



Object Oriented Programming Project4 Report

Sudoku Game

Object Oriented Programming Class 01

Mon 1 / Wed 1,2

Professor : Bong-Soo Sohn

< Team 02 >

20193057 김승아

20190323 배인경

20191673 이주연

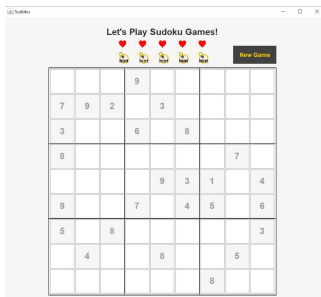
(a) project title, list of team members, brief project description (summary)

1) Project title
: Sudoku Game

2) List of team member
: 20193057 김승아, 20190323 배인경(Leader), 20191673 이주연
: All members belong to the Department of Computer Science & Engineering 19

3) Presentation speaker name
: 20193057 김승아, 20190323 배인경

4) Brief project description
: Sudoku game



The cells that make up Sudoku are subdivided into 81 cells (3x3) in total. It implemented rules that should be followed in Sudoku and added life and hint function.

: mini game for hint

To get a hint from the Sudoku game, you have to play mini games. Minigames are offered randomly among rock-paper-scissors, oddeven, and timing. Every time you try to get a hint, your chances decrease.

(b) How to compile and execute. system requirement for compilation and execution

1. Prepare the Eclipse IDE for Java Developers or IntelliJ IDEA



2. Open the Project file that you want to execute.

3. Eclipse IDE works compilation and execution at once when pressing run (ctrl + F11)

► System requirement(based on Eclipse IDE)

→ HW

: 2GB RAM, Recommend 1.8 GHz quad-core processor or above

→ OS (64-bit recommended,)

: Windows 7 service pack 1 / Windows 8.1 / Windows 10 (version 1703 or above)

(c) description on functionality of your software (what is working in your SW system).

1) Sudoku game

: Sudoku consists of 81 cells in total and is subdivided into 9cells (3 X 3).

Rules to be observed are as follows:

- Each row and column contains 1 to 9 without any overlap.
- In 3x3 boxes, 1 to 9 are included one by one without overlapping.

Additional features:

- Life: Life decreases if the value is wrong.
- Hint: Get a hint for one cell. You have to pass the mini-game.

2) Controller

: Controller update the game data according to the user's input and informed the view of the changes again.

3) Model

: Model handle the status and logic of game data.

4) View

: View shows the data of the model to the user and operates by passing the user's input to the controller.

5) Mini game

: The user performs a mini-game to get a hint.

If a user succeeds, user gets a hint and but if user fails, does not get one.

Mini games are given randomly among the games below.

- Rock, paper, scissors.
- OddEven
- Timing

(d) how you implemented (important implementation issues)

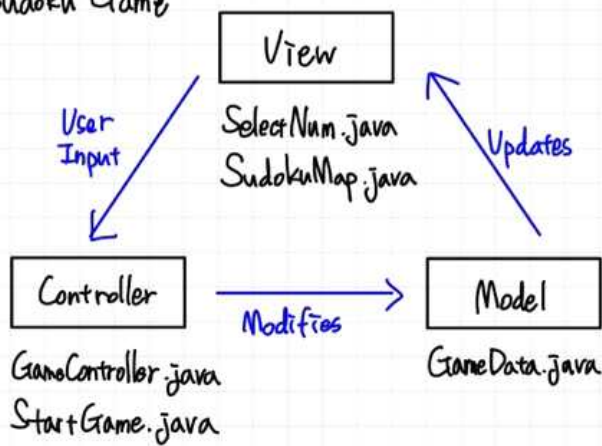
1) Initial Design Note

① Language : C++ / Java

↳ for GUI Game Program

② OOP → MVC Pattern / Overriding / Class → Bottom-Up Approach

③ Sudoku Game



④ MiniGame (for Hint)

MiniGame.java

↳ Odd / Even Game .java

↳ Rock Paper Scissors Game .java

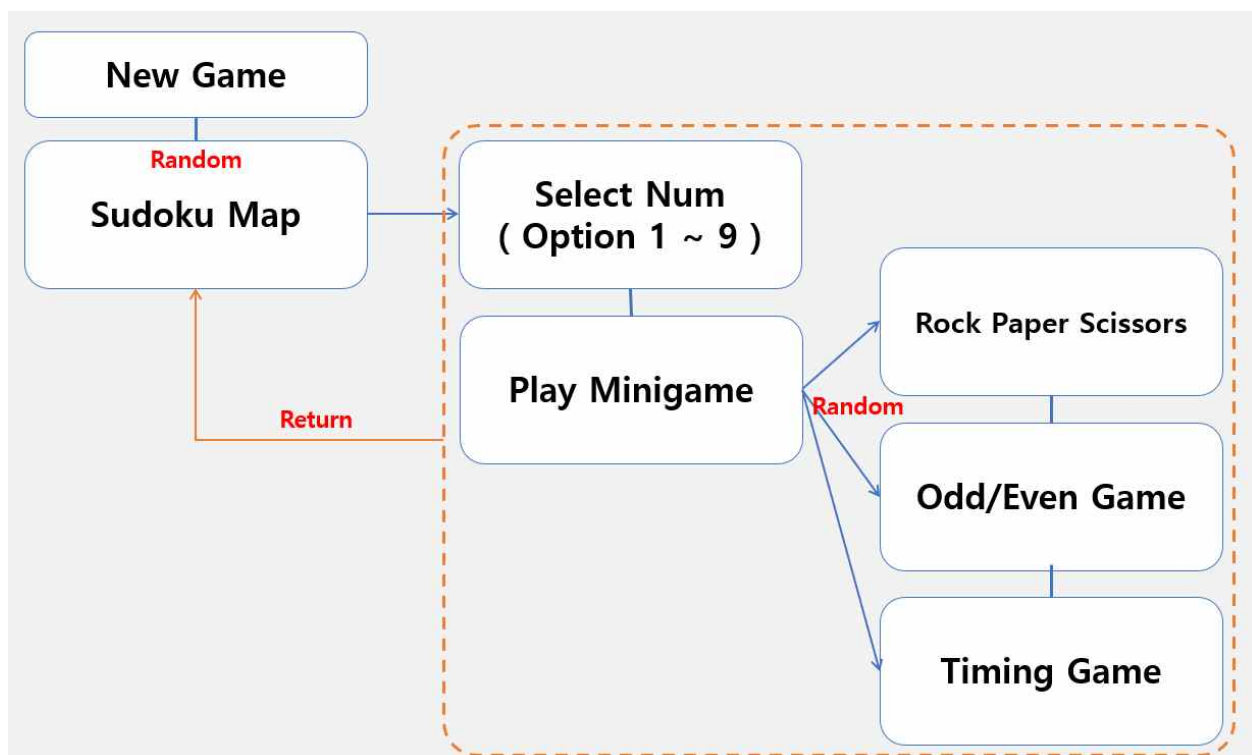
↳ Timing Game .java

↳ Result Dialog.java

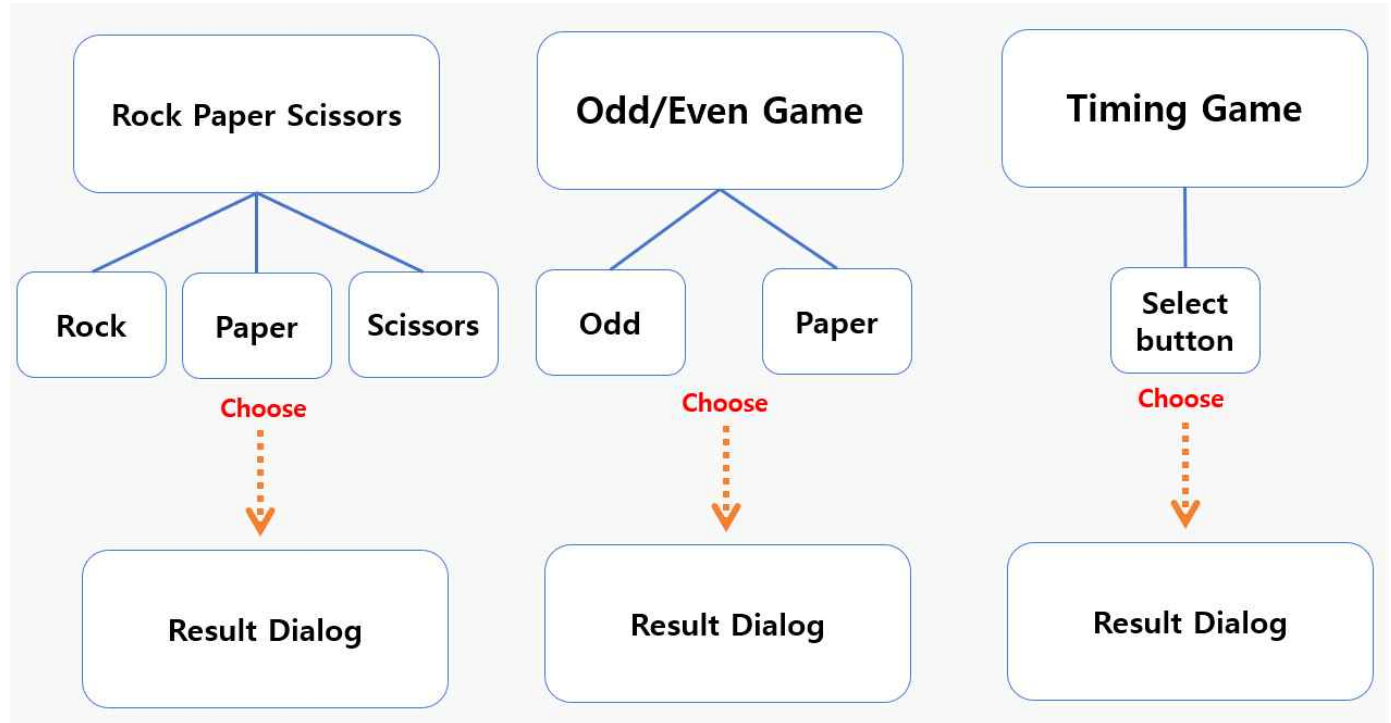
Updates

2) GUI Design Map (Confirm Game GUI options to implement)

