6.189 Final Project – Tetris!

The format for the final project will be as follows-

- You will pick a partner and work on one computer together. Make sure that you email files to each other so you each have the code you work on!
- Sit next to each other in lab. There will be an LA assigned to your area who will keep track of your progress and walk you through more difficult sections of the code.
- Both you and your partner must attend 2 of the 4 available recitation sections on Thursday and Friday. There are two **Checkoffs** that you and your partner must complete with an LA for you to get credit for doing this project. Checkoff 1 must be done in Thursday's lab section. Checkoff 2 must be done by Friday. If you just do the the checkoffs you won't complete the game, so it's worth it to work ahead on your code!
- Wednesday & Thursday we'll meet in the lecture hall for a brief reviews; on Thursday, we'll also review the answers to the 'exam' passed out during Tuesday's lecture, if you choose to do it.

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

We already had a bit of a head start. Last week, we created objects for the all the tetrominoes that had the functionality to be drawn on the screen and Project 2 used the same game board framework as Tetris. As discussed in lecture, 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 and your LA to try your best to implement the methods; we'll 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. Besides the `Block` and `Shape` classes you implemented yesterday, there are two additional classes - **Board** and **Tetris**. The `Board` class implements the functionality of the Tetris board. The `Tetris` class implements the game play, i.e. it serves as a game controller.

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

In Homework 4 you used the **GraphWin** object to create a window where you can draw objects. The `GraphWin` is a `Window` object with a `CanvasFrame` object in it. For our Tetris project, we will create a `Window` object and we will place a `CanvasFrame` in it explicitly. The `Board` class has an attribute `canvas` that is a `CanvasFrame` object. This is where the shapes will be drawn.

Run your code and make sure the empty board appears on the screen.

2. Creating a random shape

Let's make things a bit more interesting. Implement the `Tetris.create_new_shape` method, i.e. the `create_new_shape` method in the `Tetris` class. It should create a new randomly chosen shape object and return it. If you have forgotten about random numbers, go back to Homework 2 (exercise 2.4.1). If you are having trouble using the `SHAPES` attribute that contains a list of the Shape classes, look at Homework 4, problem 3 for help.

You will need a reference to the shape later to be able to move/rotate it. The `Tetris` class has an attribute `current_shape` that will hold the currently active shape. Update the `Tetris.__init__`method to display the current shape on the board (hint: take a look at the methods of the `Board` class for help).

Run your code and make sure you see the shape on the screen before you continue.

3. Keyboard Events

Before we can move the shapes around, we need to learn how to get keyboard events, e.g. when a key is pressed.

If you look at the `Tetris.__init__` method. It calls the `bind_all` method on the `Window` object to create a **key binding** that tells the `Window` object to automatically call the `Tetris.key_pressed` method when the user presses a key.

Run the code. Click on the Tetris window to make sure that the window is in focus and then press the arrow keys and the space bar. Notice the output in IDLE. The variable `key` in the `Tetris.key_pressed` method has a type string and it contains the value of the key pressed. If you press the letter `a`, `key` will have value 'a'. But, since the arrow keys and the space bar are special keys, they have the following values:

'Up', 'Down', 'Right', 'Left', 'space'

4. Moving Shapes

Modify the `Tetris.key_pressed` and `Tetris.do_move` methods to make the shapes move when the 'Left', 'Right' and 'Down' arrow keys are pressed. Take a look at the `DIRECTION` attribute of the `Tetris` class. It is a dictionary with a key that has type string and specifies the direction to move the shape, and a value (dx, dy) corresponding to how many units to move along the x and y axis respectively. Look also at the `Shape.move` method.

Don't try to implement all the functionality in the `Tetris.do_move` method 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?
- 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.can_move` method and implement part 1 described in the template file. Check if the position is within the boundaries of the board. Return True if it is and False otherwise.
2. Modify the `Block.can_move` method - fill in the code as described in the comments.
3. Now modify the `Shape.can_move` method (hint: this should utilize the `Block.can_move` method you just wrote!). Use the `Board.can_move` method for help. Note that these `can_move` methods take an additional parameter, which is a board object.
4. Finally, update the `Tetris.do_move` method, so that it first checks if the shape can move *before* it moves. The method should return True if the move was performed and False otherwise.

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 THURSDAY): Find an LA - tell him/her who your partner is, and demonstrate that your code works up to here.

