RMIT UNIVERSITY

Algorithms and Analysis
COSC 1285/2123
Assignment 2: Exploring Maze Generation and Solving Algorithms

Assessment Type	Individual assignment. Submit online via Canvas → Assignments → Assignment 2. Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/assignment FAQ/relevant discussion forums.
Due Date	Week 13, June 5, Wednesday 11:59pm
Marks	30

Please read all the following information before attempting your assignment. This is an *individual* assignment. You may not collude with any other people and plagiarise their work. Everyone is expected to present the results of their own thinking and writing. Never copy other student's work (even if they "explain it to you first") and never give your written work to others. Keep any conversation high-level and never show your solution to others. Never copy from the Web or any other resource or use Generative AI like ChatGPT to generate solutions or the report. Remember you are meant to develop the solution by yourself - assessment is designed to encourage everyone to learn, and you are not doing yourself any favours by taking short cuts. Suspected collusion or plagiarism will be dealt with according to RMIT policy.

In the submission (your PDF file for the report of Task B) you will be required to certify that the submitted solution *represents your own work only* by agreeing to the following statement:

> *I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honor code by typing "Yes":*

1 Overview

In this assignment, you will implement and explore algorithms for maze generation and solving. This assignment extends upon Assignment 1, and focuses on algorithmic design and understanding.

2 Learning Outcomes

This assessment relates to two learning outcomes of the course which are:

- CLO 1: Compare, contrast, and apply the key algorithmic design paradigms: brute force, divide and conquer, decrease and conquer, transform and conquer, greedy, dynamic programming and iterative improvement;

- CLO 5: Implement, empirically compare, and apply fundamental algorithms and data structures to real-world problems.

3 Background

In Assignment 1, we studied 2D mazes and two data structures for representing/storing them. In this assignment, we extend this to 3D mazes - in addition to rows and columns, we now have a number of levels. The aims is still to find a path from an entrance to an exit, but now we can additionally go up or down a level. See Figure 1 for an example.

