INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
LYON

LABYRINTH GAME

membre de
UNIVERSITÉ DE LYON

INSA

Contents

ABSTRACT

The main goal of our project is to create a labyrinth game with different mazes depending on the difficulties with containing a small variety of characters. To do so, we'll create various classes which will initialize different elements such as maze, display and characters.

The program will start by asking players to choose one level of difficulty and one character that they want to play with. Later we also decided to add a visibility range and different wall textures for the user to choose from.

Then once the game stars, the player will be put inside of a maze with a pre-defined radius of visibility. The main goal of the player is to find the exit as fast as possible. Higher difficulties will be much more complicated to solve and the lower ones will be much easier.

If the player manages to solve the maze, than, when he gets at the exit, he will be automatically offered the chance to play again by being shown the first display, where he can readjust all the settings.

DESCRIPTION OF PROJECT

Our main objective is to create a game which will contain different mazes with respect to different levels of difficulties. The difficulty (from 1 to 5) and the help (from 1 to 3) will be chosen by the player before the game starts. The higher the difficulty is the bigger the maze will be and help determines the radius that the player will be able to see around his/her character with 1 being the smallest radius and 3 being the biggest radius.

First of all, we create the class **Maze** which will generate the maze. Depending on the chosen difficulty by the player the number of white blocks and the complexity of the answer will change. The solution of the maze will be written as a method in this class.

Then, we create the class **Player** which will initialize the characters that will be used in the game. Moreover this class includes the method which changes the image used in the game with respect to the pressed button. Meaning that if a character is facing right and the player presses "A" then the character will face left. In order to realise this we used for different images for every character available in the game. Also we used the top view of images.

Next, the class **DisplayFilter** is where we define the method which allows the player to see the blocks around his/hers character. Furthermore, the player will choose the radius of visibility that he/she wants before the game stars.

Later on, the class **LabyrinthDisplay** is where we will be displaying the labyrinth with all of its components such as the walls, the character inside the maze, the path, background etc. Also, this class will check rather a player can make a move or not. Meaning that if the character of the player is blocked by a wall at right when player presses "D" this class will check if there is a white block where the player wants to move if yes his/hers character will move if not it won't make any move.

Finally, the class **FirstFrame** will initialize the page where the player can choose his/her character, the difficulty that he/she wants to play with and the help that he/she wants.

Of course since we'll be dealing with IHM we used some images for characters and maze. Every image that we used is designed and uploaded by ourselves.

PRINCIPLE OF ALGORITHM

1) DisplayFilter:

This object creates a Vignette in the format of a BufferedImage, which is used by the game in order to hide portions of the game which are further away than three blocks from the player.

