Object Oriented Programming with Java WS 2022 / 2023 "The Crazy Sheep Game"

Rafael Becerra Villalobos, 1427763
Martin Ruppert, 1382471
Finn Schäffler, 1350212

Frankfurt University of Applied Sciences
Nibelungenplatz 1, 60318 Frankfurt am Main, Deutschland

Abstract. *The "Crazy Sheep Game" is a simple puzzle game that consists of nine equally big pieces that have to be put together to show the solution. Each side of the pieces has either the front or back part of a sheep with a certain color. Every part and color has to be correct to solve the puzzle. This paper was made to show our solution of how this game works and what functions we implemented to succeed.*

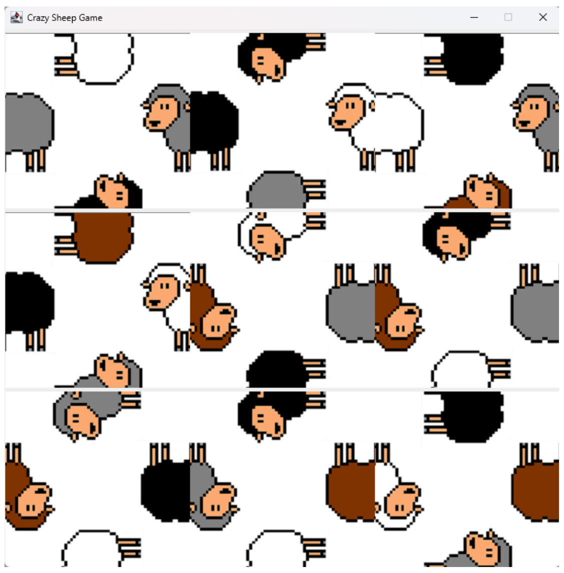


Table of Contents

1 Statutory declaration

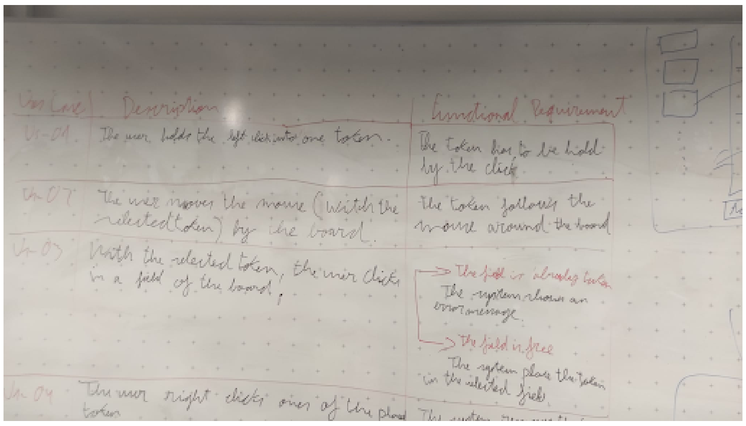
We declare that we have authored this thesis independently, that we have not used other than the declared sources / resources, and that we have explicitly marked all material which has been quoted either literally or by content from the used sources.

2 Teamwork

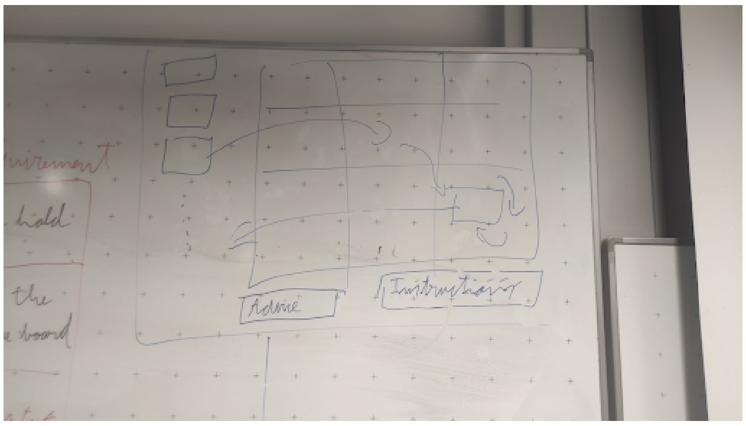
The organization in our group was a little bit complicated at the beginning, because we decided to divide the work based on the modules which were present in our first attempt to make the module organization. Afterwards, we chose to base our work on the functionalities and the different parts which the project would be involved to. Those parts include: backend programming, frontend designing, project organization and final documentation. The job made by each of us was crucial to achieve the final result of the project itself, but if we need to specify the labor made by everyone, we can say: Martin was in charge of all the backend and the actual functionality of the program itself. Finn was the designer of the frontend of the game and also has developed the 'checksolution' function. Rafael has had the role of filling and controlling the structure and design of the documentation. He has also been responsible for making the script of the AI that resolves the puzzle automatically. Besides the fundamental division of work we were open for questions and helped each others a lot. We had lots of meetings on discord and in person where we discussed issues, the progress or the next steps. In general our communication was very good and our meetings were always productive.

2.1 Brainstorm

The brainstorm was made in the initial phase of the subject, so we could have enough time to change and correct any possible mistake or design. The brainstorming took place while we were in seminars of OOP. Over there we developed all the ideas regarding the board, the module design, the first UML cases, the user interface and interaction etc.



Use Case	Description	Functional Requirement
Us-01	The user holds the left click onto one token.	The token has to be hold by the click
Us-02	The user moves the mouse (with the selected token) by the board.	The token follows the mouse around the board
Us-03	With the selected token, the user clicks in a field of the board;	→ The field is already taken The system shows an error message → The field is free The system place the token in the selected field
Us-04	The user right clicks one of the placed token	[illegible]



Our first idea was to develop a game in which each token could be taken by the user and placed into the site he/she wanted. We finally discarded this option due to the fact that, making the piece of the puzzle attached to the mouse, was more complex that it could first seem. In consequence we designed a puzzle in which each token was switchable by the user using the mouse and left click.

2.2 Work structure

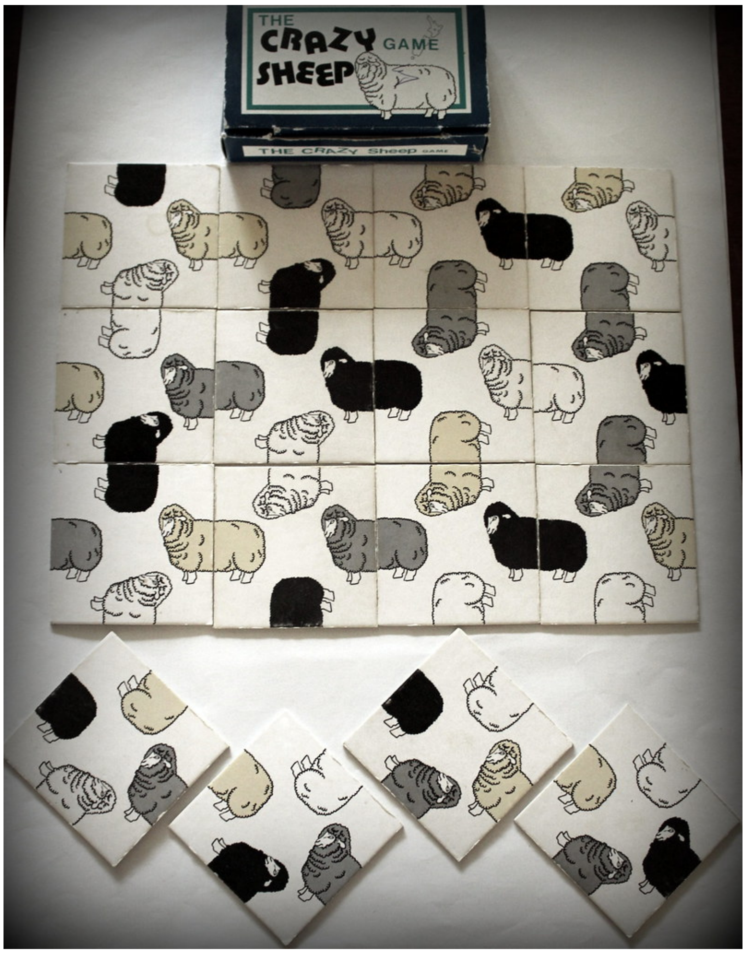
The communication between every team member was made by using whatsapp and discord. In whatsapp, we talked about ideas, concepts and possible requirements that the application would need, meanwhile in discord, we talked by voice to help each other about programming bugs, problems using libraries, decisions about the final design etc. This application was also used to portray the final decision in each part of the project. Whenever issues occured we met up and discussed them as a team. Some of these issues include: Connection of the GUI and the backend. This part was kind of complicated to design at the beginning until we discovered a way to write both necessary parts in one java application. Final state problem: The final state was correctly made, but the final screen wasn't appearing.We had a problem in our switching pictures function. A line was missing that changed the variable in our object when we switch positions. So basically the pictures were at the right position and our checksolution was working but the variables were wrong. But all of us looked for the issue and we found it relatively fast. Problem with the first design. The first game design was not fitting our backend idea or our backend capabilites at that moment, so we changed our visual idea to something that we fully understood and that we could implement from scratch.

3 Introduction

The game itself consists of a simple puzzle game for children where the objective is to match pieces with different animal body parts to create a complete picture of different coloured sheep. Every piece can have the front or the back of the body of one animal on every side of the piece itself. The goal is to attach and join the two different parts of all the animals on the board. Thereby the color has to match too.

The game finishes when you join the two parts of all the animals, not leaving any animal part without getting attached to another. The only parts which can go by free are the ones who are in the external borders of the game itself.

This "crazy sheep" game has more formats and possibilities, in fact, this is just one of the different versions that go around the internet and game boards. One game which actually inspired us to make the game look like this, was this one made by the user Gillian Everett.



This game usually works with just one design of itself, furthermore it sometimes can even have different solutions for the same game. Knowing how this game is presented normally, we decided to make two levels of it to give more content to the user. Watching other solutions for the same problem online, we saw that most of the people who developed this game were making it as a flash game for different website pages, but knowing that we didn't have to make it online (due to the programming language), we could have the possibility to charge all the pieces of the whole puzzle locally. The other extra game in our software is called "crazy dog" and, as you might think, the new level has exactly the same rules as the "crazy sheep" game, except this one is played with dog images. If you are more interested in knowing real examples of this board game, you could check out one of these URLs:

[1] and [2]

4 Problem description

In this section of the documentation, we are going to talk about the game itself, but from the point of view of the code and software engineering.

The game “crazy sheep”, as we described before, is a board game whose core is to sort out basically a puzzle, so, even before starting with the coding, first we need to understand what is the concept of the game. This requires us to think about, what are going to be the inconveniences to develop all the system, in which parts can we divide this, all the requirements and possible hazards that can affect the performance or even the functionality of the program etc.

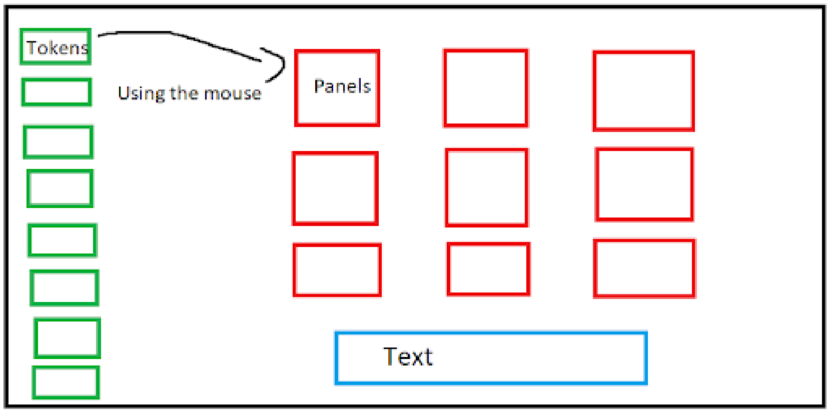
The first thing we need to do to understand and develop our game, is know how it is going to look and how it is going to be playable for the user. To make this, the best option here is to brainstorm with the rest of the members and compare all the ideas together to achieve one final result (extras versions are also allowed). One thing which is usually very useful to express your idea about the look of the game, is using a board where everyone can write, paint or delete expressions, concepts, sketches, even UML diagrams and more.

Once you have the first look of the game, now you need to think how it is going to interact with the user, which are going to be the keys to control the game itself, what are going to be all the issues which you can handle during the development and maybe some level design (just in case you want to make more than one level/game in your software).