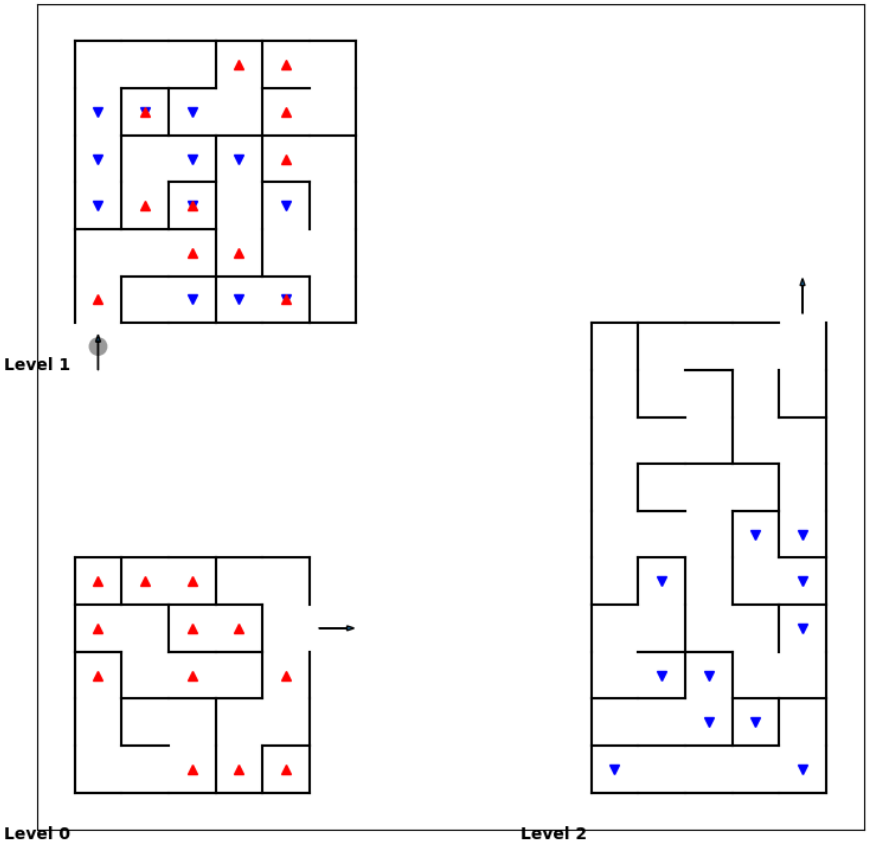

Figure 1: Sample 3D maze that has 3 levels. Level 0 is 5 by 5, level 1 is 6 by 6 and level 2 is 10 by 5. The entrances are illustrated as arrows pointing towards the maze, and exits are illustrated as arrows pointing away from the maze. The cells of the maze are indexed from 0, and the row and column indices are drawn outside the maze. Note that the bottom left cell on the lowest level (level 0) is (0,0,0), and top right cell is (0,4,4), where the the format is (level, row, column). This follows the convention that matplotlib follows. Note the exits and entrances do not all have to be on level 0, they can be on any level. The indexing of the cells in levels 1 and 2 are aligned with this indexing, e.g., the cell on level 1 that is above cell (0,0,0) on level 0 is (1,0,0). To illustrate passages between cells of different floors, a passage to a cell on the adjacent higher level is illustrated by a red triangle, while a cell on the adjacent lower level is illustrated by a blue triangle.

We are focused on *perfect* mazes, where there is always a path from entrance to exit

(by the virtue that in a perfect maze, there is always a path from a cell in the maze to any other cell in the maze).

Each cell has 4 sides on a level, and possibly one cell one level above it, and one cell one level below it. There is no standard way to reference the cells, hence we adopted the convention illustrated in Figure 1, with (0,0,0) – (level, row, column) – denoting the bottom left cell on the lowest level (always referenced from 0, or level 0), and columns increase as we travel to the right of the maze (page), and rows increases as we travel up the maze (page). Similar to Assignment 1, in order to specify the entrances and exits, we have added one row above, and one row below each level of the maze, and one column to the left, and one column to the right of each level. This means the row below the maze is referenced as -1, the row above by 5 (if we have 5 rows in the maze), the additional column to the left of the maze by -1, and the column to the right by 5 (if we have 5 columns in the maze). This allows us to still be able to use the original indexing, i.e., (0,0,0) refers to the bottom left cell of the maze on level 0, but still able to reference the location of entrances and exits. This is the same convention as Assignment 1, apart from addition of levels.

There are strategies and algorithms to *solve* a maze, i.e., find such a solution path given a maze. There are also strategies and algorithms to *generate* a maze. We will focus on these generation and solving strategies in this assignment.

4 Assessment details

The assignment is broken up into a number of tasks, to help you progressively complete the project.

Task A: Different Generation Approaches (6 marks)

To gain a better understanding of how mazes are generated, you will implement two maze generation algorithms. We also provide an implementation for your study, and to allow you to progress with maze solving even if you can't get some of the maze generators working. Each algorithm can produce different types of mazes, which is something we will explore further in Task D.

We provide the following maze generation algorithms:

- Recursive backtracking (generation) algorithm - this is the same as in Assignment 1.

We ask you to study, understand and implement two more advanced and different algorithms.

- Prim's algorithm.
- Wilson's Algorithm.

See the provide skeleton code also for additional information on how we implemented the Recursive Backtracking maze generation algorithm, particularly how we used the Maze3D data structure. Below is some description of the algorithms, but please attend the lectorials and read the online material, where further descriptions will be provided.

Recursive BackTracking Maze Generator

The recursive backtracking maze generator is essentially a DFS generator. Starting with a maze with walls between all adjacent pairs of cells, it randomly selects a cell in the maze, and perform a DFS traversal of the whole maze, from that initial cell. During the DFS traversal, we generate all the unvisited neighbours of the current cell, and randomly select one of those to go to. When we go to an unvisited cell, we remove/destroy the wall between the current cell to the selected unvisited neighbouring cell. If there is no unvisited neighbouring cell from our current one, we backtrack to the cell we were at previously. The DFS ends when we have visited all the cells in the maze.

