미니프로젝트 결과 보고서

재미있는 타자연습 - Word Bird Hunting Game

• 분반	객체지향언어2
• 학번	2271387
• 제출일	2023. 12. 11 (월)
• 소속	웹공학트랙
• 이름	서준영

1. 프로그램 개요

이 게임은 “재미있는 타자 연습”의 목적을 가지고 개발되었다. 기존의 지루한 타자 연습 게임, 단순히 화면에 나오는 단어와 문장만을 따라 입력하는 게임을 타파하고자 하는 것이 목적이다.

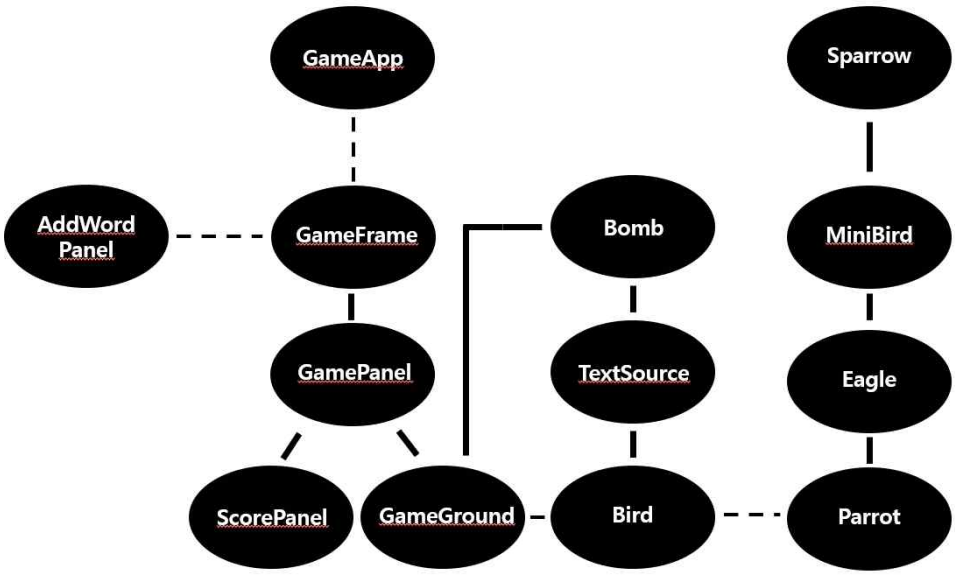
초기 화면에서 ‘게임 시작’ 버튼을 누르면 게임이 시작된다. 게임은 한글 단어로 구성되어 있다. 사용자는 게임을 시작하면 화면에 나타나는 네 마리의 새와 50% 확률로 떨어지는 폭탄을 볼 수 있다. 각각의 새와 폭탄에는 어떤 한글 단어가 적혀 있고, 이 단어를 하단의 사용자 입력창에 정확히 입력하면 된다. 또, 화면 우측에는 현재 라운드와 점수, 남은 기회를 표시하는 하트 3개가 있다. 사용자가 입력한 단어에 해당하는 단어가 적혀 있는 새는, 총에 맞은 듯 한 폭발하는 이미지로 변하며 화면에 표시된다. 이 경우 사용자는 10점을 획득하게 된다. 점수는 우측 패널에서 확인할 수 있다. 또, 사용자가 입력한 단어가 폭탄에 적혀있는 단어였다면, 폭탄은 얼음이 깨지는 듯한 모양으로 폭발하며 모든 새의 움직임이 3초간 정지된다. 폭탄은 점수를 주지는 않지만, 새의 움직임을 멈춰 사용자가 좀 더 편하게 단어를 인식하게 해 준다. 모든 새가 사냥되면 라운드가 높아지고, 높아질수록 새의 이동속도가 빨라져 단어를 알아보기 힘들기 때문이다. 이렇듯 중요한 폭탄은 좌우로만 비행하는 새와는 달리 하늘에서 땅으로 떨어진다. 폭탄이 땅에 닿기 전에 단어를 입력해 터트리지 못하면, 그 라운드에서 폭탄을 다시 만날 순 없다. 만약 급한 마음에 오답을 입력하면 하트의 개수가 하나씩 줄어들고, 모든 하트가 소모되면 게임은 패배로 끝나게 된다.

이 게임의 재미를 주는 가장 큰 포인트는 화면 상단에 출력되는 ‘잡으면 안되는 새’이다. 매 라운드마다 잡으면 안되는 새가 랜덤하게 정해지고, 화면 상단에 나타난다. 잡으면 안되는 새에 해당하는 단어를 입력하면 오히려 점수가 감소하고 하트가 줄어든다. 오답과 같은 처리를 하게 되는 것이다. 네 마리의 새가 화면에서 퍼덕이고, 폭탄이 떨어지는 상황에 정신이 없지만, 정확히 새를 알아보고 입력해야 하는 부분이 재미의 포인트가 될 것으로 기대한다.

모든 하트를 소모하고 게임이 종료되면, 사용자의 이름을 입력받는다. 사용자가 이름을 입력하면 그 이름과 점수, 라운드가 저장되어 랭킹 시스템에 활용된다. 최고 고득점자 3명은 각각 메달을 받고, 게임 실행 중에 화면 우측에 그 이름과 점수, 메달이 표시된다. 이를 통해 사용자는 경쟁하는 재미도 느낄 수 있다. 게임이 종료되면 자동으로 초기 메뉴 화면으로 되돌아간다.

초기 메뉴 화면의 ‘단어 추가’ 버튼을 누르면 단어 입력 메뉴로 이동한다. 화면 중앙의 텍스트 입력 창에 단어를 입력하면, 그 단어는 새와 폭탄 이름에 적혀 나오는 단어들 중 하나로 추가된다. 단어의 목록은 게임 폴더/source/korean.txt 파일에서 확인할 수 있다.

2. 프로그램 구조



[그림 1]

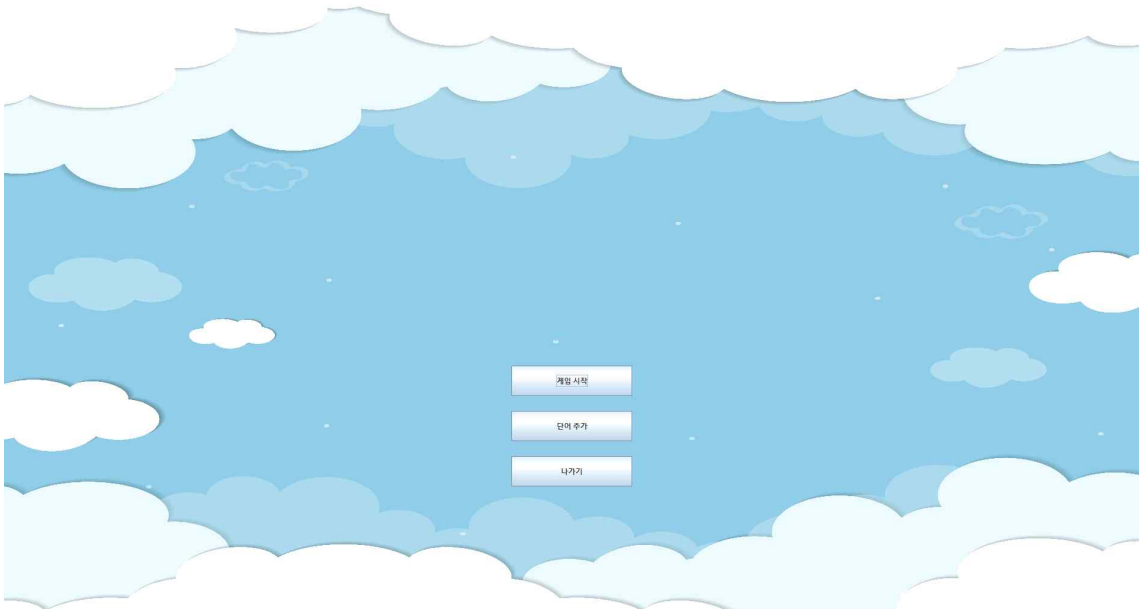
그림 1은 이 프로그램의 클래스 계층 구조를 보여준다. GameApp에는 이 프로그램의 진입점인 Main 함수가 있고, 이 함수엔 GameFrame의 생성자만 존재한다. GameFrame에는 '게임 시작', '단어 추가', '나가기' 3개의 버튼이 있고, 이는 각각 GamePanel, AddWordPanel, 게임 종료를 실행하게 된다.

먼저, GamePanel은 좌측의 GameGround와 우측의 ScorePanel로 나누어지는데, ScorePanel에는 현재 라운드, 점수, 남은 하트와 랭킹 1, 2, 3등의 기록이 표시된다.

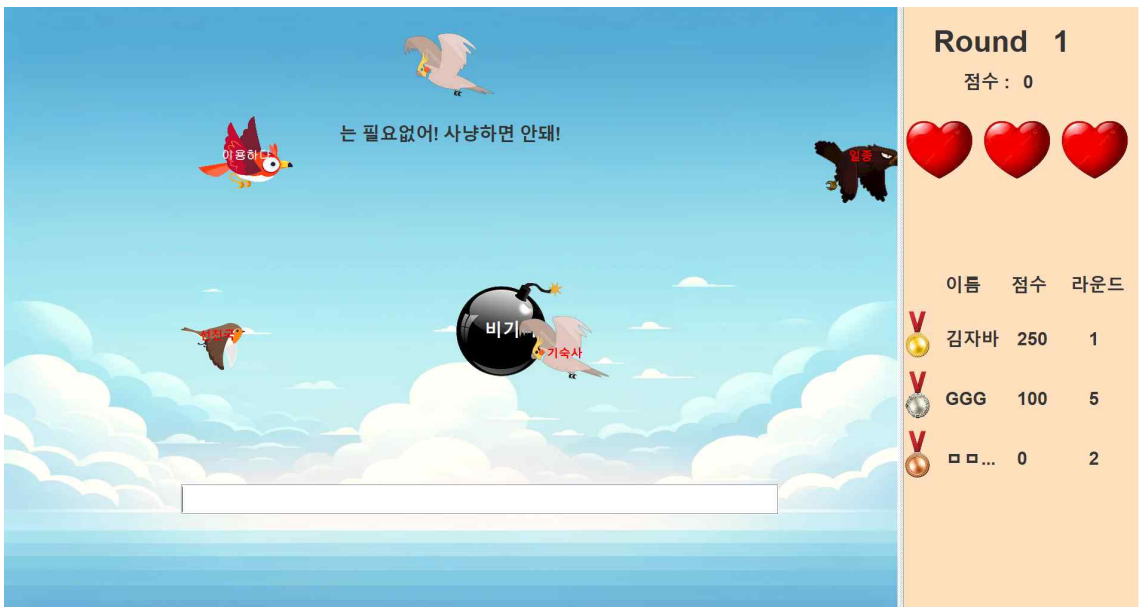
GameGround는 타자 연습 게임이 실행되는 부분으로, 단어를 가진 새와 폭탄, 잡으면 안되는 새, 사용자 입력 창이 위치한다. 이 GameGround에 날아다니는 새들은 추상클래스 Bird를 상속받아 만들어진다. 각 새의 코드 구조가 유사하기 때문에 추상클래스를 상속받도록 설계되었다. Bomb클래스는 화면에 떨어지는 폭탄을 구현하는 클래스다. 폭탄과 새의 몸에 적히게 되는 단어들은 텍스트 파일로부터 단어 목록을 받아와 랜덤하게 단어를 선정하는 TextSource에서 선택한다.

