Get the next generation of cell states

In the Game of Life a cell is a block in a square matrix surrounded by neighbours. A cell can be in two states, alive or dead. A cell can change its state from one generation to another under certain circumstances

1. A live cell with fewer than 2 live neighbours dies of loneliness
2. A dead cell with exactly 2 live neighbours comes alive
3. A live cell with greater than 2 live neighbours dies due to overcrowding
4. The remaining cells remain unchanged

Given a current generation of cells in a matrix, what does the next generation look like? Which cells are alive and which are dead? Write code to get the next generation of cells given the current generation

HINT: Represent each generation as rows and columns in a 2 D matrix. Live and dead states can be represented by integers or boolean states

GAME OF LIFE

SAY THIS IS THE STATE OF THE BOARD IN THE CURRENT GENERATION

WHAT WILL BE THE STATE OF THIS CELL?

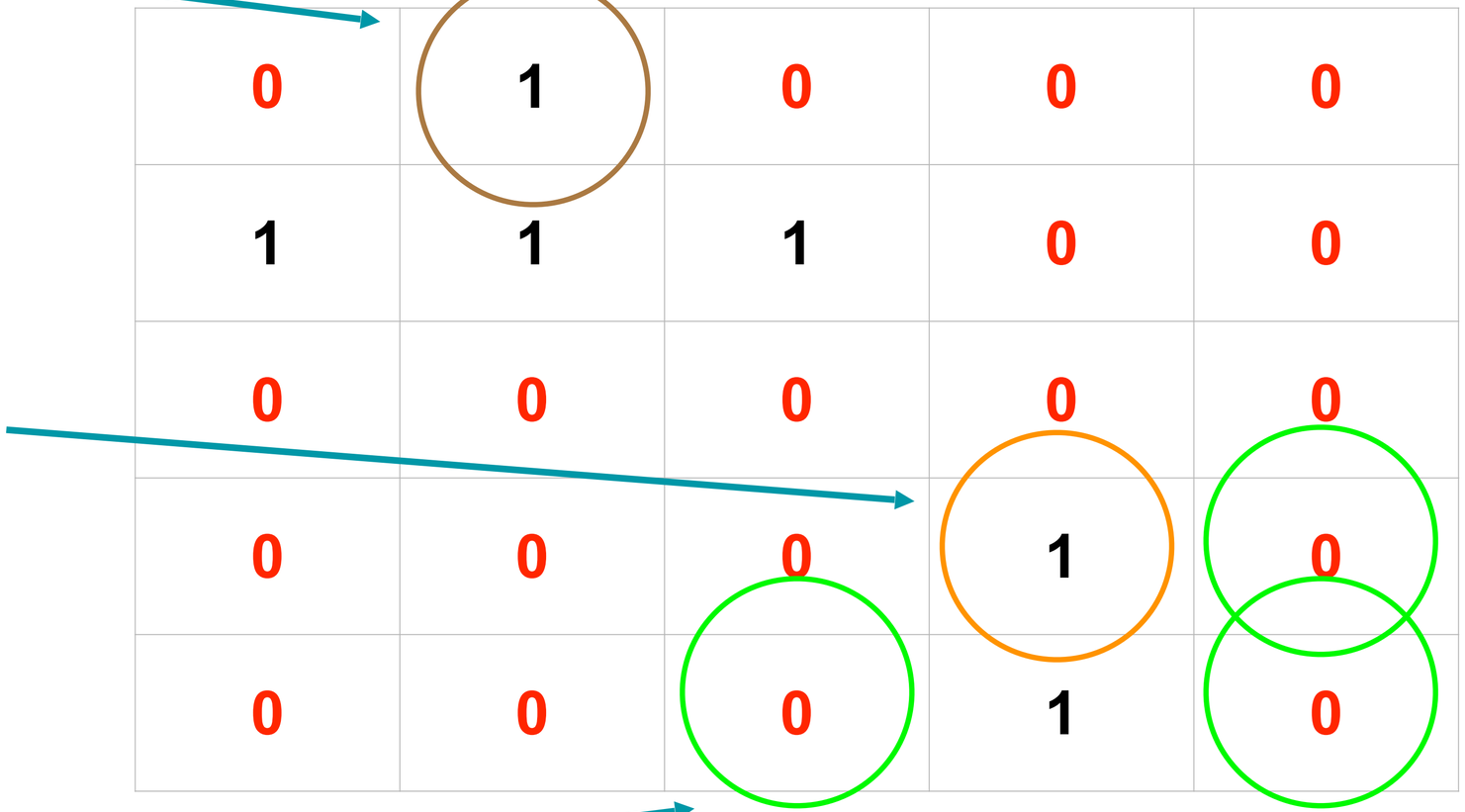
IT HAS 3 LIVE NEIGHBOURS SO IT WILL DIE OF OVERCROWDING

WHAT WILL BE THE STATE OF THIS CELL?

WITH ONLY 1 LIVE NEIGHBOUR IT WILL DIE OF LONELINESS

WHAT WILL BE THE STATE OF THESE CELLS?

