C# Tetris

5/2/2019

Project:	Recruitment test for .NET developers
Owner:	JED
Checked by:	…
Approved by:	…
Version:	1.3

CONTENTS

VERSION HISTORY

Version	Date	Author	Changes
1.0	2010-05-17	JED	Document Created
1.1	2017-03-22	OWS	Minor updates/cleanup/clarifications
1.2	2019-02-27	JAH	“Goals” and “Dev env” sections updated.
1.3	2019-05-02	JLU	Updated doc to reflect the test.

1 GOALS OF THE PROJECT

In this assignment, you will implement a simple version of the classic game of Tetris (see e.g. https://www.youtube.com/watch?v=d7fajYz0r68 to get a feeling for what the game is about). The purpose of the project is not to take the actual game of Tetris to the next level but rather to test your problem solving skills. The idea of a game is just there to define the boundaries and scope of the project.

When we measure project success the emphasis will be on

- Problem-solving techniques and methodologies
- Software design

Since we don’t want you to get stuck on technicalities such as WPF and XNA Game Studio we have provided you with a simple Windows Forms application written in C# which includes some of the tools you need to complete the task. Your mission will be to connect the dots and write the business logic for the entire game. You are allowed to search the Internet for information about C#, OOP, etc. but you are obviously not allowed to copy existing code for this particular problem. After you have handed in the assignment, you will be expected to explain your solution and talk about any decisions that you have made.

You will start from a skeleton of the application. Open "Tetris.sln" in Visual Studio, then compile and run the application. As you can see, the code for displaying the information on screen is already there, and you don't have to modify or extend it. Open the GameLogic class. This is most likely where you will make changes, among other things it contains key down/up handlers. More specifically, you should not have to modify any code in the UI directory. You will most likely add new classes to the project.

We suggest that you read through the rest of the document to get a feeling for the task ahead. When starting to code, try to solve the problems in the order they appear in the document, working in small steps to make sure that what you have implemented works well before proceeding to the next task.

Clearly, time is one of the constraints of this test but please remember that even if you don’t finish in time it doesn’t necessarily mean that you have failed.

On behalf of the Combination development team, we want to wish you good luck!

2 CONSTRAINTS

2.1 IMPLEMENTATION

- The project must be completed within 4 hours.
- The project must be implemented using the provided Tetris Forms Application.
- The provided classes can be extended, improved and changed as you see fit.

2.1.1 TETRIS FORMS APPLICATION

To get you started we have provided you with a simple windows forms application which sets up the basic components in the Tetris game environment. In the Tetris solution you will find the following classes:

- Game
- GameView
- GameViewConfig
- GameLogicBase
- GameLogic

2.1.1.1 GAME CLASS

Contains the Main method where program execution begins and ends. Creates and connects the different entities in the system.

2.1.1.2 GAMEVIEW CLASS

The GameView is a representation of the window displayed in your application. It is responsible for drawing the game elements. It is also responsible for registering timer and keyboard event handlers.

Example interaction can be seen in the constructor of GameLogic. It demonstrates how to change the color of individual cells in the GameView main area and help area.
Note that cell information is managed internally in GameView. The OnPaint method is activated on an interval and draws the current state of the game.

2.1.1.3 GAMEVIEWCONFIG CLASS

This class defines a few basic properties of the game. You will probably not need to change any of the code contained within.

2.1.1.4 GAMELOGICBASE CLASS

This base-class contains basic functionality that connects the logic to the UI.

2.1.1.5 GAMELOGIC CLASS

The GameLogic class is quite lean as you open it, but will eventually need to contain or access logic that drives the game. Besides creating new files and classes, this is the only file you are expected to change. Start your implementation here.
To mention a couple of things, it will need to provide Tetris piece management, collision detection and handling, game scoring, etcetera.