: Sudoku Screen



: Minigame Screen



3) Sudoku Game Algorithm

① checkValid

```
96 private boolean checkValid(int[][] game, int index, int[] solutionsNumber) {
97     if (index > 80)
98         return ++solutionsNumber[0] == 1;
99
100     int x = index / 9;
101     int y = index % 9;
102
103     if (game[x][y] == 0) {
104         List<Integer> numbers = new ArrayList<Integer>();
105         for (int i = 1; i <= 9; i++)
106             numbers.add(i);
107
108         while (numbers.size() > 0) {
109             int number = getPossibleNumber(game, y, x, numbers);
110             if (number == -1)
111                 break;
112             game[x][y] = number;
113
114             if (!checkValid(game, index + 1, solutionsNumber)) {
115                 game[x][y] = 0;
116                 return false;
117             }
118             game[x][y] = 0;
119         }
120     } else if (!checkValid(game, index + 1, solutionsNumber))
121         return false;
122
123     return true;
124 }
```

:1. Check if a solution is found.

2. X (of current field) is found by finding the remainder of the division of the current index by the count of fields in a row using the modulo operation.
3. Y (of current field) is found by dividing the current index by the count of fields in a row.
4. An ArrayList is filled with the numbers 1 to 9 and shuffled. Shuffling is important because otherwise you always get the same solution.

② getPossibleNumber

```

82 private int getPossibleNumber(int[][] game, int x, int y, List<Integer> numbers) {
83     while (numbers.size() > 0) {
84         int number = numbers.remove(0);
85         if (checkPossibleX(game, y, number) && checkPossibleY(game, x, number)
86             && checkPossibleBlock(game, x, y, number))
87             return number;
88     }
89     return -1;
90 }

```

: The method `getPossibleNumber(int[][], int, int, List<Integer>)` is used for obtaining the next possible number. It takes a number from the list and checks whether it is possible at the given x and y position in the given game. When found possible, the number is returned. If the list is empty and thus no number is possible at this location, -1 is returned.

③ generateGame

```

127 private int[][] generateGame(int[][] game) {
128     List<Integer> positions = new ArrayList<Integer>();
129     for (int i = 0; i < 81; i++)
130         positions.add(i);
131     Collections.shuffle(positions);
132     return generateGame(game, positions);
133 }
134
135 private int[][] generateGame(int[][] game, List<Integer> positions) {
136     while (positions.size() > 0) {
137         int position = positions.remove(0);
138         int x = position % 9;
139         int y = position / 9;
140         int temp = game[y][x];
141         game[y][x] = 0;
142
143         if (!checkValid(game))
144             game[y][x] = temp;
145     }
146
147     return game;
148 }
149

```

: Generating a game is simply achieved by constantly removing a random field and making sure the game is still valid. Valid means there is only one solution. This is achieved by the

methods below. The user should call the first method, which uses the second method. I will describe the steps again.

④ checkVaild

```
92④ private boolean checkValid(int[][] game) {
93     return checkValid(game, 0, new int[] { 0 }); // 값 대신 참조로 정수를 전달
94 }
95
96④ private boolean checkValid(int[][] game, int index, int[] solutionsNumber) {
97     if (index > 80)
98         return ++solutionsNumber[0] == 1;
99
100     int x = index / 9;
101     int y = index % 9;
102
103     if (game[x][y] == 0) {
104         List<Integer> numbers = new ArrayList<Integer>();
105         for (int i = 1; i <= 9; i++)
106             numbers.add(i);
107
108         while (numbers.size() > 0) {
109             int number = getPossibleNumber(game, y, x, numbers);
110             if (number == -1)
111                 break;
112             game[x][y] = number;
113
114             if (!checkValid(game, index + 1, solutionsNumber)) {
115                 game[x][y] = 0;
116                 return false;
117             }
118             game[x][y] = 0;
119         }
120     } else if (!checkValid(game, index + 1, solutionsNumber))
121         return false;
122
123     return true;
124 }
125
```

: A valid game has in every row, every column, and every region the numbers 1 to 9. Additionally, there should only be one solution existing. To achieve this, all open fields are filled with the first valid value. Even after finding a solution, the search continues by putting the next valid value in an open field. If a second solution is found, then the search will be stopped and the method returns false. There will always be at least one solution (hence game is an incomplete solution), so if there are less than two solutions, the game is valid and the method returns true.

⑤ checkInputNum

```
197 public void checkInputNum(int input) {  
198  
199     for (int x = 0; x < 9; x++) {  
200         for (int y = 0; y < 9; y++)  
201             check[x][y] = input == solution[x][y];  
202     }  
203  
204 }
```

: With checking user input, we compare each field in the game against the corresponding field in the solution.

4) Minigame Algorithm

① abstract class

```
9 abstract class MiniGame extends JFrame{  
10 public MiniGame() {  
11 }  
12 public abstract void run();  
13 protected abstract void init();  
14 protected abstract void start();  
15 public abstract int getResult();  
16  
17 protected ImageIcon ImageSetSize(ImageIcon icon, int width, int height) {  
18     Image ximg = icon.getImage();  
19     Image yimg = ximg.getScaledInstance(width, height, Image.SCALE_SMOOTH);  
20     ImageIcon xyimg = new ImageIcon(yimg);  
21     return xyimg;  
22 }  
23 }
```

To implement the foundation of Minigame, we used abstract classes and let each actual minigame inherit abstract classes.

Abstract methods are as follows.

- run() to activate the game screen
- Init() for initial screen configuration
- Start() to run the actual game
- getResult() to return game results

Methods for image sizing are also defined.

- ImageSetSize()

② Timing

- Random selection of pictures that need to be matched

```
72 @Override
73 protected void init() {
74     //image icon으로 변환
75     icons = new ImageIcon[11];
76     for(int i = 0; i < 11; i++) {
77         icons[i] = new ImageIcon(image_route[i]);
78         icons[i] = ImageSetSize(icons[i], 180, 180);
79     }
80
81     //inst 맞추어야 하는 그림
82     inst = new JLabel();
83     //무작위로 이미지 하나 선택
84     Random r = new Random();
85     inst_idx = r.nextInt(image_route.length-1);
86     String r_image_route = image_route[inst_idx];
87     //사진 삽입
88     ImageIcon inst_icon = new ImageIcon(r_image_route);
89     inst_icon = ImageSetSize(inst_icon, 180, 180);
90     inst.setIcon(inst_icon);
```

Randomly select one of the 10 pictures to display at the top of the minigame. We used random classes to get integers and use them as indexes for image arrays.

- An image that automatically changes over a certain interval of time.

```
47 private void changeImage(JLabel target) {
48     Timer timer = new Timer();
49     TimerTask task = new TimerTask() {
50         @Override
51         public void run() {
52             if(ischange) {
53                 if(index==10) {index=0;}
54                 target.setIcon(icons[index]);
55                 index++;
56             }
57             else if(!ischange) {
58                 timer.cancel();
59             }
60         }
61     };
62     timer.schedule(task, 0, 500);
63 }
```

Using Timer class, the picture was changed once every 0.5 seconds. When the player presses a button, the timer stops, and ischange() returns whether or not to click the button.

④ OddEven

- Determining player and computer values

```
@Override
protected void start() {
    ActionListener buttonlistener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            //player 선택
            //odd
            if(e.getSource().equals(odd)) {
                odd.setBackground(new Color(255, 0, 0));
                even.setEnabled(false);
                player = 1;
            }
            else { //even
                even.setBackground(new Color(255, 0, 0));
                odd.setEnabled(false);
                player = 0;
            }

            //computer
            Random r = new Random();
            computer = r.nextInt(2);
            if(computer == 1) {
                computer_label.setIcon(odd.getIcon());
            }else{
                computer_label.setIcon(even.getIcon());
            }

            //view result
            resultdialog.setResult(getResult());
            resultdialog.setVisible(true);
        }
    };
    odd.addActionListener(buttonlistener);
    even.addActionListener(buttonlistener);
}
```

Player: When the button is pressed, the value is determined by 1 Odd and 0 even. In gui, the button turns red and enabeled.

Computer:

The value was determined using the Random class. The computer value is determined between the 0 or 1 and displays a plot corresponding to the value of the gui.

-Result

```
@Override
public int getResult() {
    if(computer==player) {
        return 1;
    }
    return 0;
}
```

If the computer and the user have the same value, it is a success. else it is fail.

