/\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bailey Thompson

\* Game Of Life (1.1)

\* 16 September 2016

\* Game Rules: Any live cell with fewer than two live neighbours dies, as if caused by under-population.

\* Game Rules: Any live cell with two or three live neighbours lives on to the next generation.

\* Game Rules: Any live cell with more than three live neighbours dies, as if by over-population.

\* Game Rules: Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

\* Features: User has ability to pause and resume the game at any time. At any time, user can click on the game board, and the cell

\* Features: will become the opposite of what it currently is. If it was dead, it will become alive, if it was alive, it will

\* Features: become dead. Change in cell state will have effect on game logic like any normal cell would. User has the ability to

\* Features: clear the game board at any time, making all cells dead. User has the ability to randomize the amount of live cells,

\* Features: portion of live cells that should appear on the board. User has the ability to change the amount of time between

\* Features: turns, having an immediate effect on game logic. User also has the ability to change row size and column size at any

\* Features: time. If a selection pane other than the main game is closed, the pane acts as if the user did not enter anything,

\* Features: by entering the value that the user previously put. All user preferences are saved with file io so that the next time

\* Features: the user opens the game, previous settings are used. The main game frame cannot be resized by sketching the screen,

\* Features: unless amount of rows or columns in being changed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//declaring package

package gameoflife;

//declaring imports

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.io.BufferedOutputStream;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import static java.lang.Integer.parseInt;

import java.nio.file.FileAlreadyExistsException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import static java.nio.file.StandardOpenOption.TRUNCATE\_EXISTING;

import static java.nio.file.StandardOpenOption.WRITE;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.WindowConstants;

//declaring class

public class GameOfLife {

//declaring name of path file

Path file = Paths.get("GameOfLife.txt");

//declaring String variables for the displayed board of the main game frame, the temporary user input variable, and for what is saved with file io

String lifeBoard, tempSize = "", saveFile;

//declaring String array used for file io

String[] split;

//variables for what the user can choose from various settings

int maxVertical, maxHorizontal, screenWidth, screenHeight, roundTime, randomSpawning, timeCounter;

//declaring cells, both the displayed ones, and the temporary ones that are written over to displayed ones after a full turn

boolean[][] cells, tempCells;

//declaring various variables for GUI

private JFrame frame;

private JPanel middlePanel, bottomPanel;

private JLabel label;

private JButton btnPause, btnPlay, btnRandom, btnClear, btnTime, btnColumn, btnRow, btnExit;

//boolean variables for when user pauses game, when the label is being clicked on, and when the label is being edited

boolean pause, editPress, changingLabel;

//declaring main method

public static void main(String[] args) {

//sending to GameOfLife method

GameOfLife GameOfLife = new GameOfLife();

GameOfLife.GameOfLife();

}

//declaring private void method for the game setup and game logic

private void GameOfLife() {

//checking the monitor dimensions

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

//setting the maximum screen higth to the dimensions minus a little bit

screenWidth = (int) screenSize.getWidth() - 100;

screenHeight = (int) screenSize.getHeight() - 100;

//if the monitor cannot display the minimum amount of columns and rows for the game to run, the user is notified

if (((screenWidth - 10) / 14 - 1) < 25 || ((screenHeight - 70) / 16 - 1) < 25) {

String[] buttonGameMode = {"Ok"};

JOptionPane.showOptionDialog(null, "Sorry, your screen size is too small for this program to run.", "Game Of Life", JOptionPane.PLAIN\_MESSAGE, JOptionPane.PLAIN\_MESSAGE, null, buttonGameMode, buttonGameMode[0]);

System.exit(0);

}

//method Load is initialized

Load();

//the size of the cells and tempCells is set

cells = new boolean[maxVertical + 1][maxHorizontal + 1];

tempCells = new boolean[maxVertical + 1][maxHorizontal + 1];

//method FillInSave is initialized

FillInLoad();

//method FillInArray is initialized

FillInArray();

//method PrepareGUI is initialized

PrepareGUI();

while (1 != 0) {

//checking if pause is false

if (pause == false) {

//preventing automatic editing of label if it is being changed from a mouse click; until editPress is set to false again, the code is basically paused

while (editPress == true) {

try {

Thread.sleep(1);

} catch (InterruptedException ex) {

Logger.getLogger(GameOfLife.class.getName()).log(Level.SEVERE, null, ex);

}

}

//sending to method FillingAndChangingArray

FillingAndChangingArray();

//instead of using the sleep amount for the round time that the user enters, a 1 milisecond sleep thread is set the amount of times that the user specifies with round time,

