Software Team Challenge: Documentation

How to obtain source codes:

1. Github users can go to https://github.com/everfor/Maze_Challenge_ and clone the repository; everyone can download from this link and extract the zip file.
2. The necessary source codes for the challenge are "*MazeGenerator.h*", "*MazeGenerator.cpp*", "*MazeSolver.h*", and "*MazeSolver.cpp*". Import them into a project and you are ready to go!

Files Explained:

MazeGenerator.h:

- ➔ Header file for MazeGenerator class
- ➔ Includes necessary dependencies (headers)
- ➔ Contains declaration of MazeGenerator class and its methods

MazeGenerator.cpp:

- ➔ Implementation of MazeGenerator class and its methods

MazeSolver.h:

- ➔ Header file for MazeSolver class
- ➔ Contains declaration of MazeSolver class and its methods

MazeSolver.cpp:

- ➔ Implementation of MazeSolver and where the main() function lies
- ➔ You'll have to implement the SolveMaze() method here

Methods:

MazeGenerator::GenerateWalls(int width, int height):
Returns a 2D vector (at this point, you may as well consider it a 2D array) of walls of an unprocessed maze with given width and height; each cell is surrounded by 4 walls.
This method is used during the process of generating a maze and thus you will not need to call it.

MazeGenerator::GenerateMaze(int dimension):
Generates a random perfect square maze* given the dimension.

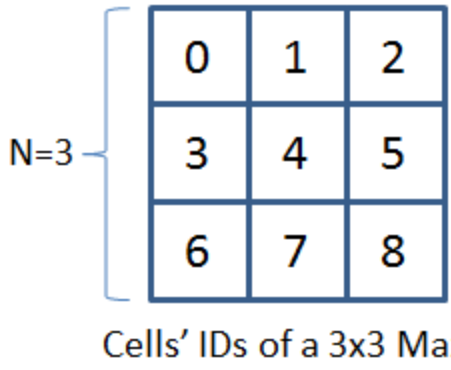
The method returns a 2D vector containing information about walls surrounding each cell. The

first dimension contains cells' IDs and the second dimension contains wall information of each cell, starting from the left wall and continues in a clockwise fashion. For a detailed explanation of the data returned, see the example below.
You will need to use this method to generate the maze to solve.
*Perfect Maze: A maze is "perfect" if it does not contain any loop and any 2 cells in it are connected by an only path. Therefore, perfect mazes are always solvable.

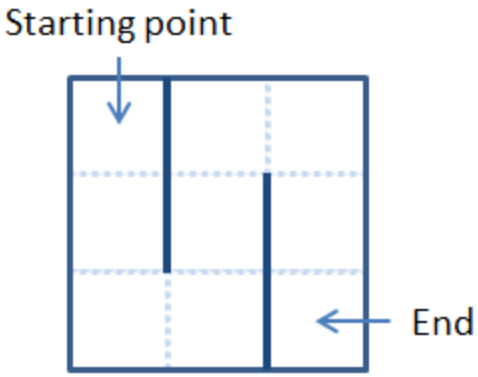
MazeSolver::SolveMaze(std::vector<std::vector<int>> walls):
This is the method you will implement.
Given a 2D vector (again, consider it as a 2D array) about wall information in a maze, the method should try to solve the maze and return a 1D vector containing the order of cell's ID along the solution path. For a detailed explanation, see the example below.

MazeSolver::ValidatePath(int dimension, std::vector<std::vector<int>> walls, std::vector<int> path):
This is a helper validation function. Given the dimension of a maze, walls in a maze and the calculated path, it will return true or false, depending on whether the path solves the maze successfully.

Example:
In this example, MazeGenerator::GenerateMaze(3) is called, and the returned data is stored in a variable called "walls". The maze looks like this:



Cells' IDs of a 3x3 Maze



Example of a 3x3 Maze

Then "walls" is a 8x4 2D array, meaning that each of the 9 cells has 4 pieces information about walls (left, upper, right and lower). The wall information starts from left wall and goes clockwise (left -> upper -> right -> lower).

As an example, walls[0][0] = 1, walls[0][1] = 1, walls[0][2] = 1, walls[0][3] = 0; this tells that cell with id 0 has left, upper and right walls, and does not has a lower wall.

More examples:
walls[1][0] = 1, walls[1][1] = 1, walls[1][2] = 0, walls[1][3] = 0
walls[2][0] = 0, walls[2][1] = 1, walls[2][2] = 1, walls[2][3] = 0

Then, MazeSolver::SolveMaze() will take in “walls” as parameter and try to find a path from cell 0 to cell 8. Since the maze is perfect there is only one (guaranteed) possible solution. This method should return a list of orders of cells’ ids along this path.

In this example, this method should return a 1D vector: {0, 3, 6, 7, 4, 1, 2, 5, 8}.

At a loss?

1. If you are new to C++, you can read this tutorial, which will walk you through from the very beginning of environment setup to really advanced stuff like multi-threading.
2. Intimidated by command lines? You may try Visual Studio or Eclipse (with C++ plugin). But remember to try getting used to command lines, as we will do most of the programming in Ubuntu for the competition.
3. Still we recommend using Eclipse with C++ plugin as your IDE for this challenge. If you are a complete beginner, just follow the tutorial here: http://www3.ntu.edu.sg/home/ehchua/programming/howto/EclipseCpp_HowTo.html
4. Don’t be afraid to speak loud! We are there to help. Still do enough research before you ask any question though.
5. **And have fun!**