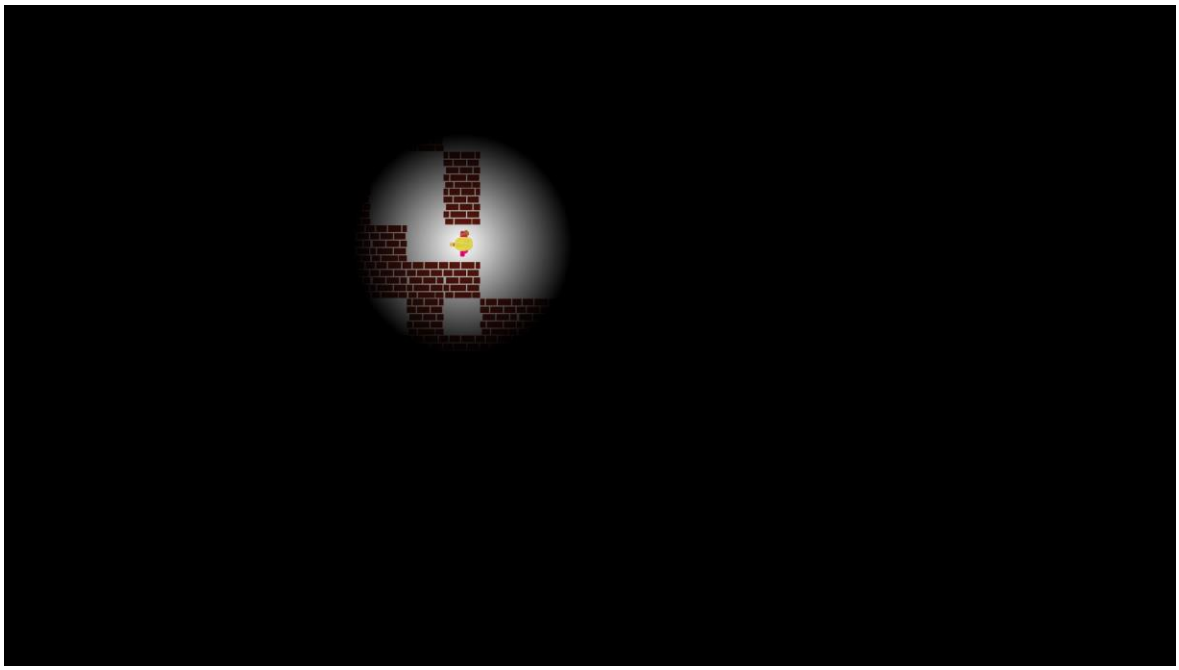
Since the algorithm is of complexity$\boldsymbol{O(n^2)}$, it is therefore only computed once and reused in the game. So that it is used once, the complexity increases. Nevertheless since it is only computed once in the beginning of the game, it does not cause any major inconveniences.

The final complexity is a function of the screen resolution **(n * m)**, expressed in screen width and screen height respectively i.e.

$$\boldsymbol{O(4 * n * m)}$$

The gradient is done linearly using the Euclidean distance and thus having a soft impairing of the view. The pixel is expressed in coordinates **(x,y)** which are given as integers. The distance is measured from the center of the screen. And since the final image is twice the final resolution, we have the distance as the following:

$$\sqrt{((\boldsymbol{n-x})^2-(\boldsymbol{m-y})^2)}$$



2) Labyrinth:

This is class where every other class is regrouped. First of all, this class makes every other class visible so that the game can be played. Secondly, the **main**() method which is found inside of this class initializes the game. Then, once the game starts with the help of various sliders and selection menus will choose a level of difficulty between 1-5 and a character between Girl, Boy or Monster(Godzilla), the visibility range, and wall texture.

3) LabyrinthDisplay:

This is the core of the game. It manipulates objects such as Player, Maze and DistanceFilter in order to achieve the final result.

The object is started using various constructors. The full featured constructor has input variables for game settings i.e. "**LabyrinthDisplay**(**int** difficulty, **int** character, **int** wall, **int** visibility)" , where difficulty would set the size of the maze, with a range of 1-5 (5 being the biggest most complex maze, and 1 the simplest and smallest one).

- Character would be an **int** from 1 to 3 corresponding to the various characters i.e. Boy, Girl, Godzilla respectively. If a different value is attributed, the game with automatically choose character 1.
- Wall has **int** values from 1 to 5. Each being a different wall texture. If a number out of the range is given, the game chooses wall 1 automatically.

- Visibility accepts an **int**, which is equivalent to the number of squares visible from your character. Technically there is no limit for the **int** accepted. But by game design it should be constricted from 1 to 3. Where 3 is maximum visibility corresponding of a visibility radius of 3 blocks.

The game sets the title as "LABYRINTH" and the program icon "Labyrinth.png".

Note: All images used in the game are custom, made by us.

The game has dynamic resolution, which is adapted to your screen. Due to the design of the game, it is restricted to standard **16:9** aspect ratio screens. Running the game on a different screen can lead to aesthetic deformations and other bugs. The refresh rate of the game is set to half the refresh rate of the screen, to reduce tear, and have a fast response to the game.

For elegance reasons, the mouse cursor and the header are both removed from the screen, resulting in a full-screen game.

