Labyrinth

Leevi Pulkkinen 910776
Computer-Science 1. year
16.4.2022

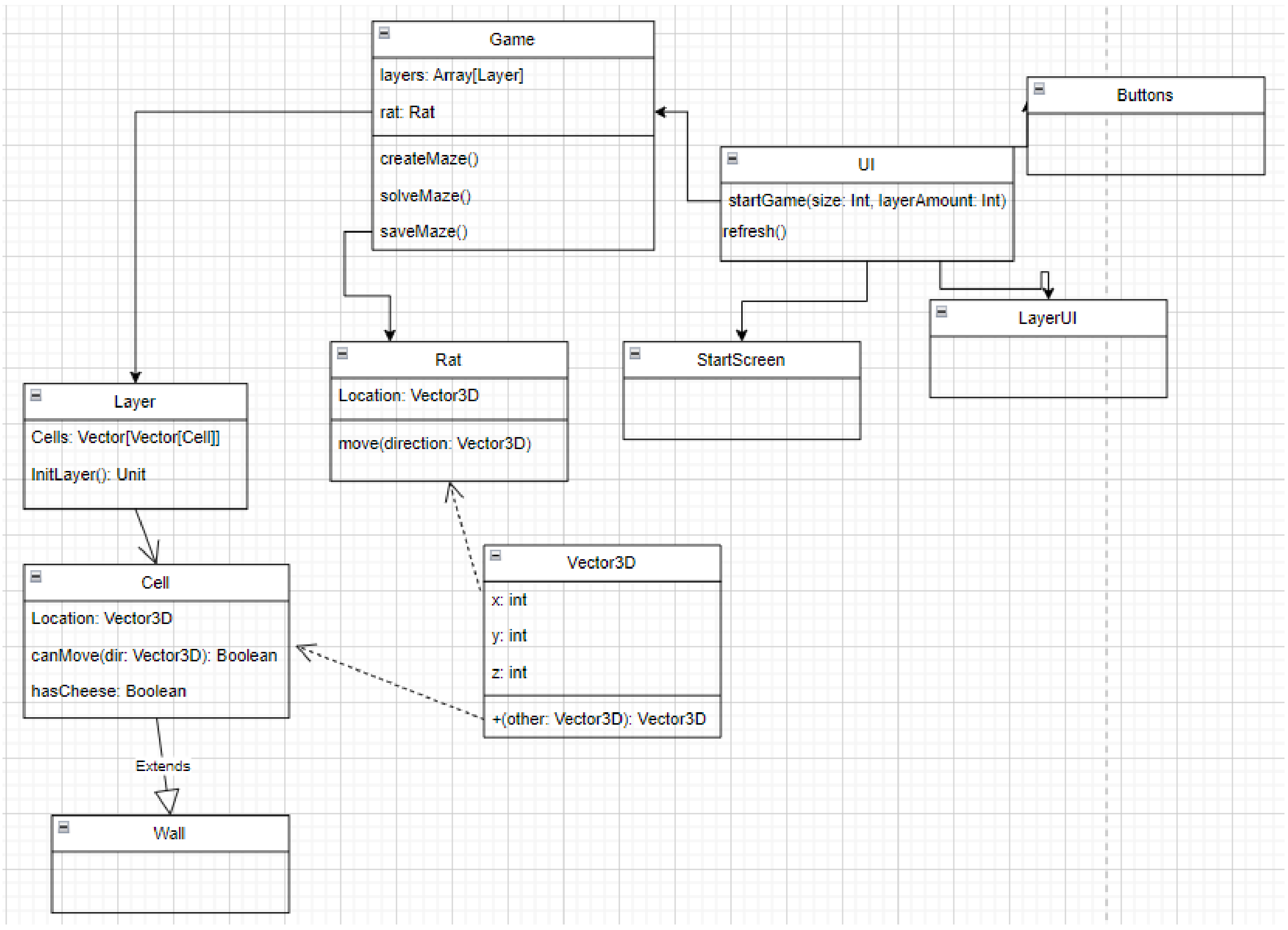
General description

The application created in this project is a labyrinth game. The maze has a rat that can be moved by user input. The aim of the game is to move the rat to the cheese found somewhere in the maze. The maze can have multiple layers that can be moved between using ladders. The size and layer counts are also decided by user input in the start screen that appears when the application is started. The requirements for moderate difficulty are covered fully and some requirements of the demanding difficulty are also covered.