//the sleep is only executed if pause is false, this is so that if pause becomes true, it stops and does not continue sleeping

for (timeCounter = 0; timeCounter < roundTime; timeCounter++) {

if (pause == false) {

try {

Thread.sleep(1);

} catch (InterruptedException ex) {

Logger.getLogger(GameOfLife.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

//method InnerGameLogic is initialized only if pause is false and the main game frame is showing

if (pause == false && frame.isShowing() == true) {

InnerGameLogic();

}

//saving the game by sending to method Save

Save();

//the button btnPlay listens only if pause is true, and sets pause to false if it is clicked

} else {

btnPlay.addActionListener((ActionEvent e) -> {

//just here to re-initialized the btnplay listener in the method PrepareGUI

});

}

}

}

//declaring private void method for the game logic

private void InnerGameLogic() {

//executes a loop in a loop to fill the 2d array

for (int vertical = 0; vertical < maxVertical + 1; vertical++) {

for (int horizontal = 0; horizontal < maxHorizontal + 1; horizontal++) {

//sets the temp number to 0

int temp = 0;

//checks all 8 neighbours if it is not a side or corner cell

if (vertical - 1 >= 0 && horizontal - 1 >= 0 && vertical + 1 <= maxVertical && horizontal + 1 <= maxHorizontal) {

if (cells[vertical - 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal + 1] == true) {

temp += 1;

}

//checks 5 neighbours if it is a side cell

} else if (vertical - 1 < 0 && vertical + 1 <= maxVertical && horizontal - 1 >= 0 && horizontal + 1 <= maxHorizontal) {

if (cells[vertical][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal + 1] == true) {

temp += 1;

}

//checks 5 neighbours if it is a side cell

} else if (vertical + 1 > maxVertical && vertical - 1 >= 0 && horizontal - 1 >= 0 && horizontal + 1 <= maxHorizontal) {

if (cells[vertical - 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal + 1] == true) {

temp += 1;

}

//checks 5 neighbours if it is a side cell

} else if (horizontal - 1 < 0 && horizontal + 1 <= maxHorizontal && vertical - 1 >= 0 && vertical + 1 <= maxVertical) {

if (cells[vertical - 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal + 1] == true) {

temp += 1;

}

//checks 5 neighbours if it is a side cell

} else if (horizontal + 1 > maxHorizontal && horizontal - 1 >= 0 && vertical - 1 >= 0 && vertical + 1 <= maxVertical) {

if (cells[vertical - 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal] == true) {

temp += 1;

}

//checks 3 neighbours if it is a corner cell

} else if (vertical - 1 < 0 && horizontal - 1 < 0 && horizontal + 1 <= maxHorizontal && vertical + 1 <= maxVertical) {

if (cells[vertical][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal + 1] == true) {

temp += 1;

}

//checks 3 neighbours if it is a corner cell

} else if (vertical - 1 < 0 && horizontal + 1 > maxHorizontal && horizontal - 1 >= 0 && vertical + 1 <= maxVertical) {

if (cells[vertical][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical + 1][horizontal] == true) {

temp += 1;

}

//checks 3 neighbours if it is a corner cell

} else if (vertical + 1 > maxVertical && horizontal - 1 < 0 && horizontal + 1 <= maxHorizontal && vertical - 1 >= 0) {

if (cells[vertical - 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal + 1] == true) {

temp += 1;

}

if (cells[vertical][horizontal + 1] == true) {

temp += 1;

}

//checks 3 neighbours if it is a corner cell

} else if (vertical + 1 > maxVertical && horizontal + 1 > maxHorizontal && horizontal - 1 >= 0 && vertical - 1 >= 0) {

if (cells[vertical - 1][horizontal - 1] == true) {

temp += 1;

}

if (cells[vertical - 1][horizontal] == true) {

temp += 1;

}

if (cells[vertical][horizontal - 1] == true) {

temp += 1;

}

}

//if the cells at the position is alive and has more than 3 neighbours or less than 2, it dies but is temporairly stored on tempCells

if (cells[vertical][horizontal] == true && (temp < 2 || temp > 3)) {

tempCells[vertical][horizontal] = false;

//if the cells at the position is dead but has 3 neighbours, it becomes alive but is temporairly stored on tempCells

} else if (cells[vertical][horizontal] == false && temp == 3) {

tempCells[vertical][horizontal] = true;

//if the cells at the position is alive and has 2 or 3 neighbours, it stays alive, and is temporairly stored on tempCells

} else if (cells[vertical][horizontal] == true && (temp == 2 || temp == 3)) {

tempCells[vertical][horizontal] = true;

