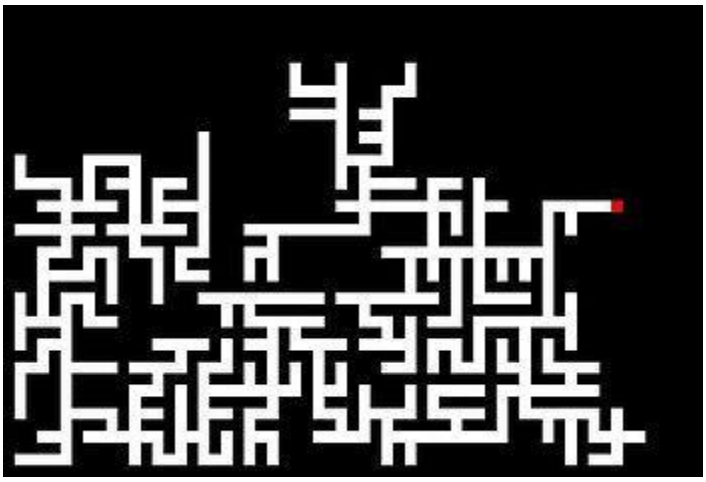
COSC1252/1254 Programming Using C++

Assignment 2: Mazes (part 2)

Due: Sunday 14th October, 11:59 pm



Overview

This assignment will require you to extend the functionality of assignment 1 by exploring different aspects of inheritance/polymorphism as well as some design patterns to produce modular code, with a focus on performance. The main requirements are:

- Generate mazes using different algorithms.
- Solve various mazes using pathfinding algorithm.
- Use polymorphism and design patterns to allow different maze generation/solving algorithms to be added easily.
- A simple class diagram representing what you think your final code will look like – do this at the start as you will be required to discuss how your original and final plan differed.
- A report discussing performance differences of the generating/solving algorithms implemented.

Details

The assignment should be done in C++, and must compile on the jupiter/saturn/titan servers using C++14. A large part of the assignment can be completed in the labs, and the exercises are directly related to the assignments.
You may work in pairs or individually.

You may enable g++ to use c++14 by entering the following command:

```
source /opt/rh/devtoolset-6/enable
```

It is probably a good idea to add this to your .bashrc file.

Class Diagram

Before getting started, draw a basic class diagram representing what you think your assignment will look like when finished. This does not have to be complete or exact but keep in mind good design principles when creating it. *Do not* do this after finishing the assignment as you will be asked to compare your original design with your finished application.

Maze generator

The program should be able to generate a maze using three different algorithms:

- Growing Tree Algorithm:
  http://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm

- Prim's Algorithms: More details available at:
  https://en.wikipedia.org/wiki/Prim%27s_algorithm
  https://en.wikipedia.org/wiki/Maze_generation_algorithm#Randomized_Prim.27s_algorithm

- Recursive Backtracking Algorithm:
  http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking

For full marks polymorphism should be used to reduce code duplication between the algorithms. The growing tree algorithm can be polymorphic in terms of the choice made for the next cell to add to the cell list. If you always choose the newest cell (the one most recently added), you'll get the recursive backtracker. If you always choose a cell at random, you get Prim's. A custom Set data structure should be implemented to speed up execution of the algorithm. A template set class with an underlying heap should be implemented to do with prims.

Note: You should not implement three algorithms separately. You need to use polymorphism.

Pathfinding

The program should be able to generate a path from cell (0,0) to cell (width-1,height-1) using Maze-routing algorithm:

- Maze-routing algorithm: https://en.wikipedia.org/wiki/Maze_solving_algorithm#Maze-routing_algorithm

Your solver should also be able to detect if the maze they are trying to solve is solvable (there is a path from the top left corner to the bottom right corner of the maze). If a maze is not solvable, you should inform the user of this.

Path Saving (SVG)

The program should be able to save the generated path in the SVG file. In addition to saving all the edges of the maze as in assignment 1, all edges belonging to the path should be saved in the colour red instead of the colour white.

Design Patterns

To promote modularity by allowing algorithms to be added/modified without affecting the rest of the program, a combination of the Prototype/Strategy/Factory Design Patterns could be used for maze generators and solvers.

Timing

Your program should be able to output to the terminal the time (in milliseconds) taken to generate the maze and to solve it.

Report

Note: For all timings you should run your program on the jupiter/saturn/titan servers. To calculate the timings you should use the std::chrono library.

Generate some mazes and save them both as binary and svg files:

- Prim using seed 0, of width/height 2000/2000
- Recursive Backtracking using seed of 0, of width/height 2000/2000

Do this 10 times, and record the mean (average) time for generation and for saving in binary/svg formats, as well as the standard deviation for your timings (You may find it easier to import your timing data into a spreadsheet program to help you with this).

Now record the time taken to solve each maze, using the maze solving algorithm (Maze-routing). Again do this 10 times and record the mean and standard deviations.