The game has 6 buttons attributed to the game-play:

- **ESC** – Exits the game
- **ENTER** – Removes the vignette, causing full visibility of the maze and showing the solution from the players' current standpoint.
- **W/UP** – Both buttons can be used to move the character up (if possible)
- **S/DOWN** – Both buttons can be used to move the character down (if possible)
- **A/LEFT** – Both buttons can be used to move the character left (if possible)
- **D/RIGHT** - Both buttons can be used to move the character right (if possible)

When the keys are pressed, they are stored in a Boolean array, which gets executed the next runtime of the game. This way lag is prevented from Event interference.

Two Events are implemented; ActionListener for the display, and KeyListener for the buttons.

Every frame of the game which is caused by "**actionPerformed**(Action Event e)" updates the time, moves the player if and only if the user requested a move, renders the new display and then paints it. This way, the game is highly responsive, with no lags or screen flickers and drops.

The render makes its processes in layers. First drawing the maze (walls, paths, solutions if enter was pressed), then the Character is superposed to its coordinates. In the end the Vignette (Visibility constraint) is superposed to the Characters position to have the desired visibility effect. If the character has won, a massive red text "You Won!" is displayed in the middle of the screen.

The "**move2**()" method (optimized version of "**move**()" which uses a very complicated system) applies a pressed button if it is possible. The checking of the possibility is done by "**canMove**(**int**, **int**)". In the end, the keyPressed array of the buttons is reset to false.

"**canMove**(**int** x, **int** y)" checks if the player can move to a requested location by converting the pixel coordinates x and y into the integer indices of the Boolean maze MAP, and it returns a true Boolean if movement is possible.

Since a lot of libraries are needed for this game, only the specific ones are used, in order to reduce game load time in the beginning and prevent potential conflicts.

4) Player:

This is a small object which contains basic coordinates and images. Every time the player is moved, the coordinates are updated accordingly.

It selects a character from a list of 3 options, as requested by the user. Each character has 4 images corresponding to right, left, up, down.

This is done so that complex linear algebra formulas are not performed which tend to result in high computational power being wasted in every move, and thus could lead into frame drops.

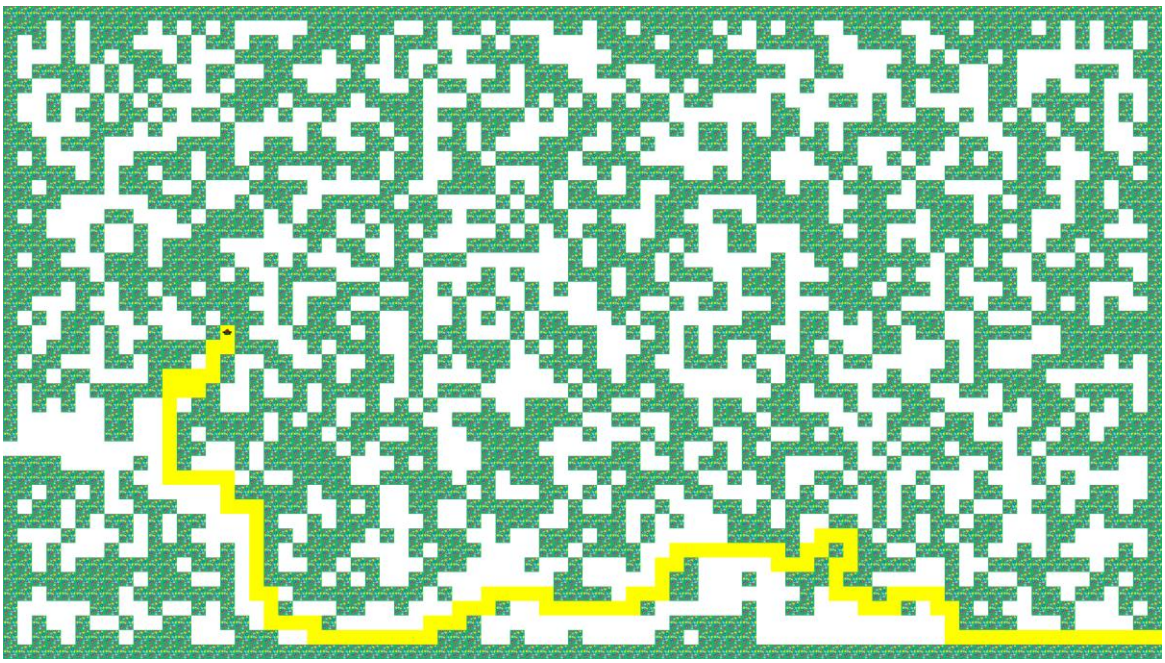
Every time the player is moved. The direction variable is also modified depending on the users' last click. From this variable, the image to be displaced is determined using the **setCurrentImage**() method.