Prim's Maze Generator

Prim's algorithm for maze generation is very similar to Prim's algorithm, particularly when we also using a graph representation of the maze. We start with a maze with walls between all pairs of adjacent cells. We select a random starting cell and put it into the visited set. We find all the neighbours of this cell, and this forms the frontier set. We select the cell/node with lowest weight in the frontier set (in our maze case, unless there is differences in moving between cells, all the edge weight/costs are the same, which means we randomly pick one). We then add that selected cell/node to the visited set, remove the wall between the selected cell and its neighbour that caused it to be added to the frontier set, remove it from the frontier set, and add all of the selected cell's unvisited neighbours to the frontier set. We repeat this until all cells/nodes are in the visited set.

Wilson's Maze Generator

Wilson's maze generation algorithm is one algorithm that can generate all mazes of any given size with equal probability - i.e., it doesn't favour one type of maze over another.

The algorithm, like the previous two, starts with a maze with walls between all pairs of adjacent cells. Also initially every cell in the maze is considered as not finalised. It then starts by randomly selecting a starting cell, that is the first cell to be considered as finalised. Select another cell, at random, that isn't finalised yet. Perform a random walk from this 2nd selected cell, until this random walk visits a cell that is finalised. Once this occurs, return to the initial selected cell, and carve a path from that cell to the last visited cell. Where the random walk looped back to a cell it has visited before (hence that cell can have two neighbouring cells it can go next), we choose the direction/cell that it most recently took - by selecting only one, we can avoid loops, which is not something we want generated (for this assignment).

Task B: Solvers (7 marks)

In the last task, we studied and implemented a number of maze generation algorithms. In this task, we will focus on maze solving algorithms. Similar to the previous task, we provide a Recursive Backtracking solving algorithm, and ask you to implement two others that are similar to each other, the Wall Following solving algorithm and the Pledge solving algorithm to improve your understanding of how maze solvers work.

We provide the following maze solving algorithms:

- Recursive backtracking (solving) algorithm.

We ask you to study, understand and implement two more advanced and different algorithms.

- Wall Following Algorithm.
- Pledge Algorithm.

See the code also for additional information on how we implemented the Recursive Backtracking solving algorithm, particularly how we used the Maze3D data structure. Below is some description of the algorithms, but please attend the lectorials and read the online material, where further descriptions will be provided.

Recursive BackTracking Maze Solver

The recursive backtracking maze solver is the equivalent of the recursive backtracking maze generator. It starts at an entrance, and from the current cell, finds an unvisited neighbouring cell and go there (and mark it visited). Then it continues until either it reaches an exit, or a deadend, by which then it backtracks to a cell where there is at least one unvisited neighbour.

Wall Following Maze Solver

This algorithm is a simple heuristic and one that a human might follow. Starting at the entrance, once we step into the maze, imagine we either reach out our right (left) hand and follow the right (left) wall. We maintain contact with our right (left) hand until we reach the exit. When we lose contact with the wall, we rotate then move forward. The rotation is according to a cyclic order. For a 2D maze and right hand rule, it will be a 90 degree turn, e.g., the order {North, East, South, West}. That is, if we were going in a North direction, when rotate, we will head East; when we are heading East, and rotate, we will be heading South, etc. This won't work for a 3D maze, but if we project going up or down a level to a 2D maze by extending the rotations we can do as follows: {North, North-East, East, South, Sourth-West, West}, where North-East is equivalent to heading in the direction of going up one level, and South-West is equivalent to heading in the direction of going down one level. When we turn/rotate, we cycle through the six directions for a 3D maze. With this extension, this algorithm can solve a 3D maze if there isn't loops, which is the case for this assignment.

