# Tetris Solution

## Overview

To store the Tetris grid, an array of unsigned short (16-bit length) is used. The 16 bits is more than enough to store the 10-block wide Tetris grid.

Shapes are stored as individual arrays of 3 unsigned shorts and are loaded from a configuration file (App.config).

The solution uses bitwise AND to detect collisions between the falling shapes and the existing grid layout and bitwise OR to populate the grid with the new shapes when a free position has been located.

## Array

The array starts at 0 at the bottom of the grid, working upwards. Bits 2-11 are used to represent the 10-block wide grid of blocks.

The below example shows a Q0 block at the bottom of the grid: -

array[0] xxxx1100000000xx

array[1] xxxx1100000000xx

array[2] xxxx0000000000xx

array[3] xxxx0000000000xx

array[…]

To get the shapes into position, the shape array is shifted left by a maximum of 8 bits (for position 0). For example, below a Q0 block has been left by 8 bits: -

shape[0] ....000000000000

shape[1] ....110000000000

shape[2] ....110000000000

array[0] xxxx0000000000xx

array[1] xxxx0000000000xx

array[2] xxxx0000000000xx

array[…]

To allow for blocks to be placed at position 8, e.g. Q8, two bits are left as a buffer on the right hand side.

Q8 block shown below: -

shape[0] ............0000

shape[1] ............1100

shape[2] ............1100

array[0] xxxx0000000000xx

array[1] xxxx0000000000xx

array[2] xxxx0000000000xx

array[…]

## Usage

The Tetris functionality is called via the static method **Tetris.ProcessFile** which accepts the following two parameters: -

* **inputFile** - The input path and filename of the file to be processed
* **outputFile** - The output path and filename of the file to be produced