Finally write a report in pdf format, highlighting the following:

- Does your final application resemble your original class diagram and why/why not? What changes did you end up making for your design that you did not foresee?
- Discuss the difference in performance of each generation/solving algorithm and potential reasons for these differences.
- Are the results for each solving algorithm what you expected and why/why not?
- Are there any improvements you could make to increase performance?

Compilation

An appropriate makefile should be used and at a minimum you should used the following flags:

```
-Wall -Wextra -pedantic -std=c++14
```

Your makefile should build each .cpp file separately and then link them together into a single executable.

Controls

In addition to the controls from assignment 1, **the program must now also accept the following arguments:**

./exe --gp … # generate a maze using Prim's algorithm
./exe --gr …# generate a maze using the recursive backing algorithm
./exe --pm … # solve the maze using the Maze-routing algorithm

Note that command line args can be given in any order, and **an appropriate error message must be given if inappropriate args are entered.**

Note that the flag --g should no longer be accepted, since it has been replaced with --gr and --gp. Both --gr and --gp should accept variable arguments as they did in assignment 1. All other flags from assignment 1 should still work in the same way.

Submission

Submit assignments under assessment tasks on canvas via MyRMIT. Assignments can be submitted prior to the assignment due date.

Submit all of your source code, Makefile and README file. Create a zip file and submit that as a single file with the name sXXXXXXX-a2.zip, where XXXXXXX is your student number.

Do not include object files, executables or output files from your program. Assignments will be compiled by markers on the jupiter/saturn/titan servers so make sure you test your assignments on those servers. If you are working in pairs only one person needs to submit, but make sure to include both student numbers in the README file. In the case of both students submitting, the latest assignment submission will be used for marking.

A few notes:

- Markers can miss functionality if the default controls are changed. Make sure alterations and additions are listed in the readme.
- If there is a bug which can be avoided in certain circumstances, describe how.
- Do not remove incomplete code from your submission as this can sometimes inform us in what you were trying to do and you will get partial marks for that.
- When commenting code focus on complex areas. Not only does this improve readability, but it shows you know what you've coded. Try to make your code as self documenting as possible with appropriate variable names, etc.
- Separate long lines of code (longer than 80 characters) and use intermediate variables. Properly structured and indented code has less bugs and receives more marks. Don't mix tabs and spaces.

- Submitting early is always a good idea. You can resubmit, the last submission will be the one used for assessments.
- If you're having trouble submitting close to the due date, contact Emily.

Marking Guide

Note: this is a rough guide, and may be subject to change.

You will be expected to use good coding standards such as indentation, variable class and function naming, etc, as well as appropriate data validation.

PA: Generates a maze using at least one algorithm and solve the maze using Maze-routing algorithm.
CR: A basic report, in addition to the PA features.
DI: Generates a maze using all three algorithms with a reasonable report, in addition to CR features.
HD: High quality implementation, which the following can contribute towards (these also affect other grades too):

- Algorithms and data structures used appropriately to ensure reasonable performance
- Good use of appropriate C++ containers
- Good use of design patterns
- Good use of modern C++ features
- A reasonable initial design with minimal coupling and maximum coherence
- A good report that shows an understanding of the algorithms implemented
- Appropriate and consistent variable function and class naming conventions
- No unnecessary dynamic memory allocation
- No memory leaks
- Good commenting
- Good makefile

More detailed breakdown of marks:

Implementing the provided algorithms: 50%

Path generation:

- Growing Tree: 13%
- Prim's Algorithm for maze generation: 8%
- Recursive Backtracking: 8%

Pathfinding:

- Maze-routing algorithm: 8%

Path saving:

- Path Saving as svg: 5 %

C++ Concepts (operator overloading, correct use of memory, use of templates, etc) (20%):

- Good use of modularity and polymorphism, operator overloading method overloading, etc: 5%
- Appropriate use of template classes: 5%
- Appropriate use of design patterns as part of your application 5%.
- Good use of c++11/14 features 5%

General coding standards (15%):

- Avoiding goto statements (2%).
- Consistent use of spaces or tabs for indentation(2%).
- Keeping line lengths of code to a reasonable maximum such that they fit in the default xterm screen width (80 characters wide) (2%).
- Commenting (including function header comments) (2%).
- Appropriate identifier names (2%).
- Avoiding magic numbers (2%).
- General code readability and maintainability (2%).
- Makefile that compiles each source file separately and then links the files together (2%)
- Makefile that uses the g++ command line arguments we specified (2%)

Note that in the above section there are more deductions here than marks available so if you do more than 7 of these things you get 0 in this section.

Written report: 15%

The written report must be submitted as a pdf and must be submitted using the turnitin assignment submission link we will create. Please note that if the report is not submitted using the turnitin assignment submission link, it will not be assessed. Please note that the timing code and the class diagram form part of the report marks. In your report, we will focus on your analysis of the performance of your program. Written expression will form part of the marks here but only a small part. If there is any confusion on any points here, please ask on the discussion board.

Demonstration:

- 8 marks check point
- 20 bonus marks