Pledge Maze Solver

The Pledge algorithm is an extension of the wall following approach. Initially, we start at an entrance and select a direction (one of the six directions in the Wall Following algorithm description above) to travel (at random), and keep moving in that direction until we hit a wall (or find the exit). We then perform wall following with either the left or right hand rule. We also keep count of the angle of turns we have made, e.g., if we consider going up a level is NE, and going down is SW (see Wall-following algorithm), then a left turn is -60, and a right turn is 60. This angle tracking, when done during the wall following step, means once we have done a series of turns where we have rotated enough times to go in the initial selected direction (angle = 0). We then proceed again with going in the initial direction until hit a wall etc. We then repeat the process until we find the exit. The Pledge algorithm is less likely than the Wall following algorithm to be stuck in loops.

Task C: Solvers with Unknown Exits (7 marks)

This is the first of two tasks where you will get opportunities to extend and explore beyond standard implementations. Typically a maze will have an entrance and an exit, and a maze solving algorithm will only need to go from the entrance to the exit, and we only seek to find a path.

However, when we have multiple entrances and exits, there are multiple paths from one of the entrances to one of the exits. We then can introduce the concept of closest entrance-exit pair.

In this task, you'll explore and design algorithms to find these closest pair of entrances and exits and a solution path between them. However, there is a number of conditions - we have to find the paths between the entrances and exits, and additionally, we include the cells explored by the path finders/solvers as the cost of finding this path. We wish to find the path of entrances and exits that:

$$\min(\text{cells\_explored}() + \text{distance}(\text{entrance}, \text{exit}))$$

where we are only given the set of entrances but not the location of the exits, hence need to locate the exits. We are given the number of exits there are. To help you get started, select one of the solvers, either provided or implemented by you, and solve the problem where we have one entrance and multiple exits. Once you solved this, then extend this to the case where there are multiple entrances (and still multiple exits).

Note that:

- The solver is assumed to not know where the maze exits are. However, you can assume it knows how many exits there are, but not the location of them.
- When the solver discovers all the exits in its exploration, the paths it took to get there from one of the entrances to each of the exits may not be the shortest path available.
- In the above case, the solver may decide to switch strategy and find the shortest path from each of the exits to an entrance.
- There is a balance between exploring the maze to find all the exits, exploring the maze to find the shortest path between an exit and entrance (remember we want to find a path that is the shortest between any entrance and exit), and minimising the amount of exploration to reduce the number of cells explored.
- There might not be one best/right solution, hence it is important in the comments in the code and in the report to explain your rationale of your chosen algorithm/strategy/approach.

To help us understand your approaches, we ask you to write a report of up to 2 pages as part of this task, in addition to your implementation.

Task D: Maximising Solver Exploration of Cells (8 marks)

This task is likely the most difficult of the Tasks A-D, and is considered as a HD task and suggested to be tackled last.

In this task, a town of maze generators and solvers is running a competition. There is a number of solvers with known solving strategies (algorithms), and the town is running

a competition where competitors generate different mazes such that the average cells explored over a number of generated mazes is maximised. The identify of the solvers are known each time a competitor is asked to generate a maze.

They also have a bonus part, where there is a mystery solver (with unknown strategy) and the idea is to generate a maze that will maximise the cell taken on average, regardless of what strateg(ies) the solver used.

Note that:

- The generator can query the passed solver for its name, which should indicate what type of solver it is.
- The solvers is one from Task B, a mystery one and we will update you about additional ones within the first 2 weeks of the assignment.
- There might not be one best/right solution, hence it is important in the comments in the code and in the report to explain your rationale of your chosen algorithm/strategy/approach.

To help us understand your approaches, we ask you to write a report of up to 2 pages as part of this task, in addition to your implementation and comments.

Task E: Recorded Interview (2 marks)

After your implementations and report are submitted, you will be asked to record your responses to a number of questions in Canvas. These questions will ask you about aspects of your implementation or report. You'll have a set time to consider the questions, make a recording then upload that recording. More details will be explained closer to submission.