③ RockPaperScissors

- Result

```
@Override
public int getResult() {
    //0:주먹 1:보 2: 가위
    //user관점에서의 결과 0: fail, 1: win, 2:draw
    int[][] result_str = {
        {2, 0, 1}, //user: 주먹
        {1, 2, 0}, //user: 보
        {0, 1, 2} //user: 가위
    };
    return result_str[this.player][this.computer];
}
```

The result of the rock-paper-scissors game was determined using a two-dimensional array. The row is the user's value, the column is the computer's value, and if you win, you return 1 if you lose, 0 if you lose, and 2 if you tie.

- Determine computer value

```
//랜덤 컴퓨터 선택
Random r = new Random();
computer = r.nextInt(3);
switch(computer) {
    case 0:
        computer_label.setIcon(rock.getIcon());
        break;
    case 1:
        computer_label.setIcon(paper.getIcon());
        break;
    case 2:
        computer_label.setIcon(scissor.getIcon());
        break;
}

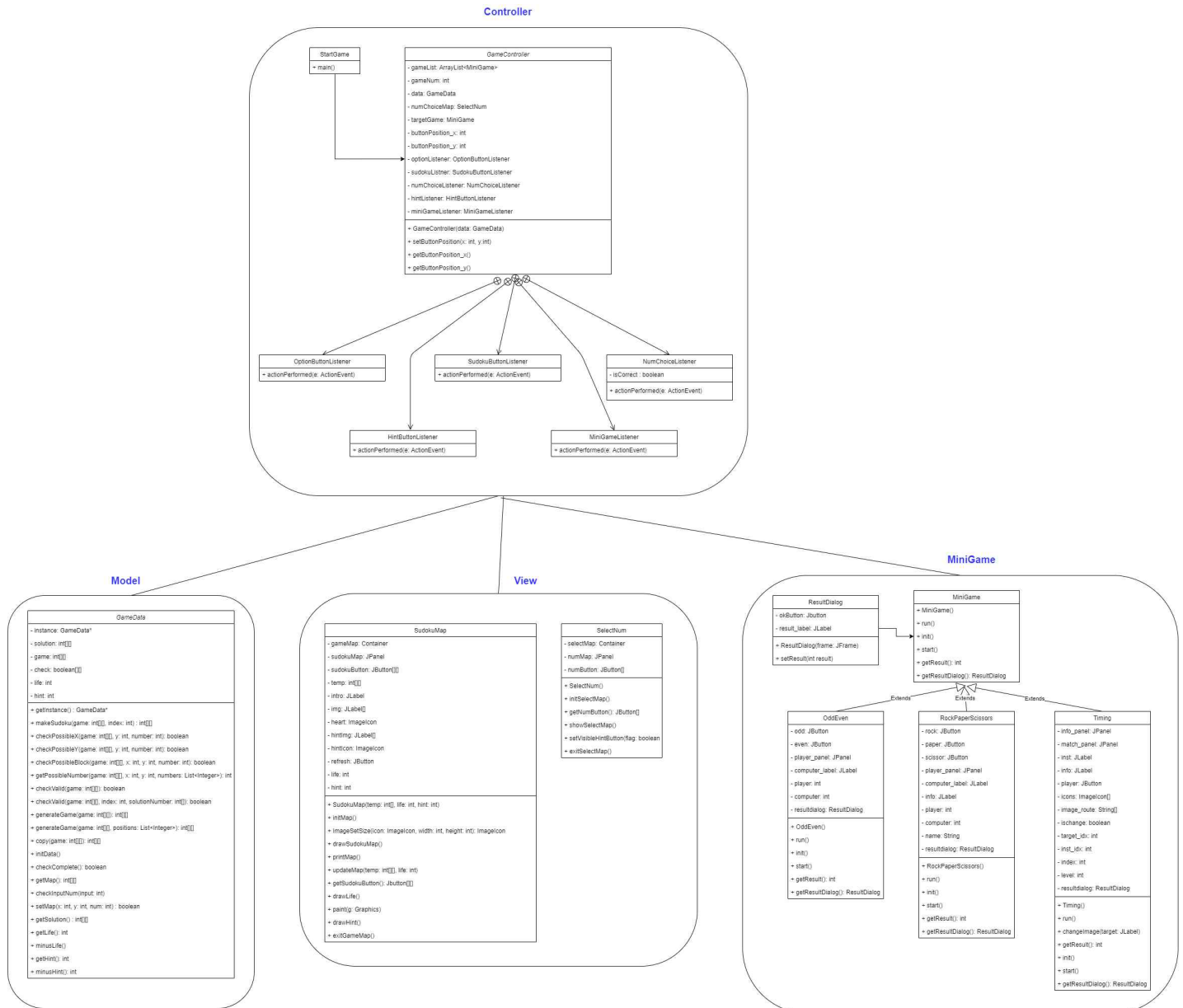
//게임 결과
//view result
resultdialog.setResult(getResult());
resultdialog.setVisible(true);
};
```

The value was determined using the Random class. The computer value displays a plot corresponding to the value of the gui.

(e) the result of UML modeling for system design

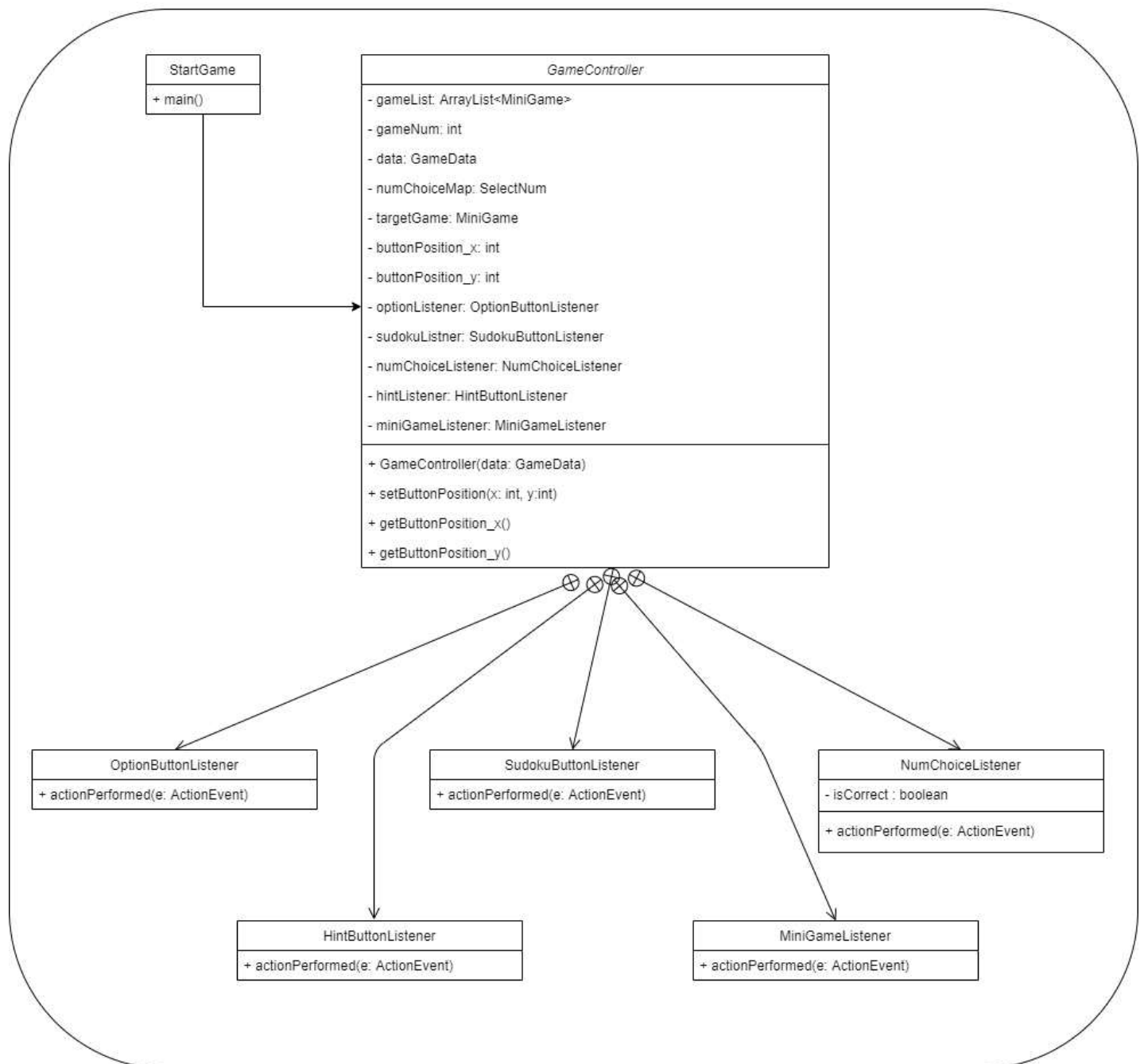
(please include "class diagram", "use-case diagram", "activity diagram" in your modeling)

1) Total UML



2) Part of UML (Controller)