THEY HAVE 2 LIVE NEIGHBOURS SO THEY COME ALIVE



GAME OF LIFE - GET THE STATE OF ONE CELL

```java
public static int getCellState(int row, int col, int[][] current) {
    int liveCount = 0;
    int lastCellIndex = N - 1;
    if (row > 0 && col > 0) {
        liveCount += current[row - 1][col - 1];
    }
    if (row > 0) {
        liveCount += current[row - 1][col];
        if (col < lastCellIndex) {
            liveCount += current[row - 1][col + 1];
        }
    }
    if (col < lastCellIndex) {
        liveCount += current[row][col + 1];
    }
    if (col > 0) {
        liveCount += current[row][col - 1];
        if (row < lastCellIndex) {
            liveCount += current[row + 1][col - 1];
        }
    }
    if (row < lastCellIndex) {
        liveCount += current[row + 1][col];
    }
    if (row < lastCellIndex && col < lastCellIndex) {
        liveCount += current[row + 1][col + 1];
    }

    return liveCount == 2 ? 1 : 0;
}
```

ASSUMING A N X N MATRIX REPRESENTING ONE GENERATION, THE LAST CELL INDEX WILL BE N - 1

CHECK THE NUMBER OF LIVE NEIGHBOURS A CELL HAS. REMEMBER THAT NEIGHBOURS ARE ALONG THE 4 SIDES AND THE CORNERS AS WELL

MAKE SURE THE MATRIX INDICES ARE VALID BEFORE ACCESSING A CELL. CELLS IN THE CORNER HAVE JUST 3 NEIGHBOURS

GAME OF LIFE - GET THE NEXT GENERATION

```
public static int[][] getNextGeneration(int[][] current) {
    int[][] next = new int[N][N];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            next[i][j] = getCellState(i, j, current);
        }
    }

    return next;
}
```

SET UP A NEW MATRIX FOR STORING THE NEXT GENERATION SO THE CHANGES YOU MAKE DON'T INTERFERE WITH CALCULATING THE STATE OF THE NEXT GENERATION

FOR EVERY CELL IN THE MATRIX CALL THE GETCELLSTATE METHOD WHICH WE JUST SET UP

Break a document into chunks

A document is stored in the cloud and you need to send the text of the document to the client which renders it on the screen (think Google docs). You do not want to send the entire document to the client at one go, you want to send it chunk by chunk. A chunk can be created subject to certain constraints

1. A chunk can be 5000 or fewer characters in length (this rule is relaxed only under one condition - see below)
2. A chunk should contain only complete paragraphs - this is a hard and fast rule
3. A paragraph is represented by the ':' character in the document
4. List of chunks should be in the order in which they appear in the document (do not set them up out of order)
5. If you encounter a paragraph >5000 characters that should be in a separate chunk by itself (@navdeep underline the exception in point 1 at this point)
6. Get all chunks as close to 5000 characters as possible, subject to the constraints above

Given a string document return a list of chunks which some other system can use to send to the client

HINT: `public List<String> chunkify(String doc)`

BREAK A DOCUMENT INTO CHUNKS

SUPPOSE THE CHUNK SIZE IS 5, RATHER THAN 5000 FOR EASE OF TESTING.

THE RESULTANT CHUNKS WOULD BE

CHUNK 1 "a:bb:"
CHUNK 2 "cc:"
CHUNK 3 "abcdef:"
CHUNK 4 "ab:c:"
CHUNK 5 "d:"

EACH CHUNK SHOULD BE < 5 CHARACTERS, WHERE 5 IS THE MAXIMUM CHUNK SIZE

SINCE THE PARAGRAPH AFTER THIS WAS > CHUNK SIZE THIS ENDED UP IN A CHUNK ALONE

VERY LONG PARAGRAPHS ARE ALWAYS IN A CHUNK BY THEMSELVES

BREAK A DOCUMENT INTO CHUNKS

```java
public static List<String> chunkify(String doc) {
    List<String> chunkList = new ArrayList<>();

    String[] paragraphs = doc.split(DELIMITER);

    String currentChunk = "";
    for (String para : paragraphs) {
        if (currentChunk.length() + para.length() > CHUNK_SIZE) {
            if (currentChunk.length() > 0) {
                chunkList.add(currentChunk);
                currentChunk = "";
            }
        }

        if (para.length() > CHUNK_SIZE) {
            if (currentChunk.length() > 0) {
                chunkList.add(currentChunk);
                currentChunk = "";
            }
            chunkList.add(para + DELIMITER);
            continue;
        }

        currentChunk += para + DELIMITER;
    }

    if (currentChunk.length() > 0) {
        chunkList.add(currentChunk);
    }

    return chunkList;
}
```

THE LIST TO HOLD THE CHUNKS OF THE DOCUMENT

GET THE PARAGRAPHS IN THE DOCUMENT, WE CREATE EACH CHUNK PARAGRAPH BY PARAGRAPH

IF ADDING THE CURRENT PARAGRAPH EXCEEDS CHUNK SIZE, THEN ADD THE CHUNK TO THE LIST TO BE RETURNED, THE CURRENT PARAGRAPH SHOULD BE IN A NEW CHUNK

IF THE PARAGRAPH ITSELF IS LONGER THAN THE CHUNK SIZE THEN ADD IT TO A CHUNK BY ITSELF

ENSURE YOU ALWAYS ADD THE DELIMITER BACK AFTER EVERY PARAGRAPH OTHERWISE PARAGRAPH BOUNDARIES WILL BE LOST

ADD THE LAST CHUNK IN IF IT IS PRESENT