Code Framework

We provide Python skeleton code (see Table 1) to help you get started and ensure we have consistent interfacing so we can have consist correctness testing.

We also listed the files that you really need to modify/implement. Unlike Assignment 1, there are more files you need to implement, hence see Table 1 to confirm those.

Notes

- If you focus on the parts with "TODO" and/or "Please implement" parts of the provided skeleton, you in fact do not need to do anything else to get the correct output formatting. mazeTester2.py will handle this.
- We will run your implementation on the university's core teaching servers, e.g., `titan.csit.rmit.edu.au`, `jupiter.csit.rmit.edu.au`, and `saturn.csit.rmit.edu.au`. If you develop on your own machines, please ensure your code compiles and runs on these machines. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, we unfortunately do not have the resources to debug each one and cannot award marks for testing. *Please note this carefully, this is a firm requirement.*

- All submissions should be compiled with no warnings on **Python 3.6.8** when compiling the files specified in Table 1 - this is the default Python3 version on the Core teaching servers. Please ensure your code runs on the core teaching servers and with this version of Python. *Please note this carefully, this is a firm requirement.*

5 Report Structure

As a guide, the report could contain the following sections:

- Your solution for Task C, and the rationale or justification behind it. (up to 2 pages in length)
- Your solution for Task D, and the rationale or justification behind it. (up to 2 pages in length)
- You can also have an appendix, which doesn't count towards the overall page count.

Clarification to Specifications

Please periodically check the assignment FAQ for further clarifications about specifications. In addition, the lecturer will go through different aspects of the assignment each week, so even if you cannot make it to the lectorials, be sure to check the course material page on Canvas to see if there are additional notes posted.

6 Submission

The final submission will consist of three parts:

- Your **Python source code** of your implementations. This must include all files, including the provided skeleton as well as any extra files you have created. Your source code should be placed into the same structure as the supplied skeleton code, and the root directory/folder should be named as `Assign2-<your student number>`. Specifically, if your student number is s12345, when `unzip Assign2-s12345.zip` is executed then all the source code files should be in directory Assign2-s12345. We use automated testing and compilation, and the testing script will expect this structure, so if is different, the script may not be able to compile your code. So please make sure not to change the structure.
- Your **written report for Tasks C and D** in PDF format, called "s12345-assign2.pdf" if your student number is s12345. Separately submit this to another submission location, for Turnitin checking.
- Then, the Python source file folder (and files within) should be zipped up and named as `Assign2-<your student number>.zip`. E.g., if your student numbers is s12345, then your code submission file should be called `Assign2-s12345.zip`, and when we unzip that zip file, then all the submission files should be in the folder Assign2-s12345.

Note: **submission of the report and code will be done via Canvas**. We will provide details closer to the submission deadline.

7 Assessment

The project will be marked out of 30.

The assessment in this project will be broken down into three parts. The following criteria will be considered when allocating marks.

Tasks A and B (**13/30**):

- Your implementation will be assessed based on the number of tests it passes in our automated testing. The tests will assess things such as whether the generated graph is perfect, whether a provided maze is solved, whether it has similar traces, when using the same random number seeds as our implementations of the algorithms.
- Your implementation will also be assessed whether it implements the requested algorithms.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence commenting and coding style will make up a portion of your marks.

Task C (**7/30**):

- Your implementation will be partially assessed based on a number of tests in our automated testing. The tests will assess things such as whether the number of cells between selected entrance and exit, the number of cells explored and how it compares with possible best solutions. This is more to provide an indication on how good the solution is.
- The clarify of the description, and the rationale/justification of the approach in the report. This is important for us to understand your reasoning.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence commenting and coding style will make up a portion of your marks.

Task D (**8/30**):

- Your implementation will be assessed based on the number of tests in our automated testing. The tests will assess things such as how close or better your implementation is to our sample implementation(s) in terms of the average number of cells explored (with same random generator seeds). This is used as an indication of how good the solution is, but not the only criteria we will use to assess your solution/implementation.
- The clarify of the description, and the rationale/justification of the approach in the report.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence commenting and coding style will make up a portion of your marks.

