THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

School of EECS
CSSE2002/7023 Semester 2, 2023
Assignment

Maze Navigator

Due 5 of October 2023 at 15:00 AEST

Abstract
The assignment instructs students to design, implement, and test a set of classes and interfaces[1] to solve a maze. With a narrative setting in the Kingdom of Labyrinthus, students will develop an application to aid the Kingdom's royal goose in navigating the maze.

Preamble

All work must be individual, except for code provided by course staff this semester. Follow guidelines from lectures and the EECS UQ Student Conduct. Direct any ambiguities to the course staff.

Note: All time references use the Australian Eastern Standard Time. Ensure to align assignment submissions accordingly.

Introduction

Set in the Kingdom of Labyrinthus, this assignment offers a blend of narrative and programming. Tasked with aiding a whimsical character, you'll utilize accessor methods, Arrays or Lists, outputs, and core programming logic. While the narrative provides context, focus on designing a robust system using classes and interfaces.

Story

Welcome, esteemed programmer! MazeMania Inc., renowned for its world-class mazes, needs your expertise. The Kingdom of Labyrinthus's annual "Grand Maze Festival" is upon us. However, Sir Wobbleton, our royal goose, has taken a fondness to one of our mazes. While charming, this poses a challenge: ensuring Sir Wobbleton can always find his way back to his duties. Your task is to develop an application to navigate a maze and ensure Sir Wobbleton's punctuality.

Your Task

Accept this assignment, and you'll:

1. Load a maze from a provided text file.
2. Display the maze either using GUI (Figure 1) or text (Figure 2).
3. Navigate the maze, either manually[2] or programmatically[3].
4. Determine whether an exit is found or if the maze is unsolvable[4].

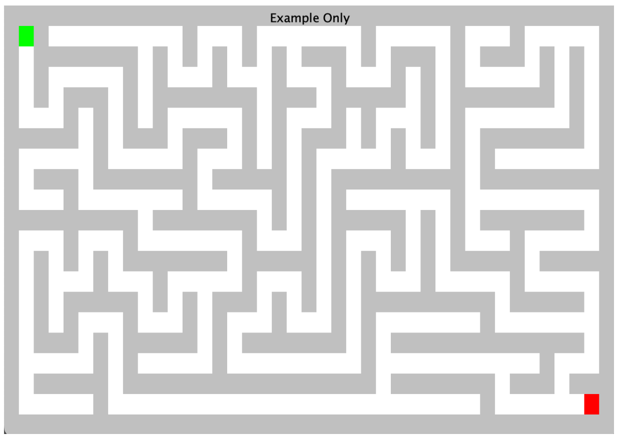

Figure 1 (GUI Display)

[1] From now on, classes and interfaces will be shortened to simply "classes"
[2] Using keyboard input (KeyListener)
[3] Advanced stretch goal
[4] Solvable maze is assumed but an unsolvable condition must considered.

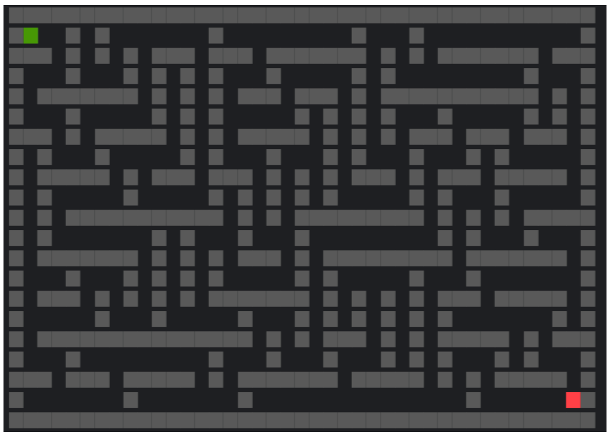
Figure 2 (Text Display – yours may vary)

To guide you in developing the file loader, we've provided an interface named FileInterface.java. Ensure that your file loader class implements this interface and meets all its stipulations. Pay close attention to the imported packages; they can offer insights on structuring your exceptions and determining the expected package locations when implementing the interface, and other classes.

Your loader should draw inspiration from the one presented in the lectures, ensuring it robustly reads the maze data and addresses any potential issues. As suggested in the provided code, make sure the loaded maze has a single start point and end point, along with other logical requirements. Ensure that all your code is thoroughly documented using JavaDoc comments.

Pro Tip

- Implement proper exception handling and ensure clear error messages.
- Close resources like the `Scanner` object if used for file reading.

File IO

- Utilize File IO to load a maze from a given text file (see *supplied files and example in Appendix A)*
  - *Create a Class that satisfies the FileInterface.java specification.*
- The maze's dimensions will be provided in the file's first line in the form of two integers, both must be odd numbers to ensure the maze has an external wall and internal paths.
- The remaining maze attributes will be provided in the following char configurations.
  - Walls - `'#'`
  - Traversable Paths - `' ' or '.'`
  - Start Point - `'S'`
  - End Point- `'E'`
- You must throw suitable exceptions if the maze isn't loaded properly, considering factors such as, but not limited to the following.
  - Uneven maze length.
  - Inappropriate characters.
  - Invalid dimensions
- Your implementation must utilise apt levels of error handling.