}

}

}

//after the whole 2d array is set onto the tempCells, the tempCells is writen onto the cells

for (int vertical = 0; vertical < maxVertical + 1; vertical++) {

System.arraycopy(tempCells[vertical], 0, cells[vertical], 0, maxHorizontal + 1);

}

}

//declaring private void method for filling in the 2d array

private void FillInArray() {

//setting changingLabel to true

changingLabel = true;

//sets the font size of the display

lifeBoard = "<html><span style='font-size:1em'>";

//loop in a loop to set every cell

for (int vertical = 0; vertical <= maxVertical; vertical++) {

for (int horizontal = 0; horizontal <= maxHorizontal; horizontal++) {

//display wether the cell is alive or dead

if (cells[vertical][horizontal] == false) {

lifeBoard += "□ ";

} else if (cells[vertical][horizontal] == true) {

lifeBoard += "■ ";

}

}

//skip a line every time a full horizontal row is displayed

lifeBoard += "<br>";

}

//setting changingLabel to false

changingLabel = false;

}

//declaring private void method for setting GUI and action listeners

private void PrepareGUI() {

//setting the frame title

if (pause == false) {

frame = new JFrame("Game Of Life");

} else {

frame = new JFrame("Game Of Life (Paused)");

}

//setting the frame layout

frame.setLayout(new BorderLayout());

//setting the frame size

frame.setSize(14 \* (maxHorizontal + 1) + 5, 16 \* (maxVertical + 1) + 95);

//making the frame non-resizable

frame.setResizable(false);

//creating a label with text as lifeBoard variable

label = new JLabel(lifeBoard, JLabel.CENTER);

//setting the first row of buttons

middlePanel = new JPanel();

//setting the second row of buttons

bottomPanel = new JPanel();

//setting buttons and what is displayed on them

btnPause = new JButton("Pause");

btnPlay = new JButton("Play");

btnRandom = new JButton("Random");

btnClear = new JButton("Clear");

btnTime = new JButton("Time");

btnColumn = new JButton("Column");

btnRow = new JButton("Row");

btnExit = new JButton("Exit");

//setting the layout of both rows of buttons

middlePanel.setLayout(new FlowLayout(FlowLayout.CENTER));

bottomPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

//setting upper row of buttons to variable middlePanel

middlePanel.add(btnPause);

middlePanel.add(btnPlay);

middlePanel.add(btnRandom);

middlePanel.add(btnClear);

//setting upper row of buttons to variable bottomPanel

bottomPanel.add(btnTime);

bottomPanel.add(btnColumn);

bottomPanel.add(btnRow);

bottomPanel.add(btnExit);

//setting various parts of the frame

frame.add(label, BorderLayout.NORTH);

frame.add(middlePanel, BorderLayout.CENTER);

frame.add(bottomPanel, BorderLayout.SOUTH);

//setting the frame to display in the middle of the monitor

frame.setLocationRelativeTo(null);

//setting the frame to close when the x button is pressed

frame.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

//setting the frame to visible

frame.setVisible(true);

//setting what happens when user clicks on the label of the frame

label.addMouseListener(new MouseListener() {

@Override

public void mouseClicked(MouseEvent e) {

//position of click is set to variable

int horizontalClickPosition = (int) Math.floor((e.getX()) / 14);

int verticalClickPosition = (int) Math.floor((e.getY()) / 16);

//prevent automatic write over of label while it is being edited from here

editPress = true;

//prevent glitching out of label when clicking buttons

if (verticalClickPosition <= maxVertical && horizontalClickPosition <= maxHorizontal) {

//flips the value of both 2d array, if true, becomes false; if false, becomes true

cells[verticalClickPosition][horizontalClickPosition] ^= true;

tempCells[verticalClickPosition][horizontalClickPosition] ^= true;

}

//sending to method FillingAndChangingArray

FillingAndChangingArray();

//enableing label editing

editPress = false;

//sending to method Save

Save();

}

//the next four mouse events are not used for anything, but must be included

@Override

public void mousePressed(MouseEvent e) {

//useless

}

@Override

public void mouseReleased(MouseEvent e) {

//useless

}

@Override

public void mouseEntered(MouseEvent e) {

//useless

}

@Override

public void mouseExited(MouseEvent e) {

//useless

}

});

//setting what happens when user clicks on pause button

btnPause.addActionListener((ActionEvent e) -> {

//sets pause to true

pause = true;

//resets the title of the frame so it says paused

frame.setTitle("Game Of Life (Paused)");

});

btnPlay.addActionListener((ActionEvent e) -> {

//setting pause to false

pause = false;

//setting the title of the frame

frame.setTitle("Game Of Life");

});

//setting what happens when user clicks on random button

btnRandom.addActionListener((ActionEvent e) -> {

boolean skip = false;

