. You can then represent a shape as an array of Coordinates. (r,c) are (row, column)

Note you could use the java.awt.Point. However, I find the Point.x, Point.y to be confusing in this case. A shape occupies a set of (row,column) logical coordinates. A TetrisGrid (below) is a 2D array of (rows high x columns wide).

You probably also find it easier if you define two constructors for Coord, namely

```
Coord (int r, int c)
Coord (Coord initial)
```

The first constructs a coordinate pair from `r,c,` the second one constructs a copy from an existing Coordinate instance.

Normally,one would have getters and setters. But just like point, allow direct access to the fields

***Back to the** Game*

The TetrisGrid Class (`TetrisGrid.java`)

The TetrisGrid is the bounded X,Y coordinate plane on which the shapes are placed. Think about it as "slots" that can be (a) empty, (b) occupied with blocks from previous pieces. (c) occupied by the current piece in play.

A TetrisShape doesn't know about other shapes, It is on the TetrisGrid where Shapes interact with one another.

The Grid is also where rules of movement are enforced.

1. The piece in play can't be moved outside of the boundaries (you might want to ignore the top boundary for simplicity)
2. a Shape can't be rotated where one of its blocks would be outside the boundary
3. a Shape can't be rotated if the result of the rotation would move it into another occupied block
4. a Shape cannot be moved (left, right, down) if that movement would put any part into an occupied block

Functionality of the TetrisGrid that will be needed

- Constructor – builds a 10 wide x 20 high grid
- Get an array (or list) of blocks that are occupied
- Determine if a shape intersects any occupied block in the grid
- Determine if a shape (at a particular offset) is completely in bounds
- Add a shape to the grid
- Remove a shape from the grid
- Determine if after placing a shape, rows have been completed
- Delete completed rows
- Override toString() for Ascii Printing/debug

The GraphicsGrid (`GraphicsGrid.java`)

The TetrisGrid must be drawn . You should be able to reuse a good fraction of the code you developed for PR6.

Something will need to eventually tell the GUI when a move has been performed and when rows have been completed (for scoring).

Hint: When telling the GraphicsGrid that a piece has moved, I would remove all the blocks that should be drawn from GraphicsGrid, retrieve an array (or ArrayList) of occupied blocks from TetrisGrid, and then re-add all the occupied blocks to GraphicsGrid. This takes care of all cases of rotating pieces, moving pieces, collapsed rows, etc. Sometimes the simple solution is best.

The Game (`Tetris.java`)

This is the GUI interface for the game. It needs to display scores (high score) present buttons and a speed slider, and indicate when a game is over.

Hint: Each time a New Game is requested, create a new TetrisGrid

The ShapeMover (`ShapeMover.java`)

The ShapeMover must move the Current Piece downwards on a timed interval. It must also respond to keystroke commands from the user. In other words, it needs to respond to keyboard events and run on a clock (think Thread). Some more Detailed requirements of the game are given below

Some hints on ShapeMover

When trying to move/rotate the current shape

1. Remove it from the TetrisGrid
2. Logically move it, or rotate it to define its new position on the Grid
3. Test if the new position is valid
    a. Test if the shape is inbounds
    b. Test if the shape intersects any of existing block
4. If the new position is valid, then place it on the GameGrid
5. If the new position is not valid
    a. Place the shape back at its original position/orientation
    b. If it was a downward move, then the shape can't move anymore

        i. In this case, you then need to test if rows are now completed
    6. At the end of a valid move, the GraphicsGrid will need to be updated.
    7. When a shape stops moving, the game needs to be informed that a shape can't be moved any more and that a new shape should be created.

Summary of Class Interactions

1. A Shape is an array of logical coordinates. It knows nothing about graphics. It can be told to rotated and report information about itself.
2. The TetrisGrid is the bounded X-Y playing field. It is the place where shapes interact with existing (placed) blocks. Hence it can enforce the rules of the game.
3. The ShapeMover is the automated mechanism of forcing the current shape to move downwards. The ShapeMover will attempt to move/rotate the shape.
4. The Tetris is the Graphical interface that sets up the All components. It responds to user input.
5. Coord is a "helper" class.
6. You will need to decide exactly how information is communicated among the various classes to achieve your goals. Ask questions and think about this for a while. Understanding how your classes should interact will make your debugging much faster

Rules/Requirements of the Game

Command-line invocation

```
$ java Tetris [blocksize]
      blocksize – Integer size of each block segement in pixels,
    default is 20.
```

This is a long section, because there a many details to get right.

1. Game Setup
    1. The Grid is initialized to
    2. Print out a usage statement if the parameters are non-sensible (e.g. negative blocksize, or size so small that game is not really playable) Empty parts of the field are colored White/background color of your choice
    3. The playing field has each grid square outlined by a black line (See Pictures above). If you choose your own background color, the lines should be apparent.
    4. You only required to use a single color for all blocks. Feel free to make blocks of any color and/or have shapes have specific colors.
    5. The Grid should centered both vertically and horizontally.
    6. You must provide a button for "New Game" and "Reset"
    7. You must provide a speed slider to change the speed of the game
    8. Above the play field are labels for Score, High Score, and an indicator for when a game is over. See Layout for the pictures above, your game should have the same layout
2. Game Initialization
    1. A random Tetris piece is created in roughly the center of the top row. Exact center is unimportant. At the beginning, part of the piece may extend above the playing grid
    2. The piece moves downwards under timed control
    3. If the speed slider is more than the minimum, that should be the speed of shape movement
    4. You do not have to react to changes in window size.