5) Maze:

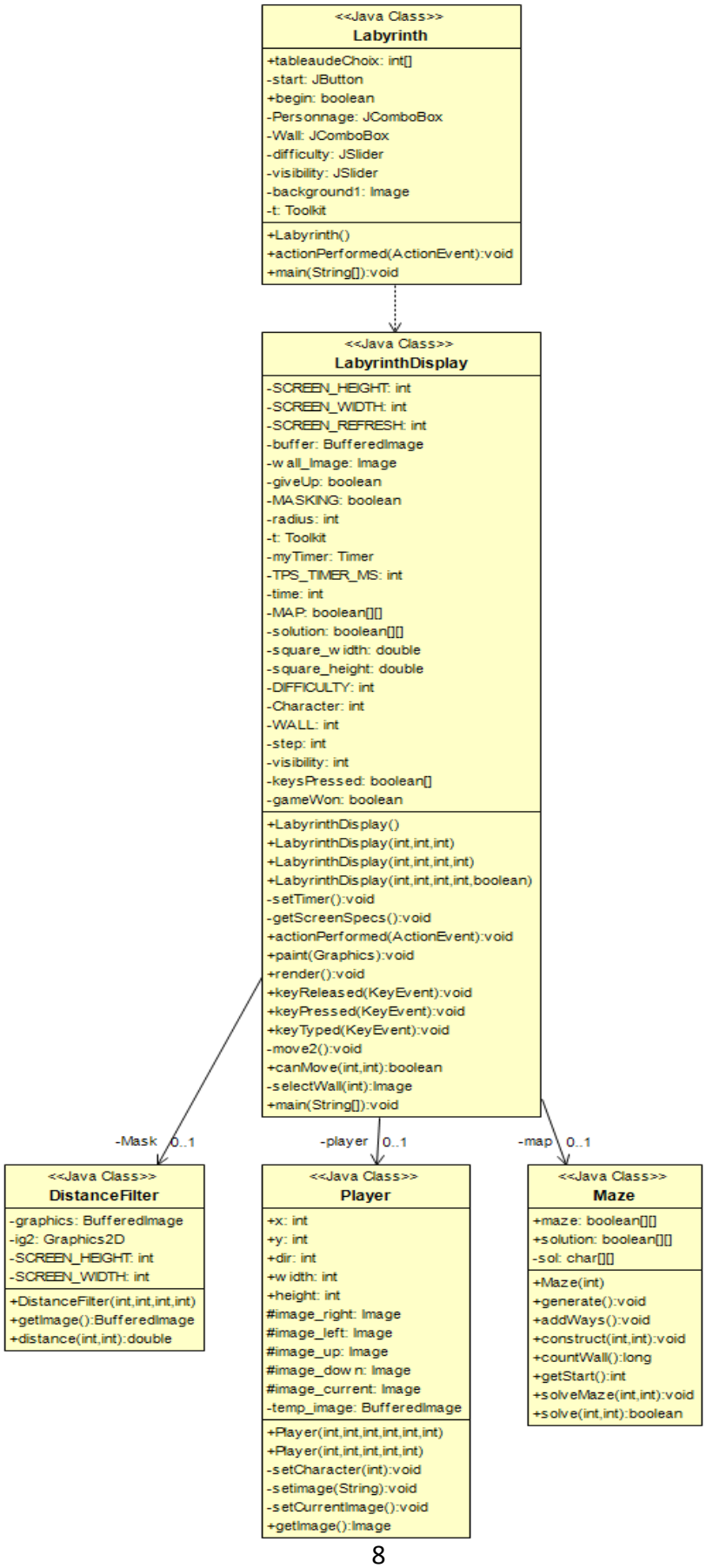
This is the class where we create the surrounding walls of the maze and also the entrance of the maze.