Controller



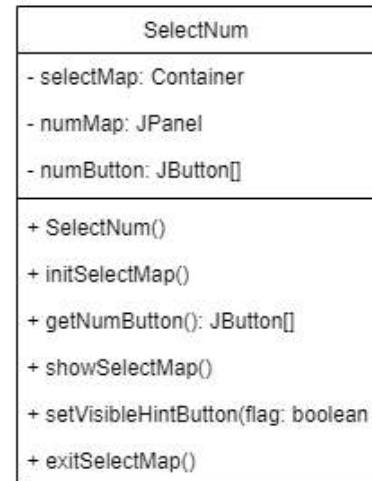
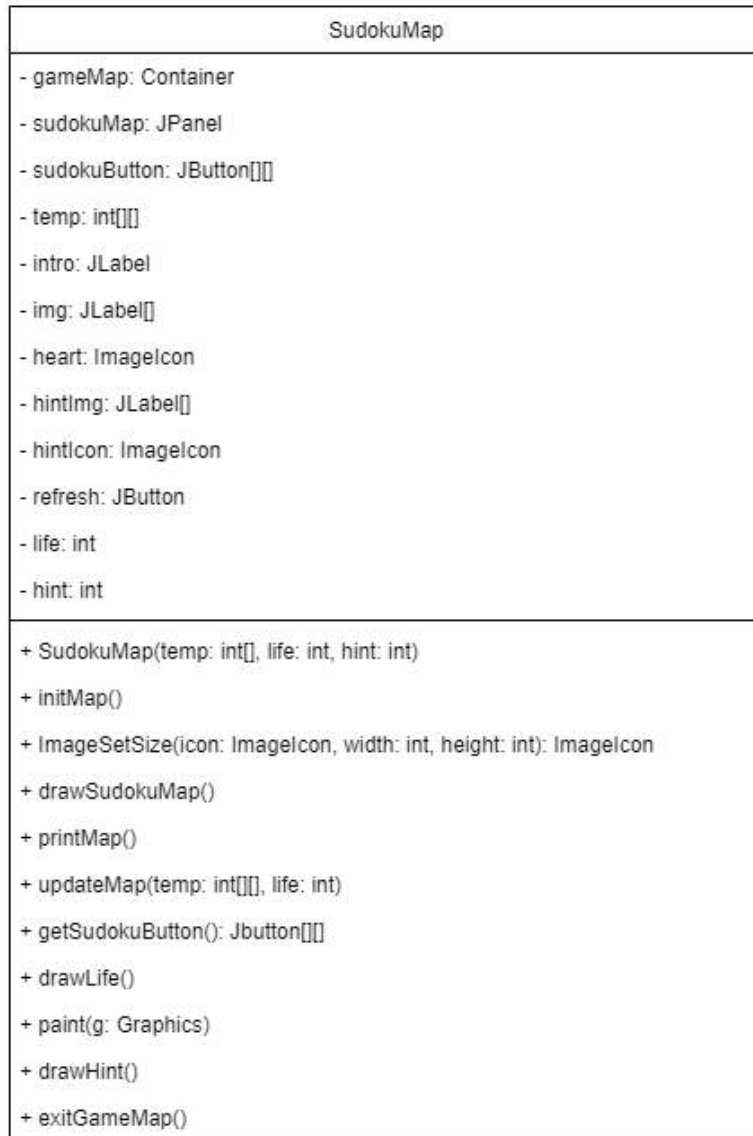
3) Part of UML (Model)

Model



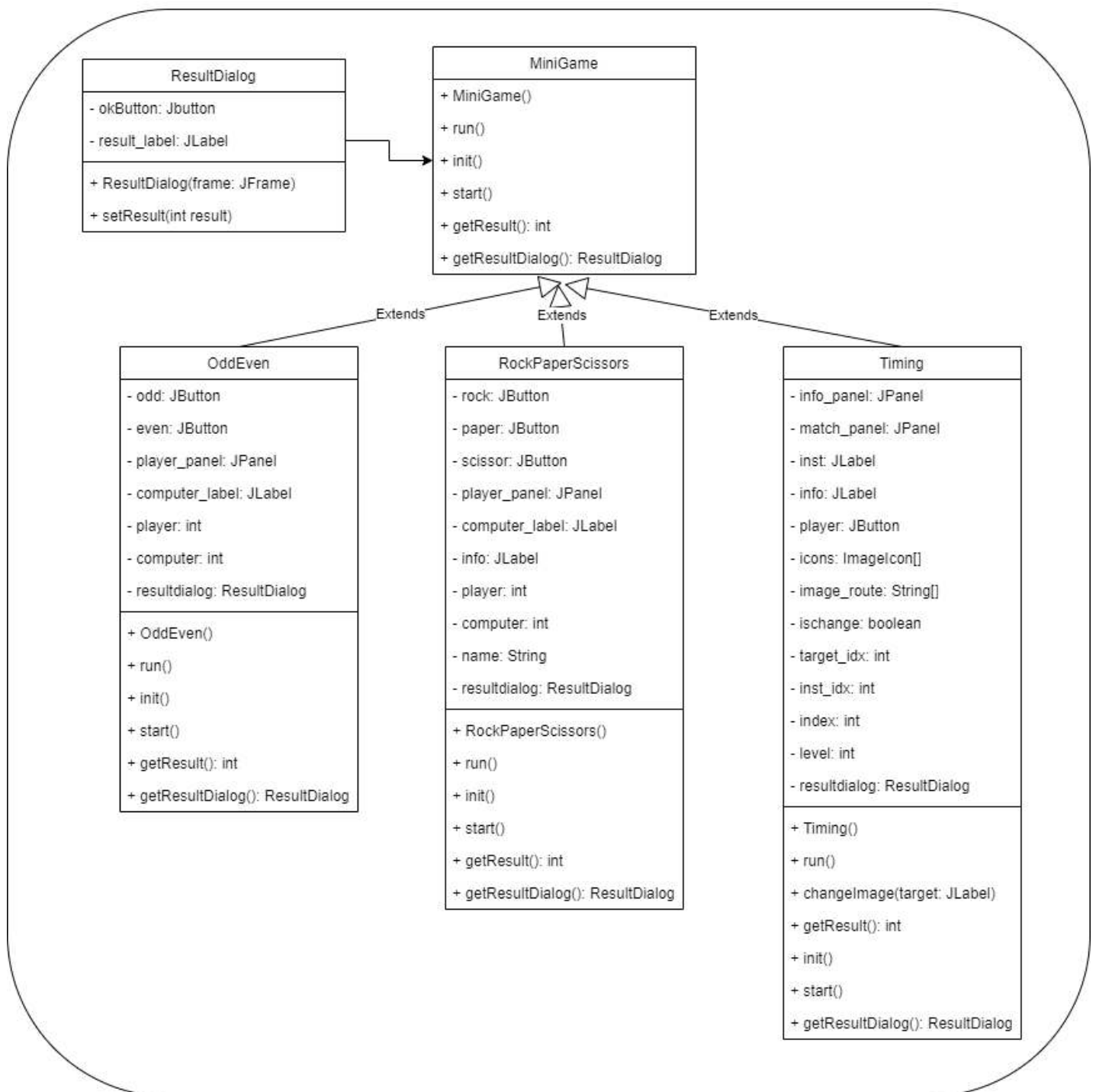
4) Part of UML (View)

View



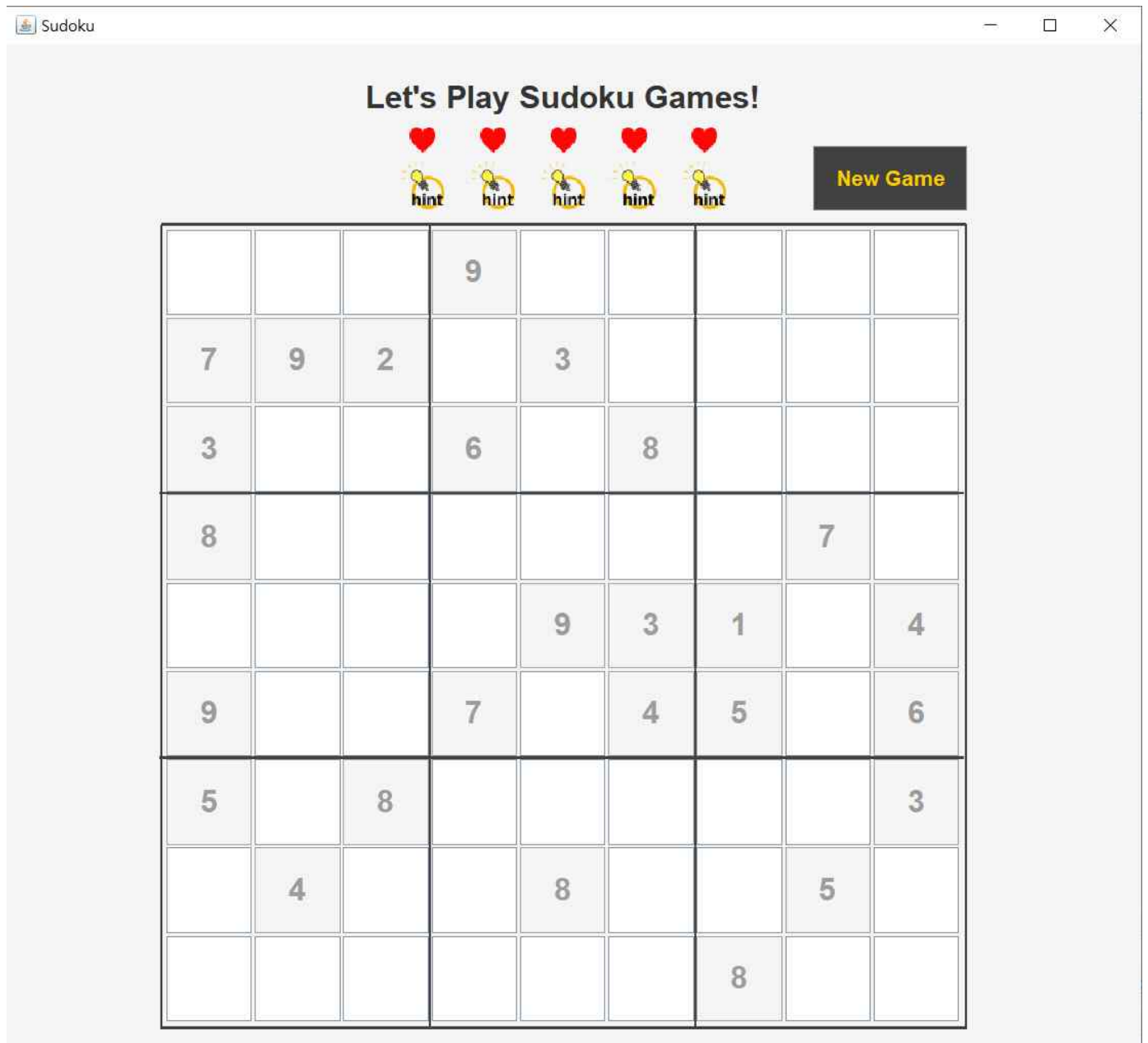
5) Part of UML (MiniGame)

