**Final** **Project** **–** **Tetris!**

The format for the final project will be as follows­

* You will pick a partner and work together. Make sure that you email files to each other so you each have the code you work on!
* You will have time to work on this project in class. There will be a schedule setup so that you may ask for my help on certain days for a set amount of time during classes. Otherwise, you should attend tutorials.
* There will be two checkpoint dates before the end of the semester. You and your partner will need to have these checkpoints completed to earn full credit for the project.

# Game Play

The goal of the project is to implement the basic game play described below.

The game starts with an empty board drawn. The board is typically 10x20 squares. The top left corner square of the board has coordinates (0, 0) and the bottom right corner square has coordinates (9, 19) (the y-axis is still flipped). A randomly chosen Tetris piece from the seven possible shapes is drawn at the top of the board. The piece starts falling at regular intervals - one square at a time.

# Basic Rules

1. The piece cannot fall into a square occupied by another piece or beyond the edge of the board.
2. When a piece hits another piece or the bottom of the board, it stops moving and a new piece appears at the top of the board.
3. As the pieces fill up the board lines form. If a complete line forms, it disappears and all the blocks above it fall down one line.
4. If a new piece can no longer be placed at the top of the board, the game ends and a “Game Over!” message is displayed.

# User Interaction

The user can use the arrow keys to move and rotate the pieces - ‘Left’, ‘Right’, ‘Down’ arrow keys move the piece left, right and down by 1 square respectively. The ‘Up’ arrow key will rotate the piece. The user can also drop a piece by pressing the spacebar. Dropping a piece means that the piece will fall down until it can no longer move and the user can longer rotate or move it in any other direction. When the piece is moved or rotated, it cannot move into another piece or over the edge of the board.

# Project Design

As discussed in class, we have prepared a starter file that has all the class and method definitions, but you will have to implement the methods to make your game work. We will do this step by step - starting small and extending the game features as we go along. You’ll work with your partner to try your best to implement the methods; I will periodically email out code to the whole class to help you along with the trickier bits.

Most of the methods that you need to implement have just one statement, **pass**, that tells Python that the method doesn’t do anything currently. All the places where you will need to add code have a comment ‘**YOUR** **CODE** **HERE**’. At the end of the project, you should have code in all the places where you find this comment.

**READ** **ALL** **THE** **INSTRUCTIONS** **IN** **A** **GIVEN** **SECTION** **BEFORE** **YOU** **START** **WRITING** **ANY** **CODE.** **MAKE** **SURE** **THAT** **YOUR** **CODE** **WORKS** **BEFORE** **MOVING** **ON** **TO** **THE** **NEXT** **SECTION.**

**1.** **Tetris** **Classes** **Overview**

Get a copy of the file tetris template.py from the course webpage.

Take a look at the file. There are several classes included in this file. Read the documentation at the top to get an idea of how everything works together. The Board class implements the functionality of the Tetris board.

Read through the file and familiarize yourself with the different classes and their attributes and methods. Take a look at the Block and Piece classes as well since they also have some additional attributes and methods. Feel free to change the color of the shapes!

In previous classwork you used **pygame.display** to create a window where you can draw objects. For our Tetris project, we will create a window surface where the game will take place.

 Run your code and make sure the window appears with half of it black and half of it gray. The gray side will contain the moving blocks. The black contains the game information.

# 2. Creating a random shape

Let’s make things a bit more interesting. We want to display random pieces on the board. Before we can do that we need to be able to draw blocks and then entire pieces onto the board.

First we need to look under the Block class and finish the draw method. This should just draw the block at the blocks x, y location. You have a couple of different ways to draw a rectangle in pygame. You can use:

pygame.draw.rect(surface, color, (x, y, length, width))

or if you have a tuple that represents the rectangle object: (x, y, length, width) – take a look at the rect property right underneath draw – then you can just write:

pygame.draw.rect(surface, color, rect)

Think of a way that you can test if a Block object will draw.

Next move onto the Piece class. Finish implementing the draw method in this class (remember that the piece is made up of several blocks; we need to draw all of them). Once you have completed this method, run the program. You should see two pieces appear in the window. If you run this again, you’ll notice that it just displays the same two pieces. Time to change that.

Look under the Board class and find the random\_piece method. This method should return a random piece from the dictionary of pieces that we have defined. However, you will notice that it only returns the first piece from the dictionary right now. Come up with a way that you can return a random piece and then replace the current return statement with your own. If you have forgotten about random numbers, look back at our previous homeworks or just google ‘python random’

 Once you think you have implemented random\_piece correctly, run the program multiple times and see if the pieces are different each time.

# 3. Keyboard Events

We have seen how we can move objects around in pygame in class already. Before we can make anything move in this game we need to map the keys to their events in pygame. You can do this in two ways.

1. Set up your own pygame program and print out each event to the screen.
2. Use the ever useful google and look up the different key representations in pygame.

You will need to figure out how to tell if a key has been pressed. Don’t add anything to your game yet!

