Object-Oriented Programming
Group Assignment #4
TDD (Tears, Despair & Debugging) Assignment

CMP_SC/INFO_TC 3330

Spring 2025

1 Objective

In this assignment, you are expected to design and implement "Tears, Despair & Debugging". In this game, you are going to construct a grid of size NxN (with a size constraint, see description for details). Your agent tries to escape from the grid by keeping track of apertures.

2 Description

In a grid, each cell is composed of 4 components, which are up, bottom, left, and right. Each component can be WALL, APERTURE, or EXIT.

You should create a grid randomly (the size of a grid can be between 3 and 7) by satisfying the following constraints:

- Grid has only one EXIT component (the EXIT must be the left component of the cell and must be on the leftmost side of the grid, as shown in the Table 1 and 2).
- Each cell has at least one APERTURE component.

After the grid construction, the game must be started. The Agent can move in four directions (UP, DOWN, LEFT, RIGHT). For each agent movement, the updated grid must be printed. The format of the grid visualization is shown in the following for the given grid below, where 'E' denotes EXIT, 'S' denotes SPACE, and 'A' denotes AGENT. When the agent reaches the EXIT cell, to escape from the grid, make sure the agent moves in the left direction, because the EXIT component is always assigned to the leftmost side of the grid, and left on the cell.

Your goal is to get the agent out of the maze! You are given JUnit5 test cases, and you must implement the project based on the test cases given to you (aka. Test-driven Development).

Table 1: Sample Initial Grid

E	S	S	S	S
S	S	S	S	S
S	S	S	S	S
S	S	S	S	S
S	S	S	S	A

Table 2: Updated Grid After Agent Moved UP

E	S	S	S	S
S	S	S	S	S
S	S	S	S	S
S	S	S	S	A
S	S	S	S	S

Important Notes

- You cannot change the given test cases.
- Make sure your code passes the given test cases.
- You can write helper methods.
- You must write a simulation class that shows that your code works (A class with a main method). Don't worry, you don't need to test your main method.
- Follow Java naming conventions, or you will lose points.
- Use packages or you will lose points.
- Add Javadoc to your code, or you will lose points.
- Export your project properly, or you will lose points.
- Don't want to 1 commit project, or commit messages like "*Adding Java code*" or "*Update code*", otherwise you will lose points. Commits must be small and meaningful with a commit message that is relevant to the code you pushed. Only I am allowed to do the above, because I am the professor of this class, and I can do whatever I want. This is my class :D
- Write your code considering edge cases. Make sure you have error controls.
- Don't ask how many points will be deducted for the notes above. There is no negotiation here. These are good practices that you must adopt and follow to have a successful career. You can try to violate one of the good practices above and see what happens :) (not recommended).
- Everyone in the group must contribute to the project. Use Git efficiently and communicate!
- If there is a group drama, you have to wait until the next group assignment to split from your group or work alone. See syllabus for details.

- **Due date:** 4/22/2025, 11:59 PM.
- **Submission:** You must submit your GitHub repository and your exported project through Canvas. Submit your GitHub link repository in a file along with your project file submission.

3 Grading Rubric

Component	Points	Description
Test Case Passing (High Priority)	**40**	All provided test cases must be passed. Partial credit will be given for passing some test cases. Tests demonstrate functional correctness.
Simulation Implementation	10	A working simulation class that runs the game and demonstrates its functionality, including proper grid visualization.
Code Quality & Readability	10	Clean, readable code with proper indentation, naming conventions (Java style), and effective use of comments. Includes use of packages, Javadoc documentation, meaningful Git commit messages
Random Grid Generation with Constraints	10	Grid is randomly generated within constraints (size 3–7, only one EXIT, each cell has at least one APERTURE. Having all cell components as APERTURE with no WALLs is invalid).
Total	**70**	