MiniGame



(f) execution results : show real examples of program execution. (use screen capture)
show that each function (기능) of the SW system is working correctly.

1) Sudoku Map (Initial Screen)

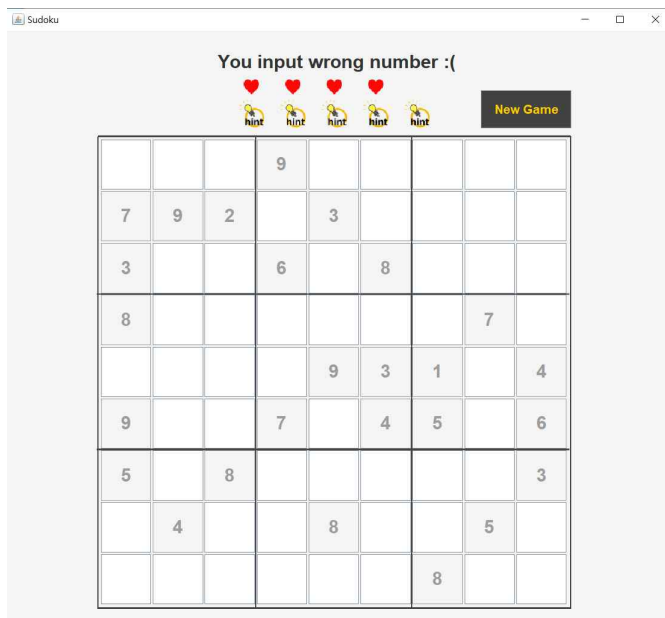


2) When you click the blank button on the Sudoku map



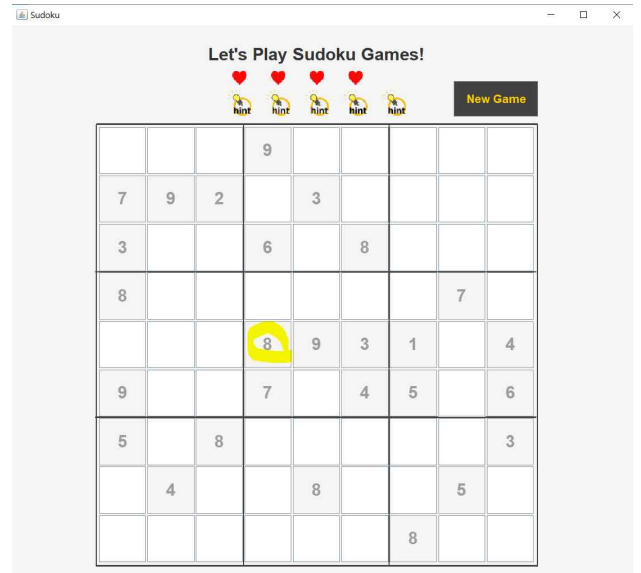
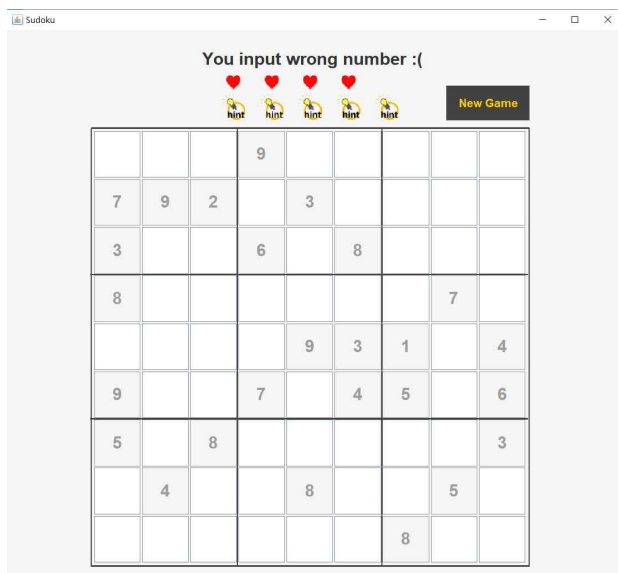
3-1) When you input wrong number

=> -1 Heart, Top message replaced with "You input wrong number :{"

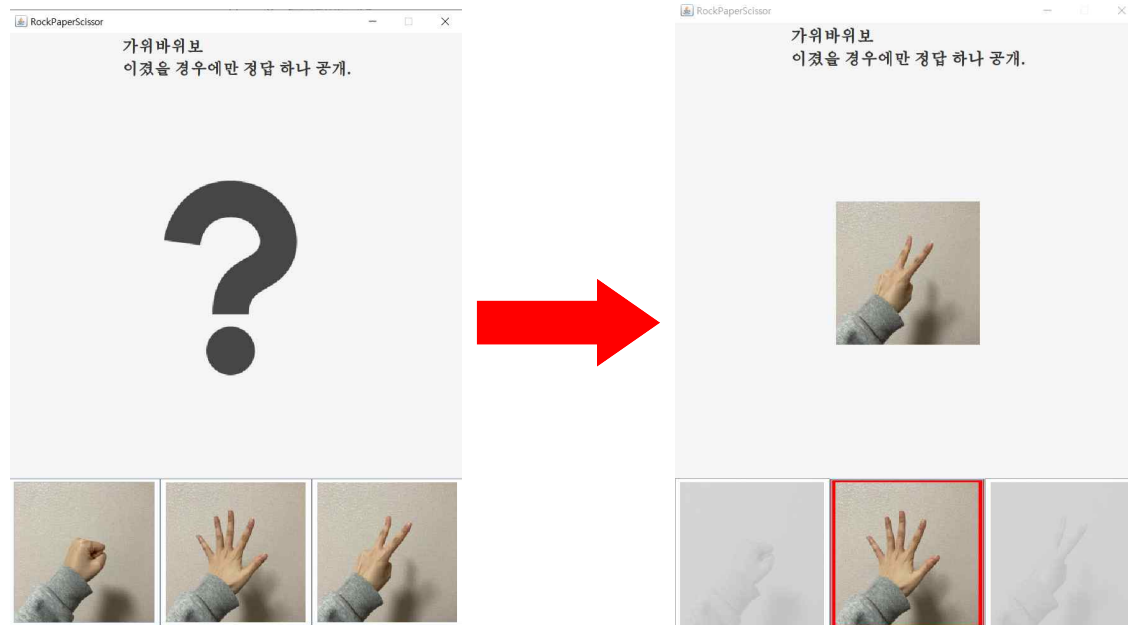


3-2) When you input right number

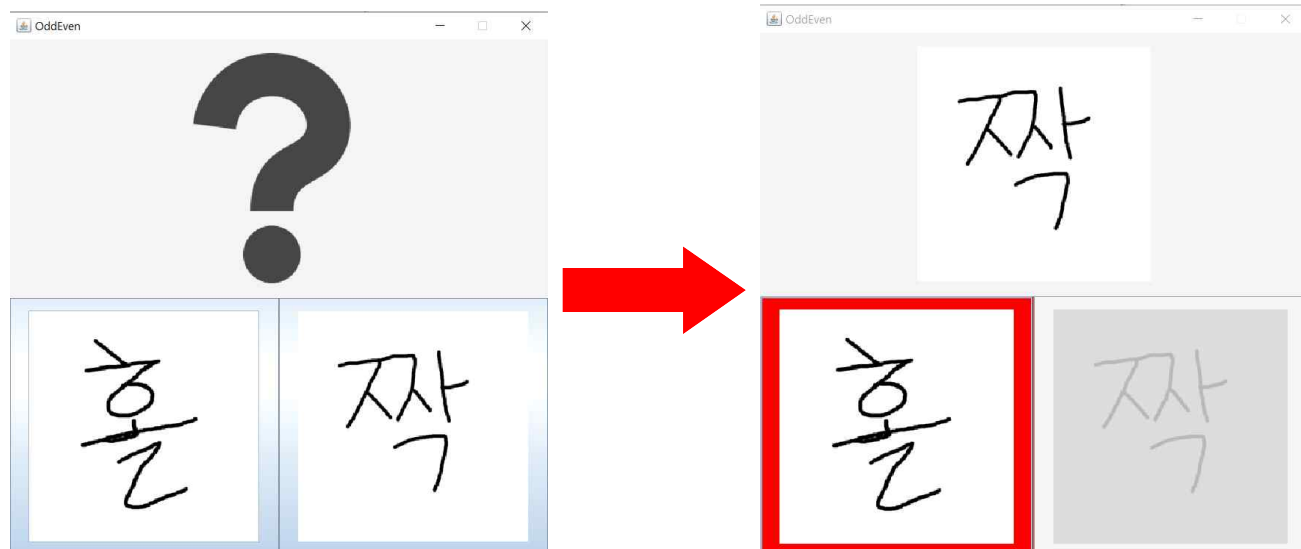
=> No Heart Change, Blanks filled with correct answer, Top message replaced with "Let's Play Sudoku Games!"