Bird를 상속받은 각 새 클래스들은 날아다니는 모습을 구현한 스레드를 실행시키고, Bomb는 폭탄이 떨어지는 모습을 구현한 스레드를 실행시킨다. GameGround는 게임의 전체적인 메인 실행 로직을 담당하는 메인 스레드와 잡으면 안되는 새를 표시하는 스레드 2개를 실행시킨다. 잡으면 안되는 새를 사냥이 끝났을 경우에 변경하기 위해 이 두 스레드는 동기화되어 있다.

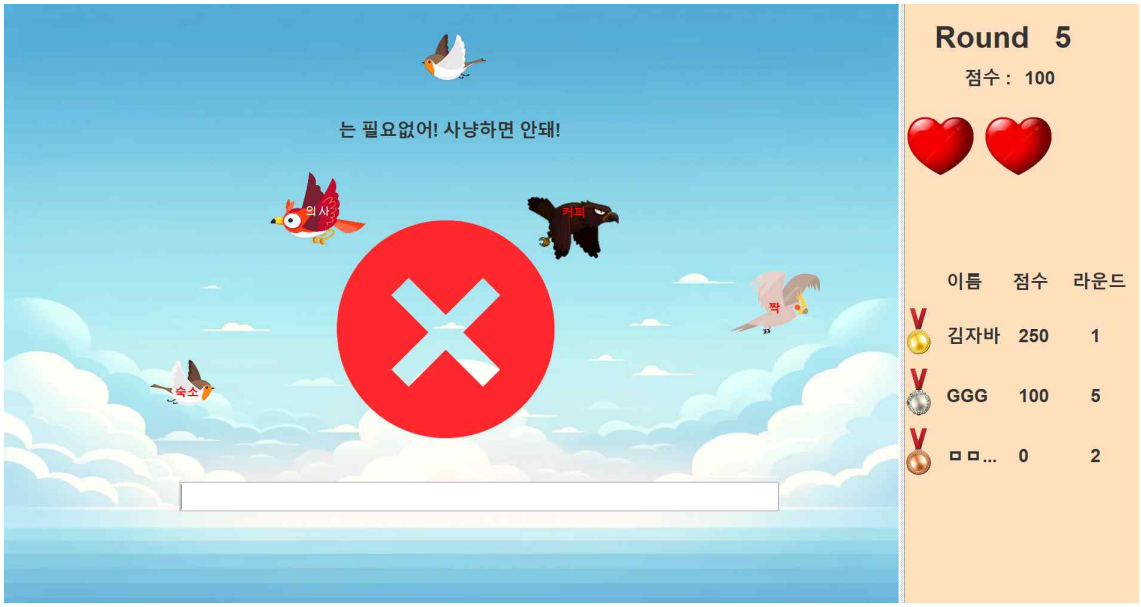
3. 프로그램 실행 모습



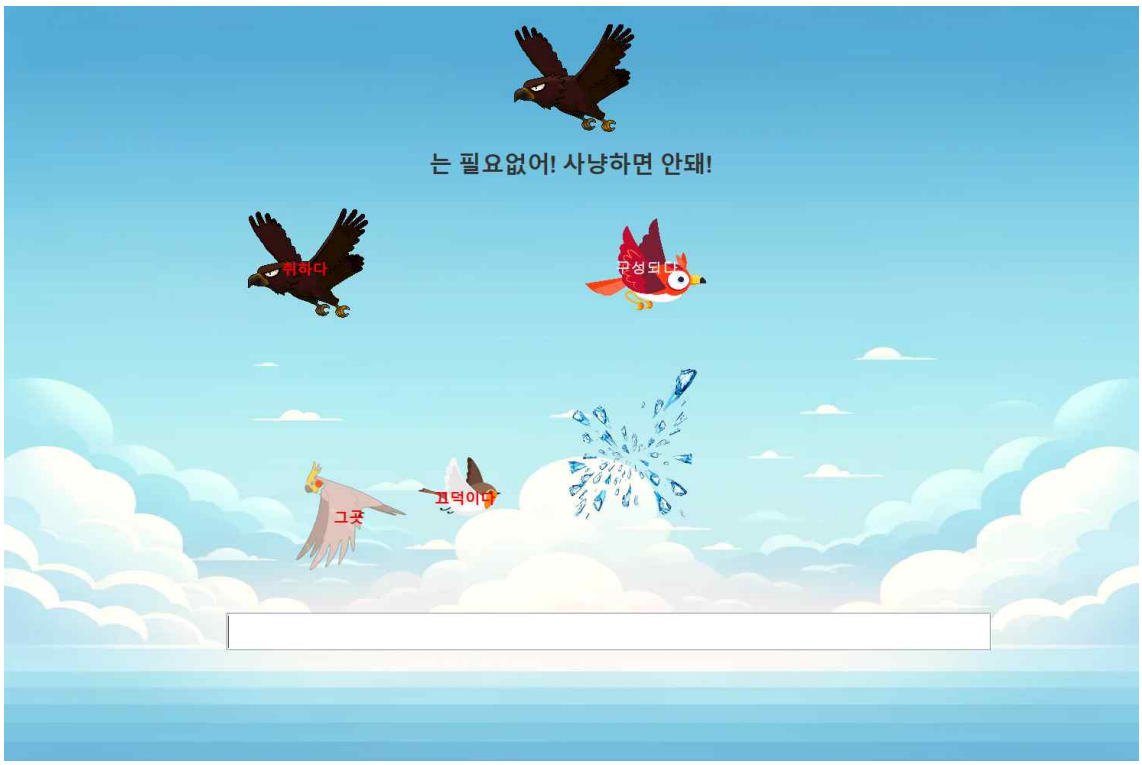
↳ 게임의 초기 실행 모습. '게임 시작' 버튼을 누르면 게임이 시작되고, '단어 추가' 버튼을 누르면 단어를 추가하는 메뉴로 이동된다.



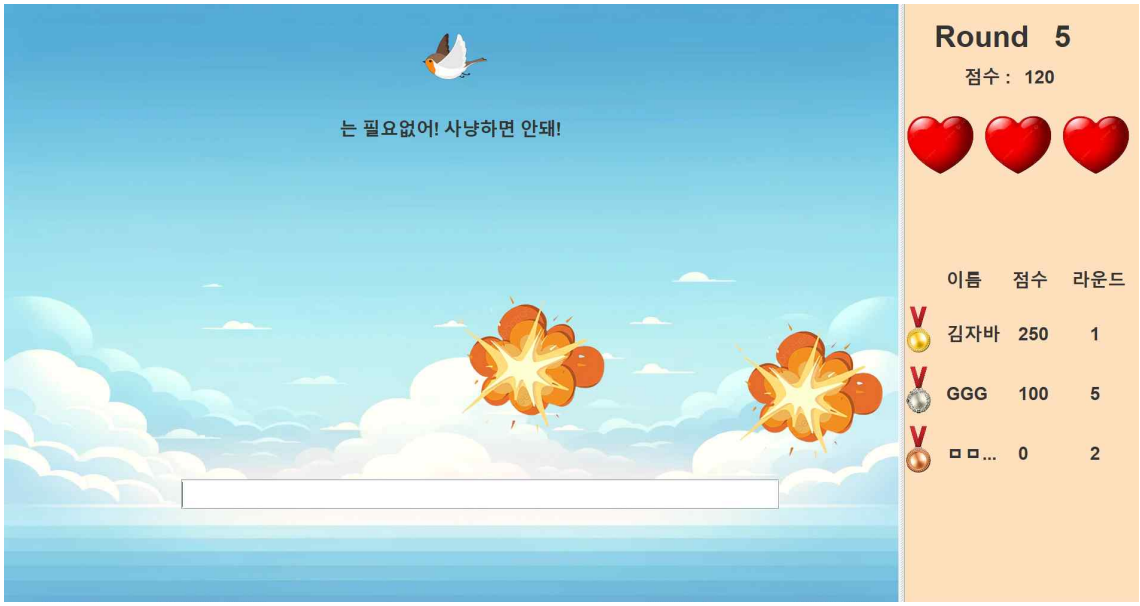
↳ '게임 시작' 버튼을 눌러 게임이 시작된 화면. 상단의 사냥하면 안되는 새와 사냥해야 하는 3마리의 새, 보너스 폭탄과 사용자 입력창을 볼 수 있다. 우측에선 라운드와 점수, 남은 하트와 랭킹을 확인할 수 있다.



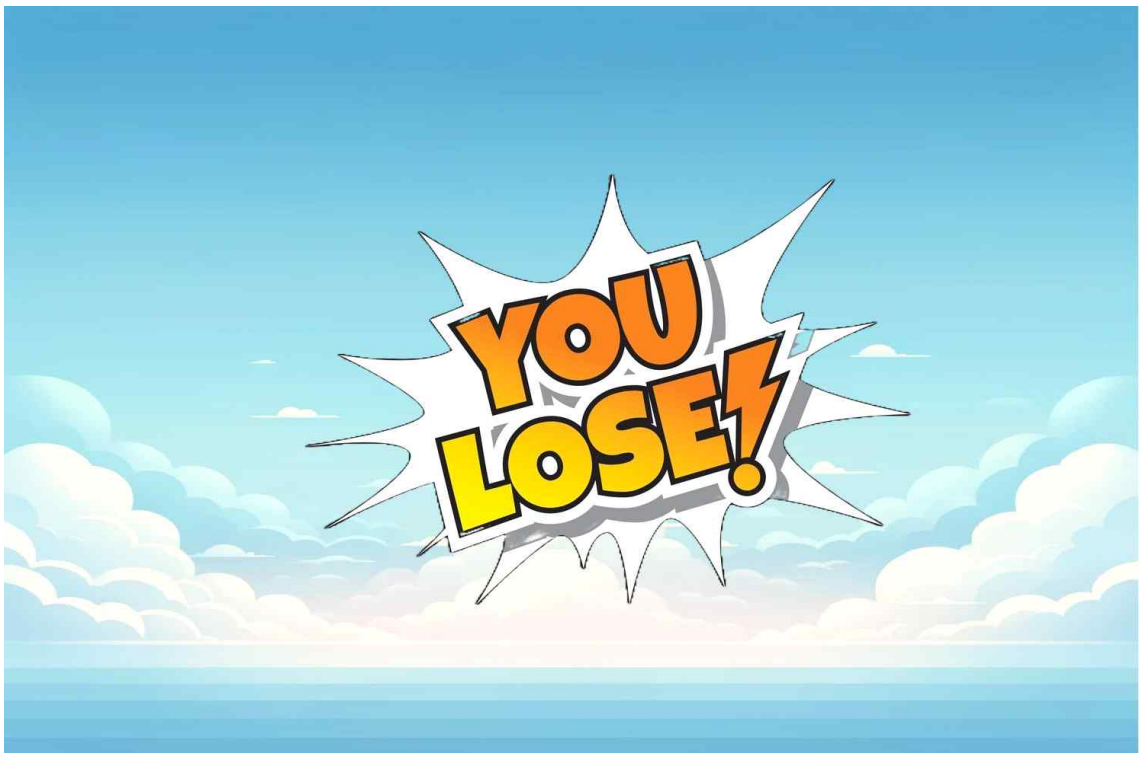
↳ 오답을 입력하는 경우 화면에 X 표시가 나타나며 하트가 감소된다. 점수도 20점 감점된다.