The walls which are found inside of the maze depend on the difficulty that will be chosen by the player before the game starts. **addWays**() is the method which controls the density of the ways and furthermore it also adds some random spots throughout the maze.

Moreover, the solution of the game no matter where the player decides to bail is shown to the player thanks to **solve**(**int** x, **int** y) method which uses the backtracking. This method is initialised by the method **solveMaze**(**int** xx, **int** yy).



STRUCTURE


<<Java Class>>
Labyrinth
+tableaudeChoix: int[]
-start: JButton
+begin: boolean
-Personnage: JComboBox
-Wall: JComboBox
-difficulty: JSlider
-visibility: JSlider
-background1: Image
-t: Toolkit
+Labyrinth()
+actionPerformed(ActionEvent):void
+main(String[]):void
<<Java Class>>
LabyrinthDisplay
-SCREEN_HEIGHT: int
-SCREEN_WIDTH: int
-SCREEN_REFRESH: int
-buffer: BufferedImage
-wall_Image: Image
-giveUp: boolean
-MASKING: boolean
-radius: int
-t: Toolkit
-myTimer: Timer
-TPS_TIMER_MS: int
-time: int
-MAP: boolean[][]
-solution: boolean[][]
-square_width: double
-square_height: double
-DIFFICULTY: int
-Character: int
-WALL: int
-step: int
-visibility: int
-keysPressed: boolean[]
-gameWon: boolean
+LabyrinthDisplay()
+LabyrinthDisplay(int,int,int)
+LabyrinthDisplay(int,int,int,int)
+LabyrinthDisplay(int,int,int,int,boolean)
-setTimer():void
-getScreenSpecs():void
+actionPerformed(ActionEvent):void
+paint(Graphics):void
+render():void
+keyReleased(KeyEvent):void
+keyPressed(KeyEvent):void
+keyTyped(KeyEvent):void
-move2():void
+canMove(int,int):boolean
-selectWall(int):Image
+main(String[]):void
-Mask
0..1
-player
0..1
-map
0..1
<<Java Class>>
DistanceFilter
-graphics: BufferedImage
-ig2: Graphics2D
-SCREEN_HEIGHT: int
-SCREEN_WIDTH: int
+DistanceFilter(int,int,int,int)
+getImage():BufferedImage
+distance(int,int):double
<<Java Class>>
Player
+x: int
+y: int
+dir: int
+width: int
+height: int
#image_right: Image
#image_left: Image
#image_up: Image
#image_down: Image
#image_current: Image
-temp_image: BufferedImage
+Player(int,int,int,int,int,int)
+Player(int,int,int,int,int)
-setCharacter(int):void
-setImage(String):void
-setCurrentImage():void
+getImage():Image
<<Java Class>>
Maze
+maze: boolean[][]
+solution: boolean[][]
-sol: char[][]
+Maze(int)
+generate():void
+addWays():void
+construct(int,int):void
+countWall():long
+getStart():int
+solveMaze(int,int):void
+solve(int,int):boolean

PROGRAM FILES

The final game contains 29 items including this report.

Java Objects

- Labyrinth.java
- LabyrinthDisplay.java
- Maze.java
- Player.java
- DistanceFilter.java

Pictures

- Cadre.png
- Character1_Right.png
- Character1_Left.png
- Character1_Up.png
- Character1_Down.png
- Character2_Right.png
- Character2_Left.png
- Character2_Up.png
- Character2_Down.png
- Character3_Right.png
- Character3_Left.png
- Character3_Up.png
- Character3_Down.png
- Lab_Background_Pic.jpg
- Labyrinth.png
- Wall1.png
- Wall2.png
- Wall3.png
- Wall4.png
- Wall5.png

Others

- Labyrinth.jar
- LabyrinthGame.pdf
- LabyrinthGame.png
- README.md