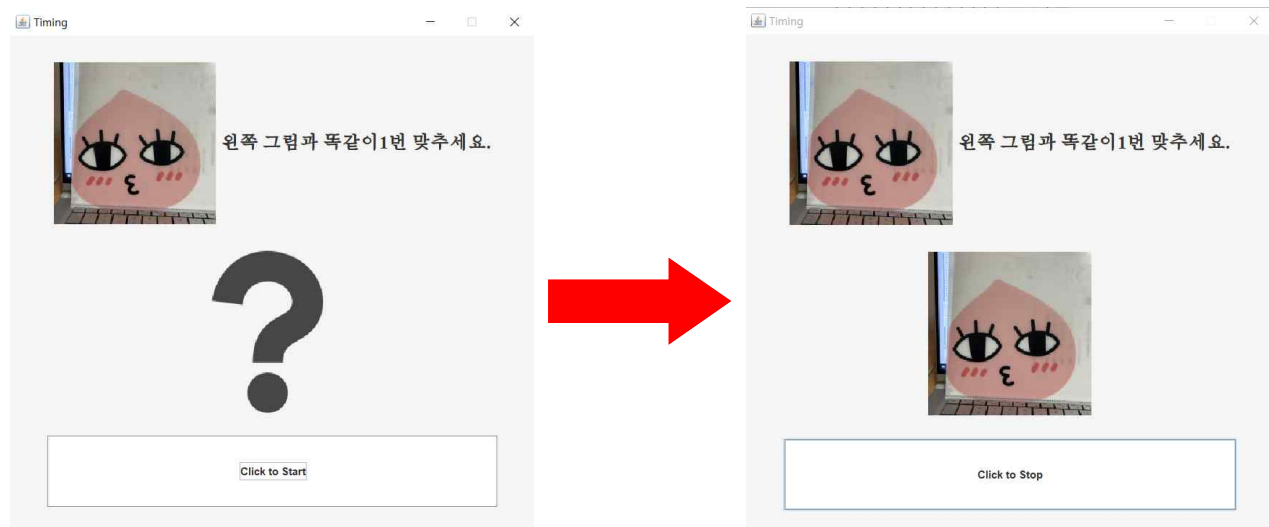
4-1) Minigame_ver1_RockPaperScissors



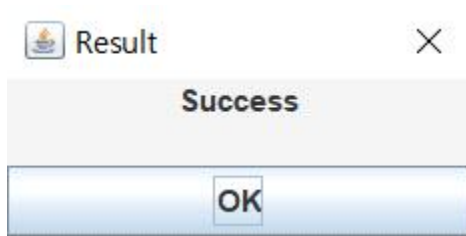
4-2) Minigame_ver2_Odd/Even Game



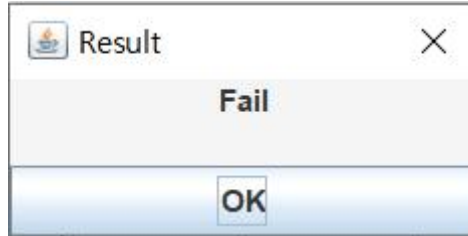
4-3) Minigame_ver3_Timing Game



5) When Minigame Finish

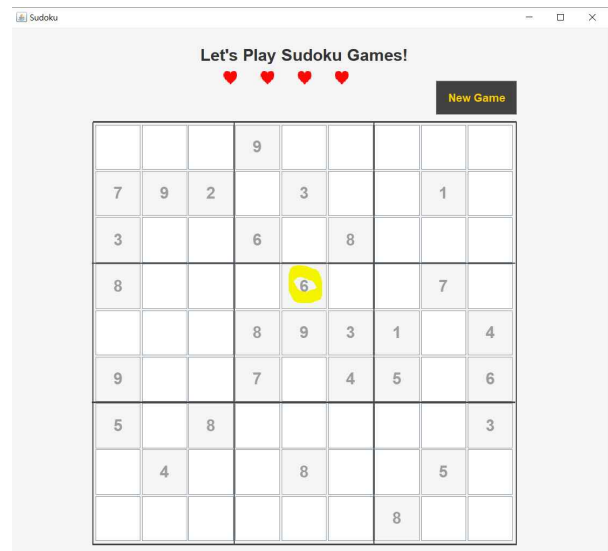
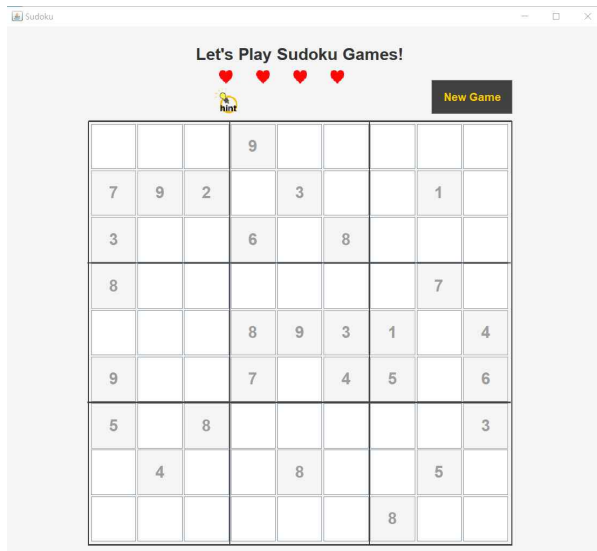


or



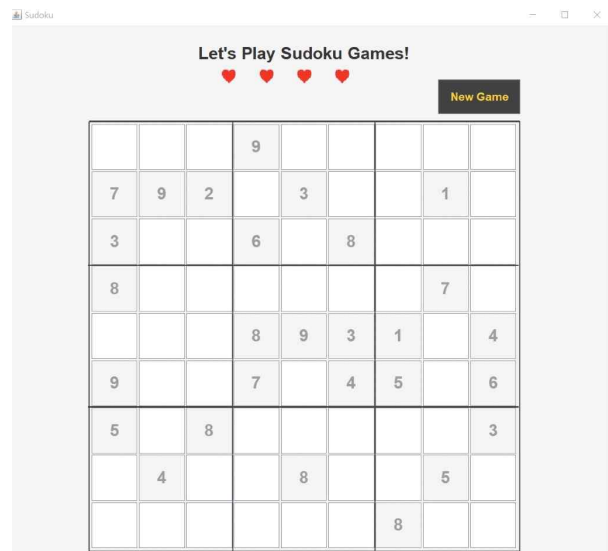
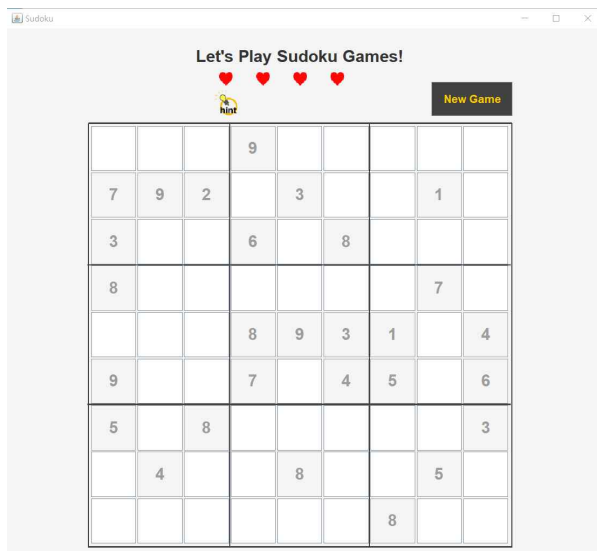
5-1) When you success the Minigame (Click the OK Button of Result Dialog_(5))