# 4. Moving Shapes

Modify the move method under the Piece class so that it will move an amount dx or dy. This part should be a simple solution.

Now look to the Board class. Find the move\_left and move\_right methods. Implement these methods so that they move the piece one block either left or right. If you looked up the information you needed from part 3, then you can head down to the game loop and add in the code to check for the left or right key being pressed. You should add statements under those checks so that if you hit the left key, the piece will move one block left; and if you hit the right key, it will move one block right. [Hint: board.move\_direction()]

*Don’t* *try* *to* *implement* *everything* *in* *the* *Board.move\_left() or Board.move\_right() yet!* - we’ll keep adding to this function in later sections. For now, just add code to move the shape in the appropriate direction as specified by the parameter.

 Run your code. Do you have a moving shape?

**5. Attention!** **Piece** **overboard!**

* What happens if you move your piece left 10 times? What about right?
* How would you ensure that the piece does not move beyond the edge boundaries? Modify your code so that a piece moves only if it can, i.e. if one of its blocks is about to fall off the edge, the entire piece won’t move.

1. Modify the Board.has\_overlap method and implement part 1 described in the template file. Check if the position of each block in a piece is outside the boundaries of the board. Return True if it is outside and False otherwise.
2. Modify the Board.move\_left and Board.move\_right methods – you need to check two things. One, check if the piece is mobile (self.current.mobile). Two, check if the blocks within the piece overlap the boundary. If it does then you need to move your piece back one. Think a little bit here… if you move the piece to the right you increase its position by one OFFSET, if it needs to move back then you need to move it by how much? Test your code and see if your pieces stay within the board when you move them left or right.
3. Now modify the Board.move\_down method. This should move you down the board in the y direction (REMINDER: the y axis is flipped) by an OFFSET every time. If the piece overlaps with the bottom boundary (i.e. located at the BOARDHEIGHT) then the piece should stop moving. Once the piece overlaps, just like before, move it back one spot and then we will set the attribute self.current.mobile = False. [Hint: first check if the y position of your current piece is less than the BOARDHEIGHT. If it isn’t then move it. If it is then set its mobile to False. After it has moved once, check if it is overlapping. If it is then move it back one spot and set mobile to False.]
4. In the game loop add the code needed to move the piece downward using the down key.

 Run your code and make sure your pieces don’t fall off the board when hit all three of the edges - left, right and bottom.

 **CHECKOFF** **1** **(Due** **FRIDAY):** **Show me sometime during the day that your code works up to this point.**

# 6. Adding a piece to the board

Now that the pieces are no longer falling off the board, let’s continue with the game. Once a piece touches the bottom edge of the board, it should be added to the board permanently and a new piece should appear at the top. How would you know that the piece touched the bottom edge? How would you add the piece to the board? What would be a useful data structure?

We are going to keep track of the state of the board using the blocks attribute of the Board object. The blocks attribute is a list of Block objects (blocks) that are currently on the board. Each block has all of its normal attributes from the Block class, such as x, y position and color.

Look at the remember\_block\_positions method under the Board class. This effectively stores the location of each block that has been placed on the board. Notice it is not keeping the information of the piece that was dropped, just the blocks! (Look up anything you don’t understand in this method) At this point, however, once a block reaches the bottom of the screen it just seems to disappear. The block is being saved into the list of blocks by our above method, but we just can’t see it. Thus, we need to draw all the blocks that get stored within the board!

1. Implement the draw\_blocks method in the Board class. It should draw all the blocks that are contained within Board.blocks. Try not to overthink this one – it should take two lines of code maximum!
2. Now you may notice how pesky it is to keep having to hit the down key to speed up the falling pieces. It would be nice if we could just hit one key and have the piece fall the rest of the way without further input. Look at the drop\_down method in the Board class. Implement this method so that the piece will move down until it can fall no further. Is there a method that you have already written that may be useful here? Do you need to put in boundary checking or is it already built into another method? Also, how will you keep the object moving down – what is useful for making repetitive actions in programming? Now go into the game loop and add code so that the drop\_down method will be used when a player hits either Enter (Return) or Space. (HINT: The key for Enter is K\_RETURN)

 Run your code and make sure that when you can drop a piece and when it reaches the bottom, it will be added to the board and a new random piece will appear at the top.

**7.** **Attention!** **Intruders!** **A.K.A. has\_overlap() Part2**

What happens if a shape tries to move to a square that is already occupied? How would you change your code to make sure that a shape doesn’t move to a square that is already taken?

Modify the Board.has\_overlap method to check if there is already a piece at the current position and return True if our piece overlaps another. It should return False otherwise. **Hint:** Use two for loops – one nested inside the other – to loop through each block in the current piece and each block on the board. Compare the x and y position for both sets of blocks. If both x and y are equal for two or more blocks, then there is an overlap!

 Run your code and make sure the pieces don’t trample each other.