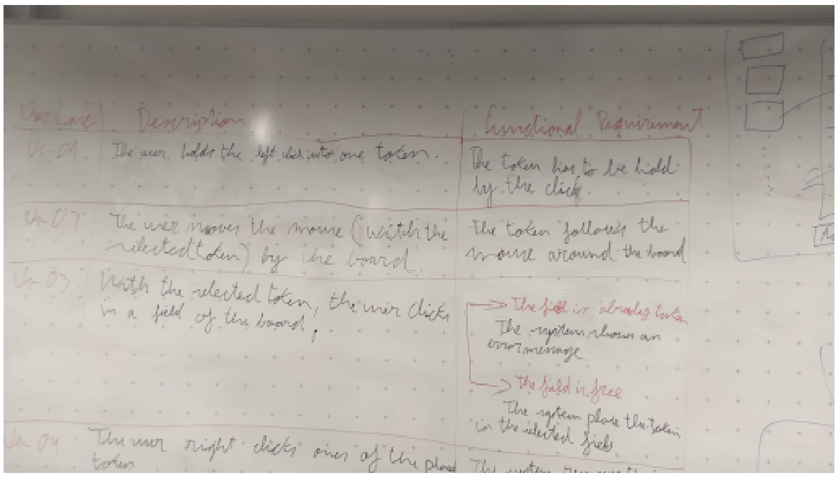
The interplay with the user is hand by hand with the UML representation and cases because it represents all the possible cases where the actors that participate in the interaction with the software. The actors are not only the users itself but also other components of the system and also facts like the time itself. However, we don’t need to forget to always test and check that everything we do or design has to be tested.

4.1 How did we solve our problem?

As we mentioned before, first we tried to resolve the user interface problem. We made this in some of the seminaries of the subject by writing on the board how it should be and how the actors of the system would interact with each other.



After having more or less what the interface should look like, we went directly to the various cases that the software could have at the moment that is executed by a user.



Use Case	Description	Functional Requirement
Us-01	The user holds the left click into one token.	The token has to be hold by the click
Us-02	The user moves the mouse (with the selected token) by the board.	The token follows the mouse around the board
Us-03	With the selected token, the user clicks in a field of the board;	→ The field is already taken The system shows an error message. → The field is free The system place the token in the selected field
Us-04	The user right clicks over of the placed token	[illegible]

By having all the possible combinations and functions that should be portrayed into the software, now it was time to make the planning, division and definition of the modules that would take responsibility in the proper functionality and performance of the software.

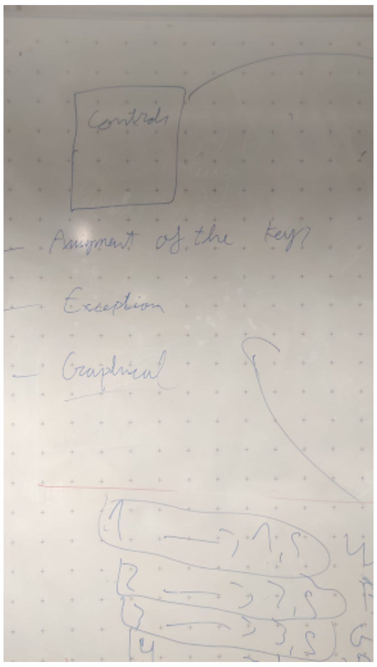
The modules which were proposed at first instance were actually 3 plus the main one. These three modules were:

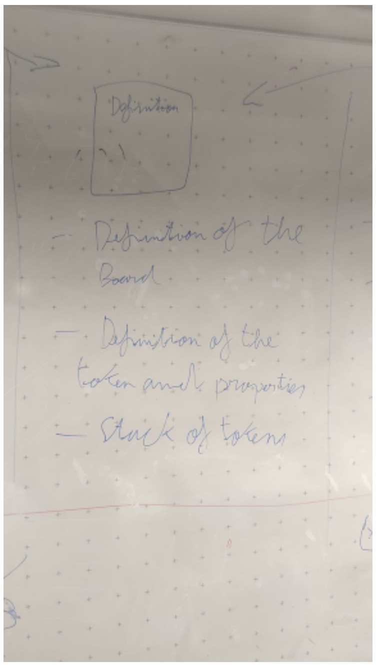
Control module: This module was going to take the responsibility of having inside all the code related to the controls of the software and how the user must use it to play the game. Basically, here we were defining how the user is going to play the game and which elements of the keyboard or mouse would be used.

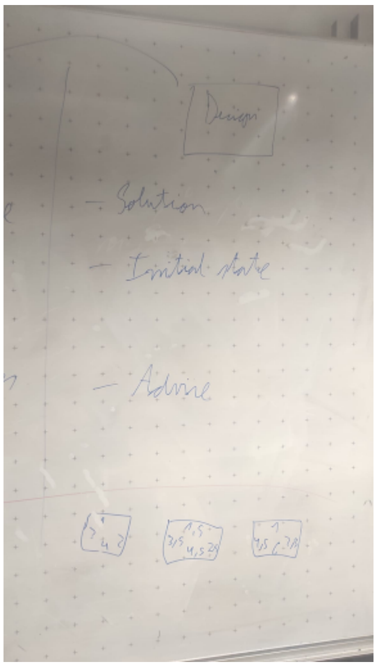
Definition: The definition module was by far the most important of all of them, not only that, it was also the one who would be in charge of define, control, use, and make portable all the objects which were going to be defined and used in all the software. It is important to remind the importance of the module, because it would make possible the creation of the puzzle and the definition of every token. Without these definitions, there would be no possibility to create any type of level design or interaction between the user and the controls. If the definition of the objects were failing, then it would be impossible to launch the game.

Design module: This one was also important, but not as much as the previous one. The purpose of this module was to generate and control the initial and final state of every game. In other words, it was created to store the design of the game and their other possible levels, including the end state solution and how they would be initialized.

Main: This last module doesn't need too much explanation because just by the name of it you can guess what it is about. The Main module is the one in charge of make the connection between all the modules.




Definition
– Definition of the Board
– Definition of the token and properties
– Stack of tokens



Now that we have revealed the initial design and requirements of the program, now we are going to present to you what has been the final solution to the problem and why we chose to change it. We will present to you the final solution to the problem in two parts, one will be about the backend and the last one will be about the design of the puzzle itself and the game.

4.2 Design and puzzle

Our previous design did not fit exactly with all the requirements that we were expecting to complete, in consequence we had to change the design of the puzzle and even the interaction that the user would have with the software to complete the requirements previously defined. Our previous design was not measured up because it had some important issues that would affect the system seriously. Some of the problems that the past design had were:

Redundancy of the variables: Java is an object oriented language, but not especially like C++. This is due to the fact that Java can only work with classes even if you are not planning to use any type of object in your program, while C++ has the ability to work with or without any type of object and not having any issues. So, the problem here is that if we define the objects from one class to another, then it could have connection issues as well as with the heredity of the classes. C++ has a powerful system to make inheritances between classes, however Java doesn't have it. So to fix this problem with the same design, you would need to define the variables and the objects from one class to another in another module.

Complex frontend: With the previous design it is possible to have a frontend which will not be fixing the size that the backend defines because of the dynamic movement of each token. To avoid the issue, we decided to make the tokens not to move dynamically and leave them statically. The only chance to move the pieces would be to make a switch in their initial positions.

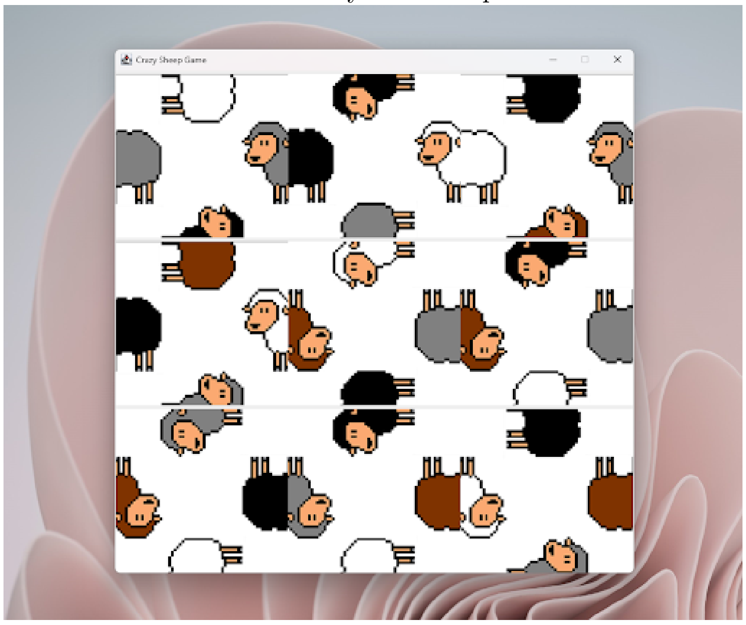
Controls not so logical: Knowing the problem with the frontend presented previously, the controls of the game could be incompatible with some keys, so, to fix this problem, we just add the function to use only the buttons present in the mouse.

Why do we need a design module?: The design module is not very useful if we are thinking in just make one of two levels, so we could add all the functions, scripts and design to the definition module. By doing this, we would also solve the problem of the data and variables redundancy.

New design, new organization: At the beginning, as I mentioned before, we made the work organization based on the modules and functions, but, if there were changes on the design, who were going to be responsible for? That's the reason why we changed the organization to functional parts of the project rather than the software itself.

With all these mistakes in mind, we changed the design of the puzzle, the frontend and the backend. The frontend would look more like a single panel with different buttons to press in order to change the positions of each token, in consequence the puzzle would look like a 3x3 panel screen with 9 tokens in some initial position ready to be changed.

The backend has its own story and design. Now the backend has 4 modules: "Main", "panel", "Objects" and "MeinFrame". I am not going to enter in detail about these modules because they will be explained later.



5 Related work

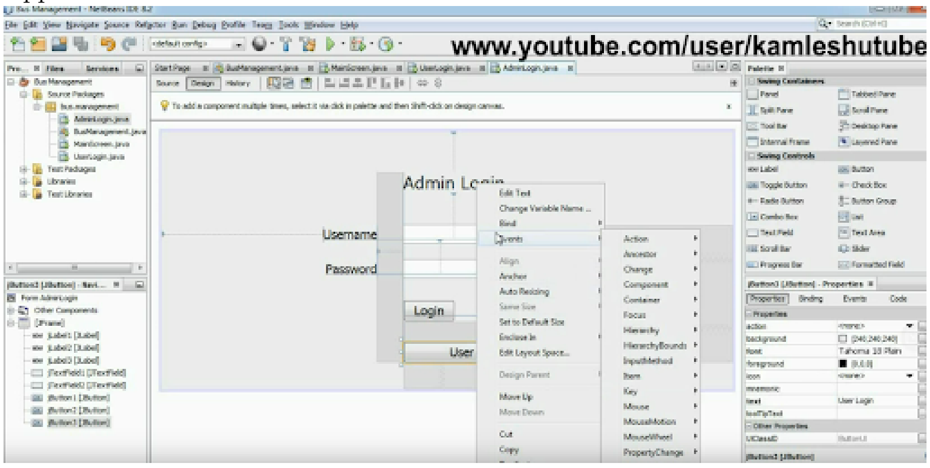
We are not professional coders yet, so, obviously, we had to watch tutorials, search in google and other forums for information related to the backend of the project and investigate the best way to achieve the goals of the future software.

[3]

This link above, was one of the first sites we visited. Over here you can find the use and management of a lot of libraries, classes, layouts and much more. If we need to stand out one thing from this page, it has been the information provided about the JPanels. These JPanels are responsible for how tokens are placed and how they can actually switch without alternating the size of the puzzle, however, to achieve the rest of functionalities and have a full knowledge of JPanels, this website has not been the best one, but it was kinda helpful.

[4]

Here you can see a link to a youtube video which has the title of “How to close a JFrame while opening another one? - Java Swing Projects”, well, the videos of this chanel, specially the list that the video follows, has been really helpful to now how to build the puzzle, what are Jframes, how to develop buttons and change the color of these ones, how to change a pointer to execute a function from a button, how to make visible or not a button, how to change from one screen to the next one and lot of other things which has been crucial to develop the application.



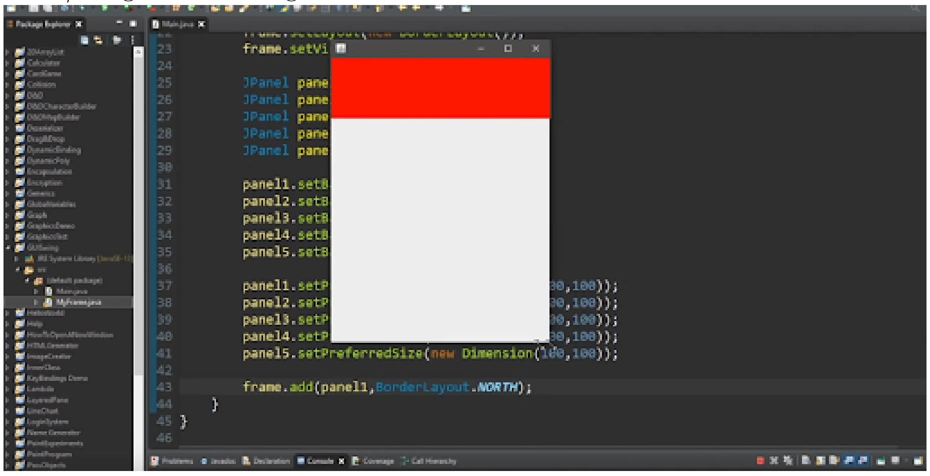
[5]

The video shows us how to make an image turn. This could seem like something easy to manage, however, let me tell you that the process of making an image turn was a challenge for us at the moment. In spite of that, the video was actually really helpful to know how to make an image to turn whenever the user wants. So, by the algorithm shown in the video, we make a function called “turn” which actually takes the piece of the puzzle, turns it 90 degrees and then changes the value inside the object which is getting imported into the function. When the object has been turned 4 times, the value changes again into 0. To call our function from another one, just use: “turn(object)”; being ‘object’ the button we want to turn.