//the loop is executed and re-executed until conditions are met

do {

tempSize = JOptionPane.showInputDialog(null, "How much of the time out of 100 should a cell spawn alive, as an integer value?", "Game Of Life", JOptionPane.PLAIN\_MESSAGE);

//if user pressed cancel or the x button, the previous value is used

if (tempSize == null) {

tempSize = String.valueOf(randomSpawning);

skip = true;

}

} while ("".equals(tempSize) || !"".equals(tempSize.replaceAll("[0123456789]", "")) || tempSize.length() > 9 || parseInt(tempSize, 10) > 100);

if (skip == false) {

//randomSpawning variable is set

randomSpawning = parseInt(tempSize, 10);

//sending to the method RandomSpawns

RandomSpawns();

}

});

//setting what happens when user clicks on clear button

btnClear.addActionListener((ActionEvent e) -> {

//setting randomSpawning to 0 so that the board clears

randomSpawning = 0;

//sending to the method RandomSpawns

RandomSpawns();

});

//setting what happens when user clicks on time button

btnTime.addActionListener((ActionEvent e) -> {

//sets tempPause to what pause is set to

boolean tempPause = pause;

//sets pause to true

pause = true;

//resets the title of the frame so it says paused

frame.setTitle("Game Of Life (Paused)");

//the loop is executed and re-executed until conditions are met

do {

tempSize = JOptionPane.showInputDialog(null, "Please insert the amount of miliseconds\nper turn as an integer value.\n1000 miliseconds = 1 second.", "Game Of Life", JOptionPane.PLAIN\_MESSAGE);

//if user pressed cancel or the x button, the previous value is used

if (tempSize == null) {

tempSize = String.valueOf(roundTime);

}

} while ("".equals(tempSize) || !"".equals(tempSize.replaceAll("[0123456789]", "")) || tempSize.length() > 9 || parseInt(tempSize, 10) == 0);

//roundTime variable is set

roundTime = parseInt(tempSize, 10);

//sending to the method Save

Save();

//setting the title of the frame if it is not paused

if (tempPause == false) {

frame.setTitle("Game Of Life");

}

//sets pause back to the original setting

pause = tempPause;

});

//setting what happens when user clicks on column button

btnColumn.addActionListener((ActionEvent e) -> {

//the frame is set to invisible

frame.setVisible(false);

//the loop is executed and re-executed until conditions are met

do {

tempSize = JOptionPane.showInputDialog(null, "Please insert the column size as an integer value.\nMust be between 25 and maximum size allowed on your monitor.", "Game Of Life", JOptionPane.PLAIN\_MESSAGE);

//if user pressed cancel or the x button, the previous value is used

if (tempSize == null) {

tempSize = String.valueOf(maxVertical + 1);

}

} while ("".equals(tempSize) || !"".equals(tempSize.replaceAll("[0123456789]", "")) || tempSize.length() > 9 || parseInt(tempSize, 10) < 25 || parseInt(tempSize, 10) > (screenHeight - 95) / 16 - 1);

//maxVertical variable is set

maxVertical = parseInt(tempSize, 10) - 1;

//sending to method Resize

Resize();

});

//setting what happens when user clicks on row button

btnRow.addActionListener((ActionEvent e) -> {

//the frame is set to invisible

frame.setVisible(false);

//the loop is executed and re-executed until conditions are met

do {

tempSize = JOptionPane.showInputDialog(null, "Please insert the row size as an integer value.\nMust be between 25 and maximum size allowed on your monitor.", "Game Of Life", JOptionPane.PLAIN\_MESSAGE);

//if user pressed cancel or the x button, the previous value is used

if (tempSize == null) {

tempSize = String.valueOf(maxHorizontal + 1);

}

} while ("".equals(tempSize) || !"".equals(tempSize.replaceAll("[0123456789]", "")) || tempSize.length() > 9 || parseInt(tempSize, 10) < 25 || parseInt(tempSize, 10) > (screenWidth - 5) / 14 - 1);

//maxHorizontal variable is set

maxHorizontal = parseInt(tempSize, 10) - 1;

//sending to method Resize

Resize();

});

//setting what happens when user clicks on exit button

btnExit.addActionListener((ActionEvent e) -> {

//exiting the program

System.exit(0);

});

}

//declaring private void method for resizing both 2d arrays

private void Resize() {

//setting the size of both 2d arrays

cells = new boolean[maxVertical + 1][maxHorizontal + 1];

tempCells = new boolean[maxVertical + 1][maxHorizontal + 1];

//sending to the method Save

Save();

