Team: The Hulk

Team members:

Georgi Ivanov (georgi.ivanov)

Kalojan Atanasov (Kalojanatanasov)

Pirin Karabenchev (pirin)

Project goal: To implement a Tetris clone. We base our solution on Windows Forms technology, aiming better user experience and studying something new during development.

Project URL: https://georgiivanov.visualstudio.com/

Used Interfaces:  
 - IDrawable – every part of the visualization have to have .Draw(Graphics surface) method and to know how to draw itself in the context of Windows Forms.

- IMovable – a set of methods, allowing Shape and Block to move to left, right and down direction.

- IGrid – describes the functionality of the holder of the grid – we have used simple array of blocks. The grid has to know its dimensions and to know if given cell is empty or not. All three interfaces are grouped in a separate folder.

The grid represents the game field and holds the building blocks of the immobilized shapes. When given shape can’t go down any more its blocks are placed in the grid. Then a check for full rows is performed – if found any the rows above them are moved one step down. Every demolished rows increases players’ score.

ShapeFactory.cs is used to produce random Shapes, which are then added in the ShapeQueue – a place, where they can be shown to the player.

Shape.cs is an abstract class which holds all the needed functionality of a shape. It is inherited by seven different classes, representing the seven basic shapes in Tetris game – I, O, J, L, T, S and Z shape. Every class knows how to initialize itself. These classes are grouped in ShapeStore folder. Another interesting property of the Shape is its ability to rotate – first we check if the new place of every block of the shape is empty and then we rotate all blocks around a pivot point, stored in the shape instance.

Every shape consists of few blocks. They are described in Block.cs. IDrawable and IMovable are implemented. The block holds instances of System.Drawing.Brush and Rectangle. They are needed for the actual drawing of the block. The block also knows it’s position in the grid (row and col) and is it possible to go to the left, right, bottom or any other specific place – it uses the grid to check if the new place is empty or not.

ScoreEntry.cs is the place where the current score of the player is stored, along with his name and the quantity of demolished rows.

ScoreManager.cs is used to save, load and reset the file with the high scores – score.dnt. Dnt comes from DoNotTouch. And because users tend to dig everywhere and a file called “score.dnt” is kind of interesting thing we have decided to encrypt it. The high scores are stored in plain text, then a hash code is calculated and appended to the text. The next step is XOR with a hardcoded string. Finally the string is written to a file. When retrieving the high scores from the store device the process goes in reversed direction. If the check sum doesn’t match we assumed that score.dnt has been changed and we reset it.

For user input we listen to the Windows Forms event KeyDown. Depending on the actual key pressed we perform the needed actions. We use the internal timer of the Form to move the active shape down.

This was a quick review of our Tetris implementation. Thank you for reading the article.