[6]

From this multimedia in youtube, we were able to make the connection between the frontend and the backend. As you just read, the knowledge that this video provided to us was crucial to develop the final product because it shows all the procedures, algorithms and steps to make a program interactable by any user. The main point of the video is the development of GUI, but it also shows other techniques and tools which are actually very useful. If you visit the channel from this youtuber, you will be able to find a lot of content and media about programming languages and coding skills.

If you have entered this last video, you would be able to see that it is 4 hours long so it contains a lot of coding parameters, techniques and methods. One part which was vital for the proper adjustment of the frames and the pictures to the marks of the software, was when he shows how to use labels and what they are used for. The labels made possible the framing between the pictures, the buttons and the marks of the software itself. Another part which was actually very practical for the software, was the BorderLayouts. These BorderLayouts are used to make possible changing the color of one of the buttons or to change the color/image of the background of our software.



```
        frame.setVi

        JPanel pane
        JPanel pane
        JPanel pane
        JPanel pane
        JPanel pane

        panel1.setB
        panel2.setB
        panel3.setB
        panel4.setB
        panel5.setB

        panel1.setP                          00,100));
        panel2.setP                          00,100));
        panel3.setP                          00,100));
        panel4.setP                          00,100));
        panel5.setPreferredSize(new Dimension(100,100));

        frame.add(panel1,BorderLayout.NORTH);
    }
}
```

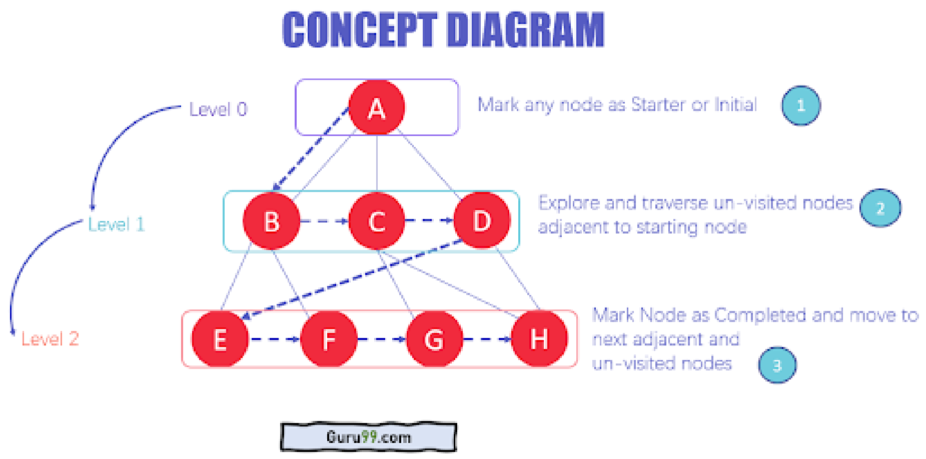
[7]

For developing the IA which solves the puzzle automatically, we were inspired by this video (the video is in spanish). It shows and explains the technique of breadth-first search (BFS) for nods. It could seem hard to understand and develop, but knowing that in our case we are just talking about different tokens and states, the moment you get the idea of how this type of searching genuinely works, you can develop it with not much difficulty.

What does this kind of mathematical searching, is starting at the tree root (or some arbitrary node of a graph) and explores the neighbor nodes first, before moving to the next level neighbors. BFS is used to find the shortest path from one node to another in a graph, or to find all nodes within a certain distance from a starting node. It is implemented using a queue data structure, where the unvisited nodes are stored in the order they are discovered, and are processed in the order they are added to the queue.

BFS can be implemented using either an iterative or a recursive approach. The basic idea behind both implementations is to maintain a queue of nodes to visit and marking them as visited when processed. In each step, the algorithm takes the node from the front of the queue, processes it, and adds its unvisited neighbors to the end of the queue. This continues until there are no more nodes to visit, or until a goal node is found.

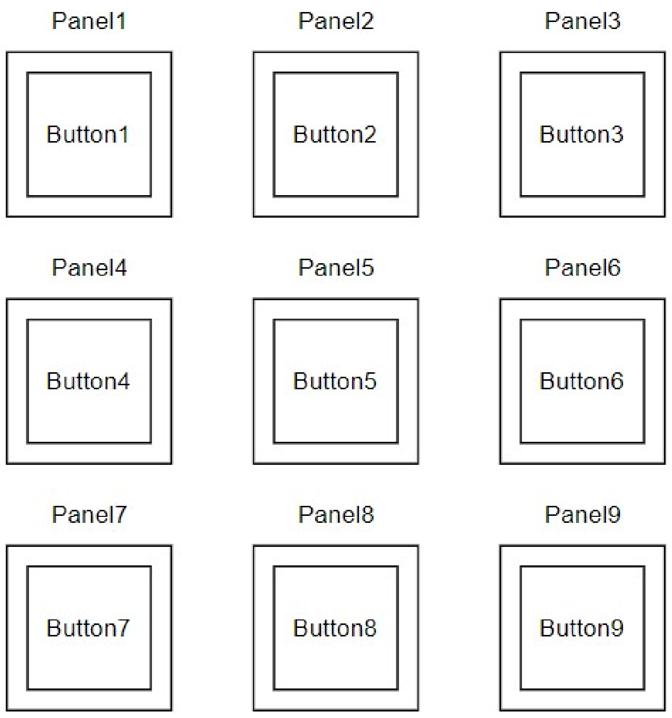
In summary, breadth-first search is a versatile and important algorithm in computer science with many practical applications, and a good understanding of it has been essential for developing this last script.


CONCEPT DIAGRAM
Level 0
Level 1
Level 2
A
B
C
D
E
F
G
H
Mark any node as Starter or Initial
1
Explore and traverse un-visited nodes adjacent to starting node
2
Mark Node as Completed and move to next adjacent and un-visited nodes
3
Guru99.com

6 Proposed approaches

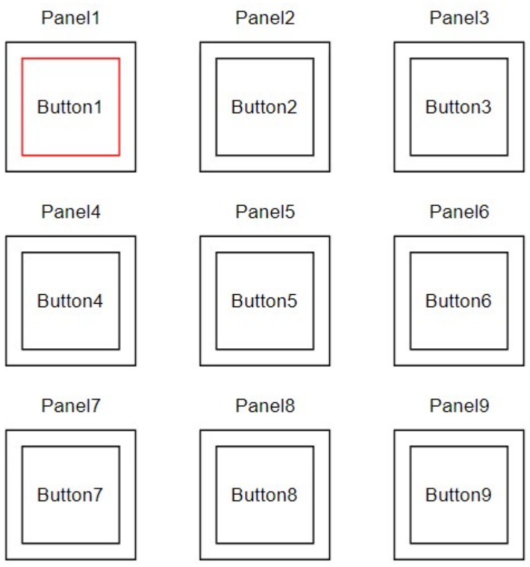
6.1 Input / Output format

Our Game has a rather simple input and output system. The user only needs to use his mouse to maneuver the screen. After choosing a game at the beginning a window opens and reveals the nine pieces of the game that can be switched with another piece or can be rotated by ninety degrees. We do that with the function MeinFrame which is an extension of the JFrame class. JFrame allows us to create the look of our game.

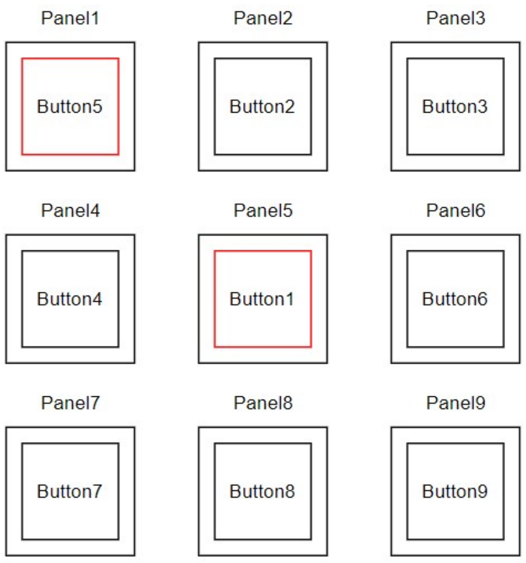


Every piece is a button that can be clicked. One button lies within one panel but is not bound to it. Therefore the pointer of each button and panel point directly at each other to connect the two components. The panels stay in a three by three layout which improves the look and accessibility of our game.

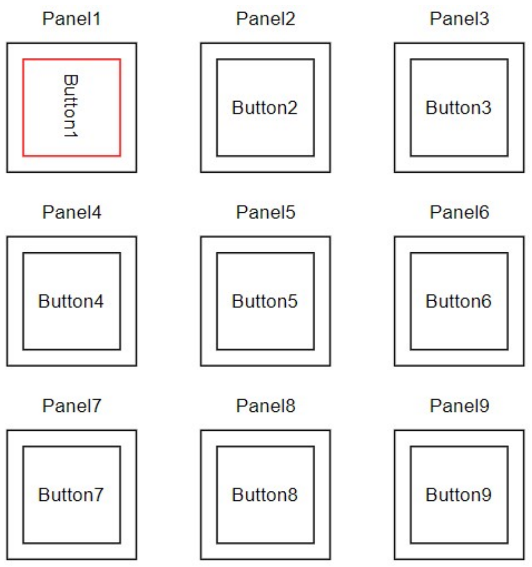
The first step to play the game is to select a piece that has to change. After getting clicked the data of the piece is being read by the system and ready to change.



If you want to switch a piece you can simply select one of the nine buttons and then click on a second one. The values and the design of the button gets overwritten with the new selected button and vice versa.



If you want to rotate a piece you can simply select one of the nine buttons and then click on the same button again. The programm executes this in a very specific way. The image gets copied and turned by ninety degrees. This has an impact on the folder where all the pictures are safed. The old image gets deleted and the new image takes its place.



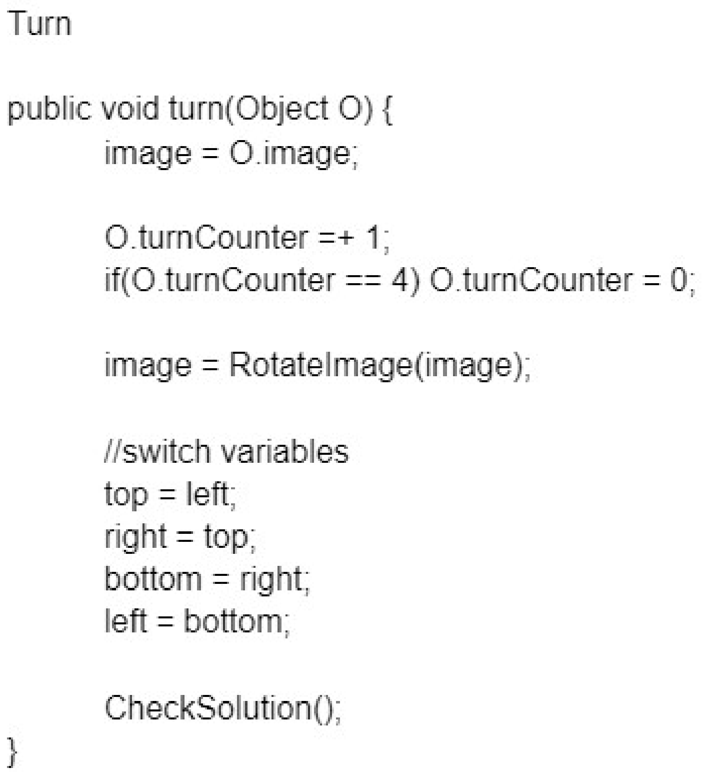
After solving the puzzle a new window opens to congratulate the player. A simple game with a simple way to play is one of the best features of this program.