# 8. Rotating a piece

Now moving a shape is easy, but how do we rotate one? Well luckily for you, I chose the easiest method possible. We defined all the different rotations that a piece could have in their respective classes. Since the pieces positions are just saved as lists, we can change the index of the list and get its new rotated shape. For more complicated objects this wouldn’t necessarily work and you may want to research how you can rotate different shapes in pygame anway. However, that won’t be necessary for the completion of this project. Let’s implement the Board.rotate() method. The piece should not be allowed to rotate off the board or into another piece. If it will rotate into another piece then it should either rotate back to its original state or keep rotating until it is in a valid position. EXTRA CREDIT: If the piece is against the wall it may not be possible for it to rotate. Code in a ‘wall kick’ to push the piece away from the wall so that it can rotate.

1. Write the code for the rotate() method.
2. Finally, modify the game loop to rotate a piece when the ‘Up’ arrow key is pressed.

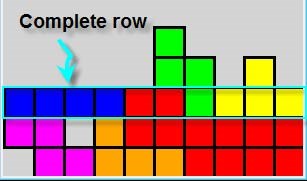
 Run your code and make sure the pieces rotate when you use the ‘Up’ arrow key. Notice what happens when you try to rotate pieces next to walls.

 **CHECKOFF** **2** **(Due** **WEDNESDAY):** **Show me sometime during the day that your code works up to this point.**

Now your game should be almost fully functional... Almost!

# 9. Checking and Removing completed lines

When a new shape is added to the board, we need to check if there are any new lines that were completed and need to be removed. If a row is complete, i.e. all the squares are occupied by blocks, it is deleted. Then all the blocks above are moved 1 square down.



The Board class has several methods to help with implementing this feature – destroy\_line (deletes a row), check\_for\_lines (creates a list of blocks in a completed row to be removed), and collapse\_hovering\_blocks (moves all rows above the given row inclusive down one square).

1. The check\_for\_lines method and collapse\_hovering\_blocks method have already been implemented for you. You need to implement the destroy\_line method. The line sent to the method is a y position value. So if the line is to be removed you should remove every block in self.blocks that happens to have the same y position value.

 Run your code and make sure that your game removes completed rows correctly!

# 11. Game Over

Final touches...

Before placing the new piece on the board, you must check if the piece can be placed into that position. If it can’t the game is over, and you should display a “Game Over” text message on the board and stop placing new pieces on the board. (Hint: look at the Board.move\_down() method. If you try to move the piece down from y = 0 and you can’t (i.e. self.has\_overlap() returns True), then can we keep playing?)

1. Inside the Board.move\_down() method put a check that will determine if the game is over.
2. Modify the Board.draw\_game\_over() method to display the game over message, if the new shape could not be drawn on the board. Notice that this method is basically implemented for you, but the make\_text method doesn’t seem to do anything. Fix this so that it will actually display something. (The benefit of this is the draw\_text method directly above will now work as well)

 Run your code and make sure that your game over message appears on the screen when you cannot add any more pieces to the board. You should now also see the rest of the text for the screen including the Next Piece, Score, Lines, Level, and Controls.

CONGRATULATIONS! You now have your very own Tetris game.

# Bells and Whistles (Optional)

If you would like to make your game a bit fancier, here are a few possible extensions you may want to try at home. We have already added a couple of features, but you can take it much further. There is some help in the starter file for some of these ideas. You may have to write new classes or methods to implement these.

## 1. Pause Game

Modify your game so that when the user presses ‘p’ or ‘P’, the game will pause until the user presses

‘p’ or ‘P’ again. By pause, we mean

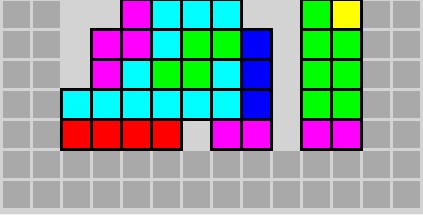
* The piece will stop falling automatically
* The user will not be able to move or rotate the piece by pressing the arrow keys while the game is paused
* There will be a message displayed on the board that says the game was paused and how to resume play.
* The board game board should not be viewable. Don’t want people cheating!

## 2. Display Gridlines

Modify your game so that when the user presses ‘g’ or ‘G’, the game board will display gridlines until the user presses ‘g’ or ‘G’ again. This has already partially been done for you – look under the draw\_ui method in the Board class.

## 3. Border around the board

Add a border of one or two squares around the board as in the snapshot below. This might be trickier than it sounds.



## 4. Set the caption and icon for the display window

You can use Google to find icon images.



## 5. Create blocks with shading. This should be implemented in the Block class. You can make it look shaded by creating one slightly larger dark rectangle and one smaller light rectangle overlaid on each other. The colors are already set up for you (normal color, light color).

## 

## 6. Add in some music!

Project Idea Taken and Modified From:

MIT OpenCourseWare

http://ocw.mit.edu

6.189 A Gentle Introduction to Programming January IAP 2011

License:

<http://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>