Labyrinth Game documentation

Bowen Xu
711483
Computer Science
Freshman
26.4

2. General description

This is a simple labyrinth game that is able to create and solve 3D mazes. Some other features are a text based UI and the ability to save and load mazes from text files.

The program places the player on the top left cell and places a goal cell on the third floor in a random cell. The goal is to guide the player to the goal cell. If the player gives up he can click the solve button to show a route to the goal. The save button saves the maze as a text file and the load button creates a new maze based on the selected text file. I originally planned to create this on the moderate difficulty but in the end I decided to create it on the easy difficulty.

3. User interface

When the program is started it immediately generates a maze for you to solve. The keys ”w”, ”a”, ”s”, ”d” are used to move around the maze and the key “e” is used to interact with the stairs. The New Game button creates a new maze for you to solve. The other buttons work as explained previously.

4. **Program structure**

The Cell class represents the individual squares in the maze. Each cell has multiple different states that are represented with various parameters with boolean values.

The Floor class represents the individual floors of the maze and each floor is made up with cells. It also has the method “prim” and “solve”. “Prim” modifies the floor into a 2D maze and “solve” shows the path to the target square of the 2D maze.

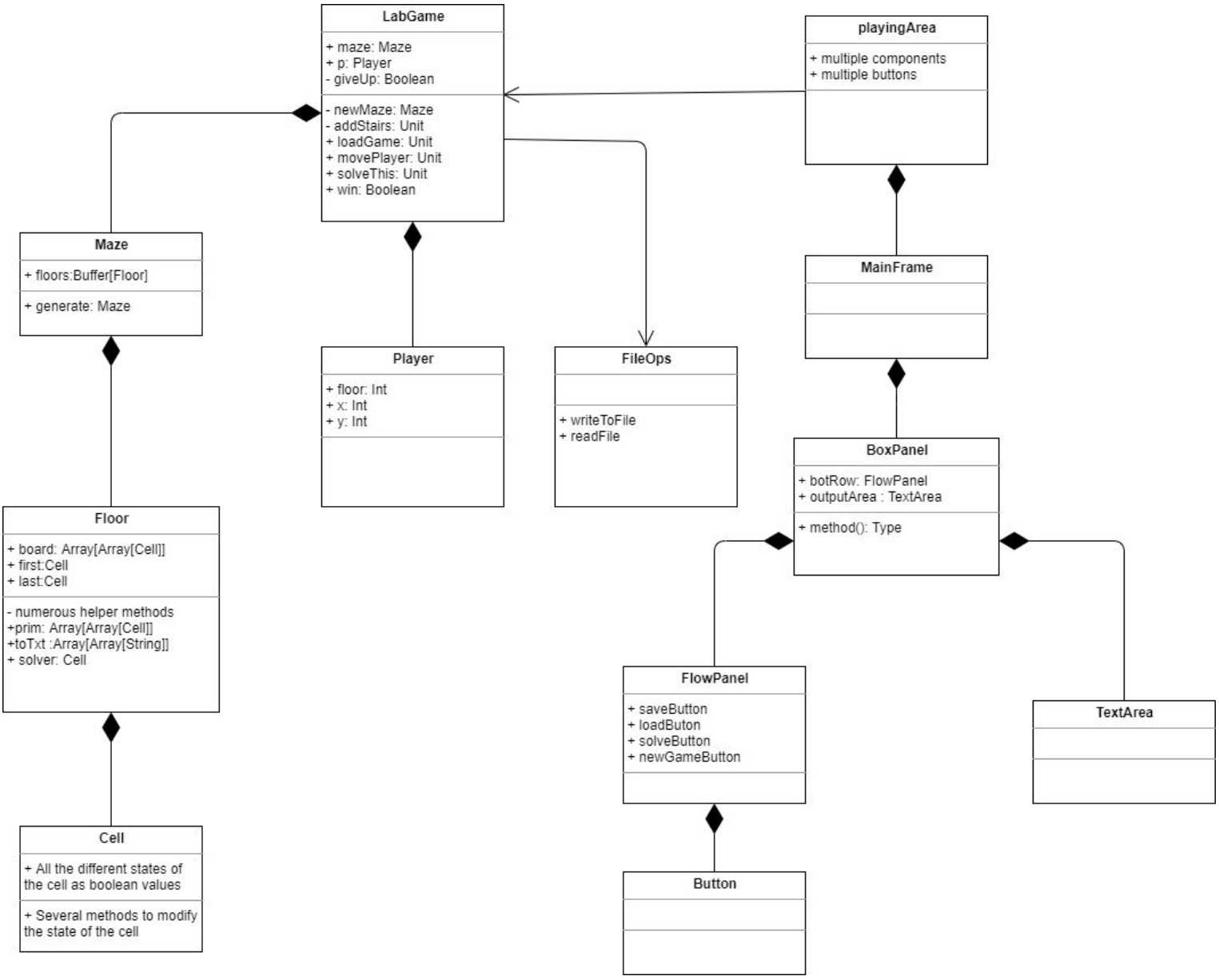
The Maze class is a collection of floors and represents the complete 3D maze. The method “generate” creates a 3 floor maze utilizing the method “prim” defined in the class Floor.