↳ 폭탄에 적힌 문자를 정확히 입력한 경우, 폭탄이 얼음 모양으로 터지면서 모든 새들의 움직임이 3초간 정지된다. 폭탄은 매 라운드마다 50%의 확률로 등장하며, 땅에 닿기 전에 단어를 입력해 터트리지 못한다면 자동으로 사라진다. 사라진 이후엔 단어를 입력해도 새들이 멈추지 않는다.

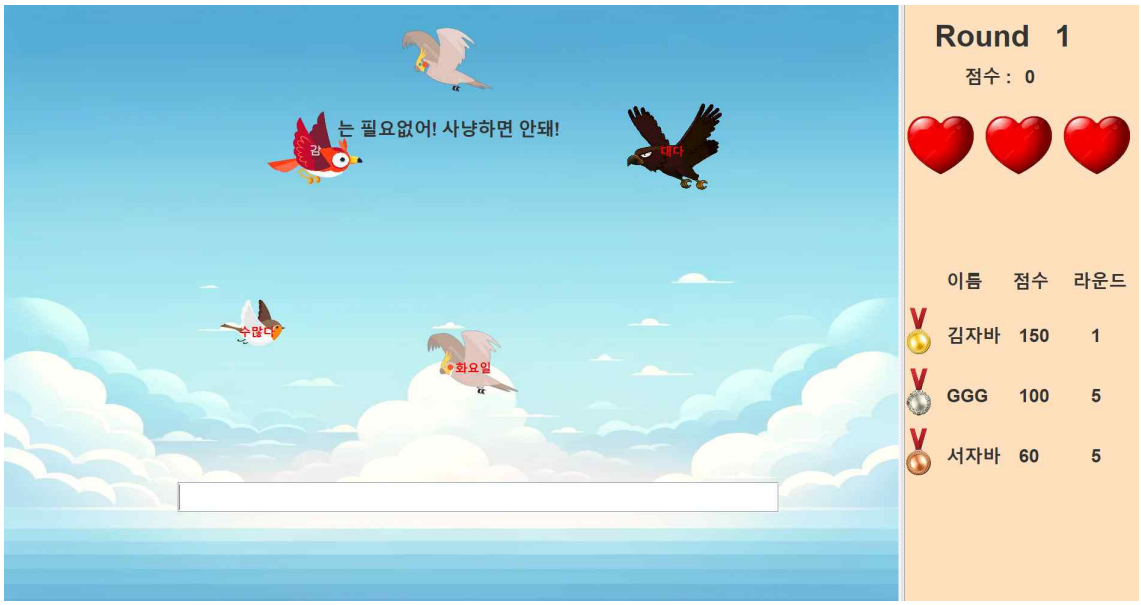


↳ 잡으면 안되는 새를 제외한 새를 모두 사냥했다면, 잡으면 안되는 새도 같이 사냥한 것과 같은 폭발 이미지로 변하며 사라진다.

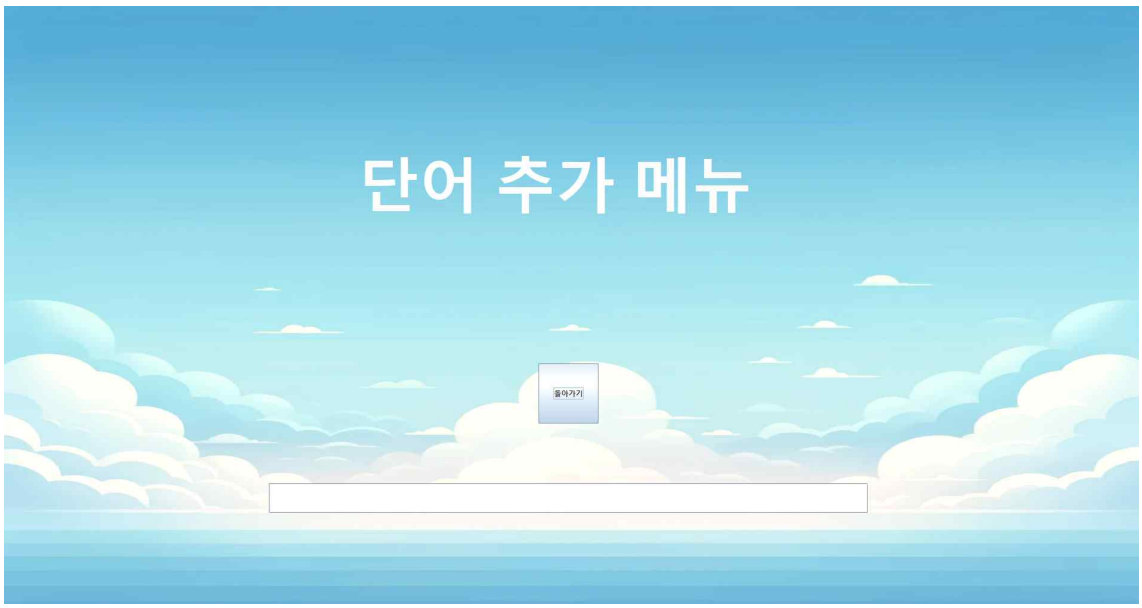


↳ 하트를 모두 소진했다면, 화면의 모든 새와 폭탄, 스코어 보드는 사라지며 패배를 표시하는 이미지가 나타난다. 이 이미지는 2초후 사라진다.

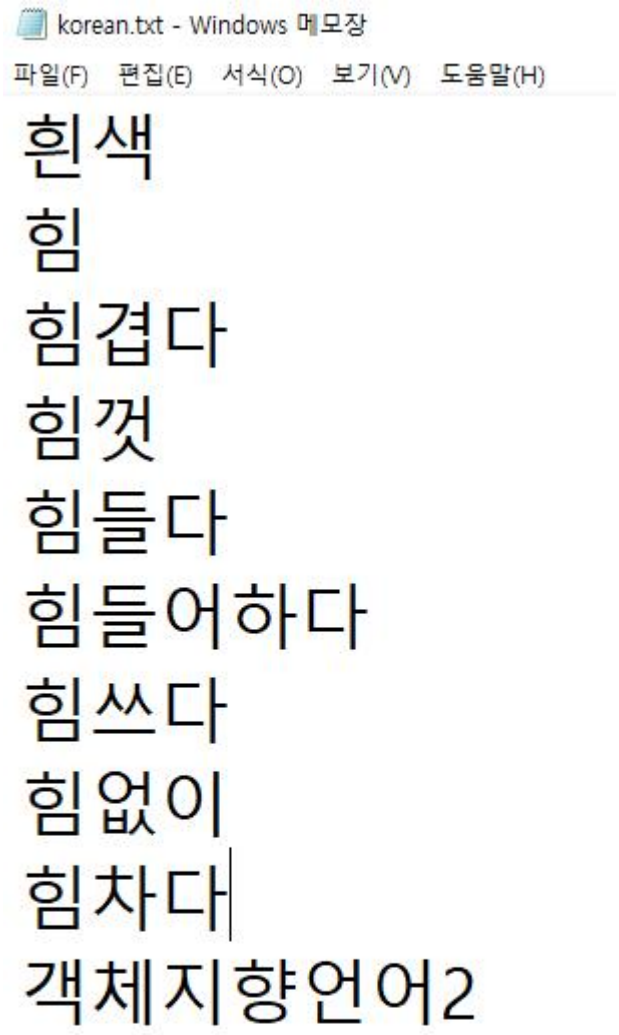
↳ 패배 이미지가 사라진 후, 사용자 이름을 입력하는 창이 나타난다. 이 때 입력한 이름은 랭킹에 저장된다. 테스트를 위해 '서자바' 라는 이름을 입력했다.



↳ 60점을 획득한 '서자바' 이름이 3위에 랭크되어 동메달을 획득한 모습을 볼 수 있다. 게임에서 획득한 점수와 라운드까지 보여준다.



↳ 게임이 종료되고 사용자 이름 입력을 마치고 나면 자동으로 초기 메뉴로 돌아간다. 이 화면은 초기 메뉴에서 '단어 추가' 버튼을 누른 후의 화면이다. 단어 추가 메뉴로 이동했다. 테스트를 위해 이 칸에 '객체지향언어2'를 입력했다.



↳ 새와 폭탄에 나타나는 문자열의 소스가 되는 txt 파일이다. 텍스트 파일의 마지막에 객체지향언어2가 추가된 모습을 볼 수 있다. 이 단어는 게임 중 언제라도 등장할 수 있다.

4. 프로그램 코드

<GameApp.java>

프로그램의 진입점(Main)이 있는 클래스

```
public class GameApp { //전체 게임 실행용(메인) 클래스
    public static void main(String[] args)
    {
        new GameFrame(); //생성자 생성으로 시작
    }
}
```

<GameFrame.java>

초기 메인메뉴 화면을 만드는 클래스

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
//초기 메인메뉴 화면 클래스
public class GameFrame extends JFrame {
    private JButton gameStart;
    private JButton exit;
    private JButton addWord;
    private GamePanel gamePanel =null;
    private JPanel menuPanel;
    private AddWordPanel addWordPanel=null;
    ImageIcon sky =new ImageIcon("images/sky.png"); //배경화면
    private GameFrame gameFrame =this;
    public GameFrame(){
        menuPanel=new menuPanel();
        setTitle("Word Bird Hunting Game");
        setSize(1920,1080);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(menuPanel);
        setVisible(true);
    }
    //메뉴 출력 패널
    class menuPanel extends JPanel {
        //배경화면 그리기
        //오버라이딩으로 컴포넌트 그릴 때 마다 배경 그리기
        @Override
        protected void paintComponent(Graphics g){
            super.paintComponent(g);
            Image skyImg = sky.getImage();
            g.drawImage(skyImg,0,0,this.getWidth(),this.getHeight(),this);
        }
        public menuPanel() {
            //예쁜 배치를 위해 배치관리자 삭제
            setLayout(null);
            gameStart =new JButton("게임 시작");
            addWord =new JButton("단어 추가");
            exit =new JButton("나가기");
            //게임시작 버튼 클릭 시 작동
            gameStart.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    getContentPane().removeAll(); // 기존 컴포넌트 제거
                    gamePanel =new GamePanel(gameFrame);
                    getContentPane().add(gamePanel, BorderLayout.CENTER);
                    getContentPane().revalidate();
                    getContentPane().repaint();
```