Video Interview 2/30:

This is a pass/fail assessment, and you'll be assessed based on your ability to answer some questions about your implement and report.

Late Submission Penalty: Late submissions will incur a 10% penalty on the total marks of the corresponding assessment task per day or part of day late, i.e, 3 marks per day. Submissions that are late by 5 days or more are not accepted and will be awarded zero, unless special consideration has been granted. Granted Special Considerations with new due date set after the results have been released (typically 2 weeks after the deadline) will automatically result in an equivalent assessment in the form of a practical test, assessing the same knowledge and skills of the assignment (location and time to be arranged by the coordinator). Please ensure your submission is correct (all files are there, compiles etc), re-submissions after the due date and time will be considered as late submissions. The core teaching servers and Canvas can be slow, so please do double check ensure you have your assignments done and submitted a little before the submission deadline to avoid submitting late. We strongly advice you submit **at least one hour before the deadline**. *Late submissions due to slow processing of Canvas or slow Internet will not be looked upon favourly, even if it is a few minutes late. Slow processing of Canvas or slow Internet will require documentation and evidence submission attempts was made at least one hour before the deadline.*

Assessment declaration: By submitting this assessment, you agree to the assessment declaration - `https://www.rmit.edu.au/students/student-essentials/assessment-and-exams/assessment/assessment-declaration`

8 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the following: `https://www.rmit.edu.au/students/student-essentials/rights-and-responsibilities/academic-integrity`.

9 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Canvas for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor. We will also be posting common questions on the Assignment 2 Q&A section on Canvas and we encourage you to check and participate in the EdForum discussion forum. However, please **refrain from posting solutions**, particularly as this assignment is focused on algorithmic and data structure design.

file	description
`mazeTester2.py`	Code that reads a configuration file and runs the appropriate generator and solver. *No need to modify this file.*
`generatorSelector.py`	Code that determines what generator to use in current run. Used in all tasks, particularly for Task D. *Please update file as needed.*
`solverSelector.py`	Code that determines what solver to use in current run. Used in all tasks, particularly for Task C. *Please update file as needed.*
`maze/maze3D.py`	Implementation of 3D maze. *No need to modify this file.*
`maze/util.py`	3D Coordinates class, and other utlity classes and methods. *No need to modify this file.*
`maze/graph.py`	Abstract class for graphs. All graph implementations should implement the Graph class (same as assignment 1). *No need to modify this file.*
`maze/adjListGraph.py`	Code that implements an adjacent list (for maze) *No need to modify this file.*
`generation/mazeGenerator.py`	Abstract class for maze generators. *No need to modify this file.*
`generation/recurBackGenerator.py`	Implementation of the recursive backtracking maze generator. *No need to modify this file.*
`generation/primGenerator.py`	Implementation of the Prim's generator algorithms. Used for Task A. *Please complete implementation.*
`generation/wilsonGenerator.py`	Implementation of the Wilson generator algorithms. Used for Task A. *Please complete implementation.*
`generation/taskDGenerator.py`	Implementation of the generator for Task D. Used for Task D. *Please complete implementation.*
`solving/mazeSolver.py`	Abstract class for maze solvers. *No need to modify this file.*
`solving/recurBackSolver.py`	Implementation of the recursive backtracking maze solver. *No need to modify this file.*
`solving/wallFollowingSolver.py`	Implementation of the Wall following algorithm solvers. Used for Task B. *Please complete implementation.*
`solving/pledgeSolver.py`	Implementation of the Pledge algorithm solvers. Used for Task B. *Please complete implementation..*
`solving/taskCSolver.py`	Implementation of the solver for Task C. Used for Task C. *Please complete implementation.*
`README.txt`	Please read this first, it mentions how to run the code. *No need to modify this file.*

Table 1: Table of provided Python skeleton files.