C++ Assigment 2 Report

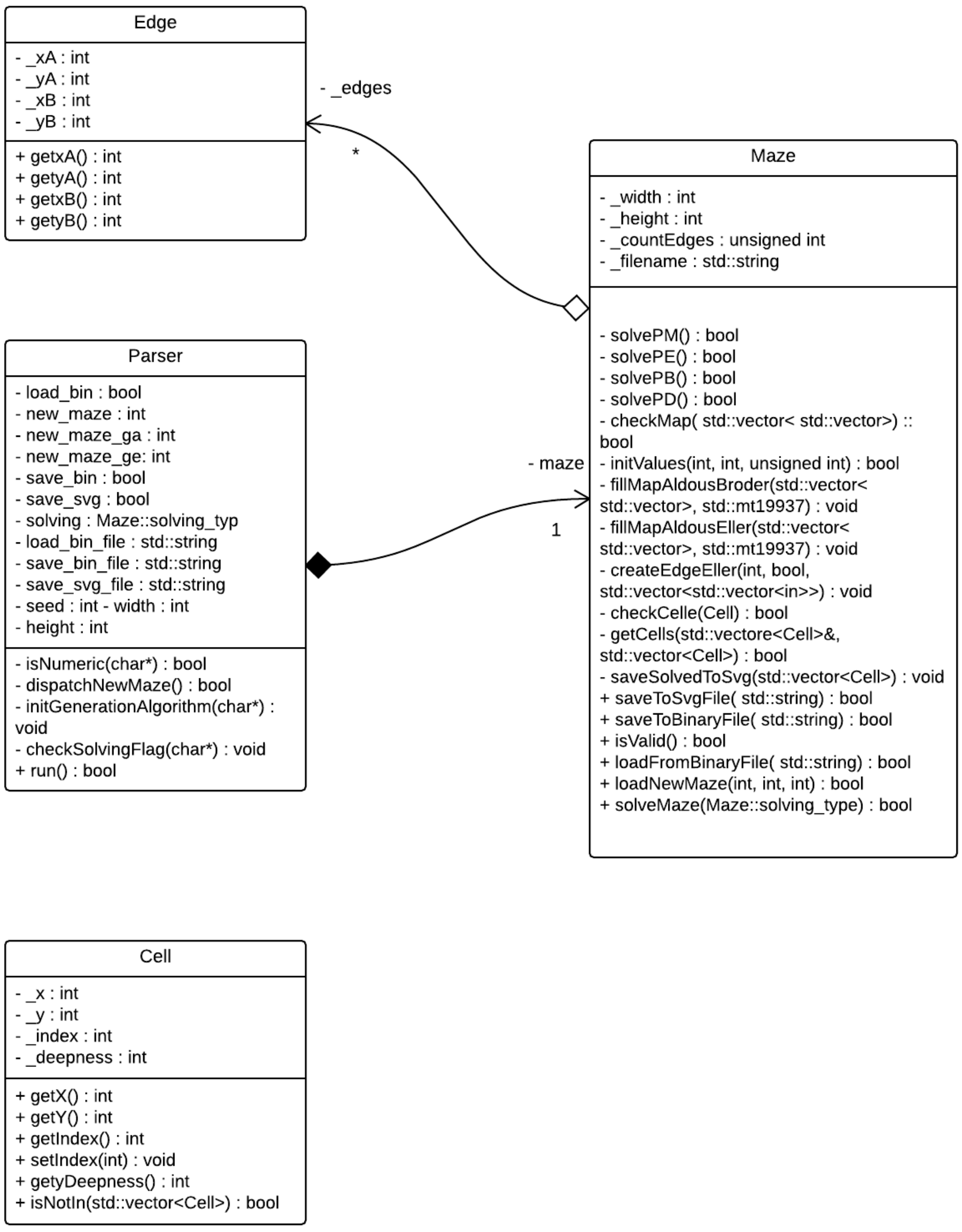
Written by :