6.2 Algorithms in pseudocode

This section talks about pseudocodes that could represent our program. For example there are more than 700 lines of code and almost all of them are well written algorithms that work in the background of the game without being noticed. one of the rather less noticeable functions is called CheckSolution and does like the name already says check if every piece is at the right spot. To be more precise the function checks if the values of each piece are in line with the next one. For example our game consists of nine pieces with four dog parts on each piece. The function looks if the color between two pieces is correct and if these two pieces do not connect each other with the same half of a dog.

```
CheckSolution

public void CheckSolution() {
        if(panel1.piece.rightColor == panel2.piece.leftColor && panel1.piece.rightPart !=
        panel2.piece.leftPart && panel1.piece.bottomColor == panel4.piece.topColor &&
        panel1.piece.bottomPart != panel4.bottomPart && … && panel8.piece.rightColor
        == panel9.piece.leftColor && panel8.piece.rightPart != panel9.piece.leftPart) {

        Open new Frame("Finished");
        }
}
```

The turn function has three different parts. First the selected image gets copied so it can be changed correctly without deleting any pictures. Because of several bugs consisting our turn function we created a variable that counts the number of rotation that each piece goes through. Every four turns the counter goes back to zero. After closing the page the turn function gets called as long as every piece is still not turned back to zero. The third part starts after rotating the image. Every piece consists of several strings that are safed within a object. Those strings consist of important information like for example the path to connect the picture with the right button. The eight different strings consists of the colors and parts of the sheep. At the end the CheckSolution gets called.

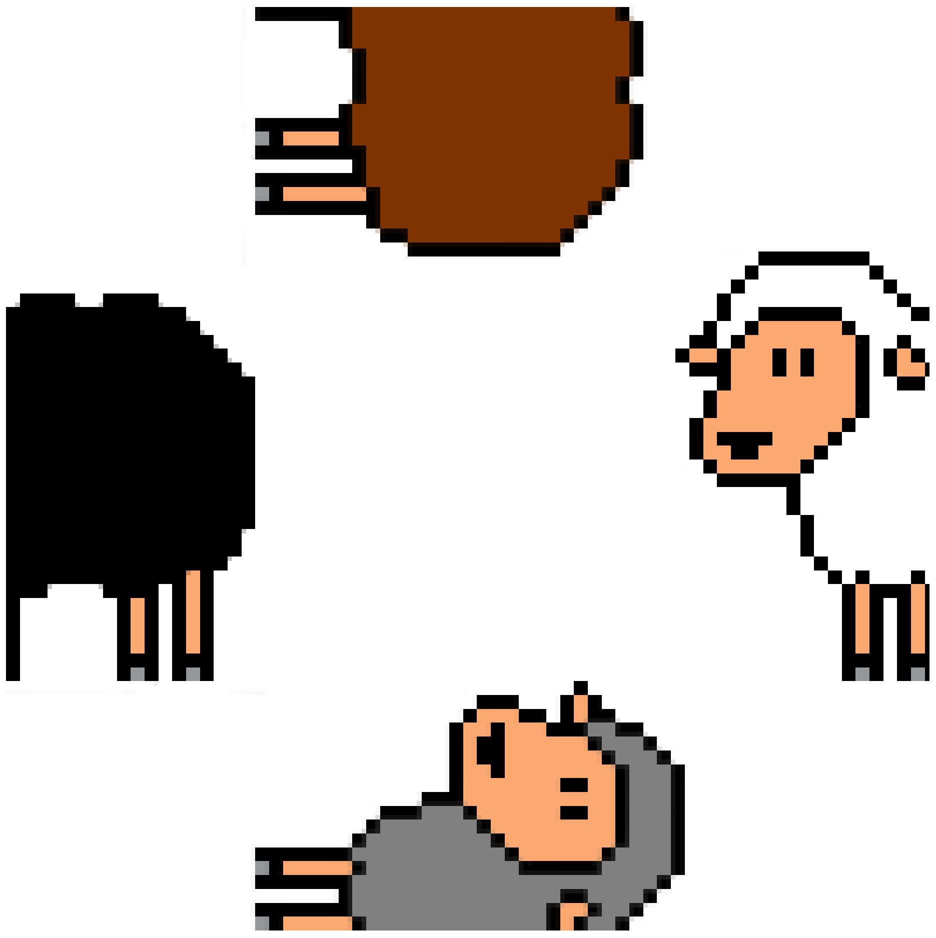


```
Turn

public void turn(Object O) {
        image = O.image;

        O.turnCounter =+ 1;
        if(O.turnCounter == 4) O.turnCounter = 0;

        image = RotateImage(image);

        //switch variables
        top = left;
        right = top;
        bottom = right;
        left = bottom;

        CheckSolution();
}
```

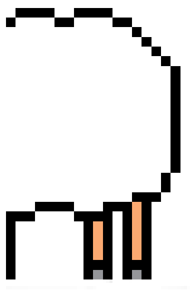
7 Implementation details

7.1 GUI details

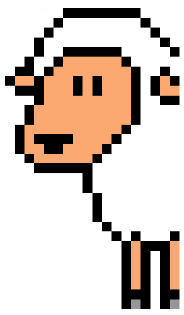
Our MeinFrame class is responsible for all the GUI changes and designs. We created each piece of our puzzle individually and colored it in a certain way so that the user can find a solution. The design idea was taken from the internet and later redesigned and changed in a way that we can use it.



Designing the pieces was difficult, first we made a blank 250 by 250 picture and started editing the front and back part of the sheeps. It was very important that the front and the back part would line up with each other when placed in the working game. After some calculating we drew the sheep between the pixels 53 and 145 so that the pieces would fit.



The front part of the sheep was more fun to create than the back because it had more details.



Both parts were added into a new image. Therefore the different parts had to rotate so that every side of the piece was filled with one part of the sheep. Two front parts and back parts were added to each image to guarantee a solution.

The colors were added in a certain way. We took pictures of the real game when its finished and colored our pieces the same way to guarantee one out of many possibilities to solve the puzzle.

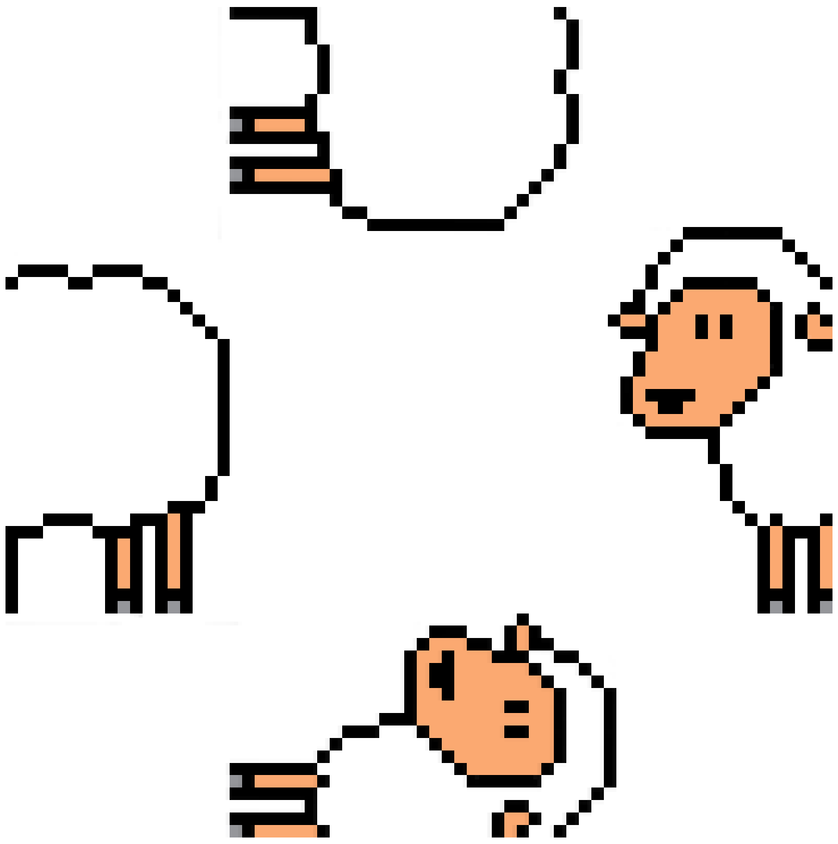


Fig. 1: Cover

7.2 Classes

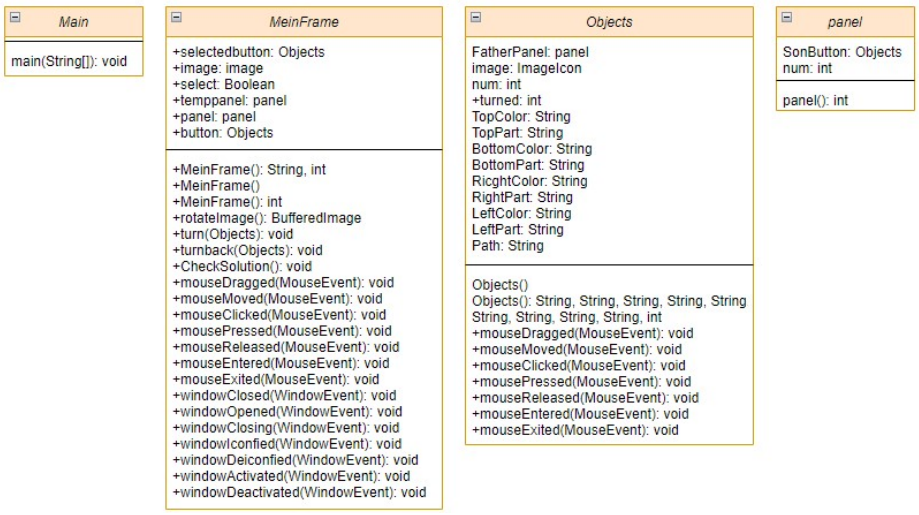
Our classes follow a certain structure. We have four different classes to reduce confusion and to keep the overview while having around 700 lines of code.

The main class calls the MeinFrame constructor and thereby opens a first window.

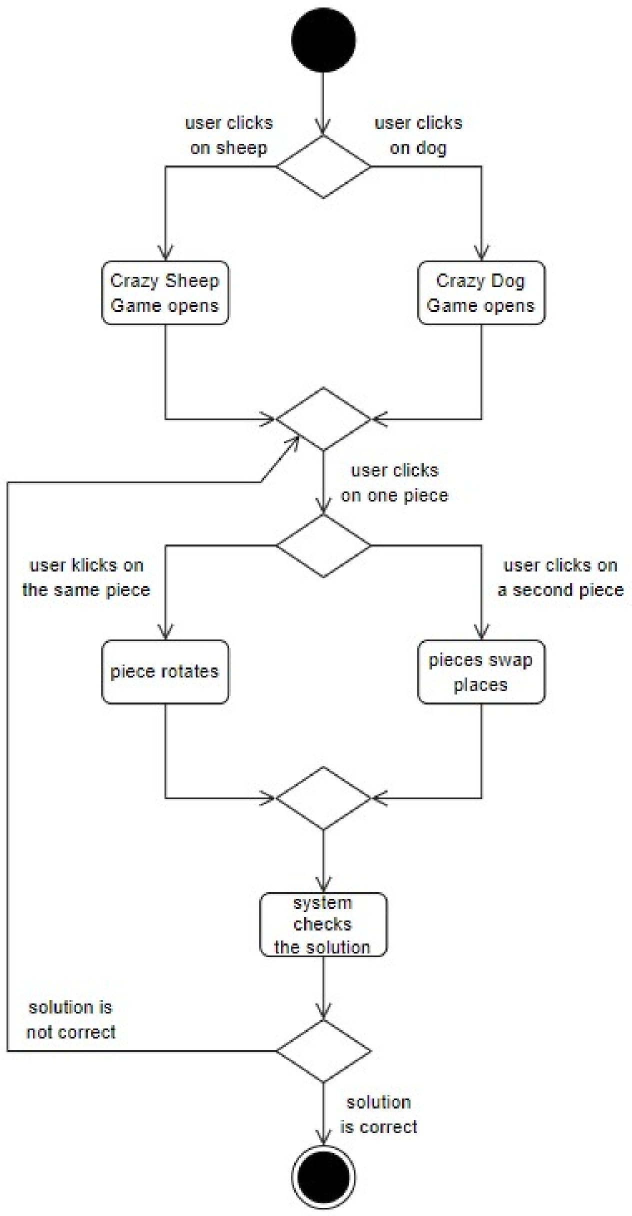
MeinFrame consists of almost every function inside our project, for example are functions like turn, turnback and CheckSolution all written inside the MeinFrame class. As an extension of JFrame it creates windows that can display anything that is programmed within a project. This class also initializes every piece as an object of the Object class. Besides initializing and visualizing objects and panels there are several so called MouseEvents that take care of the user generated input.

Objects consists of the information behind the pieces. The Objects class has a pointer that points at the panel class to create a connection. Other important information that is stored in this class are the different parts and colors and the counter of how often a piece is rotated. Then again there are several MouseEvents that have to be programmed so that the user can actually maneuver the window and its puzzle.