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 `grid` attribute of the `Board` object. The `grid` is a *dictionary* where the key is a tuple (x,y) corresponding to the square at position (x,y) on the board. The value of this key will be a `Block` object (why not a `Shape` object?) occupying the square.

Modify your code so that it adds the shape to the board, and then creates a new shape and places it at the top of the board.

1. Modify the `Board.add_shape` method so that it adds each block to the `grid` dictionary. Implement the `Shape.get_blocks` method to get the list of blocks from the `Shape` object.
2. Update the `Tetris.do_move` method such that if last move that failed was 'Down', the method will:
   - add the current shape to the board,
   - update the `Tetris.current_shape` attribute with a new random shape, and
   - draw the new shape on the board.
3. Update the `Tetris.key_pressed` method to make the piece drop to the bottom of the board if the user presses the spacebar. Remember the value of the variable `key` in this case will be 'space'.

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!

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.can_move` method to check if there is already a piece at the current position and return True only if there isn't and False otherwise. **Hint:** Use the `in` operator on the `grid` dictionary to check if there is a value (eg, a block) at the key `(x, y)`.

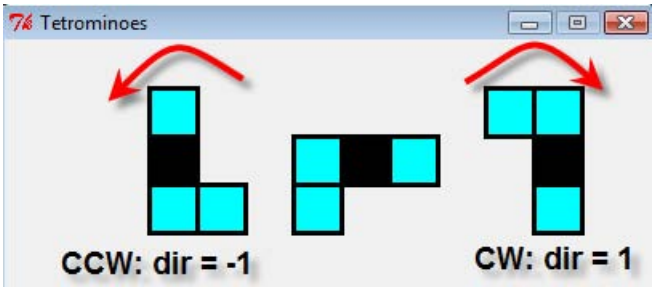
Run your code and make sure the pieces don't trample each other.

8. Rotating a piece

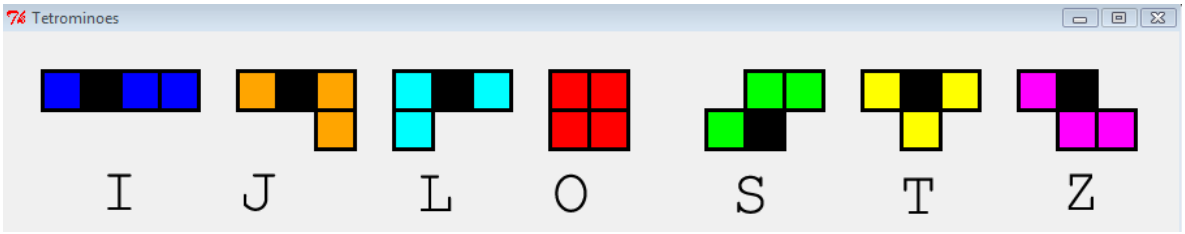
Now moving a shape is easy, but how do we rotate one? What we need to know is how to rotate a square around another square 90 degrees. Here is a formula that can help:

```
x = center.x - dir*center.y + dir*block.y
y = center.y + dir*center.x - dir*block.x
```

This formula gives the new coordinates of the `Block` object, `block`, if it is rotated around the `Block` object, `center`. The variable `dir` specifies the direction of rotation (you can find the current rotation direction using the `Shape.get_rotation_dir` method). If `dir` = 1, the block is rotated clockwise (cw), and if `dir = -1`, the block is rotated counterclockwise (ccw). In the figure below, the black square is the `center` block and the other blocks rotate around it, i.e. the black square is the center of rotation.



The different pieces, however, behave differently. J, L, and T always rotate clockwise. I, S, and Z rotate back and forth. Z and S rotate clockwise, then counterclockwise, while I goes the other way. O does not rotate. To implement rotation for each of the pieces you need to know what is the center of rotation. The black square in the figure below shows the center of rotation for each of the pieces and this is the block at index 1 in the `blocks` attribute of the `Shape` object.



1. Implement the `Shape.can_rotate` and `Shape.rotate` methods. The shapes are not allowed to rotate off the board or into another piece. The `Shape.can_rotate` should return False, if any of the blocks in the shape cannot move to its new position on the board (either because the position is beyond the boundaries or because the square is already occupied).
2. Now implement the `Tetris.do_rotate` method to rotate the current shape, if possible.
3. Finally, modify your `Tetris.key_pressed` method 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.