```
                }
            });
            //단어 추가 버튼 클릭 시 작동
            addWord.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    getContentPane().removeAll();//기존 컴포넌트 제거
                    addWordPanel =new AddWordPanel(gameFrame);
                    getContentPane().add(addWordPanel);
                    getContentPane().revalidate();
                    getContentPane().repaint();
                }
            });
            //나가기 버튼 클릭시 작동
            exit.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                }
            });
            gameStart.setSize(200,50);
            exit.setSize(200,50);
            gameStart.setLocation(860,600);
            addWord.setSize(200,50);
            addWord.setLocation(860,675);
            exit.setLocation(860,750);
            add(gameStart);
            add(addWord);
            add(exit);
        }
    }
    //다른 클래스에서 호출돼서 메인 메뉴로 돌아오게 하는 함수
    public void showMainMenu() {
        getContentPane().removeAll();
        getContentPane().add(menuPanel, BorderLayout.CENTER);
        getContentPane().revalidate();
        getContentPane().repaint();
    }
}
```

<GamePanel.java>

게임 시작 버튼을 눌렀을 때, 화면을 분할하고 게임 실행 화면을 구성하는 클래스

```
import javax.swing.*;
import java.awt.*;
public class GamePanel extends JPanel {
    ScorePanel scorePanel =new ScorePanel();
    private GamePanel gamePanel=this;
    private GameFrame gameFrame;
    GameGround gameGround =new GameGround(scorePanel,gamePanel);
    public GamePanel(GameFrame gameFrame){
        this.gameFrame= gameFrame;
        setLayout(new BorderLayout());
        //화면 구성
        splitPanel();
    }
    //게임 실행 화면을 구성하는 함수
    private void splitPanel(){
        JSplitPane hSplitPane =new JSplitPane();
        hSplitPane.setOrientation(JSplitPane.HORIZONTAL_SPLIT);
        hSplitPane.setDividerLocation(1500); // Divider 위치 설정
        add(hSplitPane);
        hSplitPane.setRightComponent(scorePanel); // 오른쪽 패널을 ScorePanel로 설정
        hSplitPane.setLeftComponent(gameGround); // 왼쪽 패널을 GameGround로 설
정
    }
    //게임 종료시 호출돼서 메인메뉴로 돌아감
    public void endGame() {
        gameFrame.showMainMenu();
    }
}
```

<GameGround.java>

게임이 실행되는 영역을 담당하는 클래스

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class GameGround extends JPanel {
    //폭탄 클래스 Bomb 레퍼런스
    Bomb bomb;
    //Bird와 Bomb에 gameGround 레퍼런스 전달하기 위한 변수
    GameGround gameGround =this;
    //잡으면 안되는 새를 출력하기 위한 변수
    protected int noHuntBirdNum;
    //scorePanel에 전달할 round 변수
    protected int round;
    //오답 입력 처리를 위한 플래그
    private boolean isNotCorrected =true;
    //4종류의 Bird 배열
    protected Bird[] birds;
    //틀렸을경우 나오는 이미지
    private ImageIcon no =new ImageIcon("images/no.png");
    //배경화면
    private ImageIcon sky =new ImageIcon("images/sky.jpg");
    //배경화면
    private Image skyImg = sky.getImage();
    //게임 패배시(하트 다 쓰면) 출력되는 이미지
    private ImageIcon loseImg =new ImageIcon("images/lose.png");
    ScorePanel scorePanel;
    //틀렸을 경우를 표기하는 레이블
    JLabel notAnswer;
    //잡으면 안되는 새를 표시할 스레드
    NohuntThread noHuntThread;
    //잡으면 안되는 새를 나타내는 이미지 배열
    protected ImageIcon[] noHuntBirdImg;
    //게임의 메인 로직 스레드
    GameThread mainGameThread;
    //사용자 입력창
    private JTextField typeInput =new JTextField(30);
    //메인 메뉴로 돌아가기 위한 GamePanel 레퍼런스
    private GamePanel gamePanel;
    @Override
    protected void paintComponent(Graphics g) {
        //배경화면 그리기용 오버라이딩
        super.paintComponent(g);
        g.drawImage(skyImg, 0, 0, this.getWidth(), this.getHeight(), this);
    }
    //메인메뉴로 돌아가기 위해 gamePanel 레퍼런스를 전달
    //점수와 랭킹 표기를 위해 scorePanel 레퍼런스 전달
    public GameGround(ScorePanel scorePanel, GamePanel gamePanel) {
```

```
this.gamePanel = gamePanel;
//시작시 라운드는 1
round =1;
birds =new Bird[4];
//각 새에 레퍼런스를 전달하며 생성
birds[0] =new Sparrow(typeInput, round, gameGround);
birds[1] =new Eagle(typeInput, round,gameGround);
birds[2] =new MiniBird(typeInput,round,gameGround);
birds[3] =new Parrot(typeInput,round,gameGround);
birds[0].getBirdLabel().setSize(300, 300);
birds[0].getBirdLabel().setLocation(200, 100);
birds[1].getBirdLabel().setSize(300, 300);
birds[1].getBirdLabel().setLocation(1000, 100);
birds[2].getBirdLabel().setSize(300,300);
birds[2].getBirdLabel().setLocation(200,400);
birds[3].getBirdLabel().setSize(300,300);
birds[3].getBirdLabel().setLocation(1000,400);
add(birds[0].getBirdLabel());
add(birds[1].getBirdLabel());
add(birds[2].getBirdLabel());
add(birds[3].getBirdLabel());
//초기(1라운드) 새 생성 끝
//잡으면 안되는 새의 이미지를 담는 변수
noHuntBirdImg =new ImageIcon[birds.length];
for (int i =0; i < birds.length; i++) {
    noHuntBirdImg[i] = birds[i].getBirdImage();
}
//오답을 입력했을 경우를 위한 레이블
//평상시에는 더해져 있다가 보이지 않음
//오답시 setIcon을 통해 잠시 X 표시를 보여줌
notAnswer =new JLabel();
notAnswer.setSize(500, 500);
notAnswer.setLocation(500, 300);
add(notAnswer);
//scorePanel 레퍼런스 설정
this.scorePanel = scorePanel;
scorePanel.setRound(round);
//사용자 입력창
typeInput.setSize(1000, 50);
typeInput.setFont(new Font("고딕체", Font.BOLD, 30));
typeInput.setLocation(300, 800);
add(typeInput);
setLayout(null);
//게임 실행 메인 스레드 시작
mainGameThread =new GameThread();
mainGameThread.start();
//사용자 입력창 액션 리스너 설정
typeInput.addActionListener(new TextActionListener());
}
//사용자 입력창 액션 리스너
class TextActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        //잡으면 안되는 새를 잡았을 경우를 검출
```

```
                    noHuntBirdNum = mainGameThread.getRandNum();
                    if (birds[noHuntBirdNum].isCorrect()) {
                        scorePanel.decrease();
                    }
                    //진짜 정답을 맞춘 경우
                    else if (birds[0].isCorrect() || birds[1].isCorrect() ||
birds[2].isCorrect() || birds[3].isCorrect()) {
                        scorePanel.increase();
                    }
                    //폭탄에 입력된 정답을 맞췄을 경우
                    //폭탄은 점수에는 관여하지 않음
                    //이 문장은 아무것도 실행하지 않지만,
                    //else의 오답인 경우에서 처리되지 않기 위해 이 문장이 필요함
                    else if (bomb!=null && bomb.isCorrect()){
                    }
                    //오답인 경우
                    else {
                        scorePanel.decrease();
                        //오답 플래그 down
                        isNotCorrected =false;
                    }
                    //사용자가 3번 틀려서 게임이 끝나는 경우
                    if(scorePanel.isLose()){
                        //랭킹 저장을 위해 round 전달
                        scorePanel.setRound(round);
                        //메인 게임 스레드의 패배함수 실행
                        mainGameThread.lose();
                    }
                    //입력창 초기화
                    typeInput.setText("");
                }
            }
            //폭탄의 답을 입력한 경우 모든 새 3초간 정지
        public void pauseAllBirds() {
            for (Bird bird : birds) {
                bird.pauseMovement();
            }
        }
        //3초 뒤 새를 움직이게 하는 함수
        public void resumeAllBirds() {
            for (Bird bird : birds) {
                bird.resumeMovement();
            }
        }
            //잡으면 안되는 새를 표시하는 스레드
            class NohuntThread extends Thread {
                //잡으면 안되는 새의 인덱스
                protected int randNum;
                //"사냥하면 안돼!" 표시용 레이블
                JLabel noHunt;
                //스레드 시작시 숫자를 전달받아 사냥하면 안되는 새로 설정
                public NohuntThread(int randNum) {
                    this.randNum = randNum;
                }
```

```
        //사냥하면 안되는 새 인덱스 설정용 함수
        public void setRandNum(int randNum) {
            this.randNum = randNum;
        }
        //메인 게임 스레드와 동기화
        //모든 새를 잡았을 경우를 위해 동기화함
        @Override
        synchronized public void run() {
            while (true) {
                //잡으면 안되는 새 이미지
                ImageIcon noHuntImg = noHuntBirdImg[randNum];
                noHunt =new JLabel(noHuntImg);
                noHunt.setText("는 필요없어! 사냥하면 안돼!");
                noHunt.setFont(new Font("고딕체", Font.BOLD, 30));
                noHunt.setSize(500, 200);
                noHunt.setLocation(500, 30);
                noHunt.setHorizontalTextPosition(JLabel.CENTER);
                noHunt.setVerticalTextPosition(JLabel.BOTTOM);
                add(noHunt);
                try {
                    //모든 새 사냥 전까지 스레드 대기
                    wait();
                } catch (InterruptedException e) {
                    //인터럽트시 제거
                    noHunt.getParent().remove(noHunt);
                    return;
                }
                //notify()시 제거
                noHunt.getParent().remove(noHunt);
            }
        }
    }
    //게임 메인 로직 담당 스레드
    class GameThread extends Thread {
        //패배 표시용 레이블
        private JLabel lose;
        //랭킹 표시를 위한 이름 변수
        private String name;
        //패배 판단용 플래그
        private boolean loseFlag=false;
        //랜덤으로 나오는 잡으면 안되는 새를 설정하는 변수
        private int randNum = (int) (Math.random() * birds.length);
        public int getRandNum() {
            return randNum;
        }
        //폭탄이 나올 확률과 위치를 바꿔주게 될 변수
        private int bombRandNum = (int)(Math.random()*100+1);
        //패배시 호출되는 함수
        public void lose(){
            //lose 레이블을 만들고 이미지를 붙여 나타낸다
            lose =new JLabel();
            lose.setIcon(loseImg);
            lose.setSize(1024,1024);
            lose.setLocation(300,0);
```

```
                //모든 새 삭제
                for(int i=0;i<birds.length;i++) {
                    GameGround.this.remove(birds[i].getBirdLabel());
                }
                //폭탄이 있었다면 삭제
                if(bomb!=null) {
                    GameGround.this.remove(bomb.getBombLabel());
                }
                //사용자 입력창 삭제
                typeInput.setSize(0,0);
                //사냥하면 안되는 새 스레드 인터럽트
                //noHuntThread의 레이블은 인터럽트 시 remove됨
                noHuntThread.interrupt();
                GameGround.this.add(lose);
                GameGround.this.repaint();
                //패배 플래그 up
                loseFlag=true;
            }
            @Override
            public void run() {
                //메인 게임 스레드가 실행되면 noHuntThread도 실행
                //1라운드를 위해 랜덤숫자 전달
                noHuntThread =new NohuntThread(randNum);
                noHuntThread.start();
                //잡으면 안되는 새는 사냥하지 않아도 잡은것으로 설정해서 다음 라운드로 넘어가게 함
                birds[randNum].setIsAlive(false);
                while (true) {
                    try {
                        //폭탄 숫자는 1~100까지 나온다
                        //50 미만이면 생성되므로 50%확률로 폭탄 생성
                        //폭탄이 여러개 생기는 경우를 방지하기 위해 bomb==null 조건
                        if(bombRandNum<50 && bomb==null){
                            //폭탄의 문자열 입력 검증을 위해 입력창 레퍼런스 전달
                            //폭탄 삭제를 위해 gameGround 레퍼런스 전달
                            bomb =new Bomb(typeInput,gameGround);
                            bomb.getBombLabel().setSize(300,300);
                            //랜덤한 위치에 생성되는 폭탄
                            bomb.getBombLabel().setLocation(bombRandNum*25,0);
                            add(bomb.getBombLabel());
                        }
                        //패배 플래그 up이면
                        if(loseFlag){
                            //1초간 대기 후 lose 이미지 제거
                            sleep(1000);
                            GameGround.this.remove(lose);
                            //패배시 사용자 이름을 남기기 위한 입력창 생성
                            //이름은 ranking.txt파일에 저장됨
                            JLabel inputName =new JLabel("이름을 입력하세요.");
                            JTextField inputNameField =new JTextField(30);
                            inputName.setSize(1000,30);
                            inputName.setFont(new Font("고딕체",Font.BOLD, 30));
                            inputName.setBackground(Color.RED);
                            inputName.setLocation(630,300);
```

```
                inputNameField.setSize(1000,30);
                inputNameField.setLocation(300,500);
                add(inputName);
                add(inputNameField);
                //사용자 이름 입력창에 대한 액션 리스너
                inputNameField.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        name = inputNameField.getText();
                        //ranking.txt에 이름을 남기는 기능은 scorePanel 클
래스에 있음

                        scorePanel.setPlayerName(name);
                        //이름 입력이 완료되면 초기 메뉴화면으로 돌아감
                        gamePanel.endGame();
                    }
                });
                repaint();
            }
            if (!isNotCorrected) {//오답인경우
                //숨겨져 있던 notAnswer 레이블에 X 이미지 표시
                notAnswer.setIcon(no);
                repaint();
                //2초후 이미지 삭제
                sleep(2000);
                notAnswer.setIcon(null);
                repaint();
                //다시 오답 플래그 up
                isNotCorrected =true;
            }
            //모든 새가 사냥됐는지 검증하기 위한 플래그
            boolean isAllBirdDead =true;
            for (int i =0; i < birds.length; i++) {
                if (birds[i].getIsAlive()) {//살아있는 새가 있다
                    isAllBirdDead =false;
                    break;
                }
            }
            sleep(100);
            if (isAllBirdDead) {//사냥 완료한 시점에
                //사냥하면 안되는 새를 바꾸기 위한 동기화
                synchronized (noHuntThread) {
                    //사냥하면 안되는 새의 플래그를 올려 잡은 것으로 인식
                    birds[randNum].setCorrectFlag(true);
                    //다음 라운드로 증가
                    round++;

scorePanel.roundNumLabel.setText(Integer.toString(round));
                    for (int i =0; i < birds.length; i++) {
                        if (birds[i] !=null) {//NullPointer예외 방지
                            //기존 새 스레드가 정지될 때 까지 대기
                            birds[i].getThread().join();
                        }
                    }
                    //라운드가 바뀌면 새로운 새 생성
```

```
                                //라운드 변수가 새들에게 전달되면서 속도가 빨라진다
                                birds[0]      =new      Sparrow(typeInput,      round,
gameGround);
                                birds[1] =new Eagle(typeInput, round, gameGround);
                                birds[2]      =new      MiniBird(typeInput,      round,
gameGround);
                                birds[3]=new Parrot(typeInput,round,gameGround);
                                birds[0].getBirdLabel().setSize(300, 300);
                                birds[0].getBirdLabel().setLocation(200, 200);
                                birds[1].getBirdLabel().setSize(300, 300);
                                birds[1].getBirdLabel().setLocation(1000, 200);
                                birds[2].getBirdLabel().setSize(300, 300);
                                birds[2].getBirdLabel().setLocation(200, 500);
                                birds[3].getBirdLabel().setSize(300,300);
                                birds[3].getBirdLabel().setLocation(1000,400);
                                add(birds[0].getBirdLabel());
                                add(birds[1].getBirdLabel());
                                add(birds[2].getBirdLabel());
                                add(birds[3].getBirdLabel());

                                //새로운 사냥하면 안되는 새 인덱스 설정
                                this.randNum = (int)(Math.random() * birds.length);
                                //사냥하면 안되는 새 스레드 깨움
                                noHuntThread.notify();
                                noHuntThread.setRandNum(randNum);
                                birds[randNum].setIsAlive(false);
                                repaint();
                                //폭탄의 확률과 위치도 초기화
                                bombRandNum = (int)(Math.random()*100+1);
                                //사냥이 끝났으면 bomb=null로 초기화해서 사용자 입력
창에서 조건으로 사용
                                bomb=null;
                            }
                        }
                    } catch (InterruptedException e) {
                        return;
                    }
                }
            }
        }
}
```