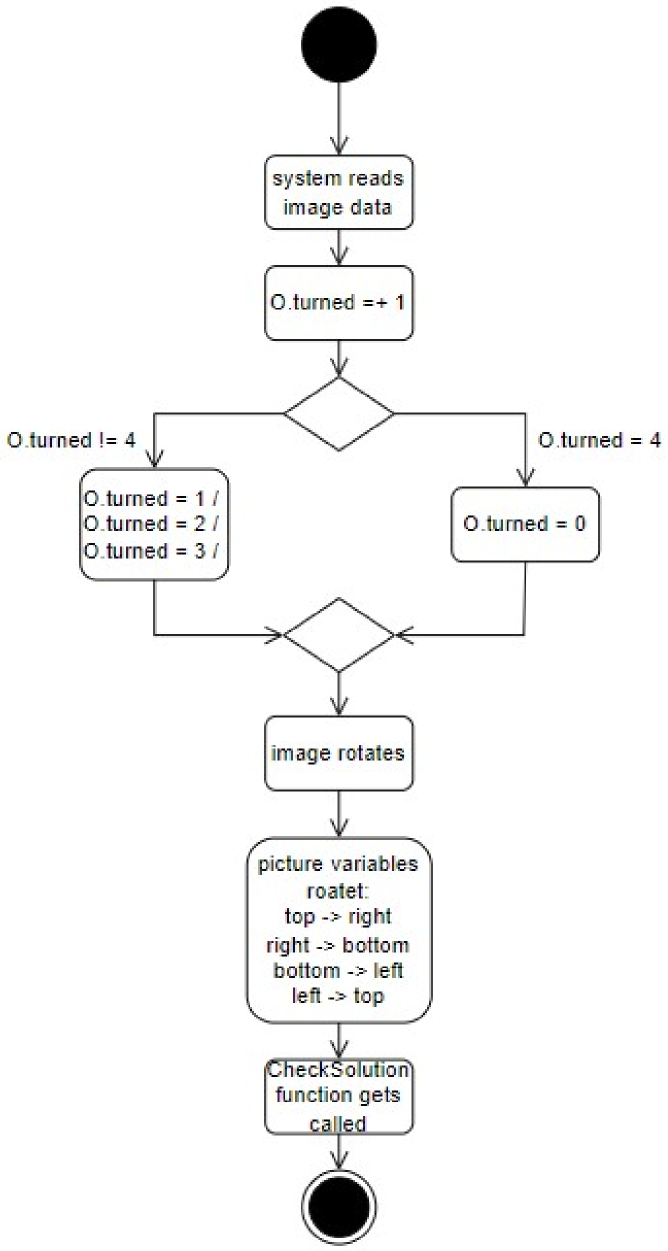
The panel class creates nine panels that contain a button (piece) each. This class do not need any auto-generated MouseEvent methods because the user is not supposed to interact with them. They have to stay in there position.


Main
main(String[]): void
MeinFrame
+selectedbutton: Objects
+image: image
+select: Boolean
+temppanel: panel
+panel: panel
+button: Objects
+MeinFrame(): String, int
+MeinFrame()
+MeinFrame(): int
+rotateImage(): BufferedImage
+turn(Objects): void
+turnback(Objects): void
+CheckSolution(): void
+mouseDragged(MouseEvent): void
+mouseMoved(MouseEvent): void
+mouseClicked(MouseEvent): void
+mousePressed(MouseEvent): void
+mouseReleased(MouseEvent): void
+mouseEntered(MouseEvent): void
+mouseExited(MouseEvent): void
+windowClosed(WindowEvent): void
+windowOpened(WindowEvent): void
+windowClosing(WindowEvent): void
+windowIconfied(WindowEvent): void
+windowDeiconfied(WindowEvent): void
+windowActivated(WindowEvent): void
+windowDeactivated(WindowEvent): void
Objects
FatherPanel: panel
image: ImageIcon
num: int
+turned: int
TopColor: String
TopPart: String
BottomColor: String
BottomPart: String
RicghtColor: String
RightPart: String
LeftColor: String
LeftPart: String
Path: String
Objects()
Objects(): String, String, String, String, String
String, String, String, String, int
+mouseDragged(MouseEvent): void
+mouseMoved(MouseEvent): void
+mouseClicked(MouseEvent): void
+mousePressed(MouseEvent): void
+mouseReleased(MouseEvent): void
+mouseEntered(MouseEvent): void
+mouseExited(MouseEvent): void
panel
SonButton: Objects
num: int
panel(): int

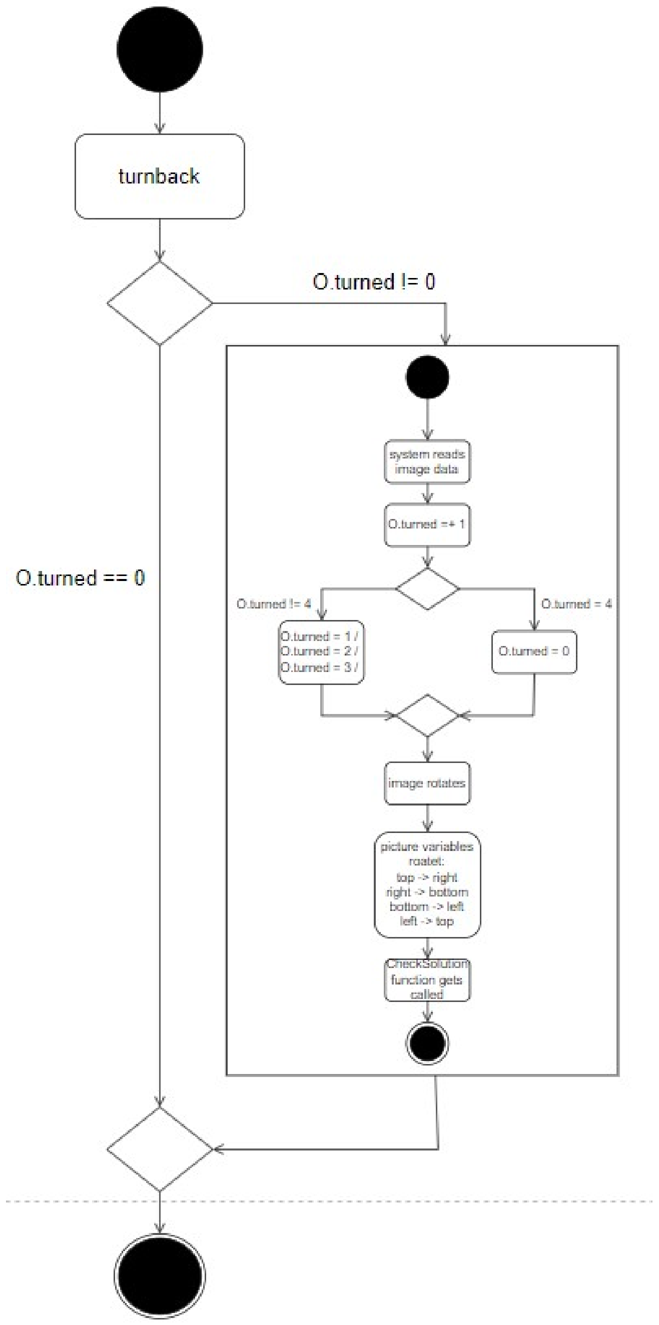
7.3 UML diagrams



Actitvity diagrams allow us to understand the concept behind a programm. This diagram shows the general way of how our game has to work.


system reads image data
O.turned =+ 1
O.turned != 4
O.turned = 4
O.turned = 1 / O.turned = 2 / O.turned = 3 /
O.turned = 0
image rotates
picture variables roatet: top -> right right -> bottom bottom -> left left -> top
CheckSolution function gets called

This activity diagram shows the turn function and how it works.


turnback
O.turned != 0
O.turned == 0
system reads image data
O.turned =+ 1
O.turned != 4
O.turned = 4
O.turned = 1 / O.turned = 2 / O.turned = 3 /
O.turned = 0
image rotates
picture variables roatet: top -> right right -> bottom bottom -> left left -> top
CheckSolution function gets called

This diagram shows the turnback function and includes the turn method inside of it.

7.4 Used libraries

We used a few libraries to create our program.

java.awt.* is the abstract window toolkit and javax.swing.* is mainly reffered to as part of the Java Foundation class. Both allow the user to create windows. One of our most important feature JFrame and JLabel is part of java.swing and is the reason of how the game looks.

java.awt.event.* is part of the abstract window toolkit. It allows the user of our program to take part in it and maneuver through the window. This library adds MouseEvents such as mousePressed, mouseReleased, mouseClicked and many more. It also adds MouseListener and MouseMotionListener to interact with the panels and buttons.

java.io.* adds different input and output functions that have an impact of what the game reads and prints out for us to see.

7.5 Codesnippet

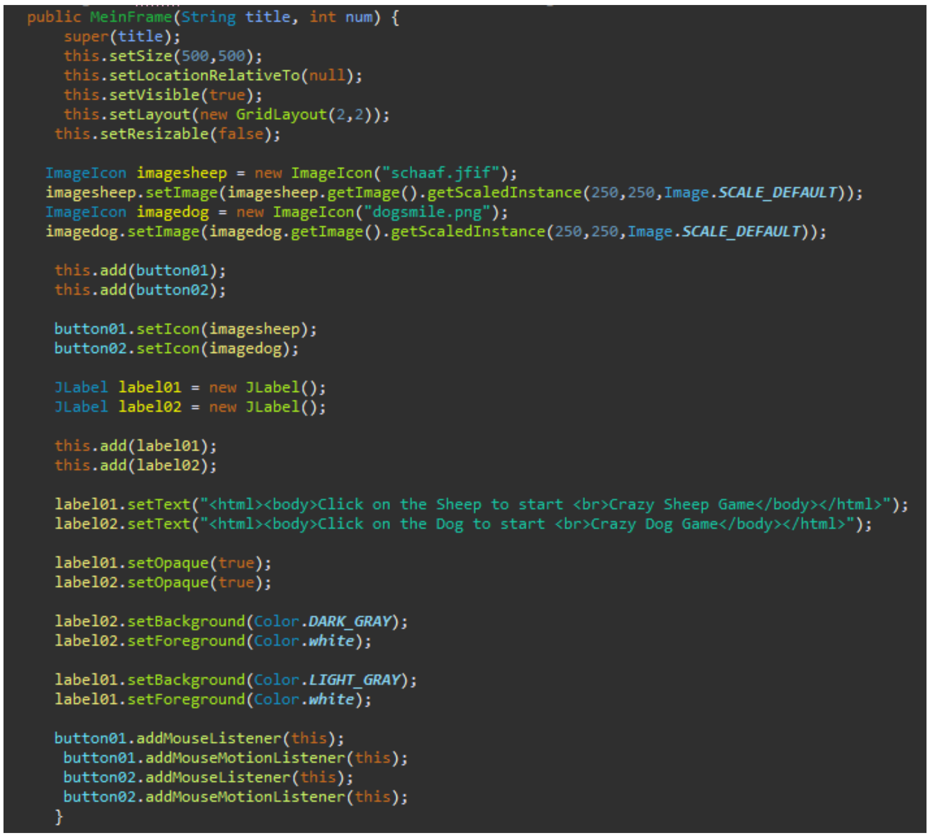
The first codesnippet shows the first frame of our game and how it is initialized. The first paragraph sets the size and design of what the frame should look like, for example the title, the size, the layout and if the frame is suppossed to be resizable.

The second paragraph inserts the two pictures of a sheep and a dog which were added to make the user choose, which game he or she wants to play.

The third and forth paragraph add two buttons that receive the icons of the images mentioned before.

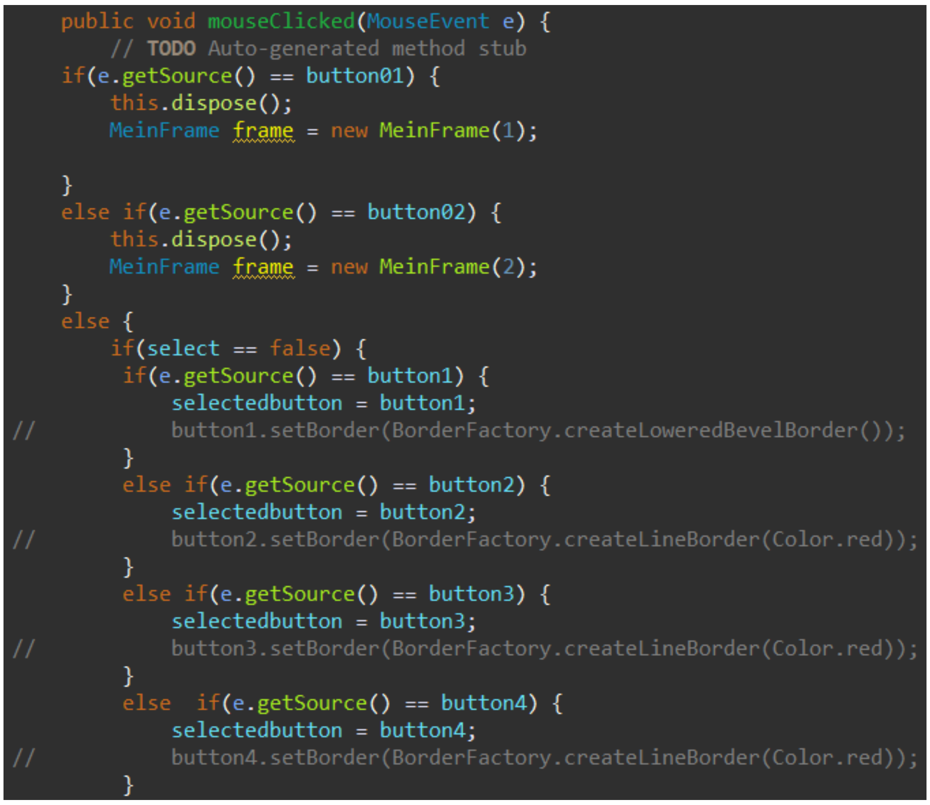
The next few paragraphs add two JLabels that are used to print out text for example "Click on the sheep to start, Crazy Sheep Game" or "Click on the dog to start, Crazy Dog Game".