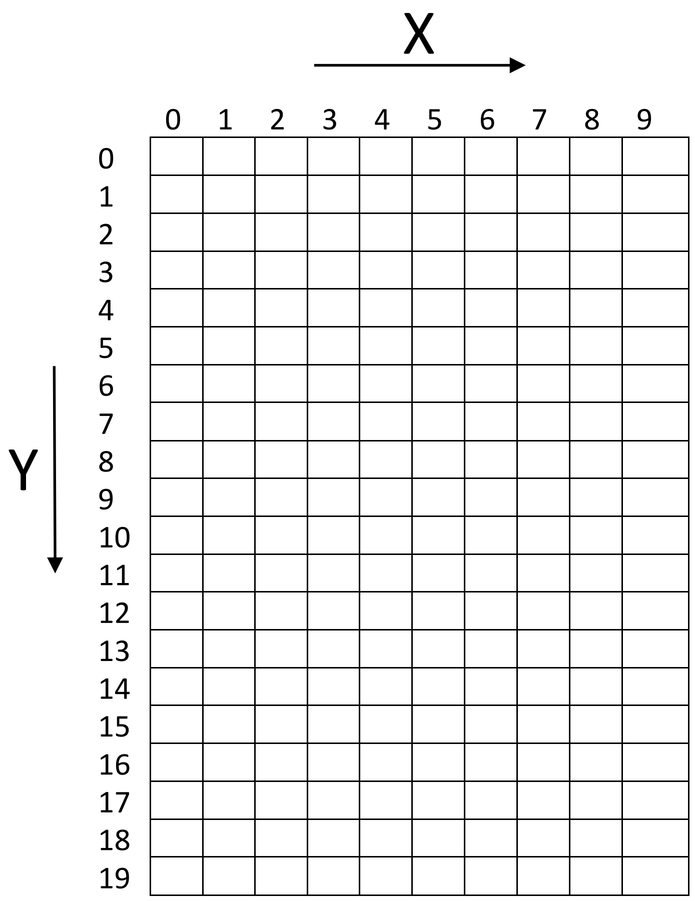
Development environment
The tools will be restricted to

- Microsoft Visual Studio
- C# programming language
- .NET Framework

3 FUNCTIONAL REQUIREMENTS

The definition of Tetris specifies that the shape of each new falling piece is to be selected randomly.

The board is a grid of cells, having 10 columns, and 20 rows, for a total of **10 * 20 = 200** cells.



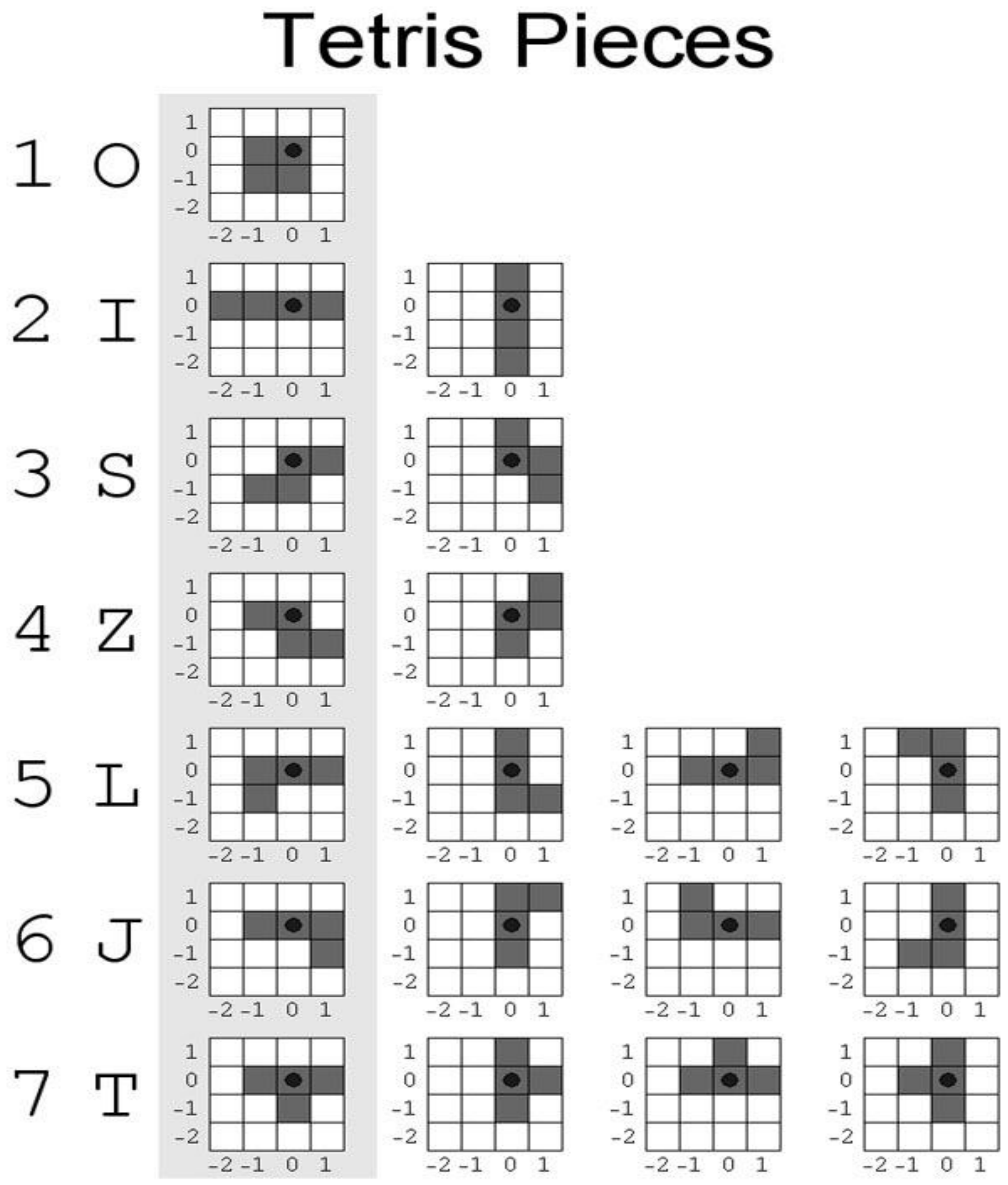
Each cell can either be unoccupied (empty) or occupied (full).