User interface

The application can be started by running the GUI object. When the application is running a start screen appears on the screen. At the start screen the user sees some general instructions for using the app and is also asked to decide the size and the number of layers in the maze. The game can be started by pressing the start game button. When the game starts the user sees a rat in a maze. The rat can be moved using the arrow keys and ladders can be used by pressing the spacebar. If the user doesn't find the cheese and decides to give up, the solution to the maze can be shown by pressing the "solve maze" button. When the rat finds the cheese, the user is asked if they want to play again. If the answer is yes, the user is directed back to the start menu, otherwise the application is stopped. The format of the maze can be saved to a file by pressing the "Save to a file" button.

Program structure



User interface:

Objects GUI, StartScreen, Buttons and the class LayerUI form the user interface of the program.

The GUI object takes care of making the application window, initializing the program, and starting the game. GUI contains the logic that decides what is shown on the screen. GUI has the method startGame that creates a new game.

StartScreen takes care of accepting user input when the app is started and draws sliders and some general instructions to the screen.

When the game is ongoing, LayerUI draws the contents of the current maze layer to the screen, monitors user input from the keyboard and updates the screen. Buttons works together with LayerUI by drawing two buttons to the screen and checking for user input.

Labyrinth components:

Classes Layer and Cell have the job of modeling the labyrinth of the game. A Cell is a single square in the maze that has a location. Main method of the Cell class is the canMove method, that tell whether it is possible to move to certain directions from the cell.

The Layer class creates the required number of cells for a single layer and stores them in a 2D array. The main method of the layer class is the initLayer method because it creates the required cells.

Wall object is a cell that can't be accessed. Walls are only used for error handling purposes and they are not really that important.

Game logic and Labyrinth:

The Game class oversees the game logic and the labyrinth in general. The game class creates the required layers and the rat and keeps track of the current layer that is shown on the screen. Game class implements the maze creating, solving, and saving methods.

The Rat class simply models the rat that is in the maze. The rat can be moved by the move method.

Helper classes and objects:

The Vector3D class is used to model the locations of cells and the rat using coordinates. Directions are also modeled by using the Vector3D class (for example North is Vector3D(0,1,0))

The constants object is used to store values and pictures used in the app. This allows for easy modification of the app because a variable needs to be replaced from one place only.

The program structure was quite natural to implement since the project was built from the bottom up. The structure of the classes modeling the labyrinth (Cell, Layer, Vector3D, Rat) is quite good and I don't think a drastically different solution would have been practical. In hindsight I think the GUI class contains too much logic when it's focus should be on the UI. A clearer separation between the GUI and game class would have been more ideal for the clarity of the project.

Algorithms

The algorithms used in this project are the recursive backtracker algorithm for creating the maze, and the left-hand rule for solving the maze.

The recursive backtracker creates the maze by “carving” a path. The algorithm starts at the top left corner of the maze and continues creating a path at one of the six available directions. Since the rat can move to four directions on a layer and between layers using a ladder, there are a total of six directions the path can be carved to (In the case there is only one layer, only four directions are available.). Every time a new path is carved, the new cell is added to a stack. The algorithm keeps carving the path until the current cell is surrounded by cells that have already been visited. When this situation arises, cells are popped from the stack until a cell is found that has unvisited neighboring cells. If all the cells are popped from the stack, all the cells have been visited. The maze formed by this algorithm is a so-called perfect maze since it has no loops and a single path from one cell to another.

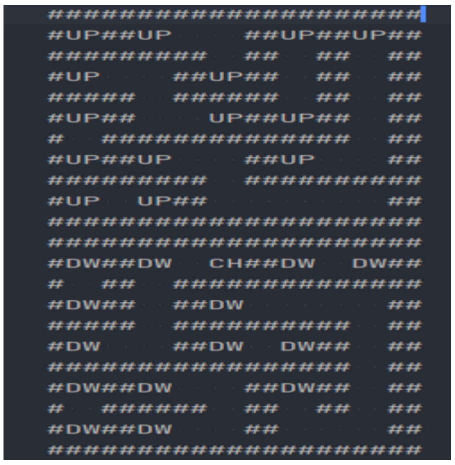
The left-hand rule is a simple but effective algorithm for solving the maze. The algorithm attempts to find a path from the rat's location to the cheese by using the left-hand rule, which means that every time an intersection occurs, a left turn is taken. Usage of this algorithm is possible because the maze has no loops. While attempting to find the path, the algorithm keeps track how many times each cell has been visited. The shortest path to the cheese can be found by following the cells that have been visited once.