<Bird.java>

각 새들이 상속받게 될 추상 클래스

```
import javax.swing.*;
    //단어를 가진 새
    //모든 새들의 부모 클래스가 되는 추상클래스
public abstract class Bird {
    //현재 진행중인 라운드. GameGround에서 설정돼서 새의 속도를 결정하는데 사용
    protected int round;
    protected JTextField userInput;//게임그라운드 입력창 레퍼런스 연결
        //textSource에서 받아와서 단어 설정
    protected String word;
    //살아있는지에 대한 Flag. NoHuntBird와 단어 사냥에서 사용
    private boolean isAlive =true;
    //GameGround 메소드와 remove, add를 위한 레퍼런스
    protected GameGround gameGround;
    //단어 소스를 위해 사용하는 레퍼런스
    protected TextSource textSource =new TextSource();
    //폭탄 사용을 위한 플래그
    protected boolean isMoving =true;
    public Bird(JTextField userInput, int round,GameGround gameGround) {
        //userInput은 새에 적혀있는 문자열과 입력을 확인하기 위한 레퍼런스
        //round는 새의 속도 조절을 위해 GameGround가 전달
        //gameGround는 remove와 add를 위해 레퍼런스 전달
        this.gameGround=gameGround;
        this.round = round;
        this.userInput = userInput;
        //textSource의 getNext()호출로 다음 단어로 지정
        this.word = setWord();
    }
    //폭탄이 터지면 실행. 새 움직임 일시정지
    public void pauseMovement() {
        isMoving =false;
    }
    // 새 움직임을 재개하는 메소드
    public void resumeMovement() {
        isMoving =true;
    }
    //textSource에서 txt 파일을 읽어 다음 단어를 읽어오는 메소드
    public String setWord() {
        return textSource.getNext();
    }
    //현재 word 반환 메소드
    public String getWord() {
        return word;
    }
    //살아있는지 반환. noHuntThread와 단어 입력 창에서 사용
    public boolean getIsAlive() {
        return isAlive;
    }
```

```
    //noHuntThread에서 사냥하면 안되는 새를 잡았다고 판단할 때 사용
    public void setIsAlive(boolean isAlive) {
        this.isAlive = isAlive;
    }
    //추상 메서드
    //상속받은 하위 새 클래스들에서 사용
    public abstract boolean isCorrect();
    public abstract void moveWings();
    public abstract JLabel getBirdLabel();
    public abstract ImageIcon getBirdImage();
    public abstract void setCorrectFlag(boolean correctFlag);
    public abstract Thread getThread();
}
```

<Eagle.java>

Bird를 상속받아 Eagle을 구현하는 클래스

```
import javax.swing.*;
import java.awt.*;
public class Eagle extends Bird {
    private JLabel eagleLabel;
    private Thread eagleThread;
    private boolean correctFlag =false;
    private int right =1;
    //현재 방향을 나타내는 변수
    //오른쪽으로 향하고 있으면 1
    private int current =1;
    @Override
    public Thread getThread() {
        return eagleThread;
    }
    //새들이 날개를 다르게 한 모양에 대한 이미지 4개
    //날개의 퍼덕임을 위해서 4개의 이미지(날개 위 아래 + 새의 머리 좌 우 향한 모습)
    private ImageIcon eagle1 =new ImageIcon("images/eagle1.png");
    private ImageIcon eagle2 =new ImageIcon("images/eagle2.png");
    private ImageIcon eagle3 =new ImageIcon("images/eagle3.png");
    private ImageIcon eagle4 =new ImageIcon("images/eagle4.png");
    private ImageIcon shot =new ImageIcon("images/shot.png");
    //맞았는지 여부를 판단할 플래그
    public void setCorrectFlag(boolean correctFlag) {
        this.correctFlag = correctFlag;
    }
    //Eagle 생성시 단어 입력창, 라운드, gameGround에 대한 레퍼런스 전달
    public Eagle(JTextField userInput, int round, GameGround gameGround) {
        super(userInput,round,gameGround);
        eagleLabel =new JLabel();
        eagleLabel.setIcon(eagle1);
        eagleLabel.setHorizontalTextPosition(JLabel.CENTER); // 텍스트 위치 조정
        eagleLabel.setVerticalTextPosition(JLabel.CENTER);
        eagleLabel.setForeground(Color.RED); // 텍스트 색상
        eagleLabel.setFont(new Font("고딕체", Font.BOLD, 20)); // 텍스트 폰트
        //사냥이 완료되면 Eagle 객체 자체가 사라지게 설계됐으므로 단어는 생성 시 설정
        eagleLabel.setText(this.word);
        //gameGround에서 스레드 인터럽트를 위해 스레드 레퍼런스를 반환하기 위한 레
퍼런스 변수
        this.eagleThread =new EagleThread();
        eagleThread.start();
    }
    //사냥하면 안되는 새 이미지에 사용되는 이미지 리턴
    @Override
    public ImageIcon getBirdImage(){
        return eagle1;
    }
    //맞았다면
```

```
//Eagle의 getWord와 입력창의 텍스트가 동일하다면
@Override
public boolean isCorrect() {
    if (getWord().equals(userInput.getText())) {
        //정답 플래그 up
        correctFlag=true;
        //죽은 것으로 처리
        setIsAlive(false);
        return true;
    }
    else return false;
}
//이 JLabel의 레퍼런스 리턴
@Override
public JLabel getBirdLabel() {
    return eagleLabel;
}
//오른쪽으로 가게 하는 함수
//X좌표가 1300이 넘으면 방향 설정 프래그 right 1로 설정(다시 왼쪽으로)
public void goRight() {
    eagleLabel.setLocation(eagleLabel.getX() +40, eagleLabel.getY());
    if (eagleLabel.getX() >1300) {
        right =1;
    }
}
//왼쪽으로 가게 하는 함수
//X좌표가 0보다 작아지면 방향 설정 프래그 right 0으로 설정(다시 오른쪽으로)
public void goLeft() {
    eagleLabel.setLocation(eagleLabel.getX() -40, eagleLabel.getY());
    if (eagleLabel.getX() <0) {
        right =0;
    }
}
//날개 퍼덕임을 구현하는 함수
//현재 상태에 따라 이미지를 다르게 해서 나는 것처럼 구현
public void moveWings() {
    if (current ==0 && right ==1) {
        eagleLabel.setIcon(eagle1);
        current =1;
    } else if (current ==1 && right ==1) {
        eagleLabel.setIcon(eagle2);
        current =0;
    } else if (current ==0 && right ==0) {
        eagleLabel.setIcon(eagle3);
        current =1;
    } else if (current ==1 && right ==0) {
        eagleLabel.setIcon(eagle4);
        current =0;
    }
}
class EagleThread extends Thread {
    @Override
    public void run() {
        while (true) {
```

```
                try {
                    //폭탄 정답을 맞추면 imMoving = false;
                    if(isMoving) {
                        moveWings();
                        //right Flag에 따라 방향 함수 다르게 호출
                        if (right ==1) {
                            goLeft();
                        } else {
                            goRight();
                        }
                    }
                    //맞았다면
                    if (correctFlag) {
                        //새가 총에 맞은 이미지로 레이블 변경
                        eagleLabel.setIcon(shot);
                        eagleLabel.repaint();
                        //새에 적혀있는 문자열 없애기
                        eagleLabel.setText("");
                        sleep(1000);
                        //새 이미지 제거
                        gameGround.remove(eagleLabel);
                        gameGround.repaint();
                        //인터럽트 처리를 통한 스레드 스케쥴링 멈춤
                        this.interrupt();
                    }
                    //sleep 매개변수 시간으로 날갯짓 시간 구현 + 초당 날아가는 거리
변경
                    //round는 GameGround에서 현재 라운드를 전달
                    //round가 올라갈수록 더 파닥거리고, 더 빨리 날아간다
                    sleep(500/round);
                } catch (InterruptedException e) {
                    return;
                }
            }
        }
    }
}
```