9. Automatically moving pieces

We learned how to use the `after` method on the GraphWin object to animate graphics objects. We can use the same method on the `Window` object as well (take a look at the `Tetris.animate_shape` method). The method will move the shape down one square once every second.

Modify the `Tetris.__init__` method to start animating the shape once it is drawn.

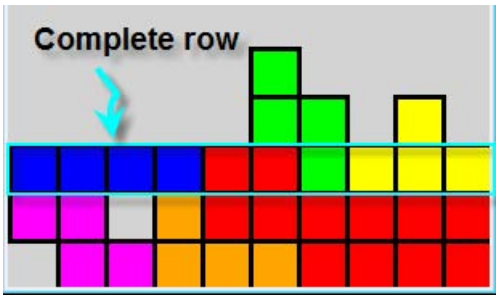
Run your code and make sure the piece falls down on its own.


CHECKOFF 2 (Due FRIDAY): Find an LA and demonstrate that your code works up to here.

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

10. Removing completed lines

When a new shape is added to the board, we need to check if there are any new rows 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 - `delete_row` (deletes a row), `is_row_complete` (returns True if all squares in the given row are occupied), `move_down_rows` (moves all rows above the given row inclusive down one square), and `remove_complete_rows` (checks if there are any complete rows and removes them, and then moves all rows above down one).

1. Implement all the four methods described above. Recall the `del` command for dictionaries, listed in the dictionary "cheat sheet" of Exercise 3.3.
2. Then, modify the `Tetris.do_move` method so that every time a shape can no longer move and is added to the board, it checks if any rows have been completed and removes them.

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 implementation of `Board.draw_shape` - can you use this to help you figure out when the game is over?)

1. Implement the `Board.game_over` method to display the "Game Over!!!" message. If you didn't do the digital clock problem from Homework 4, or you forgot how to draw text, look at the optional exercise on that assignment.
2. Modify the `Tetris.do_move` to display the game over message, if the new shape could not be drawn on the board.

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.

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. There is no help in the starter file for these. You will have to write any classes or methods necessary yourself. You might also need to modify some of the existing methods.

1. Scores and Levels

Modify your game so it keeps track of the score as the user is playing. You can pick your own strategy for scoring the game. In general, you might want to give extra points if the user removes multiple rows at once and perhaps a hefty bonus if they remove 4 rows at once (that's called a Tetris). Create a `ScoreBoard` class that will create a new `CanvasFrame` object where the score will be displayed. Take a look at the `Board` class for example of how to do that. The `CanvasFrame` objects are displayed below one another in the window. So, depending on whether you want to the scores to be displayed above or below the tetris board, you will have to select the order in which the `Board` and `ScoreBoard` objects are created.

Think about what methods you would need to add to the `ScoreBoard` class and what objects it will have to interact with to be able to keep track of the score.

You can also add different levels. For example, if the user's score passes a certain threshold, they will move up a level. Moving up a level means that the pieces start falling down faster. Look at the `Tetris` object to figure out what attributes you would need to modify to make the pieces fall faster. Make your `ScoreBoard` class keep track and display the current game level as well.

2. Piece Preview

To aid your game play, create a preview for the upcoming piece. Create a `PiecePreview` class that will also create a `CanvasFrame` where the next piece will be drawn. You need to figure out what methods will be necessary to communicate between the `Tetris` object and the `PiecePreview` object in order to both display the next piece and update the current piece when it is necessary.

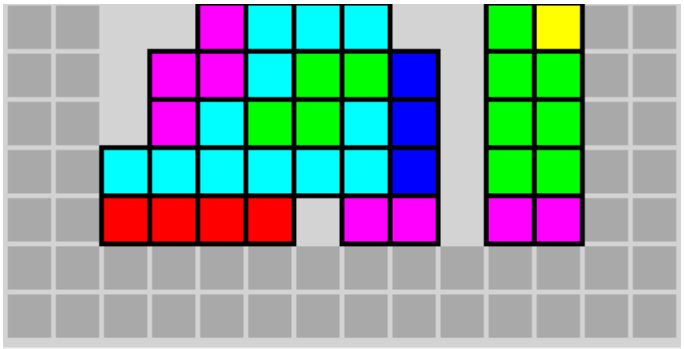
3. Pause Game

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.

4. 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.



6.189 A Gentle Introduction to Programming

January IAP 2011