3. Shape Movement
    1. See above for the key movement (you will need to implement the KeyListener interface. it makes sense for the ShapeMover to be the listener)
    2. If the user does nothing, the current shape moves downward on its own.
4. Scoring and obstacles
    1. Each time a shape moves downwards or under user control, 10 points should be added to the current score
    2. The high score should keep track of scores between new games, if the current score would be higher than the high score, the high score should update immediately.
    3. The game is over if no more pieces can be put on the grid.
5. Timed movement
    1. The slow speed moves the Shape ~once/second
    2. The fast speed moves the Shape ~20 times per second
    3. The speed of the Shape should increase every 2000 points until the maximum is reached.
    4. The ShapeMover should not ever sleep for longer than 50ms. This is to keep the interactivity of the game. It's easiest to think of moving once/second as counting down twenty, 50ms sleeps and then moving the shape.
    5. If a user moves/rotates a shape, the countdown should start again
6. Reset Button
    1. The speed should be reset to the slow speed
    2. The high-score should be reset
    3. The current game should stop immediately
7. Game Over
    1. When the snake violates a rule of the game, all movement should stop and the GAME OVER! Indicator should be shown.
8. These requirements may be amended/clarified during the assignment.

Development Strategy

This is a big program and you must develop in stages, the final program is roughly 700 -800 lines of code (some of the code you have already written, so it's less). Take a deep breath. And plan to work on this assignment in small chunks. Celebrate the progress as you make it. What follows is a suggested plan of attack You should be looking to do 2-4 steps at a time. Estimate of the whole project is somewhere between 10 and 20 hours. (You have TWO weeks and potentially a partner). If you find yourself getting stuck, ask for help. **We really want you to succeed at this program**. The strategy is pretty detailed so that you can make good, measured progress. This is a very challenging assignment.

When you get things working at various steps, make copies of your working code and store in a safe place.

1. Forget about graphics as you start out. You need to get the TetrisShape and Coord Classes coded correctly. Together they total 150 – 175 lines of code.
2. Write the Coord class and test it. This shouldn't take too long, It's handy for your Coord class to think about (rows,columns) instead of (x,y). We specify graphics width x height, but arrays are indexed (row, column) (The transpose of graphics).
3. To test your TetrisShape class, I suggest creating a "throw away" test class called ShapeTest.java that allows you create either a specific shape or a Random shape and the print it "graphically" (using an array of chars). Make sure that the getter with offset works. Make sure

rotation works. Your command-line codes from previous assignments (like ArrayPlay) can be used as a starter "interpreter". Create shape, print on a small grid, rotate it. print. etc. Do this with all 7 seven shapes.

4. Write the first edition of the TetrisGrid that allows you to put shapes on the grid, test for interaction with other shapes. You can test functionality in stages.
    1. Place a shape specifically at a location.
    2. Test bounds
    3. Test intersection with other shapes. etc.
    4. Printing the grid
5. To assist in your testing of TetrisGrid, create second testing program (perhaps, ShapeTest2.java)

6. Modify ShapeTest2 to support the hitting keys for movement. Don't attempt this with KeyListener, Use Scanner – 'h' followed by return moves the current shape left, 'l' followed by return moves the shape right, 'd' followed by return moves the shape down, etc. This allows you test all the logical functionality of the game in a step by step manner, without threads. Don't move on until this right. Essentially, you are creating an Ascii version of Tetris with no Threads
7. For printing TetrisGrid, each element of the grid can be a character. Use a '.' for an empty space, a "#" for a an occupied block,
    1. You need to initialize an empty grid, create a new Snake and place the snake at the top center of the grid.
    2. you need to implement move and grow methods in GameGrid that will move/grow a snake AND check that the move/grow was valid (Stayed within the GameGrid)
8. A complete TetrisGrid for the Professor is about 150 lines of code
9. Now it's probably time to make the first graphical version. You can (and probably should) reuse much of PR6. Still use your keyboard (+ return) to move blocks You are simply moving from ascii graphics to Swing graphics. Getting the graphics right means the each logical square on the GameGrid is KxK pixels.
10. When you have the keyboard-based (+ return working well), it's time to build a ShapeMover class. Do this in two steps. First, have the ShapeMover implement the KeyListener interface. If you have things working correctly, the keyboard controls will work interactively (without return) At this point you have basic movement in a graphics environment working (We haven't even started on the gui, yet). Note, you have to add the KeyListener to the JPanel (GraphicsGrid) and you have to click on the canvas to make the listener active.
11. The next step for ShapeMover is take away the manual "d" movement and make ShapeMover a Thread. Fix the timestep so that it automatically moves the Shape downwards every second. Have the run loop pause for 50ms, if the user doesn't hit a valid for twenty consecutive pauses, then move the Shape downwards one row. Reset that counter each a valid command key is
12. At this point, you have basic, automated Shape movement with keyboard input working. Play with your program for while. Check that updates are correct, drawing is correct and no exceptions occur during testing.
13. Time to start on the GUI. Lay out the components so that the scores, buttons, and sliders show up where you want them. At this stage I wouldn't add any actionListeners. Just get the layout to look right.
14. Next, make sure that when you hit the "New Game" button, things work as you expect (GAME OVER! Goes away, score is zero, TetrisGrid is created.) You now need Tetris to be an ActionListener.
15. If you've gotten to 15, you've have made excellent progress. The rest of the assignment makes the game more fun. Honestly, you are on the downhill slope now. If you only get this far and