=> -1 Hint, No Heart Change, Blanks filled with correct answer



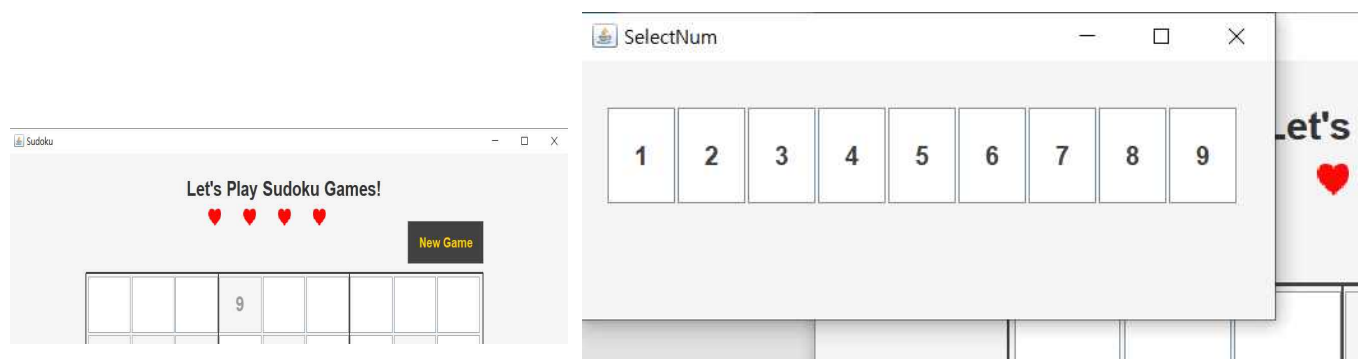
5-2) When you fail the Minigame (Click the OK Button of Result Dialog_(5))

=> -1 Hint, No Heart Change, Blanks doesn't fill with correct answer but still blank



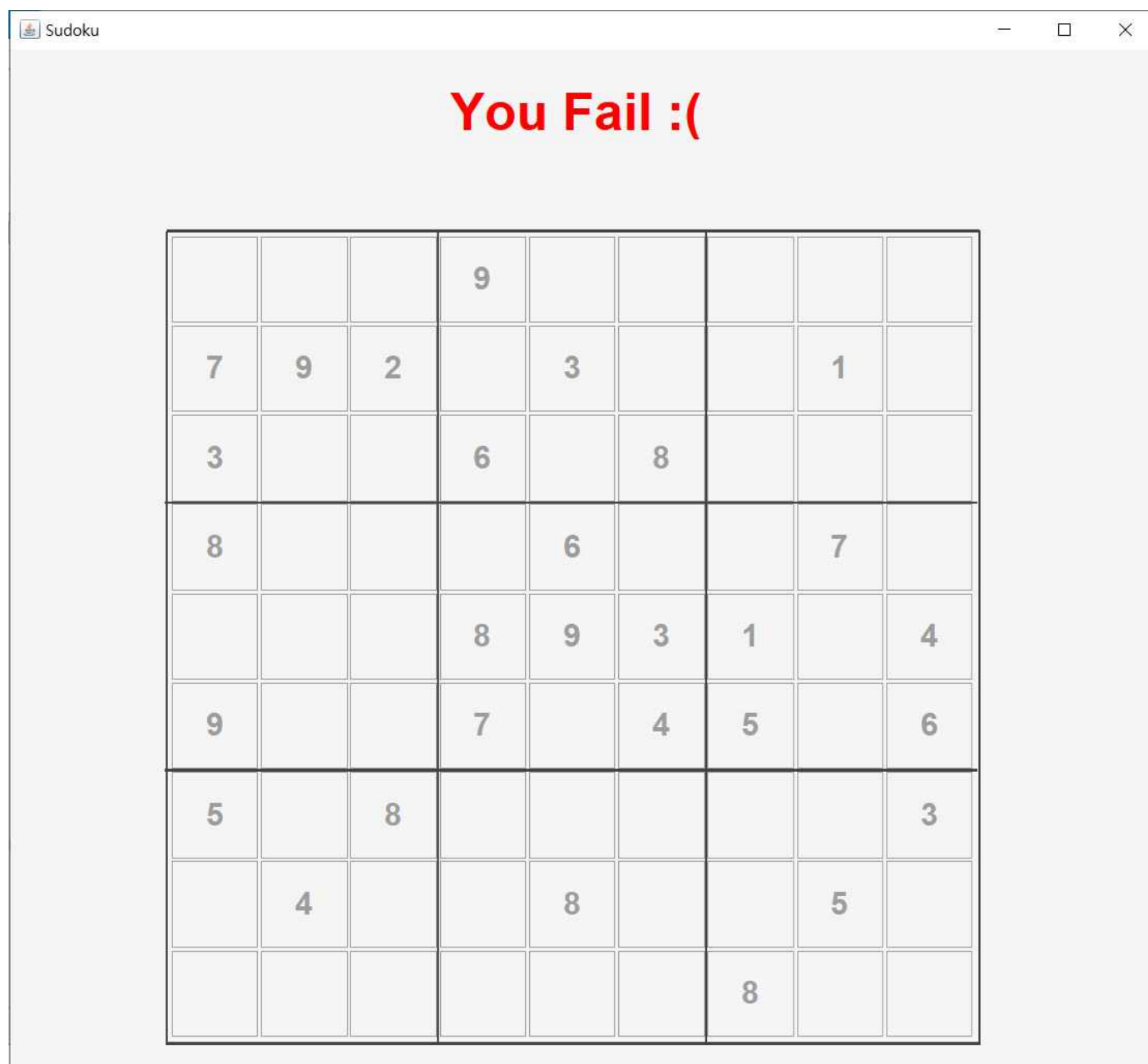
6-1) When you've used up all five Hints

=> No more Hint ICON, No more 'Play the mini-game and get answer' Button



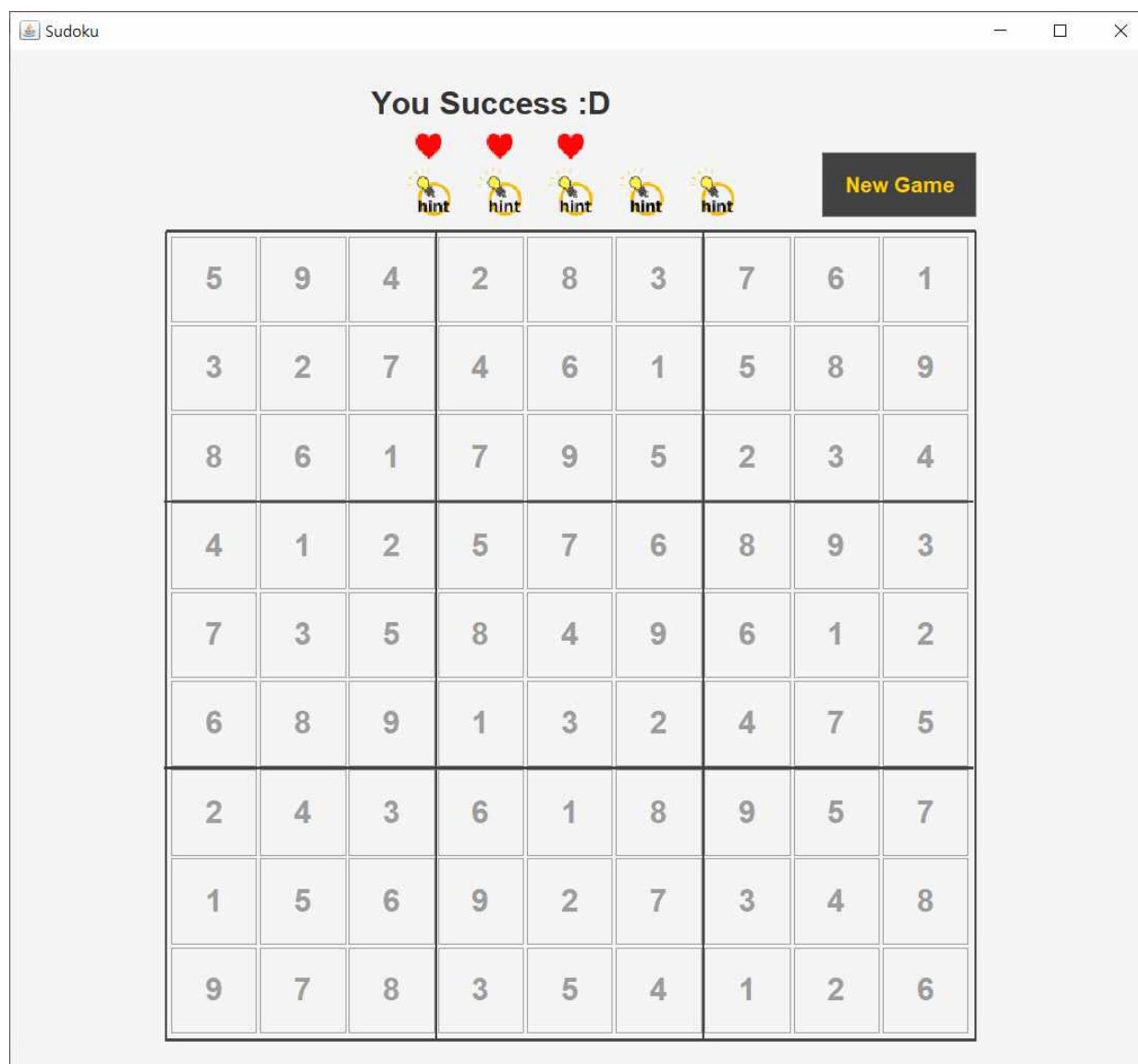
6-2) When you've used up all five Hearts

=> The Game is end, No more 'New Game' Button, There is no clickable space on the Sudoku map. Top message replaced with "You Fail :("