<miniBird.java>

Bird를 상속받아 miniBird를 구현하는 클래스

```
import javax.swing.*;
import java.awt.*;
public class MiniBird extends Bird {
    private JLabel miniBirdLabel; // MiniBird의 레이블
    private Thread miniBirdThread; // MiniBird의 스레드
    private boolean correctFlag =false; // 정답 여부를 판단하는 플래그
    @Override
    public Thread getThread() {
        return miniBirdThread; // MiniBird 스레드 반환
    }
    private int right =1; // 현재 방향을 나타내는 변수, 오른쪽으로 향하고 있으면 1
    private int current =1; // 날개 퍼덕임의 현재 상태
    // MiniBird의 날개 퍼덕임을 위한 이미지, 날개 위 아래 + 새의 머리 좌 우 향한 모습
의 4개 이미지
    private ImageIcon miniBird1 =new ImageIcon("images/minibird1.png");
    private ImageIcon miniBird2 =new ImageIcon("images/minibird2.png");
    private ImageIcon miniBird3 =new ImageIcon("images/minibird3.png");
    private ImageIcon miniBird4 =new ImageIcon("images/minibird4.png");
    private ImageIcon shot =new ImageIcon("images/shot.png"); // 새가 맞았을 때의 이
미지
    public void setCorrectFlag(boolean correctFlag) {
        this.correctFlag = correctFlag; // 정답 플래그 설정
    }
    // MiniBird 생성자, 단어 입력창, 라운드, GameGround에 대한 레퍼런스 전달
    public MiniBird(JTextField userInput, int round, GameGround gameGround) {
        super(userInput, round, gameGround);
        miniBirdLabel =new JLabel();
        miniBirdLabel.setIcon(miniBird1); // 초기 이미지 설정
        miniBirdLabel.setHorizontalTextPosition(JLabel.CENTER); // 텍스트 위치 조정
        miniBirdLabel.setVerticalTextPosition(JLabel.CENTER);
        miniBirdLabel.setForeground(Color.RED); // 텍스트 색상
        miniBirdLabel.setFont(new Font("고딕체", Font.BOLD, 20)); // 텍스트 폰트
        miniBirdLabel.setText(this.word); // 생성 시 설정되는 단어
        miniBirdThread =new MiniBirdThread(); // MiniBird 스레드 생성 및 시작
        miniBirdThread.start();
    }
    @Override
    public ImageIcon getBirdImage() {
        return miniBird1; // 사냥하면 안되는 새 이미지 반환
    }
    @Override
    public boolean isCorrect() {
        if (getWord().equals(userInput.getText())) {
            correctFlag =true; // 정답 플래그 설정
            setIsAlive(false); // 죽은 것으로 처리
            return true;
        } else return false;
```

```
}
@Override
public JLabel getBirdLabel() {
    return miniBirdLabel; // 이 JLabel의 레퍼런스 반환
}
public void goRight() {
    miniBirdLabel.setLocation(miniBirdLabel.getX() +10, miniBirdLabel.getY());
    if (miniBirdLabel.getX() >1300) {
        right =1; // 오른쪽으로 이동 중 방향 변경
    }
}
public void goLeft() {
    miniBirdLabel.setLocation(miniBirdLabel.getX() -10, miniBirdLabel.getY());
    if (miniBirdLabel.getX() <0) {
        right =0; // 왼쪽으로 이동 중 방향 변경
    }
}
public void moveWings() {
    // 날개 퍼덕임 구현, 현재 상태와 방향에 따라 다른 이미지 설정
    if (current ==0 && right ==1) {
        miniBirdLabel.setIcon(miniBird1);
        current =1;
    } else if (current ==1 && right ==1) {
        miniBirdLabel.setIcon(miniBird2);
        current =0;
    } else if (current ==0 && right ==0) {
        miniBirdLabel.setIcon(miniBird3);
        current =1;
    } else if (current ==1 && right ==0) {
        miniBirdLabel.setIcon(miniBird4);
        current =0;
    }
}
class MiniBirdThread extends Thread {
    @Override
    public void run() {
        while (true) {
            try {
                if (isMoving) {
                    moveWings(); // 날개 퍼덕임
                    if (right ==1) {
                        goLeft(); // 왼쪽 이동
                    } else {
                        goRight(); // 오른쪽 이동
                    }
                }
                if (correctFlag) {
                    // 맞았을 경우 처리, 이미지 변경 및 레이블 제거
                    miniBirdLabel.setIcon(shot);
                    miniBirdLabel.repaint();
                    miniBirdLabel.setText("");
                    sleep(1000);
                    gameGround.remove(miniBirdLabel);
                    gameGround.repaint();
```

```
                        this.interrupt(); // 스레드 종료
                    }
                    sleep(500 / round); // 날아가는 속도 조절
                } catch (InterruptedException e) {
                    return;
                }
            }
        }
    }
}
```

<Parrot.java>

Bird를 상속받아 Parrot을 구현하는 클래스

```
import javax.swing.*;
import java.awt.*;
public class Parrot extends Bird {
    private JLabel parrotLabel; // Parrot의 레이블
    private Thread parrotThread; // Parrot의 스레드
    private boolean correctFlag =false; // 정답 여부를 판단하는 플래그
    @Override
    public Thread getThread() {
        return parrotThread; // Parrot 스레드 반환
    }
    private int right =1; // 현재 방향을 나타내는 변수, 오른쪽으로 향하고 있으면 1
    private int current =1; // 날개 퍼덕임의 현재 상태
    // Parrot의 날개 퍼덕임을 위한 이미지, 날개 위 아래 + 새의 머리 좌 우 향한 모습의
4개 이미지
    private ImageIcon parrot1 =new ImageIcon("images/parrot1.png");
    private ImageIcon parrot2 =new ImageIcon("images/parrot2.png");
    private ImageIcon parrot3 =new ImageIcon("images/parrot3.png");
    private ImageIcon parrot4 =new ImageIcon("images/parrot4.png");
    private ImageIcon shot =new ImageIcon("images/shot.png"); // 새가 맞았을 때의 이
미지
    public void setCorrectFlag(boolean correctFlag) {
        this.correctFlag = correctFlag; // 정답 플래그 설정
    }
    // Parrot 생성자, 단어 입력창, 라운드, GameGround에 대한 레퍼런스 전달
    public Parrot(JTextField userInput, int round, GameGround gameGround) {
        super(userInput, round, gameGround);
        parrotLabel =new JLabel();
        parrotLabel.setIcon(parrot1); // 초기 이미지 설정
        parrotLabel.setHorizontalTextPosition(JLabel.CENTER); // 텍스트 위치 조정
        parrotLabel.setVerticalTextPosition(JLabel.CENTER);
        parrotLabel.setForeground(Color.RED); // 텍스트 색상
        parrotLabel.setFont(new Font("고딕체", Font.BOLD, 20)); // 텍스트 폰트
        parrotLabel.setText(this.word); // 생성 시 설정되는 단어
        parrotThread =new ParrotThread(); // Parrot 스레드 생성 및 시작
        parrotThread.start();
    }
    @Override
    public ImageIcon getBirdImage() {
        return parrot1; // 사냥하면 안되는 새 이미지 반환
    }
    @Override
    public boolean isCorrect() {
        if (getWord().equals(userInput.getText())) {
            correctFlag =true; // 정답 플래그 설정
            setIsAlive(false); // 죽은 것으로 처리
            return true;
        } else return false;
```

```
    }
    @Override
    public JLabel getBirdLabel() {
        return parrotLabel; // 이 JLabel의 레퍼런스 반환
    }
    public void goRight() {
        parrotLabel.setLocation(parrotLabel.getX() +25, parrotLabel.getY()-5); // 오른쪽
으로 이동
        if (parrotLabel.getX() >1300) {
            right =1; // 오른쪽으로 이동 중 방향 변경
        }
    }
    public void goLeft() {
        parrotLabel.setLocation(parrotLabel.getX() -25, parrotLabel.getY()+5); // 왼쪽으
로 이동
        if (parrotLabel.getX() <0) {
            right =0; // 왼쪽으로 이동 중 방향 변경
        }
    }
    public void moveWings() {
        // 날개 퍼덕임 구현, 현재 상태와 방향에 따라 다른 이미지 설정
        if (current ==0 && right ==1) {
            parrotLabel.setIcon(parrot1);
            current =1;
        } else if (current ==1 && right ==1) {
            parrotLabel.setIcon(parrot2);
            current =0;
        } else if (current ==0 && right ==0) {
            parrotLabel.setIcon(parrot3);
            current =1;
        } else if (current ==1 && right ==0) {
            parrotLabel.setIcon(parrot4);
            current =0;
        }
    }
    class ParrotThread extends Thread {
        @Override
        public void run() {
            while (true) {
                try {
                    if(isMoving) {
                        moveWings(); // 날개 퍼덕임
                        if (right ==1) {
                            goLeft(); // 왼쪽 이동
                        } else {
                            goRight(); // 오른쪽 이동
                        }
                    }
                    if (correctFlag) {
                        // 맞았을 경우 처리, 이미지 변경 및 레이블 제거
                        parrotLabel.setIcon(shot);
                        parrotLabel.repaint();
                        parrotLabel.setText("");
                        sleep(1000);
```

```
                        gameGround.remove(parrotLabel);
                        gameGround.repaint();
                        this.interrupt(); // 스레드 종료
                    }
                    sleep(700/round); // 날아가는 속도 조절
                }
                catch (InterruptedException e) {
                    return;
                }
            }
        }
    }
}
```