3.1 TETRIS PIECES

There are seven (7) standard Tetris pieces, with the following letter names:

{ O, I, S, Z, L, J, T }

The letter names are inspired by the shapes of the pieces.



The dot at **(0,0)** coincides with board position **(5,0)** when the piece first appears.
The first column shows the initial "orientations".

In the following, the word "orientation" is used to describe any state of a piece, within a set of allowed states, that can result from a counterclockwise rotation event.

Changing "orientation" from a specified "orientation" of an "**I**", "**S**", or "**Z**" piece, requires the combination of a rotation and a translation. Therefore, the word "orientation" is used here to mean something more than rotation alone. However, "orientation" can change only in response to a counterclockwise rotation event, and the cycle of distinct "orientations" for each piece approximates, or matches, the cycle resulting from pure rotations.

The special use of the word "orientation" in this context is nearly equivalent to the meaning of the word "rotation" or "angle", but the word "orientation" is used instead of "rotation" to attempt to bring attention to the fact that some pieces require more than rotation to produce the set of allowed states resulting from counterclockwise "rotation" events.

Pieces can only switch orientations (or undergo a specific horizontal or vertical translation) if the resulting state of the piece would not have any occupied (full) cells beyond the area of the board and would not have any occupied cells which overlap any currently occupied cells of the board.
(In this rule, the occupied (full) cells of the piece are not considered as part of the "currently occupied cells of the board").

In the following comments, any reference to a result of a counterclockwise rotation event is made with the assumption that such a rotation can actually be performed, given the existing conditions of the piece and the board.

The "**O**" (box) piece only has a single orientation, and does not change the locations of any of its occupied (full) cells in response to any counterclockwise rotation event.

The "**I**" (line) piece has two possible orientations, initially appearing in a horizontal orientation. The "**I**" piece alternates between the two orientations in response to successive counterclockwise rotation events.

The "**S**" and "**Z**" pieces each have two possible orientations. These pieces each alternate between two orientations in response to successive counterclockwise rotation events.

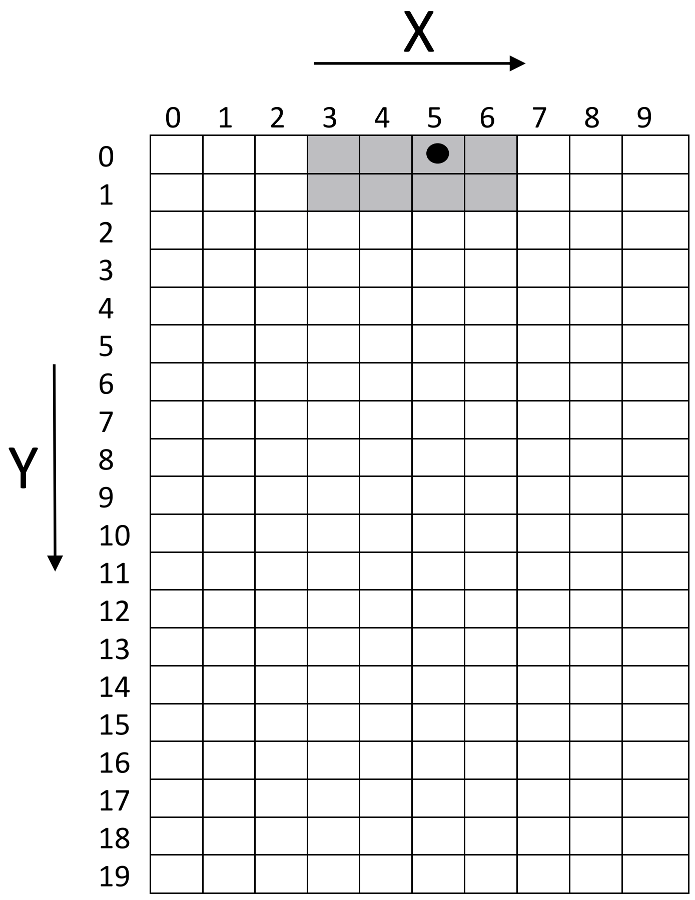
The "**L**", "**J**", and "**T**" pieces each have four possible orientations, and these orientations are the results of simple rotations about center points on the shapes.