Display

- The maze can be displayed as GUI or text output.
  - Default to text output; use GUI only when 'GUI' appears in the command-line arguments[5]
  - Supply the filename via Command-Line[6] or a GUI dropdown menu[7].
  - Research advanced navigation methods like Depth-first, Breadth-first, A*, etc[8].

[5] EXAMPLE: `java Launcher GUI`
[6] EXAMPLE: `java Launcher Maze002.txt` and `java Launcher GUI Maze002.txt`
[7] Advanced GUI stretch goal.
[8] Extremely advanced stretch goal.

- Adopt the MVC paradigm for efficient code structure across different displays.

Maze Navigation

- Navigate the maze starting from the 'S' position.
  - Take one step at a time, avoiding walls.
  - Employ lambda expressions 'up', 'down', 'left', and 'right'.
  - Validate movements using criteria such as an isTraversable boolean.
  - Track your path in the maze's 2D array, marking valid paths and backtracked routes (see *Figure 3*).

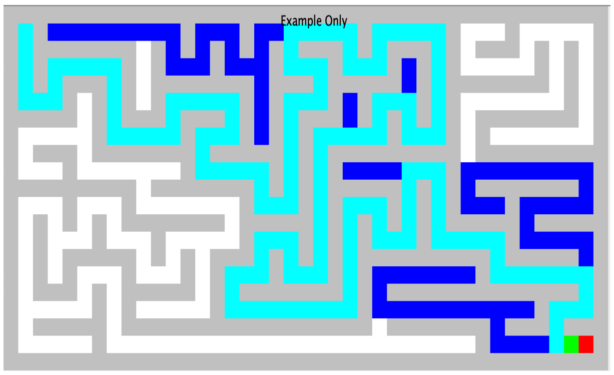

Figure 3 Depicting GUI version of Traversed Path and backtracked pathways.

Classes & Subclasses

- Design appropriate classes and subclasses, informed by lectures.
  - Consider a `Position` class for maze coordinates.
  - Use `MazeComponent` for maze symbols and positions.
  - Implement specific classes for maze elements like Wall, Path, EndPoint, etc.
  - Feel free to design other classes as needed.

Launcher

- Kickstart the application with a `Launcher` class holding the main method. This class will determine if the maze will be displayed in GUI or text form based on command-line arguments, see Display section above.

Testing

- Your maze loader will undergo electronic testing.
- Create two Junit Testing classes to demonstrate your code's functionality.

This assignment will not only test your technical prowess but also your ability to encapsulate real-world challenges within object-oriented paradigms. Your success will be determined by how efficiently and effectively you can manoeuvre through the maze, both in the narrative and in code. You are being given the freedom to create this project, using your own ideas and intuition. The focus should be on delivering a functioning application that tackles the contracted expectations, as fully as you can. Your functioning code, and the level at which you have been successful, your code style, code commenting, and use of both basic and advanced Java implementations will be examined in accordance with the standards we have discussed in lectures, practicals, and tutorials. For some students this may seem challenging. However, it is important to recognise the creative nature of programming and hone this tenet in line with your learning. Your ability to follow guidelines, code independently, and solve programming problems will be challenged.

And remember, the happiness of the Royal Kingdom of Labyrinthus and Sir Wobbleton rests on your capable shoulders. So…unleash your coding prowess and ensure that our charming goose finds his way home every time!

Happy coding!

Supplied Material

- File Loader Interface
- Maze text test data

Given the materials, you have the autonomy to choose which packages, classes, and methods you'll develop, leveraging your newly acquired skills. This approach not only ensures the uniqueness of your work but also reflects your grasp on the subject matter, programming syntax, and professionalism.

Evaluation Criteria

Symbol	**Mode**	**Description**
FT	Hybrid (Electronic/In-Person)	Functionality according to the interface specification.
CF	In-Person	Conformance to your defined specification
CS:SL	In-Person	Code Style: Structure, layout, and adherence to style guide.
CS:CR	In-Person	Code style: Style Guide Compliance
JU	Electronic	Junit testing for accurate and robust interface implementation.