Yixuan Zhang s3380293
Gregoire Ducharme s3615602

I. Performance report

Size / Algorithm	Depth first	Breadth first
20 * 20	- 27 ms	- 143 ms
	- 27 ms	- 148 ms
	- 25 ms	- 147 ms
	- 26 ms	- 140 ms
	- 26 ms	- 136 ms
	- 26 ms	- 142 ms
	- 28 ms	- 136 ms
	- 36 ms	- 137 ms
	- 28 ms	- 137 ms
	- 29 ms	- 136 ms
25 * 25	- 28 ms	- 342 ms
	- 29 ms	- 341 ms
	- 28 ms	- 356 ms
	- 30 ms	- 331 ms
	- 28 ms	- 326 ms
	- 32 ms	- 328 ms
	- 31 ms	- 327 ms
	- 29 ms	- 326 ms
	- 28 ms	- 327 ms
	- 31 ms	- 322 ms
7 * 7	- 4 ms	- 4 ms
	- 4 ms	- 4 ms
	- 8 ms	- 4 ms
	- 5 ms	- 4 ms
	- 4 ms	- 3 ms
	- 6 ms	- 5 ms
	- 5 ms	- 4 ms
	- 4 ms	- 3 ms
	- 4 ms	- 4 ms
	- 4 ms	- 4 ms
6 * 20	- 8 ms	- 14 ms
	- 6 ms	- 15 ms
	- 9 ms	- 17 ms
	- 7 ms	- 15 ms
	- 6 ms	- 14 ms
	- 8 ms	- 14 ms
	- 9 ms	- 15 ms
	- 7 ms	- 16 ms
	- 7 ms	- 14 ms
	- 8 ms	- 14 ms

II. Class diagram


Edge
- _xA : int
- _yA : int
- _xB : int
- _yB : int
+ getxA() : int
+ getyA() : int
+ getxB() : int
+ getyB() : int
- _edges
*
Maze
- _width : int
- _height : int
- _countEdges : unsigned int
- _filename : std::string
- solvePM() : bool
- solvePE() : bool
- solvePB() : bool
- solvePD() : bool
- checkMap(std::vector< std::vector>) :: bool
- initValues(int, int, unsigned int) : bool
- fillMapAldousBroder(std::vector< std::vector>, std::mt19937) : void
- fillMapAldousEller(std::vector< std::vector>, std::mt19937) : void
- createEdgeEller(int, bool, std::vector<std::vector<in>>) : void
- checkCelle(Cell) : bool
- getCells(std::vectore<Cell>&, std::vector<Cell>) : bool
- saveSolvedToSvg(std::vector<Cell>) : void
+ saveToSvgFile(std::string) : bool
+ saveToBinaryFile(std::string) : bool
+ isValid() : bool
+ loadFromBinaryFile(std::string) : bool
+ loadNewMaze(int, int, int) : bool
+ solveMaze(Maze::solving_type) : bool
- maze
1
Parser
- load_bin : bool
- new_maze : int
- new_maze_ga : int
- new_maze_ge: int
- save_bin : bool
- save_svg : bool
- solving : Maze::solving_typ
- load_bin_file : std::string
- save_bin_file : std::string
- save_svg_file : std::string
- seed : int - width : int
- height : int
- isNumeric(char*) : bool
- dispatchNewMaze() : bool
- initGenerationAlgorithm(char*) : void
- checkSolvingFlag(char*) : void
+ run() : bool
Cell
- _x : int
- _y : int
- _index : int
- _deepness : int
+ getX() : int
+ getY() : int
+ getIndex() : int
+ setIndex(int) : void
+ getyDeepness() : int
+ isNotIn(std::vector<Cell>) : bool

The class diagram is pretty similar to the class diagram of assignment 1.

The only new class if "Cell", which is used while solving. This class possess the coordinate of the cell, along with it index if the possible cell the path could have taken and it deepness in the algorithm solving.

It's not really different from what was expected.

III. Algorithm performance

As seen below the depth first algorithm is quicker to resolve mazes generated using our maze generator. Excluding the 7 * 7 maze where the breadth first looks quicker, but the time taken by each algorithm isn't long enough to jump to any conclusions.

This can be explained by the fact mazes generated using Aldous Broder's algorithm have only one solution.

The process of solving is as described below:

- Depth first:
    - Having a maze possessing only solution, once the solving algorithm found the good way, it doesn't have to go backward that often and just continues until it reaches the end of the maze
- Breadth first:
    - In the same circumstance the algorithm will still explore every path possible to solve the maze. Therefore losing a large amount of time in dead end. In some special case, for example, when the solution of the maze is one of the last to be explored by the depth first algorithm, the breadth first can be quicker (see 7 * 7 example below).

IV. Results

The maze generation algorithm use does not provide a perfect maze each time. Which means, the solving is not successful every time.

Therefore, we added in the "README" and the "test" script some seeds which allow to generate perfect maze and see how the solving is working

V. Enhancement

The algorithm might be faster using more temporary variable instead of instantiating variable and pushing them back to the vector, then popping them out if the path is incorrect.

VI. Overall

We had issue regarding Dijkstra’s algorithm since it doesn’t apply to the way our maze is thought. This solving method has not been implemented despite, understanding it and its value.

The Eller’s generation algorithm still suffer some imperfection and doesn’t give perfect maze, but the concept is also understood.

Deadlines being really sharp and with all the flow of assignments at this time of the semester, we decided to start this assignment using code already produce for assignment 1 and taking in consideration the feedback we had about it. This has obviously put some limitation on what could be done for assignment 2, but starting from scratch seemed like an impossible project.