<Sparrow.java>

Bird를 상속받아 Sparrow를 구현하는 클래스

```
import javax.swing.*;
import java.awt.*;
public class Sparrow extends Bird { // 참새, 10점
    private JLabel sparrowLabel; // Sparrow의 레이블
    private Thread sparrowThread; // Sparrow의 스레드
    private int right =0; // 현재 방향을 나타내는 변수, 오른쪽으로 향하고 있으면 0
    private int current =0; // 날개 퍼덕임의 현재 상태
    private boolean correctFlag =false; // 정답 여부를 판단하는 플래그
    @Override
    public Thread getThread() {
        return sparrowThread; // Sparrow 스레드 반환
    }
    // Sparrow의 날개 퍼덕임을 위한 이미지, 날개 위 아래 + 새의 머리 좌 우 향한 모습
의 4개 이미지
    private ImageIcon sparrow1 =new ImageIcon("images/sparrow1.png");
    private ImageIcon sparrow2 =new ImageIcon("images/sparrow2.png");
    private ImageIcon sparrow3 =new ImageIcon("images/sparrow3.png");
    private ImageIcon sparrow4 =new ImageIcon("images/sparrow4.png");
    private ImageIcon shot =new ImageIcon("images/shot.png"); // 새가 맞았을 때의 이
미지
    public Sparrow(JTextField userInput, int round, GameGround gameGround) {
        super(userInput, round, gameGround); // 상위 클래스 생성자 호출
        sparrowLabel =new JLabel();
        sparrowLabel.setIcon(sparrow1); // 초기 이미지 설정
        sparrowLabel.setHorizontalTextPosition(JLabel.CENTER); // 텍스트 위치 조정
        sparrowLabel.setVerticalTextPosition(JLabel.CENTER);
        sparrowLabel.setForeground(Color.WHITE); // 텍스트 색상
        sparrowLabel.setFont(new Font("Monospaced", Font.BOLD, 20)); // 텍스트 폰트
        sparrowLabel.setText(this.word); // 생성 시 설정되는 단어
        this.sparrowThread =new SparrowThread(); // Sparrow 스레드 생성 및 시작
        sparrowThread.start();
    }
    public void setCorrectFlag(boolean correctFlag) {
        this.correctFlag = correctFlag; // 정답 플래그 설정
    }
    @Override
    public ImageIcon getBirdImage() {
        return sparrow1; // 사냥하면 안되는 새 이미지 반환
    }
    @Override
    public boolean isCorrect() {
        if (getWord().equals(userInput.getText())) {
            correctFlag =true; // 정답 플래그 설정
            setIsAlive(false); // 죽은 것으로 처리
            return true;
        } else return false;
    }
```

```
    @Override
    public JLabel getBirdLabel() {
        return sparrowLabel; // 이 JLabel의 레퍼런스 반환
    }
    public void goRight() {
        sparrowLabel.setLocation(sparrowLabel.getX() +20, sparrowLabel.getY()); // 오
른쪽으로 이동
        if (sparrowLabel.getX() >1300) {
            right =1; // 오른쪽으로 이동 중 방향 변경
        }
    }
    public void goLeft() {
        sparrowLabel.setLocation(sparrowLabel.getX() -20, sparrowLabel.getY()); // 왼
쪽으로 이동
        if (sparrowLabel.getX() <0) {
            right =0; // 왼쪽으로 이동 중 방향 변경
        }
    }
    public void moveWings() {
        // 날개 퍼덕임 구현, 현재 상태와 방향에 따라 다른 이미지 설정
        if (current ==0 && right ==1) {
            sparrowLabel.setIcon(sparrow1);
            current =1;
        } else if (current ==1 && right ==1) {
            sparrowLabel.setIcon(sparrow2);
            current =0;
        } else if (current ==0 && right ==0) {
            sparrowLabel.setIcon(sparrow3);
            current =1;
        } else if (current ==1 && right ==0) {
            sparrowLabel.setIcon(sparrow4);
            current =0;
        }
    }
    class SparrowThread extends Thread {
        @Override
        public void run() {
            while (true) {
                try {
                    if (isMoving) {
                        moveWings(); // 날개 퍼덕임
                        if (right ==1) {
                            goLeft(); // 왼쪽 이동
                        } else {
                            goRight(); // 오른쪽 이동
                        }
                    }
                    if (correctFlag) {
                        // 맞았을 경우 처리, 이미지 변경 및 레이블 제거
                        sparrowLabel.setIcon(shot);
                        sparrowLabel.repaint();
                        sparrowLabel.setText("");
                        sleep(1000);
                        gameGround.remove(sparrowLabel);
```

```
                        gameGround.repaint();
                        this.interrupt(); // 스레드 종료
                    }
                    sleep(600 / round); // 날아가는 속도 조절
                } catch (InterruptedException e) {
                    return;
                }
            }
        }
    }
}
```

<Bomb.java>

새의 움직임을 멈추는 폭탄을 구현하는 클래스

```
import javax.swing.*;
import java.awt.*;
public class Bomb {
    //GameGround에 입력하는 사용자 입력창
    private JTextField userInput;
    //Bomb add와 remove를 위한 레퍼런스
    private GameGround gameGround;
    protected JLabel bombLabel;

    private ImageIcon bombImg =new ImageIcon("images/bomb.png");
    private ImageIcon bombShotImg =new ImageIcon("images/bombshot.png");
    //폭탄에 적힐 단어 선정을 위해 textSource 레퍼런스 전달
    protected TextSource textSource =new TextSource();
    //폭탄에 적힌 단어
    private String word;
    //gameGround에서 중단시키기 위한 스레드 레퍼런스
    private BombThread bombThread;
    //맞았는지에 대한 플래그
    private boolean correctFlag =false;
    Bomb(JTextField userInput, GameGround gameGround) {
        this.userInput = userInput;
        this.gameGround = gameGround;
        bombLabel =new JLabel();
        bombLabel.setIcon(bombImg);
        this.word = textSource.getNext();
        bombLabel.setText(word);
        bombLabel.setForeground(Color.WHITE);
        bombLabel.setFont(new Font("고딕체", Font.BOLD, 30));
        bombLabel.setHorizontalTextPosition(JLabel.CENTER); // 텍스트 위치 조정
        bombLabel.setVerticalTextPosition(JLabel.CENTER);
        //bomb 객체 생성시 스레드 실행
        //bomb가 없으면 스레드가 작동하지 않는 최적화를 위해
        //gameGround에서 bomb 답 맞추면 참조 변수를 null로 초기화
        bombThread =new BombThread();
        bombThread.start();
    }
    public JLabel getBombLabel() {
        return bombLabel;
    }
    public String getWord() {
        return word;
    }
    //맞았다면
    public boolean isCorrect() {
        if (getWord().equals(userInput.getText())) {
            correctFlag=true;
            return true;
```

```
        } else return false;
    }
    //폭탄이 떨어지는 것을 구현한 메서드
    class BombThread extends Thread {
        @Override
        public void run() {
            while (true) {
                try {
                    bombLabel.setLocation(bombLabel.getX(), bombLabel.getY() +10);
                    bombLabel.repaint();
                    //Y좌표가 600 이상(입력창보다 아래쯤)이면 폭탄 삭제 후 인터럽트
                    if(bombLabel.getY()>600){
                        gameGround.remove(bombLabel);
                        gameGround.repaint();
                        this.interrupt();
                    }
                    //정답이 맞았으면 폭탄이 얼음으로 터지는 이미지를 보여주고 폭탄
문자열 초기화
                    //3초간 얼음을 보여주고 새들 모두 정지
                    //그 후 이미지 제거
                    if(correctFlag){
                        gameGround.pauseAllBirds();
                        bombLabel.setIcon(bombShotImg);
                        bombLabel.setText("");
                        gameGround.repaint();
                        sleep(3000); // 3초 동안 모든 새들의 움직임을 멈춤
                        gameGround.resumeAllBirds();
                        gameGround.remove(bombLabel);
                        gameGround.repaint();
                    //3초 지난 뒤에 인터럽트를 통해 스레드 스케쥴링 멈춤
                        this.interrupt();
                    }
                    sleep(100);
                } catch (InterruptedException e) {
                    return;
                }
            }
        }
    }
}
```

<TextSource.java>

txt 파일에서 단어를 읽어 새와 폭탄에 적힐 단어를 선정하는 클래스

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;
import java.util.Vector;
//word source 파일에서 단어를 읽어오는 클래스
public class TextSource {
    //원본단어 5965개
    //로딩 속도를 위해 벡터의 초기 크기 6000개로 설정
    //벡터 내 랜덤 단어를 쉽게 얻어내고, 추가하기 위해 배열 아닌 벡터 사용
    private Vector<String> wordVector =new Vector<String>(6000);
    public TextSource(){
        //int count =0;
        try {
            //source 폴더/ korean.txt를 FileRead
            Scanner scanner =new Scanner(new FileReader("source/korean.txt"));
            while(scanner.hasNext()){
                String word = scanner.nextLine();
                wordVector.add(word);
                //count++;
            }
            //System.out.println(count);

        }
        catch (FileNotFoundException e){//예외처리
            System.out.println("no file exist");
            System.exit(0);
        }
    }
    //벡터의 다음 단어 리턴
    //새와 폭탄 문자열 표시에 사용
    public String getNext(){
        //벡터의 크기 변수에 저장
        int wordVectorsize = wordVector.size();
        //벡터 크기만큼 random Int 생성
        int idx = (int)(Math.random()*wordVectorsize);
        //그 인덱스에 해당하는 단어 리턴
        return wordVector.get(idx);
    }
}
```

<ScorePanel.java>

게임 화면 우측에 나타나는 스코어, 랭킹을 표시하는 클래스

```
import javax.swing.*;
import java.awt.*;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
import java.util.List;
public class ScorePanel extends JPanel {
    private int round=1;
    private String playerName;
    private int score;
    private ImageIcon heartImg=new ImageIcon("images/heart.png");
    private Vector<JLabel> hearts =new Vector<>(3);
    private JLabel scoreLabel =new JLabel(Integer.toString(score));
    public JLabel roundNumLabel =new JLabel(Integer.toString(round));
    private HashMap<String,ScoreRound> scoreRoundHashMap=new HashMap<>();
    // RGB 코드로 예쁜 색깔 설정
    Color bgcolor =new Color(255,218,185);

    //HashMap에 사용될 클래스
    //점수와 라운드를 묶어서 객체화
    protected class ScoreRound{
        private int score;
        private int round;
        public ScoreRound( int score, int round){
            this.score=score; this.round=round;
        }
    }
    public ScorePanel(){
        this.setBackground(bgcolor);
        try {
            //txt 파일의 각 라인을 모두 읽고
            Scanner scanner =new Scanner(new FileReader("source/ranking.txt"));
            while (scanner.hasNext()){
                StringTokenizer stk =new StringTokenizer(scanner.nextLine(),",");
                String name = stk.nextToken();
                int score = Integer.parseInt(stk.nextToken());
                int round = Integer.parseInt(stk.nextToken());
                ScoreRound s =new ScoreRound(score,round);
                //객체화 해서 <name,객체>쌍을 만들어 Hashmap에 모두 입력
                scoreRoundHashMap.put(name,s);
            }
        }
        catch(FileNotFoundException e){
            System.out.println("can't open File!");
            System.exit(0);
        }
```