LEVEL ACHIEVED	**FT**	**CF**	**CS:SL**	**CS:CR**	**JU**	**Overall**
Exceptional (7)	At least 85% of tests are successful	Expert conformance (no more than 2 violations)	Mastery of style (no more than 4 violations)	Variable naming is masterful. Comments are significantly numerous, meaningful, detailed and useful. Code design follows best practice. Programming follows best practices.	At least 85% of faulty solutions are identified	FT7;CF6+; CS 6+; JU 6+
Advanced (6)	At least 75% of tests are successful	Extensive conformance (no more than 3 violations)	Extensive compliance with style (no more than 6 violations)	Variable naming is substantially appropriate. Comments are extensively numerous, meaningful, detailed and useful. Code design is substantially approaching best practice. Programming substantially approaches best practice.	At least 75% of faulty solutions are identified	FT 6+; CF 5+; CS 5+; JU 5+
Proficient (5)	At least 65% of tests are successful	Good conformance (no more than 4 violations)	Good compliance with style (no more than 8 violations)	Variable naming is good. Comments are generally meaningful, detailed and useful. Code design is good. Programming practices are good.	At least 65% of faulty solutions are identified	T 5+; CF 4+; CS 4+; JU 4+
Functional (4)	At least 50% of tests are successful	Adequate conformance (no more than 5 violations)	Adequate compliance with style (no more than 10 violations)	Variable naming is adequate. Comments are adequate. Code design is adequate. Programming practices are adequate.	At least 50% of faulty solutions are identified	(FT4+orJU 4+; FT 3+) and CS:SL 4+ and CS:CR 3+
Developing (3)	At least 30% of tests are successful	Superficial conformance (no more than 6 violations)	Superficial compliance with style (no more than 12 violations)	Variable naming is adequate. Commenting is superficial. Code design is adequate. Programming practices are adequate.	At least 30% of faulty solutions are identified	(FT3+orJU 3+; FT 2+) and CS:SL 3+
Minimal Achievement (2)	At least 20% of tests are successful	Deficient conformance (no more than 7 violations)	Deficient compliance with style (no more than 14 violations)	Variable naming is deficient. Commenting is deficient. Code design is deficient. Programming practices are deficient.	At least 20% of faulty solutions are identified	(FT2+orJU 2+) and CS:SL 2+
Absence (1)		No demonstrated evidence of ability to apply concepts in the field of study.				

Functionality Marking

Functionality is assessed in person. We will gauge if you've aptly implemented classes, properties, and methods as taught. Be prepared to discuss and defend your design choices. Your score will reflect your understanding of the project, competency, and the professionalism of your code.

Submission

- Platform: Gradescope
- Detailed submission instructions will be provided on Blackboard (under Assessment → Assignment). Submissions before the release of these instructions won't be accepted.
- Ensure timely submission. Late submissions will incur penalties as per the ECP.
- Multiple submissions are allowed before the deadline; however, only the final one before the due date will be evaluated.

Late Submission

Submissions post the deadline on 5th October 2023 will incur penalties as detailed in the Electronic Course Profile. Please avoid last-minute rushes. Any submission that begins before 15:00 but concludes post 1:00 will be subject to penalties. Fairness mandates no exceptions for individual students.

Common Style Guide Errors:	
Metric	**Errors**
Naming	Examples: String temp;, char a;, int myVar;
Commenting	Inadequate Javadoc comments for classes, methods, constructors, or variables. Sparse or uninformative inline comments.

Code Design	Misuse of class member variables, duplicated code sections, overly lengthy functions, illogical code layout.
Programming Practices	Inadequate use of inheritance, poor exception handling, using magic numbers without context, e.g., object.someMethod(50); // Clarify 50's significance.

What to Submit
Your project should include:

`src/` for packages and .java files, described in the Javadoc.
`test/` for JUnit test class files.

Example Structure (yours may vary):

```
src/io/YourIOFile.java

test/io/YourTestingClass.java (Note test directory, not src!)
...
```

Make sure to declare the correct package in your classes and interfaces. Avoid submitting other file types (e.g., .class files). Your JUnit tests will be individually compiled against a sample solution, excluding other test files..

Assignment Extensions
Extension requests should be made via my.UQ as outlined in section 5.3 of the Electronic Course Profile. Direct emails to the course coordinator for extensions will be redirected to my.UQ.

Remark Requests
For assignment remarks, refer to:|
https://my.uq.edu.au/information-and-services/manage-my-program/exams-and-assessment/querying-result
Remember, your score can remain unchanged, decrease, or increase upon re-evaluation.

Embracing Practical Coding Proficiency
In contrast to previous years, this assignment is designed specifically to ensure that you can confidently apply coding skills in practical scenarios. By successfully completing this assignment, you'll not only earn your grade but also solidify a personal testament to your coding abilities. We believe in your potential, and this project will be a significant milestone in your programming journey.

example Maze002.txt file

```
21 41
#########################################
#S#         #   #       #     #   #     #
# ####### # # # # # ### # ### # ### # # #
# #     # #   #   #   #   # # #     # # #
# # ### # ####### ### ##### # ####### # #
#   # # # #     # #   # #   # #       # #
##### # ### ### # # ### # # # # ####### #
#     #     #   # # #     #   # #       #
# ### ####### ##### # ######### #########
#   #       #     # # #       #         #
######### ####### # # ##### # # ####### #
#   #   #       #   # #   # # #   #     #
# # # # ####### ##### # # # # ##### #####
# #   #   #     #   # # #   #     #     #
# # ##### # # ### # # # ######### ##### #
# # #   #   # #   #   # #       #       #
# ### # ##### # ####### # ############# #
#     # #     #         #           #   #
# ##### ################# ####### ### ###
#     #                         #      E#
#########################################
```