Solution for Conway’s Game of Life

1 Applied SOLID OO design principles and created class GameOfLife and Validator classes which implemented IGameOfLive and IValidator interfaces accordingly. This design is good for future extension and unit tests.

2 In method InitializeWorld, initialize 2-demension array World. Use Validator class to check whether the input is valid.

3 In the Evolve method, only use one 2-D array \_world. Use three buffers to temporarily store data of each line then update \_world array later. The space complexity of my solution is n\*n + 3\*n bits. Compare to other solutions which use two 2-D arrays (space complexity is 2\*n\*n bits), my solution save nearly half of the memory if the grid is massive. Some other solution may use the combination of one 2-D array and one List<int> (for storing the neighbour numbers). This seems save one 2-D array. But it actually consumes more memory. Because int size is 32 bit and Boolean size is 1 bit. Use other data structures to store int for the whole grid will consume 32 times memory of 2-D Boolean array.

4 In the private method GenerateBufferForLine, I pass bool[] buffer parameter as reference(use ref keyword). The benefit of using “ref” keyword is it passes bool[] buffer as reference rather than create a new copy of bool[]. This will reduce memory consumption and reduce the number of times the garbage collector will run.

5 Create public read only properties: World, Columns and Rows. Expose those properties to external world so other developers can use GameOfLife class as an API to develop Conway’s game of life UI. Class GameOfLife also stores the current state of the “World” internally so that one copy of 2-D array is shared across the whole class. It avoids being passed as a parameter to multiple methods and created several unnecessary copies. This helps to reduce the overall memory consumption.

6 Use Lazy initialization for 2-D array “World”. It will deferred the creation until it is first used. It is very useful when dealing with large data set. It improves both performance and memory consumption when we are dealing with massive grid.

7 Create unit test project to test all the scenarios. I didn’t only test 5 x 5 gird. I also tested 15 x 15 gird to ensure the results are still correct. I didn’t only test World evolves once, but also test scenarios that World evolves several times.

8 Use functional programming style for selecting lines in “World” grid. This makes code more readable and concise.

9 In the Main method, I evolves the world four times. The results will be printed on the console. It’s interesting to see that the world was destroyed after evolving 4 times in In the 5 x 5 torus gird.

Future improvement:

Because of the time limit, I copied the logic of game rules and calculating neighbour numbers. I should be able to re-write the logic in more concise and readable way. My main focus for this task is memory consumption.