GridManager.cs:

-Creates a grid filled with random blocks based on the entered row and columncount (InitializeGrid()). If there are no blastable areas initially, it generates a new grid (ClearGrid()).

-Checks whether a block is blastable. This is done by calling the CheckAndDestroyBlocks method for the selected block, and if it can be blasted, it ensures the blocks are destroyed.

-The method that checks how many blocks can be blasted is GetConnectedBlocks(). This method uses BFS to determine how many adjacent blocks of the same color are connected.

-After blocks are destroyed, they fall down using the DropBlocksWithAnimation() and MoveBlock() methods.

-Instead of shuffling the grid when a deadlock occurs, I implemented an algorithm to ensure there is always a blastable area on the grid. This algorithm checks for matchable blocks (HasMatchableBlocks()) and adjusts the grid accordingly by dropping new blastable blocks.(RefillGrid(), CreateSuitableBlock(), HasMatchableBlocksForPosition()).

Block.cs:

-Calls the GetConnectedBlocks() method from the GridManager class to determine how many neighboring blocks of the same color exist and changes to the appropriate sprite accordingly.

-The clicking mechanic works for both mouse clicks (for testing purposes) and touch inputon mobile devices.

-To ensure synchronization, blocks cannot be clicked while any block is falling on the screen.