The last paragraph adds MouseListener and MouseMotionListener for the two buttons.



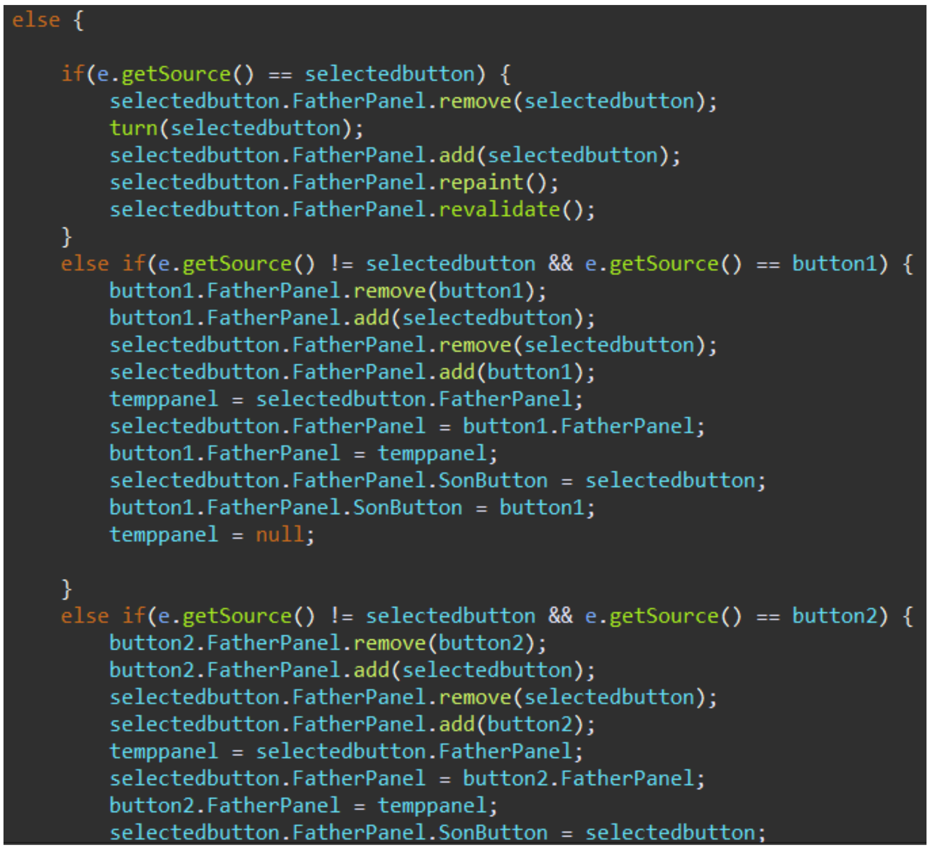
```
public MeinFrame(String title, int num) {
    super(title);
    this.setSize(500,500);
    this.setLocationRelativeTo(null);
    this.setVisible(true);
    this.setLayout(new GridLayout(2,2));
   this.setResizable(false);

  ImageIcon imagesheep = new ImageIcon("schaaf.jfif");
  imagesheep.setImage(imagesheep.getImage().getScaledInstance(250,250,Image.SCALE_DEFAULT));
  ImageIcon imagedog = new ImageIcon("dogsmile.png");
  imagedog.setImage(imagedog.getImage().getScaledInstance(250,250,Image.SCALE_DEFAULT));

   this.add(button01);
   this.add(button02);

   button01.setIcon(imagesheep);
   button02.setIcon(imagedog);

   JLabel label01 = new JLabel();
   JLabel label02 = new JLabel();

   this.add(label01);
   this.add(label02);

   label01.setText("<html><body>Click on the Sheep to start <br>Crazy Sheep Game</body></html>");
   label02.setText("<html><body>Click on the Dog to start <br>Crazy Dog Game</body></html>");

   label01.setOpaque(true);
   label02.setOpaque(true);

   label02.setBackground(Color.DARK_GRAY);
   label02.setForeground(Color.white);

   label01.setBackground(Color.LIGHT_GRAY);
   label01.setForeground(Color.white);

   button01.addMouseListener(this);
    button01.addMouseMotionListener(this);
    button02.addMouseListener(this);
    button02.addMouseMotionListener(this);
   }
```

The method MouseEvent is auto-generated and takes care of several things. Clicking the mouse within the open window creates a so called MouseEvent. Every movement with the mouse gets detected. The next picture shows what happens if one of the buttons get clicked. The system reads the source of the button and creates a copy that is called selectedButtpn.



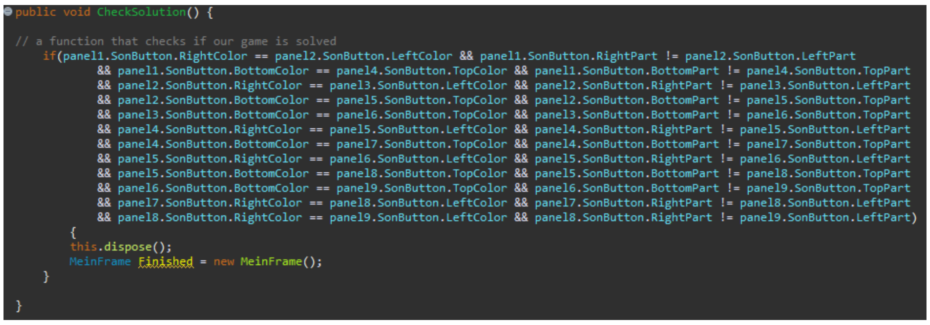
```
    public void mouseClicked(MouseEvent e) {
        // TODO Auto-generated method stub
    if(e.getSource() == button01) {
        this.dispose();
        MeinFrame frame = new MeinFrame(1);

    }
    else if(e.getSource() == button02) {
        this.dispose();
        MeinFrame frame = new MeinFrame(2);
    }
    else {
        if(select == false) {
         if(e.getSource() == button1) {
             selectedbutton = button1;
//           button1.setBorder(BorderFactory.createLoweredBevelBorder());
         }
         else if(e.getSource() == button2) {
             selectedbutton = button2;
//           button2.setBorder(BorderFactory.createLineBorder(Color.red));
         }
         else if(e.getSource() == button3) {
             selectedbutton = button3;
//           button3.setBorder(BorderFactory.createLineBorder(Color.red));
         }
         else  if(e.getSource() == button4) {
             selectedbutton = button4;
//           button4.setBorder(BorderFactory.createLineBorder(Color.red));
         }
```

Through the method mouseClicked one button is selected and gets swapped with the next button that is clicked on. This only happens if the new clicked button is not the same as the selectedButton. If thats the case the FatherPanel of the new clicked button gets removed and the selectedButton gets overwritten with the new button.



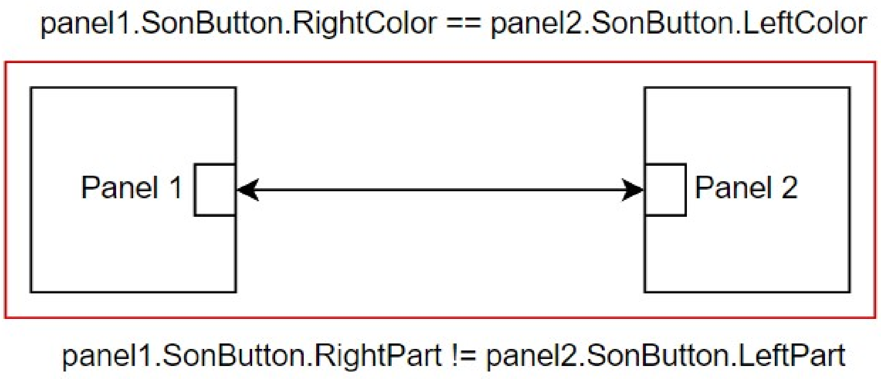
```java
else {

    if(e.getSource() == selectedbutton) {
        selectedbutton.FatherPanel.remove(selectedbutton);
        turn(selectedbutton);
        selectedbutton.FatherPanel.add(selectedbutton);
        selectedbutton.FatherPanel.repaint();
        selectedbutton.FatherPanel.revalidate();
    }
    else if(e.getSource() != selectedbutton && e.getSource() == button1) {
        button1.FatherPanel.remove(button1);
        button1.FatherPanel.add(selectedbutton);
        selectedbutton.FatherPanel.remove(selectedbutton);
        selectedbutton.FatherPanel.add(button1);
        temppanel = selectedbutton.FatherPanel;
        selectedbutton.FatherPanel = button1.FatherPanel;
        button1.FatherPanel = temppanel;
        selectedbutton.FatherPanel.SonButton = selectedbutton;
        button1.FatherPanel.SonButton = button1;
        temppanel = null;

    }
    else if(e.getSource() != selectedbutton && e.getSource() == button2) {
        button2.FatherPanel.remove(button2);
        button2.FatherPanel.add(selectedbutton);
        selectedbutton.FatherPanel.remove(selectedbutton);
        selectedbutton.FatherPanel.add(button2);
        temppanel = selectedbutton.FatherPanel;
        selectedbutton.FatherPanel = button2.FatherPanel;
        button2.FatherPanel = temppanel;
        selectedbutton.FatherPanel.SonButton = selectedbutton;
```

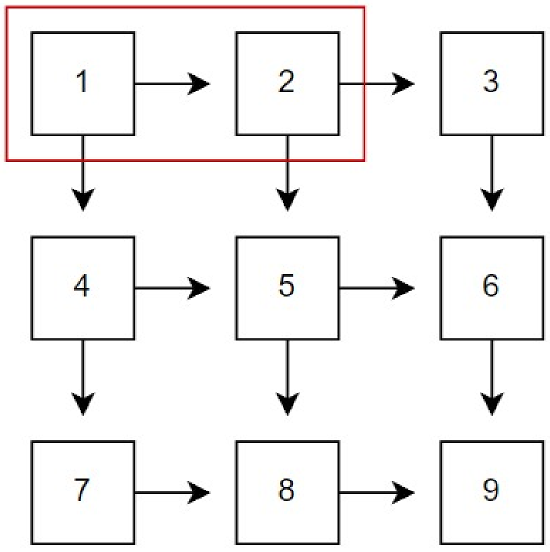
The CheckSolution method consists of one small if statement. The system therefore checks if the RightColor of panel1.SonButton is equal to the LeftColor of panel2.Sonbutton and if the RightPart of panel1.SonButton is not equal to the LeftPart of panel2.SonButton. This goes on for every other panel until panel9 is reached. If the statment is true the final window opens and congratulates the player for finishing the game.



