# CmpE 160 Project 2
# Smart Tetris

Can Tunca, Cagatay Yildiz, Haluk Bingol

Spring 2015

**Due:** April 29th, 2015; 09.00 a.m.
**How to submit:** Will be announced later.

## 1 Overview

In this project, you will implement the classic game of Tetris. To make things interesting, in addition to the regular Tetris where a person plays the game, you will also implement a simple AI, so that the computer can play the game by itself.

If you have never played Tetris before, you can see what it is like here: http://goo.gl/LHKkzZ

For this project, you do not have any obligations regarding the usage of classes and such. We provide you with some classes to help you out with the visual aspects of the game, but you may modify them or not use them at all if you like. In the next section, we briefly explain these classes, if you choose to use them.

## 2 Details of the Given Classes

We give you a drawing framework very similar to the one we gave you for the first project. The framework consists of the same three classes and one interface (`GUI`, `Canvas`, `AnimationCanvas` and `Animatable`).

A new method `fillGridSquare` is added to `Canvas`. This method fills the grid square at the specified coordinates with the given color. The coordinates are of type `int`, and the coordinate (0,0) corresponds to the lowest leftmost square. This functionality is basically all that you need to implement the visuals of Tetris, since all Tetris pieces are basically combinations of little squares.

We also added a `removeObject(Animatable obj)` method to `AnimationCanvas`, so now you can remove objects from the canvas when you are done with them.

As mentioned before, you do not have to use these classes. It is totally up to you. If you choose to use them you may change them as you wish. Alternatively you may write you own visualization code from scratch, and not use these classes at all.

An important thing to note: You still need to implement keyboard interaction functionality on top of the given code. You may find some information about this issue in the Tips and Important Remarks section at the end of this document.

## 3 Rules of Tetris

In this section, we provide you with the basic rules of Tetris. Since Tetris is a well known game, we won't be giving you a complete list of rules. We only clarify the points that may be ambiguous. The little details that would change the game only slightly are not all that important, and we will not be checking your implementation rigorously for every rule. Your implementation would be perfectly OK if the core rules are satisfied, and the game more or less feels like Tetris.

### 3.1 General Rules

The Tetris game board is composed of a grid of 20 rows and 10 columns. Pieces of different shapes, that are combinations of little square blocks, fall from above, and the goal is not to let the pile of blocks to reach the top edge of the board. A row of the pile is deleted only when all of its columns are occupied by a block. Anyway, you already know this stuff :) If you don't, just play the Tetris example given in the Overview section, and you will have an idea soon enough.

The game ends when a part of the pile touches or exceeds the top edge of the board. You should inform the user (via a Game Over message) and pause the animation when the game ends.

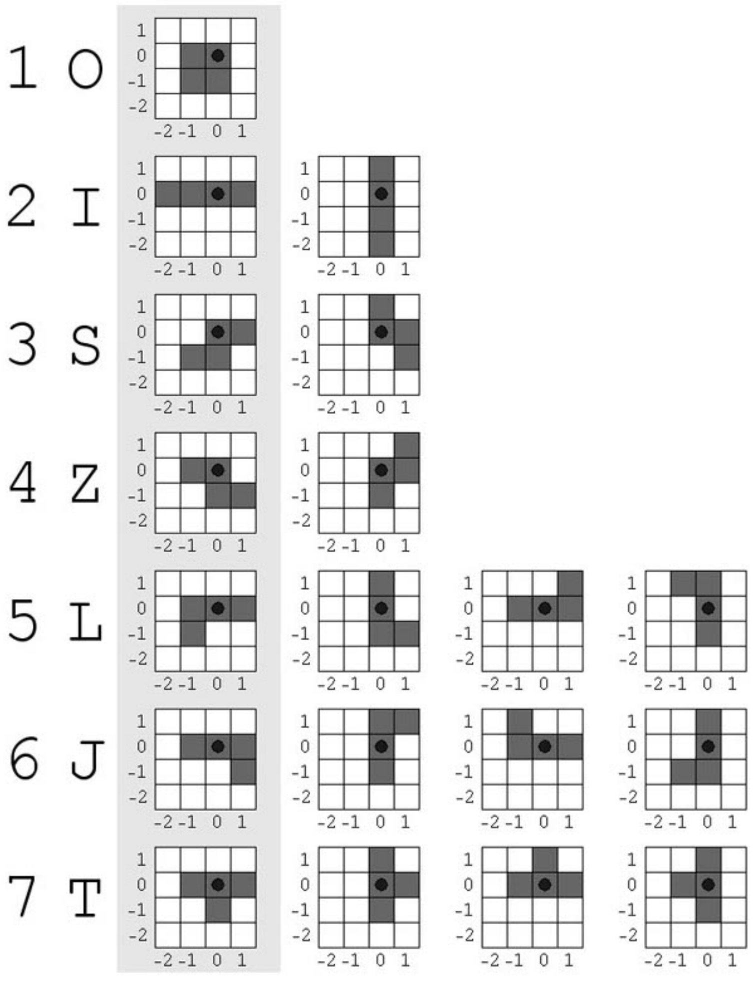

Figure 1: Tetris pieces and their rotations

Some versions of Tetris also keep a player score, that is increased depending on the number of rows eliminated at once. For this project, you are not obliged to implement such a feature. It is up to you. If you decide to implement this feature, please make sure that it is visible on the user interface, so that we may see and appreciate it.