//setting the size of the frame

frame.setSize(14 \* (maxHorizontal + 1) + 5, 16 \* (maxVertical + 1) + 95);

//sending to method FillingAndChangingArray

FillingAndChangingArray();

//making the frame visible again

frame.setVisible(true);

//centering the frame in the middle of the monitor

frame.setLocationRelativeTo(null);

}

//declaring private void method for randomly spawning live cells to 2d array

private void RandomSpawns() {

//declaring variable

int randomRandomSpawns;

//loop in a loop used to fill 2d arrays

for (int vertical = 0; vertical < maxVertical + 1; vertical++) {

for (int horizontal = 0; horizontal < maxHorizontal + 1; horizontal++) {

//only execute if randomSpawning is not 0 and user has not pressed the clear button

if (randomSpawning != 0) {

//random number generator

randomRandomSpawns = (int) (Math.random() \* 100);

//if the randomly generated number is greater or equal to 0 and less than the randomSpawning variable, both 2d arrays are set to true at that position

if (randomRandomSpawns >= 0 && randomRandomSpawns < randomSpawning) {

cells[vertical][horizontal] = true;

tempCells[vertical][horizontal] = true;

//if the randomly generated number is equal to or greater than the randomSpawning variable, both 2d arrays are set to false at that position

} else {

cells[vertical][horizontal] = false;

tempCells[vertical][horizontal] = false;

}

//execute if randomSpawning is 0 or the user has pressed the clear button

} else {

cells[vertical][horizontal] = false;

tempCells[vertical][horizontal] = false;

}

}

}

//sending to method FillingAndChangingArray

FillingAndChangingArray();

//setting the timeCounter to 0

timeCounter = 0;

//sending to the method Save

Save();

}

//declaring private void method used for filling in the array and changing the label

private void FillingAndChangingArray() {

//making sure the label is currently not being changed

if (changingLabel == false) {

//sending to method FillInArray

FillInArray();

//set label from variable lifeBoard

label.setText(lifeBoard);

}

}

//declaring private void method used for filling in the array after loading the game settings

private void FillInLoad() {

for (int vertical = 0; vertical < maxVertical + 1; vertical++) {

for (int horizontal = 0; horizontal < maxHorizontal + 1; horizontal++) {

if (parseInt(split[vertical \* (maxHorizontal + 1) + horizontal + 4], 10) == 1) {

cells[vertical][horizontal] = true;

}

}

}

}

//declaring private void method used for loading from file io

private void Load() {

try {

//trying to create file

Files.createFile(file);

//executed if file already exists

} catch (FileAlreadyExistsException x) {

//file is read from and saved to variable saveFile is file already exists

try (InputStream in = Files.newInputStream(file);

BufferedReader reader = new BufferedReader(new InputStreamReader(in))) {

String line;

while ((line = reader.readLine()) != null) {

//content of file is saved to saveFile

saveFile = line;

}

} catch (IOException y) {

System.err.println(y);

}

} catch (IOException x) {

System.err.println(x);

}

//if the file does not contain anything since it was just created, default variables are used for saveile

if (saveFile == null) {

saveFile = "24 24 200 0";

for (int counterFillFalse = 0; counterFillFalse < 25 \* 25; counterFillFalse++) {

saveFile += " 0";

}

}

//a String array is created and each part of the array is saved to from saveFile seperated by spaces

split = saveFile.split("\\s+");

//variable maxVertical is the first number

maxVertical = parseInt(split[0], 10);

//variable maxHorizontal is the second number

maxHorizontal = parseInt(split[1], 10);

//variable roundTime is the third number

roundTime = parseInt(split[2], 10);

//variable pause is the fourth number

if (parseInt(split[3], 10) == 1) {

//setting pause to true

pause = true;

}

}

//declaring private void method used for saving with file io

private void Save() {

//saveFile is created using the four main variables, seperated by spaces

saveFile = maxVertical + " " + maxHorizontal + " " + roundTime;

if (pause == false) {

saveFile += " 0";

} else {

saveFile += " 1";

}

//saving the 2d array to file

for (int vertical = 0; vertical < maxVertical + 1; vertical++) {

for (int horizontal = 0; horizontal < maxHorizontal + 1; horizontal++) {

if (cells[vertical][horizontal] == false) {

saveFile += " 0";

} else {

saveFile += " 1";

}

}

}

//saveFile is converted to byte data

byte data[] = saveFile.getBytes();

//byte data is saved to file using file io

try (OutputStream out = new BufferedOutputStream(

Files.newOutputStream(file, WRITE, TRUNCATE\_EXISTING))) {

out.write(data, 0, data.length);

} catch (IOException x) {

System.err.println(x);

}

}

}