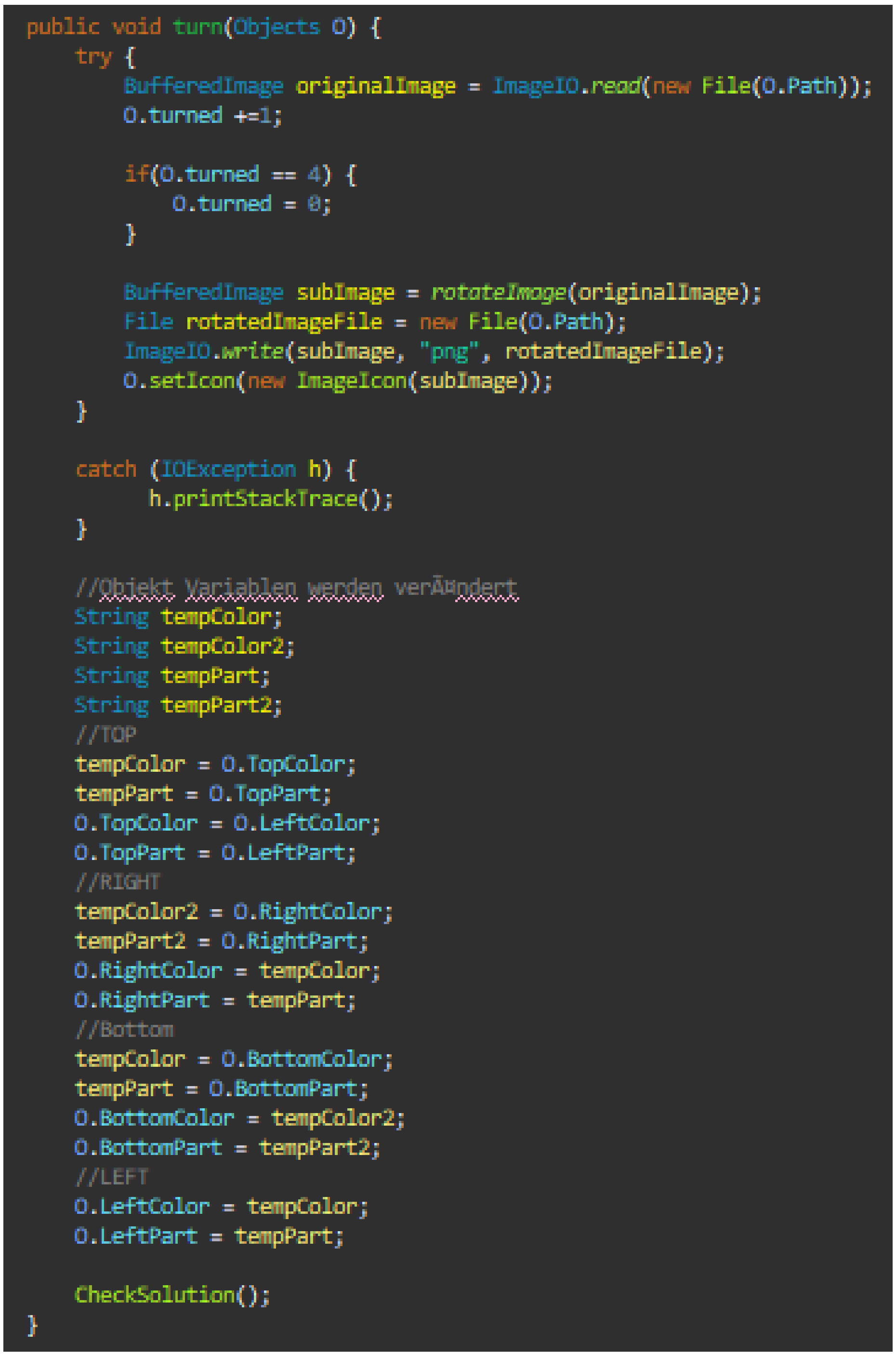
```java
public void CheckSolution() {

 // a function that checks if our game is solved
    if(panel1.SonButton.RightColor == panel2.SonButton.LeftColor && panel1.SonButton.RightPart != panel2.SonButton.LeftPart
            && panel1.SonButton.BottomColor == panel4.SonButton.TopColor && panel1.SonButton.BottomPart != panel4.SonButton.TopPart
            && panel2.SonButton.RightColor == panel3.SonButton.LeftColor && panel2.SonButton.RightPart != panel3.SonButton.LeftPart
            && panel2.SonButton.BottomColor == panel5.SonButton.TopColor && panel2.SonButton.BottomPart != panel5.SonButton.TopPart
            && panel3.SonButton.BottomColor == panel6.SonButton.TopColor && panel3.SonButton.BottomPart != panel6.SonButton.TopPart
            && panel4.SonButton.RightColor == panel5.SonButton.LeftColor && panel4.SonButton.RightPart != panel5.SonButton.LeftPart
            && panel4.SonButton.BottomColor == panel7.SonButton.TopColor && panel4.SonButton.BottomPart != panel7.SonButton.TopPart
            && panel5.SonButton.RightColor == panel6.SonButton.LeftColor && panel5.SonButton.RightPart != panel6.SonButton.LeftPart
            && panel5.SonButton.BottomColor == panel8.SonButton.TopColor && panel5.SonButton.BottomPart != panel8.SonButton.TopPart
            && panel6.SonButton.BottomColor == panel9.SonButton.TopColor && panel6.SonButton.BottomPart != panel9.SonButton.TopPart
            && panel7.SonButton.RightColor == panel8.SonButton.LeftColor && panel7.SonButton.RightPart != panel8.SonButton.LeftPart
            && panel8.SonButton.RightColor == panel9.SonButton.LeftColor && panel8.SonButton.RightPart != panel9.SonButton.LeftPart)
        {
        this.dispose();
        MeinFrame Finished = new MeinFrame();
    }

}
```

The next picture visualizes the mentioned method from before.


panel1.SonButton.RightColor == panel2.SonButton.LeftColor
Panel 1
Panel 2
panel1.SonButton.RightPart != panel2.SonButton.LeftPart

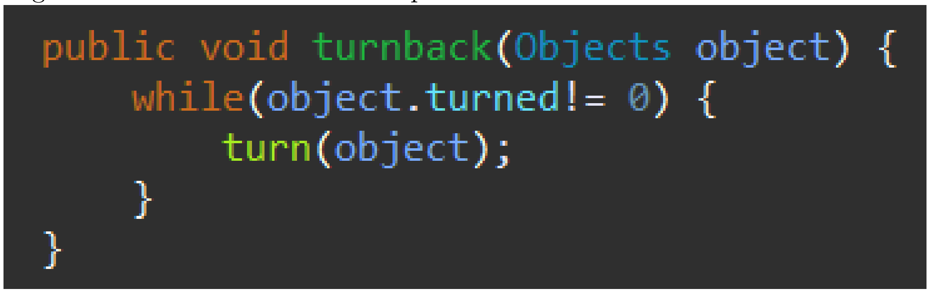


The turn method gets called when a player selects the same button twice. Every button has a counter turned that keeps track of how many times a button rotated while playing the game. The turned value goes up by one and if the value is equal to four the value goes back to zero. Rotating a picture consists of two different parts. First the picture has to be copied and replaced with a ninety degree turned picture of the original image. In the second part the variables of the object get changed. The image gets rotated by ninety degree. Top values replace the right values. Right values replace the bottom values. Bottom values replace the left values and left values replace the top values.



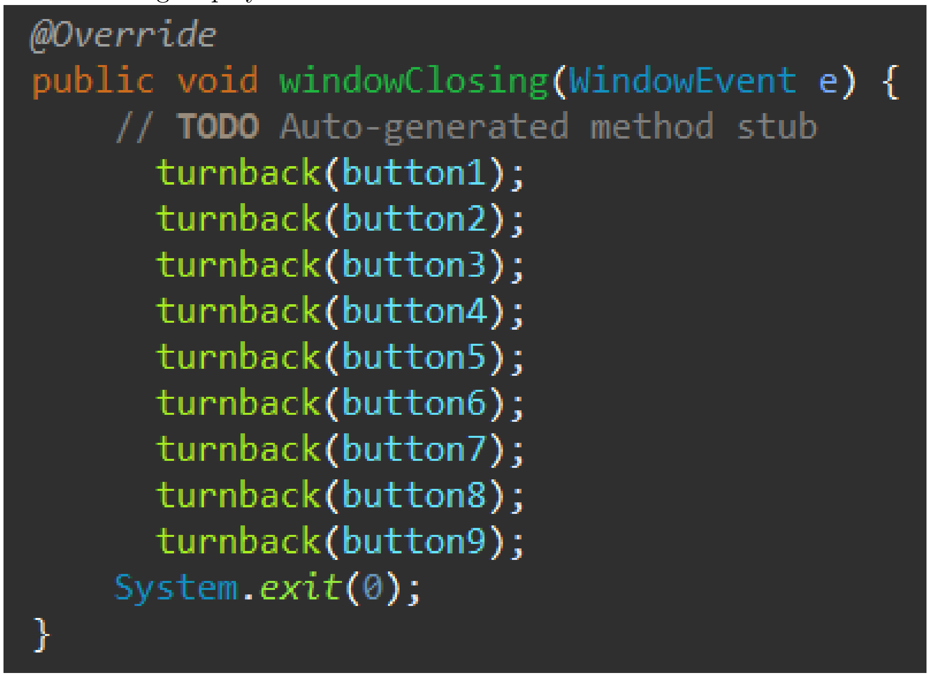
```
public void turn(Objects O) {
    try {
        BufferedImage originalImage = ImageIO.read(new File(O.Path));
        O.turned +=1;

        if(O.turned == 4) {
            O.turned = 0;
        }

        BufferedImage subImage = rotateImage(originalImage);
        File rotatedImageFile = new File(O.Path);
        ImageIO.write(subImage, "png", rotatedImageFile);
        O.setIcon(new ImageIcon(subImage));
    }

    catch (IOException h) {
          h.printStackTrace();
    }

    //Objekt Variablen werden verÃ¤ndert
    String tempColor;
    String tempColor2;
    String tempPart;
    String tempPart2;
    //TOP
    tempColor = O.TopColor;
    tempPart = O.TopPart;
    O.TopColor = O.LeftColor;
    O.TopPart = O.LeftPart;
    //RIGHT
    tempColor2 = O.RightColor;
    tempPart2 = O.RightPart;
    O.RightColor = tempColor;
    O.RightPart = tempPart;
    //Bottom
    tempColor = O.BottomColor;
    tempPart = O.BottomPart;
    O.BottomColor = tempColor2;
    O.BottomPart = tempPart2;
    //LEFT
    O.LeftColor = tempColor;
    O.LeftPart = tempPart;

    CheckSolution();
}
```

Every button has a counter turned that keeps track of how many times a button rotated while the game was open. If the program closes without every piece being rotated back to its initial state the game will initialise the variables wrong the next time it gets opened. The turnback function calls the turn function as long as the turned variable is not equal to zero.



```
public void turnback(Objects object) {
    while(object.turned!= 0) {
        turn(object);
    }
}
```

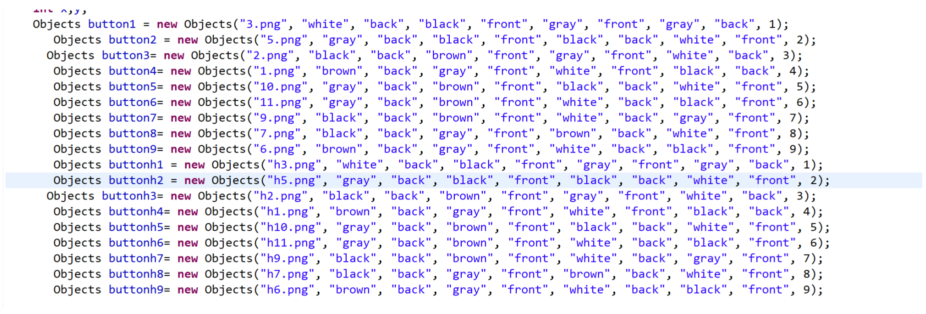
If the window is being closed or the puzzle is correct the function window-Closing gets active. This method is similar to the auto-generated MouseEvent function and needs no extra lines of code. However in this case we add the method turnback for every button so that the game has no problems initializing the next time it gets played.



```
@Override
public void windowClosing(WindowEvent e) {
    // TODO Auto-generated method stub
      turnback(button1);
      turnback(button2);
      turnback(button3);
      turnback(button4);
      turnback(button5);
      turnback(button6);
      turnback(button7);
      turnback(button8);
      turnback(button9);
    System.exit(0);
}
```

8 Experimental results and statistical tests

We tested all of our Methods with different variables and parameters. If any issues occurred we fixed them immediately and our final testing was successful. We checked the functionality of the game by changing variables in the objects/buttons. We changed each variable individually and then played the game. Changing the variables regarding the color and the part of the sheep led as expected to an unsolvable version of the game. We initialized the game with different pictures, but with correct assign variables to the pictures and that led to a perfectly working game. By implementing a second game we can proudly say that our game is adjustable and expandable.