The Player class contains the location info of the player.

The LabGame class makes the game. It is responsible for moving the player, calling all of the methods required to create a maze. The method newMaze creates a 3 Floor maze and sets it up, for example selects the goal square etc. Then the method addStairs creates stairs to passable cells with a certain probability to make it more 3D. The method solveThis solves the whole maze from the current location of the player. The loadGame method creates a new maze from the given file.

The FileOps class contains the methods used to write and read files.

The user interface is made up from a MainFrame with a BoxPanel, which contains a TextArea which shows the labyrinth as characters and a FlowPanel with all of the buttons.



5. Algorithms

The maze generation uses the randomized prim's algorithm:
A Floor consists of a 2 dimensional array of cells.
A Cell can be Passable or not be Passable.
Start with a Floor full of Cells in state not Passable.
Pick a random Cell, set it to state Passage and calculate its frontier cells. A frontier cell of a Cell is a cell with distance 2 in state not passable and within the floor.
While the list of frontier cells is not empty:
Pick a random frontier cell from the list of frontier cells.
Let neighbors(frontierCell) = All cells in distance 2 in state Passable. Pick a random neighbor and connect the frontier cell with the neighbor by setting the cell in between to state Passable. Calculate the frontier cells of the chosen frontier cell and add them to the frontier list. Remove the chosen frontier cell from the list of frontier cells.

The solving algorithm is a recursive backtracker. This algorithm works like this:
If we're at a not passable cell or an already visited cell, return failure
Else if we're at the goal cell, then return success
Else, add the cell in the pathlist and recursively travel in all four directions. If failure is returned, remove the cell from the list and return failure. The pathlist will contain a unique path when the goal is found.