6-3) When you finished solving Sudoku

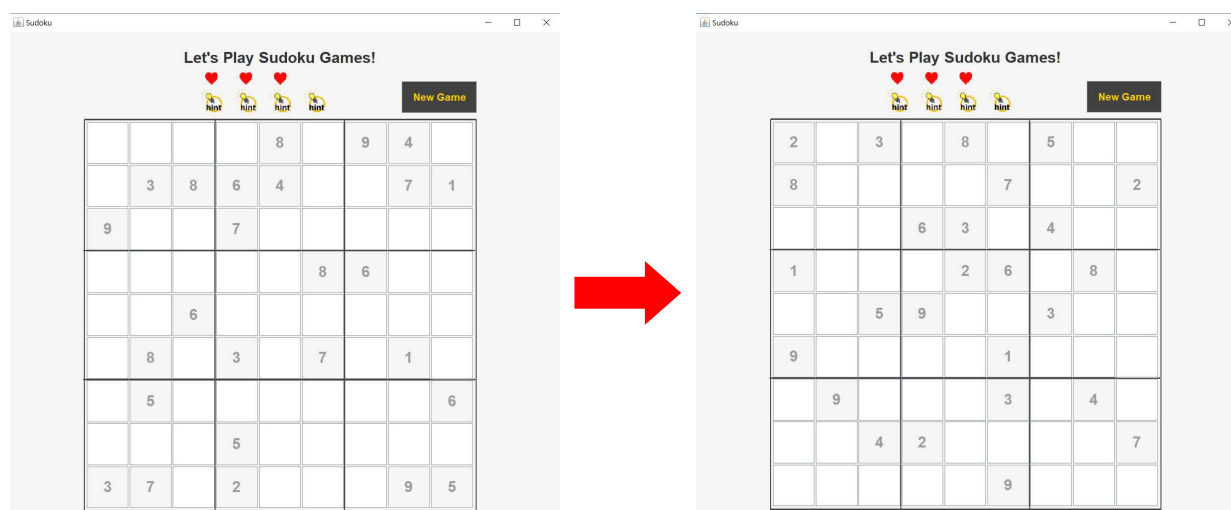
=> You can click 'New Game' Button, Top message replaced with "You Success :D"



7) When you click 'New Game' Button

=> No Heart Change, No Hint Change, Just Change Sudoku Map

=> If your Heart is left, you can change the sudoku map infinitely.



(g) explain how you applied object oriented concepts to the development for your project. also explain what you felt and learned from the project.

① Method Overloading

=> Two methods have the same name, but have different numbers of factors or data types. They are work as component of object oriented program. We use method overloading, So a method that functions the same can be used as one name. Also, we can save the name of the method. The follow is the example of method overloading in our code.

: checkValid(int[][] & checkValid(int[][], int index, int[])

```
92 private boolean checkValid(int[][] game) {
93     return checkValid(game, 0, new int[] { 0 }); // 값 대신 참조로 정수를 전달
94 }
95
96 private boolean checkValid(int[][] game, int index, int[] solutionsNumber) {
97     if (index > 80)
98         return ++solutionsNumber[0] == 1;
```

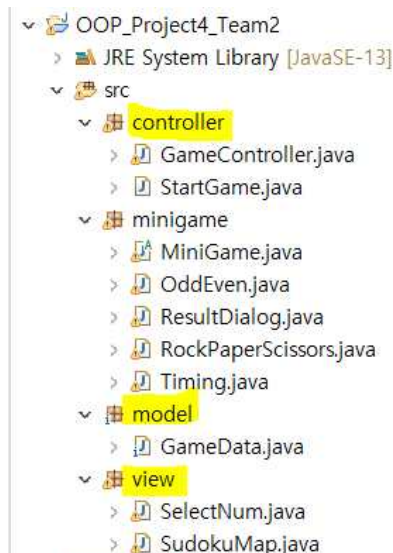
▲ Part of GameData.java

: generateGame(int[][]) & generateGame(int[][],List<Integer>)

```
127 private int[][] generateGame(int[][] game) {
128     List<Integer> positions = new ArrayList<Integer>();
129     for (int i = 0; i < 81; i++)
130         positions.add(i);
131     Collections.shuffle(positions);
132     return generateGame(game, positions);
133 }
134
135 private int[][] generateGame(int[][] game, List<Integer> positions) {
136     while (positions.size() > 0) {
137         int position = positions.remove(0);
```

▲ Part of GameData.java

② Using MVC pattern



```

OOP_Project4_Team2
├── JRE System Library [JavaSE-13]
├── src
│   ├── controller
│   │   ├── GameController.java
│   │   └── StartGame.java
│   ├── minigame
│   │   ├── MiniGame.java
│   │   ├── OddEven.java
│   │   ├── ResultDialog.java
│   │   ├── RockPaperScissors.java
│   │   └── Timing.java
│   ├── model
│   │   └── GameData.java
│   └── view
│       ├── SelectNum.java
│       └── SudokuMap.java
```

=> We used MVC patterns to make games object-oriented. "Model" Packages handled the status and logic of game data. Since the game data should be kept as one, we used a single tone pattern. "View" Package shows the data of the model to the user and operates by passing the user's input to the controller. The "Controller" Package updated the game data according to the user's input and informed the view of the changes again. By using MVC pattern, we were able to design the program flexibly by avoiding the dependence of view and model and by communicating indirectly with a controller.

③ Inheritance

: It is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods.

```
b 9 abstract class MiniGame extends JFrame{
```

▲ Part of Minigame.java

```
public class OddEven extends MiniGame{
public class RockPaperScissors extends MiniGame{
public class Timing extends MiniGame{
```

▲ Part of Inherited games

④ Abstraction

: Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

```
b 9 abstract class MiniGame extends JFrame{
10     public MiniGame() {
11     }
12     public abstract void run();
13     protected abstract void init();
14     protected abstract void start();
15     public abstract int getResult();
16
17     protected ImageIcon ImageSetSize(ImageIcon icon, int width, int height) {
18         Image ximg = icon.getImage();
19         Image yimg = ximg.getScaledInstance(width, height, Image.SCALE_SMOOTH);
20         ImageIcon xyimg = new ImageIcon(yimg);
21         return xyimg;
22     }
23 }
```

▲ Minigame.java

```

//벽에 닿지 않고 출구로 캐릭터를 탈출시키는 게임
public class OddEven extends MiniGame{

    public OddEven(){
    }

    @Override
    public void run() {
    }

    @Override
    protected void init() {
    }

    @Override
    protected void start() {
    }

    @Override
    public int getResult() {
    }
}

```

▲ Part of OddEven.java : Override the method of abstract class

(h) Conclusion

By applying what we learned during the semester while taking the "2020.2 Object Oriented Programming", we created the Sudoku game in JAVA language. Unlike traditional Sudoku, Sudoku, which we have implemented, has a difference in that it provides hints on Sudoku answers through simple mini games. Therefore, it implemented automatic Sudoku problem generation algorithm according to Sudoku game rules, number input to Sudoku game board, and three mini games (Rock Paper Scissors Game, Odd/Even Game, Timing Game) functions for getting Sudoku number hints.

In carrying out above task, We have implemented on the basic ideas of object-oriented modeling and design. We designed it according to the process described in the class. As a result, structural completeness was high. Also, we could feel the benefits of using object-oriented language. It was more convenient to modify the code after implementation as well as the development process.

We felt that we were better than the IIKH Team Project we had earlier. Rather than simply focusing on implementing Sudoku games, they tried to increase the efficiency of code by using many object-oriented elements and achieved satisfactory results. In addition, the GUI was implemented for the convenience of the user, thinking that there was a limit in the production of the game to proceed with the console window.

Explain What you felt and learned from the project

<김승아>

Inheritance was not used when implementing minigame for the first time. The same method was repeated in Rock-paper-scissors, OddEven, and Timing, resulting in unnecessary work. Also, to change the whole minigame, i had to do the same many times. By using abstract class, common parts of minigame were abstracted and used, and all of the above problems were solved. It has become easier to expand and modify. Thanks to inheritance, the process of connecting to Sudoku Game was also easy. It was also convenient to use objects in terms of managing game data during the game. Object orientation also served as an advantage in collaboration with team members. It was easy to share the work. This team project is satisfied with both the process and the results.

<배인경>

In order to make the most of the object-oriented concepts I learned during the semester, I thought a lot about the topic selection. Therefore, we decided to create our own unique Sudoku game, and to use JAVA language instead of C++ language used in the past tasks to make the game easier to play from the user's perspective with GUI rather than playing it in the console window. At first, it was burdensome to use JAVA language, which was not covered recently, but in conclusion, it is satisfying to organize the program code

object-oriented. It was good to have a good division of work with team members in the process of coding Sudoku program. Especially, I am proud as a team leader to submit the final project in this class with my current team members more satisfactory result than last time. If I have a chance, I want to decorate the GUI prettier than now.

<이주연>

The application of MVC pattern in this project seemed to be the most interesting. I had previously learned about MVC patterns in the departmental academic club, so I played a role in setting the overall structure of the program in this project4. I am proud that MVC pattern was applied to this our sudoku game. I could learn more by writing code of the controller package and view package. Clearly, I felt that the program was more object-oriented and flexible because we design the program with modularization by using MVC patterns.

In order to differentiate itself from the existing Sudoku games, we added mini games on the condition that they provide hints, and this part seems to be very interesting. After making the game by paying attention to the UI by using Java's GUI, I could feel the completeness of the project visually. In particular, this team project was helpful and fun because the role distribution was very good with the team members.