Data Structures

Immutable Vector3D class:
Vector3D stores a x, y and z coordinates and supports some simple calculations between Vector3D objects such as addition. Vector3D is used to model the locations of cells and the rat. Vector3D is easy to use and is preferred instead of using plain integers for coordinates, because all three of the coordinates are now stored in a single place.

Cells in a layer are stored in a 2-dimensional Vector, and the whole labyrinth is stored as an array of layers.

Files and internet access



```
#####################
#UP##UP    ##UP##UP##
#########  ##  ##  ##
#UP    ##UP##  ##  ##
#####  ######  ##  ##
#UP##    UP##UP##  ##
#  ##############  ##
#UP##UP    ##UP    ##
#########  ##########
#UP  UP##          ##
#####################
#####################
#DW##DW  CH##DW  DW##
#  ##  ##############
#DW##  ##DW        ##
#####  ##########  ##
#DW    ##DW  DW##  ##
#################  ##
#DW##DW    ##DW##  ##
#  ######  ##  ##  ##
#DW##DW    ##      ##
#####################
```

The maze structure can be saved to a file. The structure is saved in the following format:

# => wall

" " => space that the rat can move to

CH => cheese

UP => ladder up

DW => ladder down

Graphics of the game are png files included in the pictures folder of the project.

Testing

methods of the program were tested mostly by giving example input in REPL and viewing if it amounted to desired behavior. A lot of testing in the latter parts of the project were done visually, for example just playing the game and moving the rat in the maze when building the solving algorithm. No unit tests were made.

Known bugs and missing features

A bug that seems to appear sometimes is the game over notification appearing twice when the cheese is found. I have not been able to replicate this bug consistently. The maze structure can't be saved to a file, after the cheese has been found, and this could be considered a missing feature.

3 best sides and 3 weaknesses

pros:

Playability: The game is smooth to play, has basically no loading times and can be played again after completion easily.

Algorithms: I think the algorithms are the coolest and most interesting part of this project.

Overall code quality: I think the overall code quality and clarity is on a good level.

cons:

The start screen graphics: The start screen is quite rough, and that is because its appearance was not a priority in this project.

Method clarity: All the methods of the program are fully functional and offer the needed functionality, but I believe most of them could have been implemented in more "elegant" ways. However, I did not think major refactoring was necessary for the purposes of this project.

Repetition of code: Some parts of the code are quite repetitive, but I could not come up with a better solution for these parts.

Deviations from the plan, realized process and schedule

The project was mostly made according to the project plan. The project was built from the bottom up, starting with Vector3D, Cell and Layer. Next a basic user interface was implemented with GUI and the rat was also added. After that maze creation and rat moving logic was added. General game logic, additional UI features, maze solving, and saving were added last. Majority of the time used to work with the project was used in building graphical features and the game logic. The amount of time needed to implement the algorithms was overestimated in the plan.

Final evaluation

Overall, I think this project has succeeded and covered my expectations for the quality of the project. This application is simple, easy to use, smooth and has some graphics that bring more life to the game. The coding process has been relatively straight forward and rewarding. Obviously, this project is far from perfect in all aspects, for example code clarity and efficiency could always be improved. This app could be improved by adding new features, such as a scoring system. I think data and class structures are planned correctly, however their implementation in practice is not completely polished. I think the program is easy to scale and new features could be added quite easily. This is true especially for the maze generating and solving algorithms, that are implemented in their own methods. If I started this project from the beginning, I wouldn't make any drastic changes to the process. However, it would be beneficial to take some time to think about the implementation of "simple" methods, instead of making them in a way that first comes to mind.

References

https://opengameart.org/content/wall-texture-walltexturepng

https://www.youtube.com/watch?v=Kmgo00avvEw&t=10s

https://opengameart.org/content/pack-of-ladders

https://opengameart.org/content/lab-rat-labyrinth

https://papunet.net/materiaalia/kuvapankki/kuvat/juusto-0

https://opengameart.org/content/molten-metal-texture

https://www.astrolog.org/labyrnth.htm

https://www.youtube.com/watch?v=Y37-gB83HKE&t=1304s

https://www.scala-lang.org/api/2.12.5/scala-swing/scala/swing/