have no errors, your score for functionality is already in the 75 percent range.

16. Work on getting communication working between the ShapeMover and the GUI. The *mover needs to report to the GUI* every time a valid move is made or when shape can't move any longer.
17. Add speed adjustment. Every 2000 points, the speed should increase. One way to think about this is there are 20 speed steps. Increase the speed by one speed step every 2000 points. If you want, if the game starts at speed level 10, it doesn't need to increase the speed to 11 until the score is 20000
18. Play your game. Enjoy the fruits of a long programming assignment.

Grading (total of 200 points)

160 points – Does your program compile and run properly. If all aspects are met, then you will receive full credit. Various misfeatures or bugs will lose appropriate points. We will run your program. Since the assignment does not specify particular signatures of methods, we can only test your final product.

40 points – Commenting/indentation/style. Please see the Style guide. You Must put your NAME:, LOGIN:, ID: in every file you turn in. Please use ALL CAPS for the NAME, LOGIN, ID labels. Use Javadoc-style comments on all public methods that you define.

Are you using private methods and variables?
Do you use accessor methods?
Are your methods overly long?
Are you repetitive in code?
Are the comments informative?
Are Variable names appropriate?
Are the classes defined used in a reasonable way?

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR7" program that will turn in the following files

```
Coord.java
TetrisShape.java
TetrisGrid.java
GraphicsGrid.java
ShapeMover.java
Tetris.java
```

No other files will be turned in and and they **must be named exactly as above.** BundlePR7 uses the department's standard turnin program underneath. If you want to define other classes, they should be helper or inner classes defined in one or more

the above files.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

```
$ /home/linux/ieng6/cs11s/public/bin/bundlePR7
```

Some of you have had trouble turning in programs and sometimes turn in the wrong version. To address this, a A HIGHLY Recommended sequence of commands is the following: (You are not ready to turn in files if you get any errors)

```
$ rm -i *.class
$ javac Coord.java TetrisShape.java TetrisGrid.java
GraphicsGrid.java Tetris.java
$ java Tetris
$ /home/linux/ieng6/cs11w/public/bin/bundlePR7
```

The above will remove your compiled java classes (asking you first if you want to remove them, say yes to *.class files don't make a mistake and delete .java files!!!), then will compile the files you are about to turn in.

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Suggestion: if you have classes that compile and do some or most of what is specified, turn them in early. When you complete all the other aspects of the assignment, you can turn in newer (better) versions.

Don't forget to turn in your best version of the assignment.

START EARLY! ASK QUESTIONS!
HAVE FUN PLAYING YOUR SNAKE GAME!

Some Frequently Asked Questions

Can I change the colors of the grid,head,snake,obstacle? Yes. Feel free to.

Can I add other elements to the game interface? Yes. As long as the required functionality is achieved, you may add extra information.

I see the word "callback" in the instructions, what is that? It's a well-defined method on an object. Suppose Object A has callback named callme(), If Object B has a reference to A (called a), then when something occurs that should be of interest to an A, B invokes a.callme(). In this code, the GameGrid is calling back to the GUI to indicate that the snake has moved. In turn, the GUI calls back to the GameGrid when a new obstacle should be placed on the Grid.

This program is enormous? Does anybody finish it? It is big. You can have a partner. Many people finish it completely. If you get through this program, you've learned quite a bit. You must develop in stages or you will not realistically complete the assignment.

Arrays or ArrayList, which should I use? I suggest both. The GameGrid itself is most easily implemented as two-dimensional array. Each cell in the GameGrid is one of four things: EMPTY, SNAKE HEAD, SNAKE BODY, or OBSTACLE. You could use anything to represent that state of a grid cell (integer constants, Colors, your own Object type). The coordinates of the Snake itself could be stored in an Array or an ArrayList.

I have KeyListener in my GraphicsGrid (JPanel), I don't see the key presses. **Why?** This may be a focus issue. Clicking on the grid, should set the keyboard focus correctly.

How come your rules of snake are different than my phone's? That's just the way it is.

Can I add extra classes and/or not use as many classes as specified? You may do both, but you cannot add/takeaway any of the defined files. If you want an "Empty class", create the file, and put in comments as to why you aren't using that class. If you want to add a class, make it a helper class or inner class defined in one of the files above.