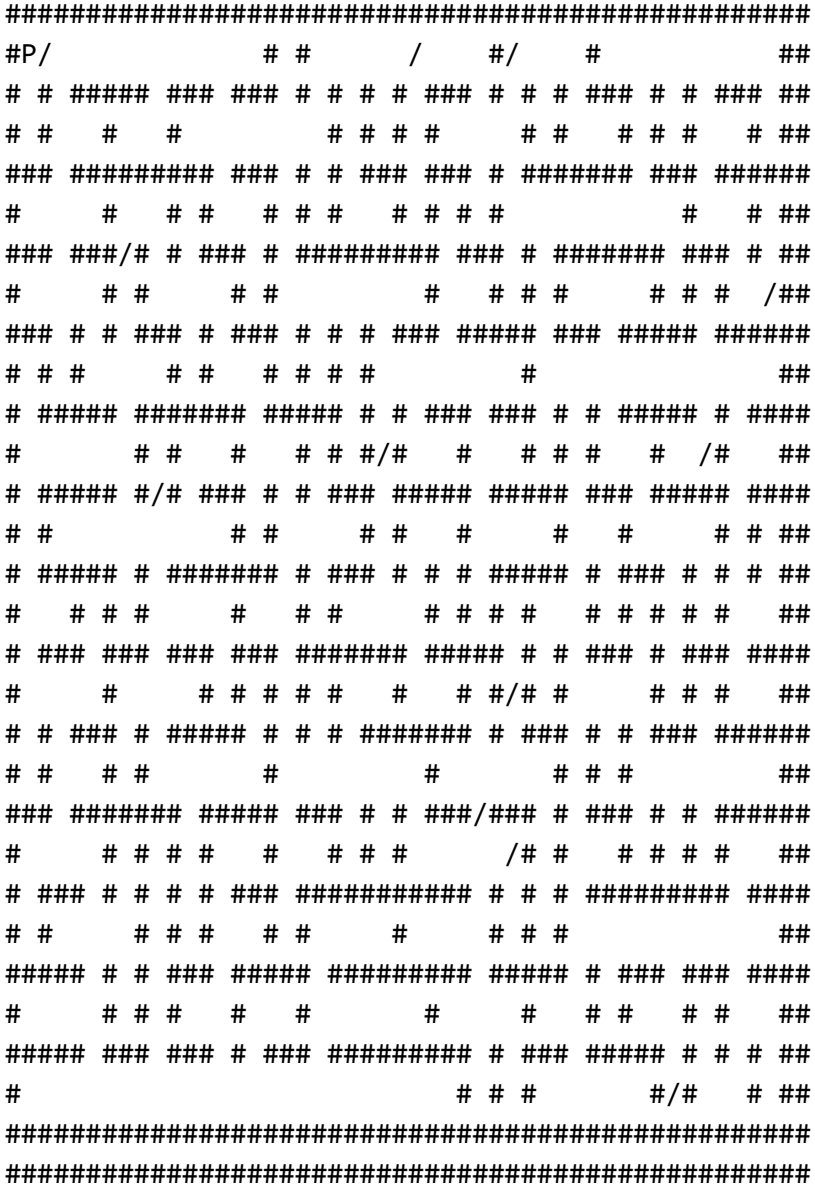
6. Data structures

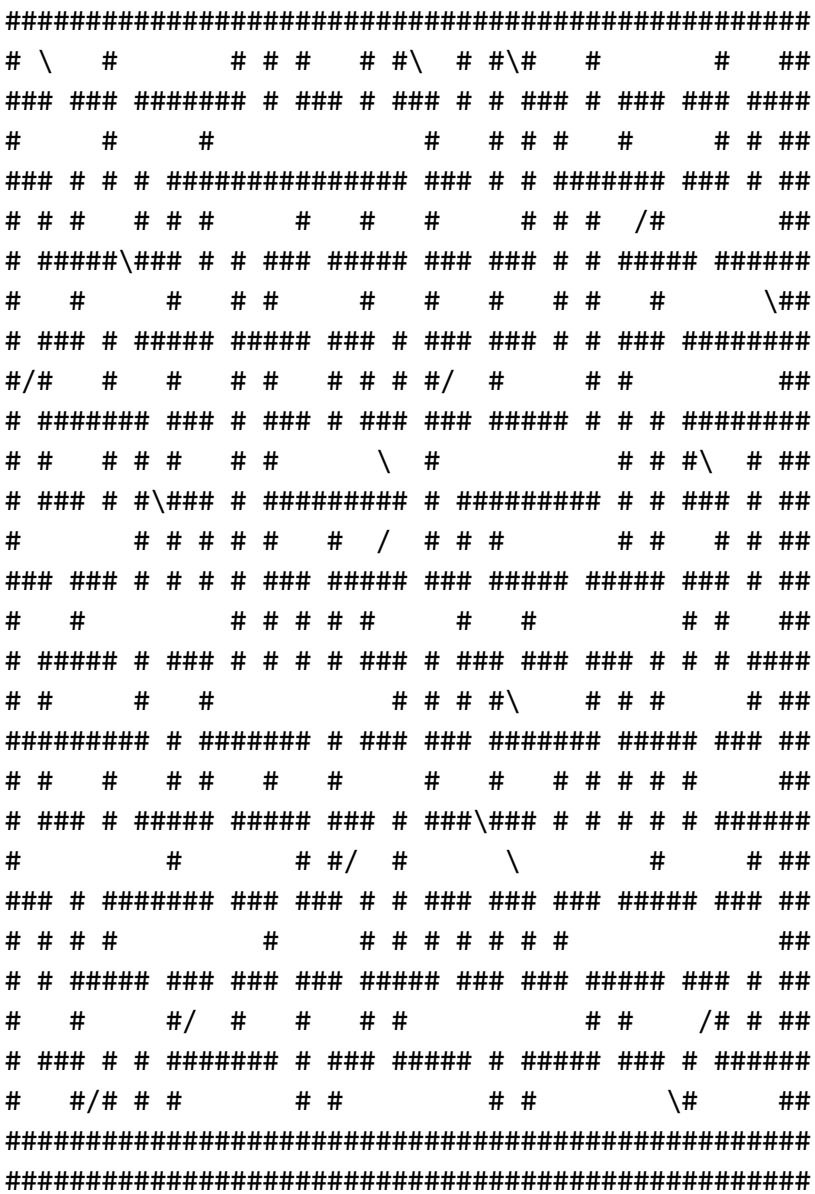
My program mainly uses 2D arrays to store information. I chose this because it seemed like the most intuitive way to represent a labyrinth. Buffers are used to track the frontierCells, neighbors and the path cells. I chose buffers for this task because it is possible to remove and add elements to it.