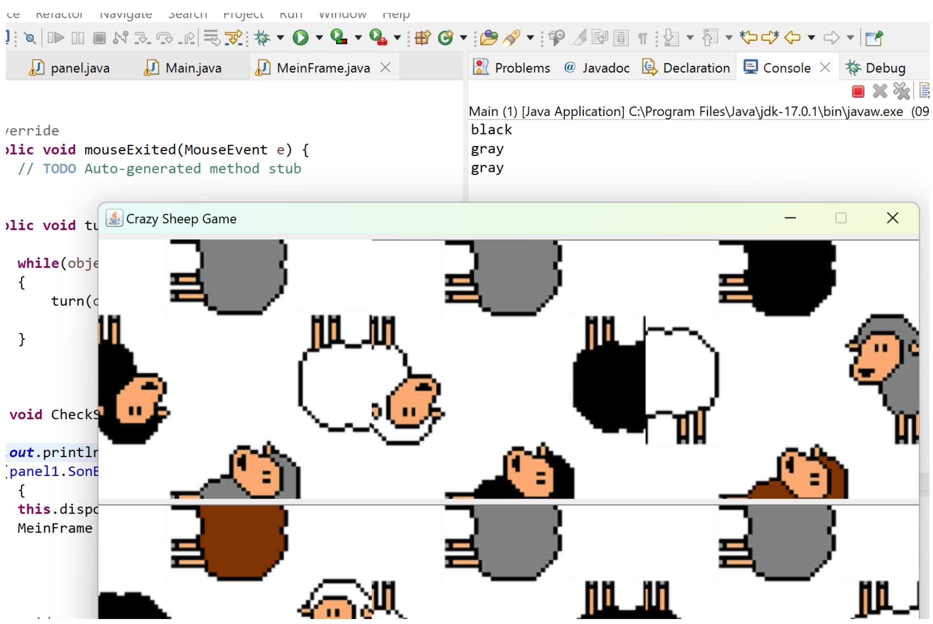


```
Objects button1 = new Objects("3.png", "white", "back", "black", "front", "gray", "front", "gray", "back", 1);
   Objects button2 = new Objects("5.png", "gray", "back", "black", "front", "black", "back", "white", "front", 2);
  Objects button3= new Objects("2.png", "black", "back", "brown", "front", "gray", "front", "white", "back", 3);
   Objects button4= new Objects("1.png", "brown", "back", "gray", "front", "white", "front", "black", "back", 4);
   Objects button5= new Objects("10.png", "gray", "back", "brown", "front", "black", "back", "white", "front", 5);
   Objects button6= new Objects("11.png", "gray", "back", "brown", "front", "white", "back", "black", "front", 6);
   Objects button7= new Objects("9.png", "black", "back", "brown", "front", "white", "back", "gray", "front", 7);
   Objects button8= new Objects("7.png", "black", "back", "gray", "front", "brown", "back", "white", "front", 8);
   Objects button9= new Objects("6.png", "brown", "back", "gray", "front", "white", "back", "black", "front", 9);
   Objects buttonh1 = new Objects("h3.png", "white", "back", "black", "front", "gray", "front", "gray", "back", 1);
   Objects buttonh2 = new Objects("h5.png", "gray", "back", "black", "front", "black", "back", "white", "front", 2);
  Objects buttonh3= new Objects("h2.png", "black", "back", "brown", "front", "gray", "front", "white", "back", 3);
   Objects buttonh4= new Objects("h1.png", "brown", "back", "gray", "front", "white", "front", "black", "back", 4);
   Objects buttonh5= new Objects("h10.png", "gray", "back", "brown", "front", "black", "back", "white", "front", 5);
   Objects buttonh6= new Objects("h11.png", "gray", "back", "brown", "front", "white", "back", "black", "front", 6);
   Objects buttonh7= new Objects("h9.png", "black", "back", "brown", "front", "white", "back", "gray", "front", 7);
   Objects buttonh8= new Objects("h7.png", "black", "back", "gray", "front", "brown", "back", "white", "front", 8);
   Objects buttonh9= new Objects("h6.png", "brown", "back", "gray", "front", "white", "back", "black", "front", 9);
```

For example we switched the names of the buttons. That means the game gets initialized differently, but still if we solve the puzzle we get our success screen. But if we change one color or part to a color that's not existing in our game like red, then even if we solve the puzzle there´s no success screen. So our results regarding the basic function of the game were positive, since the game is working as intended if we initialize it differently and its not working if we change necessary variables. We also tested the rotating and moving of pictures. We implemented lines like

System.out.println(panel1.sonbutton.BottomPart)

to see our variables after every action. Then we started to rotate and move pictures. During the testing we always checked visually if our variables are changing correctly. In more than 10 minutes of testing all variables were correct. Based on that we assume that our functions for rotating and moving are working properly.



```
erride
lic void mouseExited(MouseEvent e) {
  // TODO Auto-generated method stub

lic void tu

  while(obje
  {
       turn(

  }

void Check

out.printl
panel1.SonE
  {
  this.disp
  MeinFrame
```

```
Main (1) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (09
black
gray
gray
```

For example in this picture u can see our console. Before rotating the picture in the first panel the bottomcolor value was black. After rotating it should be grey as u can see. We tested all of our panels like that.

```
//TOP
tempColor = O.TopColor;
tempPart = O.TopPart;
O.TopColor = O.LeftColor;
O.TopPart = O.LeftPart;

//RIGHT
tempColor2 = O.RightColor;
tempPart2 = O.RightPart;
O.RightColor = tempColor;
O.RightPart = tempPart;

//Bottom
tempColor = O.BottomColor;
tempPart = O.BottomPart;
O.BottomColor = tempColor2;
O.BottomPart = tempPart2;

//LEFT
O.LeftColor = tempColor;
O.LeftPart = tempPart;
```

This picture shows how our variables have to change when we rotate a picture. That's working properly. We had different tester that aren't related to the work. Those testers played the game for quiet a long time. Not all of them could solve it but we can proudly say that no issues occurred during the testing. From that time on we assumed that our game is working properly.

Since we barely had to use algorithms our testing phase was short. We only had to test a couple of methods. In these methods we usually wrote the algorithms completely from scratch on our own. Since our ideas for the algorithms always worked we never had to use any other algorithms. We also tried to make our methods as efficient as possible.

In conclusion we can state that our work was completely checked and we didn't find any issues at all. We designed the in our eyes most efficient algorithm for each method so we didn't try a couple of different one that we could compare. During our planning time we discussed these algorithms and all of our game designers agreed the ones we currently have implemented.

9 Conclusion and future work

9.1 Teamwork

In the beginning we needed some time to get used to each other. All of us had lots of ideas and we were brainstorming a lot. After a bit of working on the project we realized that we need a better structure. Then we thought about splitting the work in fundamental aspects that can be done by one person. Like this we made sure that everything that everyone was working on could be included in the final application. We met weekly and discussed the progress of our work and discussed further steps and problems. Since then, our teamwork was very good. We had great communication in Discord/WhatsApp and in person. We were always fast with answering questions and helping each other's. We got along very well and it was easy to work together.

Especially great was that everyone had very good knowledge about the parts they were working on and everyone was able to explain it and answer question very precisely. Our team made a lot from scratch and even thought that took a lot of time and maybe slowed us down a bit, it led to a very good understanding of our working application and that's why everyone was able to tell very clearly what he did. Also everyone was flexible towards the worksplitting and we always found agreements. Like this we were able to work efficiently since everyone took his tasks very seriously. Whenever someone didn't understand something or a problem occurred, then we met on discord shared our screens and discussed every issue. No one was working on his tasks without motivation and we had a very good general progress. All in all we were pleased with our teamwork and it was a great group project for us.

9.2 What we learned

Our team learned a lot through these months. Our first mention should be the incredibly know-how that we had to get about JFrame, JPanels, JButtons, JLabels and the libraries behind these classes. Our knowledge about objects in general increased a lot, because we needed very specific objects and methods for our application. How we address something correctly was very important for our work and we were able to manifest that knowledge very fast. Most of our objects needed an ActionListener and it was a great thing to learn since u are able to visualize it very well. For example finally understanding how a WindowListener or a MouseListener works was great, because then we could implement those and see the change directly. We had to go though a lot of sources for that but we tried a lot around with all of these methods and got used to them. The rotation and switching of pictures was very challenging, since at the beginning our frame wouldn´t even refresh. But these challenging functions gave us such a great understanding when we finally managed to solve them.

This was the first UI we worked on and we had a lot of fun and learned a lot especially because everything we did impacted the frame that we were seeing. We learned a lot about the usage of images in applications and we designed most images ourself. Through that we also practiced our design skills and being creative was good for us as well. It also gave us a better idea how to structure an application and the necessary work. Designing and discussion the application first and making a good structure is very important and after our first week we made it a high priority.

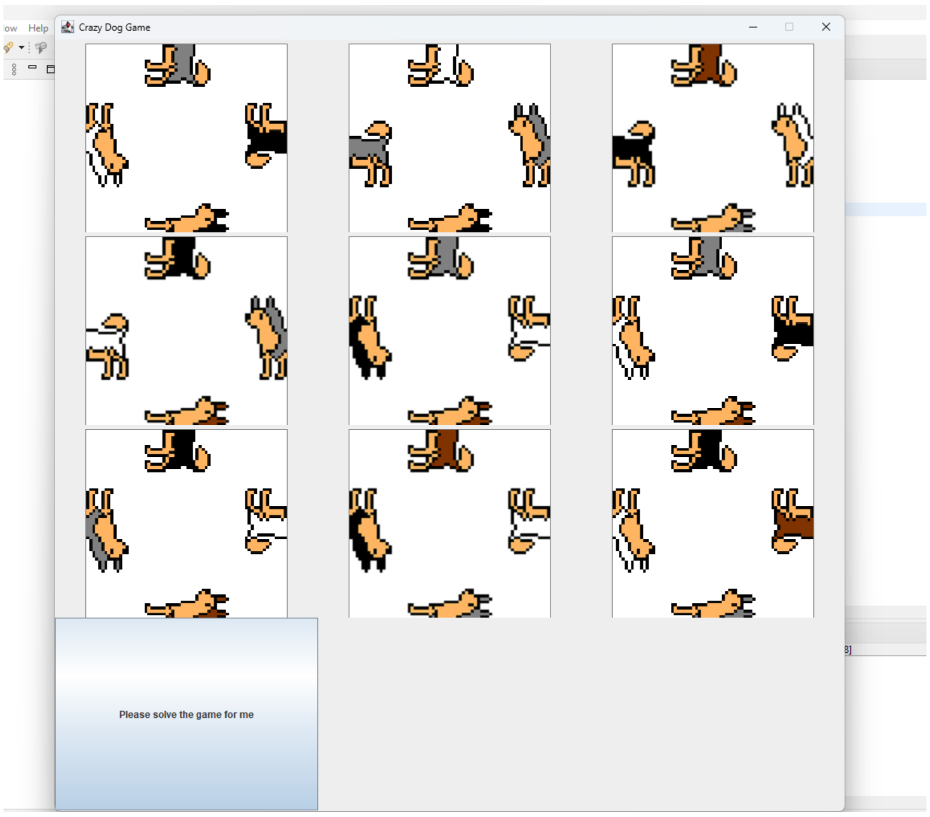
We also looked at a couple of algorithms or ideas on how to write a function that solves the game on its own. And even though we couldn´t implemented our designed algorithm correctly, it gave us a lot to think about and we got a first introduction to very complex algorithms.

We also improved our teamwork a lot and we got used to writing applications in groups. Which worked very well for us. Our understanding of the structure of an application like this and also our new knowledge about splitting work in relevant tasks is something that we take home from this project. In Conclusion we can definitely say that we learned a lot and challenging us led to a great increase in our knowledge of java.

9.3 Ideas for future developement

We designed our application in an expandable way and we would like to make a 4x4 game and include it as well. This would be a different type of game and it would maybe have a different motivation for players. We also wanted to add a button that gives a hint. Like “that picture is rotate correctly” or “this picture belongs in the second spot” since the game is kind of hard to solve for some people.

Down below we started working on a button that solves the puzzle for you, where you can watch the movement of the pictures. Sadly, we didn´t manage to implement that correctly until the handing in of this report. It was a non-necessary part and we wanted to make sure that everything is running correctly and in the end there was not enough time work on this extrapart. But its something that we´ll definitely look into in the future.



Our team also thought about a more beautiful UI. There are a couple of things that we´ll investigate in the future. For example, a "Moving "Grassbackground" or a save my game button.

All of these things are tasks that weren´t necessary in our eyes but as a development team we would love to improve our work much more in the future. But since the time ran out our focus was on making the application without and non-working parts. We´re happy that we managed to implement a second game in time and you can be exited since this application made us motivated to keep working and learning and improving our coding skills.

10 References

10.1 Links

[Number] Description: Link

[1]Crazy Sheep Game by Gillian Everett:
https://www.flickr.com/photos/gilleverett/8650751040

[2]Crazy Sheep (16 Piece):
https://www.mightyape.co.nz/product/crazy-sheep-16-piece/21015426

[3]Gailer Tutorials:
http://www.gailer-net.de/tutorials/java/Notes/chap62/ch62_19.html

[4]"3 - How to close a JFrame while opening another one? - Java Swing Projects" by KAMLESHUTUBE
https://www.youtube.com/watch?v=sDoAH9_Aj2k

[5]"Learning Java: Part 28: Image Rotation" by BrandonioProductions
https://www.youtube.com/watch?v=vxNBSVuNKrc

[6]"Java GUI: Full Course" by Bro Code
https://www.youtube.com/watch?v=Kmgo00avvEw

[7]"Programa de Búsqueda primero en profundidad y en anchura (GRAFOS) en JAVA" by Tony
https://www.youtube.com/watch?v=2QIVfIgRvyI

10.2 Image Descriptions