When a piece first appears on the board, the piece has its "major axis" in a horizontal orientation, and the piece is at the top of the board. Therefore, no pieces are initially capable of having their orientations changed. The piece must descend by one row to have the possibility of having its orientation changed. Once a piece has fallen by one row on the board, all piece orientations can be attained (assuming the piece is not too close to the side walls or to the current pile of pieces).

Additionally, each Tetris piece has a unique color.

3.2 PIECE CREATION

The following diagram shows the (4 cell * 2 cell) region on the board where all pieces appear when created.



When a new piece first appears on the board, its origin coincides with the dot on this diagram, and the piece will be completely contained by the shaded area on this diagram.

When a new game is started, a full free-fall delay elapses, and on the next game iteration a piece is spawned at the top of the board. During normal game play, when a specific game iteration "lands" a piece, a full free-fall delay elapses and on the next game iteration a piece is spawned at the top of the board.

3.3 CONTROLS

During game play, the following inputs are available:

Input	Description
Left	Request to translate left by one column
Right	Request to translate right by one column
Rotate	Request to do a counterclockwise rotation
Drop	Request to instantly drop the piece

All inputs take effect on the rising-edge of the positive input (on button press, as opposed to button release). When a button press occurs, this counts as a request.

Requests to rotate can be executed if there is no overlap between the desired orientation and set cells on the current board (excluding the falling piece), and if the desired orientation has no set cells outside the board area.

Requests to translate can be executed if there is no overlap between the desired translated configuration and set cells on the current board (excluding the falling piece), and if the desired translated configuration has no set cells outside the board area.

A piece can be dropped immediately without having to wait for the free-fall delay.

A piece can be translated several times to the left or right, and subsequently dropped, all without having to wait for the free-fall delay.

Because a newly spawned piece cannot possibly be rotated (because it is stuck against the top edge of the board), the player must wait for one free-fall delay to elapse if rotations are desired or required.

3.4 PIECE LANDING

If a piece is simply falling, it falls by a single row during each game iteration. As this is allowed to continue, the piece moves from a place without contact to horizontal surfaces to a place that has contact with horizontal surfaces. Once that last iteration occurs, the pieces are in resting contact.

When a piece lands it becomes part of the static pile.

A completed row is a row of the pile in which all cells are occupied. When a completed row is eliminated from the pile, and the rows above the eliminated row are shifted down by one row to eliminate the gap, this counts as a completed "line".

Immediately after the piece lands, the pile is checked for completed rows, and all completed rows are eliminated.

Up to four rows can be completed simultaneously. The following table gives the upper limit on lines completed simultaneously by a single piece:

Piece	**Max simultaneously completed rows**
“O”	2
“I”	4
“S”	2
“Z”	2
“L”	3
“J”	3
“T”	2

3.5 LEVELS

Tetris has 10 difficulty levels, numbered 1 (one) through 10 (ten), with level 1 being the "least difficult".

3.6 FALLING ITERATION DELAY

Tetris has a real-time delay between successive line free-fall iterations that is a function of the current difficulty level.

Level	Delay, seconds
1	0.50
2	0.45
3	0.40
4	0.35
5	0.30
6	0.25
7	0.20
8	0.15
9	0.10
10	0.05

If the board is empty, and there is no user input, a spawned piece at level 1 lands in approximately 10 seconds, and a spawned piece at level 10 lands in approximately 1 second.

3.7 SCORE

The number of points per landed piece increase with each level. A quick drop is rewarded more points than a normal landing.

Completing a line rewards more points than simply landing a piece. Completing two lines at a time rewards more points than completing a single line, etcetera.