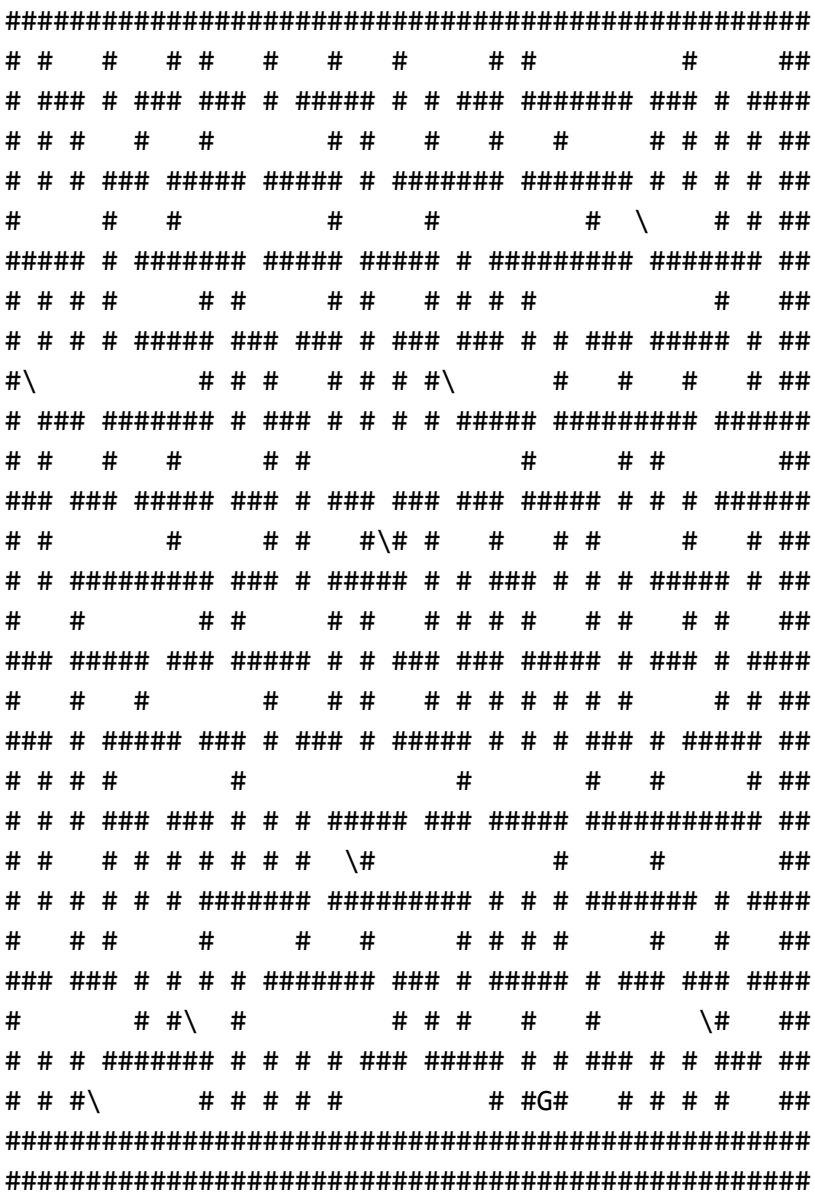
Arrays are mutable, indexed collections of values and Buffers are used to create sequences of elements incrementally by appending, prepending, or inserting new elements. It is also possible to access and modify elements in a random access fashion via the index of the element in the current sequence.