## 3.2 Pieces and Rotations

There are a total of 7 different pieces in Tetris, which can be named after the letter they visually resemble: O, I, S, Z, L, J, T. Note that some of these pieces are mirror images of the others. The pieces are presented in Fig. 1. The default orientations of the pieces are presented in the first column, and these are the initial orientations when the pieces enter the board from above. The dots on the pieces should be aligned to the grid square with the coordinates (5,19) when the pieces first appear on the board (note that the bottom leftmost grid square is assumed to have the coordinates (0,0)). The other columns in the figure show the possible rotations. Pieces should be rotated 90 degrees counter-clockwise, in the way shown in the figure, each time the *Rotate* button is pressed. Some pieces have less than four orientations since the other orientations would be copies of the ones shown.

Each new piece will fall at a constant speed (It is important that this speed is easily changeable through the code or the GUI, in case we'd like to speed up/slow down the game). The player should be able to move a piece sideways (left or right) however he/she wishes. The player also should be able to rotate the piece. It is also desirable to have a fast-fall functionality, to be used when the player believes he/she correctly aligned the piece. Some versions immediately sends (teleports) the piece to its place when the fast-fall button is pressed, while other versions merely speed up the fall. It is totally up to you; you may choose whichever way you prefer.

A suggestion for the key scheme is to use the arrow keys. Left/right arrows move the piece sideways. Up arrow rotates the piece and down arrow executes fast-fall.

If you have any other questions about the rules, or are curious about how Tetris was invented, you may visit the site: http://colinfahey.com/tetris/tetris.html

Please do not ask us questions about the nuances and specifics of all the little details of the game. As we have mentioned above, the small details are not important as long as they do not significantly change the game itself. So, you may assume anything you like for such details.

# 4 What You Need to Do

**Implement regular Tetris.** You are required to implement the regular Tetris game as described above, that can be played by a person via the keyboard. This is the most important task in this project and it will constitute the majority of your grade. So do not attempt the other tasks before finishing this task.

**Implement simple AI.** You are required to implement a very primitive AI (Artificial Intelligence) that can play the game by itself. AI is a limitless subject with lots of different methods and approaches, and we definitely don't want you to get lost trying to find the best solution. We want an AI that shows only a little sign of intelligence and is somewhat better than randomly placing the pieces. This task is to encourage you to do some thinking on how the dynamics of the game can be understood through a computer's perspective. So, please do not ask us questions about which approach to use, since it would defeat this purpose.

If you manage to code a good AI, it would be a good idea to place two canvases side by side and allow a person to play against the AI you designed (The new `addCanvas(...)` method of `GUI` allows the addition of multiple canvases side by side). Of course, to be fair, the exact same sequence of pieces should be presented to both the player and the AI. It would also be nice if you could synchronize the two canvases. For instance the AI may make its moves only when the player makes his/her moves. The idea of two canvases is totally optional.

**(Optional) Implement evil mode.** This task is totally optional and solely for your own fun. The experience you get while coding the AI may actually be used for evil purposes :) A common complaint by many Tetris players is the never-coming long piece (I). You may actually cause this! You may implement an insanely difficult evil mode where, depending on the situation of the board, bad pieces come. Imagine this: you ask a friend of yours if he/she could test the Tetris game you have coded for a course project, and secretly give the evil version and watch him/her go crazy :) Of course you would not want your friend to notice something weird going on, so you must not consistently present the same pieces over and over again.

**Javadoc and Project Report.** As usual, we require you to provide javadoc comments for your methods and classes. Additionally, we also request a short project report. This report should contain the instructions on how to play your game (e.g., the keys to interact with the game). You must also give brief information about the classes you implemented. Moreover, you must briefly explain the approach you used to implement your AI, so that we may appreciate it! You must also tell us exactly how to switch to the AI mode or the evil mode, if you have implemented them, so we can test them.

## 5 Tips and Important Remarks

- This is a difficult project. It is strongly advised that you start the project as soon as possible. Completing it in the last couple of days may not be possible. Also, there will be no deadline extensions.

- About capturing key presses in `Java`: The following resources may help you out. Also remember, Google is your friend!
  http://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html
  http://www.edu4java.com/en/game/game4.html

- Remember, object oriented programming is there to ease our efforts! The problem presented in this project is quite big, trying to implement everything in a single class may make it much more difficult. So, divide and conquer! Implement one thing at a time. A suggestion regarding the class hierarchy of the project: Define separate classes for each of the 7 different pieces that extend a general `Piece` class, and override the rotation functions. Moreover, creating a separate class (for instance, `TetrisGame`) that implements the game logic and serves as a middle-ground between the drawing framework and the different Tetris pieces is advisable. You may also use this class to capture the key presses (by implementing the `KeyListener` interface).

- Representing the pieces and the pile of blocks present on the board is an important part of the project. You are advised to use the different data structures you have learned in the class (ArrayList, Stack, Queue etc.) to your advantage. Especially for the AI, the representation of the state of the existing block pile is very important for the AI to be able to make intelligent decisions. Also note that, for the AI, the most important thing is the top surface of the pile, so you may want to think of a clever way to represent this surface, in addition to the remainder of the pile.

- Your program must be responsive. When the player presses the sideway movement buttons, he/she would like to see the results immediately. This means that the frames per second (fps) value of the `AnimationCanvas` should not directly dictate the speed of the falling pieces, since it merely defines the refresh rate of the animation, which may be much higher than the movement rate of the objects. Setting a low fps value to slow things down would also reduce the responsiveness of your game. Here is an example on the correct way to do it: you may fix the fps to 30 and define a separate variable that determines the falling speed with respect to this value. For instance, if you want a falling speed of one row per second, you would then perform the falling movement only once in 30 calls to the `move(...)` function. If you want a falling speed of two rows per second, then you would perform movement once in 15 calls, and so on...

Good luck and have fun!