7. Files and internet access

The program deals with text files.
For example:







8. Testing

Testing was done mostly by printing out the mazes and looking at them manually. The file operations were tested by comparing their outputs to the actual mazes or files. The generation algorithm was tested by printing out the created mazes and determining by eye if they were properly created. The solving algorithm was tested by marking the cells that are part of the solution and then printing out the maze and looking at it if the route leads to the goal. No unit tests were utilized during testing. The testing process was very different compared to my original plan.

9. Known bugs and missing features

The loadGame method does not handle incorrect file formats, so the program crashes if the file that it reads from is formatted incorrectly. For example if the number of rows, row length and characters are not correct, it will crash. To fix this the method needs to first check the size of the file to make sure the row and row length are good and then check if there are unidentified characters. If these requirements are not met it should throw out an exception and handle it properly.

10. 3 best sides and 3 weaknesses

The generation algorithm for individual floors works nicely (it produces what it is supposed to produce, perfect mazes) and the solving algorithm works well too (it will find a solution if it exists). The end product is playable, so that is a good thing. A major weakness is the scalability, the size of the labyrinth including number of floors, height and width of the individual floors are hard coded. I'm not sure how to fix this problem, it would be required to at least rethink several methods. The 3D factor is pretty weak in my program. All of my floors are perfect mazes and the goal is located on the highest floor. So the way to complete the game is to go up to the highest floor. If there were "dead areas" it would make the 3D factor more interesting as some areas can only be accessed via particular stairs. Also handling of improper inputs is almost completely ignored in my program.

11. Deviations from the plan, realized process and schedule

The generation algorithm and everything that it required were implemented first. After that the solving algorithm was created. After those I implemented the playability portions of the program and ran it from the console, then I created a UI for it. The process followed my original plan for the most part. My time estimate was completely off, in reality I used a fraction of the estimated time (about 50 hours). The biggest difference is that I lowered my difficulty from moderate to easy because the scalability and GUI would have taken too much time and rethinking as I started working on this project too late.

12. Final evaluation

My project is a generally functional game. It doesn't handle some bad inputs properly and the scalability is poor. Also making extensions and changes is going to be troublesome due to my project's implementation and structure. The generation method could be tweaked to produce

more challenging mazes but that would require a more complex algorithm. Also the solving method could be tweaked to find the shortest possible route to the goal but that too would require a more complex algorithm. In conclusion the solution methods could be better but I can't think of other data structures that I could have used and in general I am happy with my class structure. One of the biggest downfalls is handling of incorrect inputs, especially with the loadGame and readFile methods.

If I would start the project again from the beginning I would start working on it properly earlier. This would give me more time to think of better implementations and solutions to my issues, especially the parts that I hard coded in due to the lack of time. Without the parts that were hard coded in the labyrinths could have a freely determinable size.

13. References

https://stackoverflow.com/questions/29739751/implementing-a-randomly-generated-maze-using-prims-algorithm

https://www.baeldung.com/java-solve-maze

http://www.astrolog.org/labyrnth/algrithm.htm

https://en.wikipedia.org/wiki/Maze_generation_algorithm

14. Appendixes