```
        RankingPanel rankingPanel =new RankingPanel();
        rankingPanel.setBounds(0,400,420,680);
        add(rankingPanel);
        setLayout(null);
        JLabel round =new JLabel("Round");
        JLabel score =new JLabel("점수 : ");
        scoreLabel.setFont(new Font("고딕체",Font.BOLD,30));
        score.setFont(new Font("고딕체",Font.BOLD,30));
        round.setFont(new Font("고딕체",Font.BOLD,50));
        roundNumLabel.setFont(new Font("고딕체",Font.BOLD,50));
        round.setSize(250,50);
        round.setLocation(50,30);
        score.setSize(100,50);
        scoreLabel.setSize(100,50);
        score.setLocation(100,100);
        scoreLabel.setLocation(200,100);
        roundNumLabel.setSize(50,50);
        roundNumLabel.setLocation(250,30);
        //목숨을 의미하는 하트 이미지 3개 생성
        //하트 이미지를 배열로 관리해서 오른쪽부터 편하게 삭제 가능
        for(int i=0;i<3;i++){
            hearts.add(new JLabel());
            hearts.get(i).setIcon(heartImg);
            hearts.get(i).setSize(120,120);
            hearts.get(i).setLocation(130*i,180);
            add(hearts.get(i));
        }
        add(score);
        add(scoreLabel);
        add(round);
        add(roundNumLabel);
    }
    //맞췄을 때
    //gameGround의 액션리스너가 호출
    public void increase(){
        score+=10;
        scoreLabel.setText(Integer.toString(score));
    }
    //틀렸을 때
    //gameGround의 액션리스너가 호출
    //틀렸으니까 하트도 하나 삭제
    public void decrease(){
        score-=20;
        if(score<0) score =0;
        scoreLabel.setText(Integer.toString(score));
        lostHeart();
    }
    //하트를 모두 소진했을 경우를 검출
    public void lostHeart(){
        //만약 하트가 아직 남았다면
        if(hearts.size()>0) {
            //하트를 인덱스로 관리해서 오른쪽 하트부터 삭제
            this.remove(hearts.get(hearts.size() -1));
            this.repaint();
```

```
            hearts.remove(hearts.size() -1);
        }
    }
    //패배한 경우를 검출하는 함수
    //만약 남은 하트가 하나도 없는 경우 true 리턴
    public boolean isLose(){
        if(hearts.size()==0){
            return true;
        }
        else return false;
    }
    //현재 라운드 설정 함수
    //ranking.txt 입력 시 사용
    public void setRound(int round){
        this.round=round;
    }
    //게임 종료 후 gameGround의 사용자 이름 입력 창에서 호출하는 함수
    //ranking.txt 입력 시 사용
    public void setPlayerName(String playerName){
        this.playerName=playerName;
        //객체 생성 후
        ScoreRound sr =new ScoreRound(score,round);
        //HashMap에 입력
        scoreRoundHashMap.put(playerName,sr);
        //ranking.txt에 저장하는 함수
        saveScoresToFile();
    }
    //ranking.txt에 저장하는 함수
    public void saveScoresToFile() {
        try {
            FileWriter writer =new FileWriter("source/ranking.txt",true);
                writer.write(playerName +","+ score +","+ round +"₩n"); // 파일에 쓰기
                writer.close();
        } catch (IOException e) {
            System.out.println("can't open File!");
            System.exit(0);
        }
    }
    public class RankingPanel extends JPanel {
        private ImageIcon goldImg =new ImageIcon("images/gold.png");
        private ImageIcon silverImg =new ImageIcon("images/silver.png");
        private ImageIcon bronzeImg =new ImageIcon("images/bronze.png");
        JLabel [] rankOne =new JLabel[3];
        JLabel [] rankTwo =new JLabel[3];
        JLabel [] rankThree =new JLabel[3];
        JLabel name =new JLabel("이름");
        JLabel score =new JLabel("점수");
        JLabel round =new JLabel("라운드");
        JLabel blank =new JLabel("");
        JLabel goldMedal =new JLabel();
        JLabel silverMedal =new JLabel();
        JLabel bronzeMedal =new JLabel();
        public RankingPanel() {
            this.setBackground(bgcolor);
```

```
                setLayout(null);
                goldMedal.setIcon(goldImg);
                silverMedal.setIcon(silverImg);
                bronzeMedal.setIcon(bronzeImg);
                goldMedal.setSize(100,100);
                silverMedal.setSize(100,100);
                bronzeMedal.setSize(100,100);
                name.setFont(new Font("고딕체",Font.BOLD,30));
                score.setFont(new Font("고딕체",Font.BOLD,30));
                round.setFont(new Font("고딕체",Font.BOLD,30));
                blank.setSize(100,30);
                name.setSize(100,30);
                score.setSize(100,30);
                round.setSize(100,30);
                blank.setLocation(0,0);
                name.setLocation(70,50);
                score.setLocation(180,50);
                round.setLocation(280,50);
                goldMedal.setLocation(0,100); //1등 금메달
                silverMedal.setLocation(0,200); // 2등 은메달
                bronzeMedal.setLocation(0,300);// 3등 동메달
                add(blank);    add(name);    add(score);    add(round);    add(goldMedal);
add(silverMedal); add(bronzeMedal);
                for(int i=0;i<3;i++){
                    rankOne[i] =new JLabel();
                    rankTwo[i] =new JLabel();
                    rankThree[i] =new JLabel();
                }
                //랭킹을 산정을 위해 scoreRoundHashMap의 키를 리스트로 변환
                List<String> keys =new ArrayList<>(scoreRoundHashMap.keySet());
                // 리스트를 사용자의 점수 순으로 내림차순으로 정렬
                keys.sort((k1, k2) -> Integer.compare(scoreRoundHashMap.get(k2).score,
scoreRoundHashMap.get(k1).score));
                // 정렬된 리스트 순회
                // 1,2,3등만 사용
                for (int i =0; i < keys.size(); i++) {
                    String playerName = keys.get(i);
                    ScoreRound s = scoreRoundHashMap.get(playerName);
                    // 상위 3명에게 메달 할당
                    if (i ==0) { // 1등
                        rankOne[0].setText(playerName);
                        rankOne[1].setText(String.valueOf(s.score));
                        rankOne[2].setText(String.valueOf(s.round));
                    } else if (i ==1) { // 2등
                        rankTwo[0].setText(playerName);
                        rankTwo[1].setText(String.valueOf(s.score));
                        rankTwo[2].setText(String.valueOf(s.round));
                    } else if (i ==2) { // 3등
                        rankThree[0].setText(playerName);
                        rankThree[1].setText(String.valueOf(s.score));
                        rankThree[2].setText(String.valueOf(s.round));
                    }
                }
                for(int i=0;i<3;i++){
```

```
                rankOne[i].setSize(100,50);
                rankTwo[i].setSize(100,50);
                rankThree[i].setSize(100,50);
                rankOne[i].setFont(new Font("고딕체", Font.BOLD, 30));
                rankTwo[i].setFont(new Font("고딕체", Font.BOLD, 30));
                rankThree[i].setFont(new Font("고딕체", Font.BOLD, 30));
            }
            for(int i=0;i<3;i++){
                rankOne[i].setLocation(70 +120*i,130);
                rankTwo[i].setLocation(70 +120*i, 230);
                rankThree[i].setLocation(70+120*i,330);
                add(rankOne[i]);
                add(rankTwo[i]);
                add(rankThree[i]);
            }
        }
    }
}
```

<AddWordPanel.java>

단어 추가 버튼을 눌러 사용자가 입력하는 단어를 추가하는 기능을 구현한 클래스

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileWriter;
import java.io.IOException;
public class AddWordPanel extends JPanel {
    private JLabel title =new JLabel("단어 추가 메뉴");
    //사용자 이름 입력창 레퍼런스
    private JTextField input =new JTextField(30);
    //GampFrame의 showMainmeu() 호출을 위한 버튼
    private JButton backMenuPanel =new JButton("돌아가기");
    //단어 추가를 위한 FileWriter
    private FileWriter writer;
    //GampFrame의 showMainmeu() 호출을 위한 레퍼런스 변수
    private GameFrame gameFrame;
    //배경화면
    private ImageIcon sky =new ImageIcon("images/sky.jpg");
    private Image skyImg = sky.getImage();
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(skyImg, 0, 0, this.getWidth(), this.getHeight(), this);
    }
    //GameFrame의 showMainmenu() 호출을 위해 생성자에서 GameFrame을 인자로 받
음
    public AddWordPanel(GameFrame gameFrame){
        this.gameFrame=gameFrame;
        try {
            writer =new FileWriter("source/korean.txt",true);
        } catch (IOException e) { //예외처리
            System.out.println("can't open File! // 생성자 에러");
            System.exit(0);
        }
        setLayout(null);
        title.setSize(1000,200);
        title.setFont(new Font("고딕체",Font.BOLD,100));
        title.setForeground(Color.WHITE);
        title.setLocation(600,200);
        input.setSize(1000, 50);
        input.setFont(new Font("고딕체", Font.BOLD, 30));
        input.setLocation(450, 800);
        backMenuPanel.setSize(100,100);
        backMenuPanel.setLocation(900,600);
        input.addActionListener(new InputActionListener());
        backMenuPanel.addActionListener(new ActionListener() {
            //메인메뉴로 돌아가는 버튼을 클릭했을 때 호출
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                try {
                    //돌아갈 때 Writer 객체 close
                    writer.close();
                } catch (IOException event) {
                    System.out.println("can't open File! // 버튼에러");
                    System.exit(0);
                }
                //예외가 발생하지 않으면 메인메뉴로
                gameFrame.showMainMenu();
            }
        });
        add(title);
        add(input);
        add(backMenuPanel);
    }
    class InputActionListener implements ActionListener{
        //단어추가 필드에 단어를 입력 후 엔터를 누르면 호출
        @Override
        public void actionPerformed(ActionEvent e) {
            //입력창에서 단어를 읽어서
            String word =  input.getText();
            //txt 파일에 입력하고
            addWord(word);
            //입력창 초기화
            input.setText("");
        }
    }
    //korean.txt 파일에 단어를 추가하는 함수
    public void addWord(String word){
        try {
            writer.write(word +"₩n"); // 단어 추가 후 줄바꿈
            writer.flush(); //버퍼 비워버리기
        } catch (IOException e) {
            System.out.println("can't open File! // addword 함수에러");
            System.exit(0);
        }
    }
    }
```

5. 프로젝트 결론

모바일&스마트시스템 강의와 함께 완성한 인생 두 번째 프로젝트였다. 프로젝트를 진행하며 실력이 늘어난 지금에서 돌아보면, 간단하게 떨어지는 단어를 맞추면 점수가 오르는 정도의 게임이었으면 큰 어려움 없이 완성했을 것 같다. 어려움에서 배운다는 생각을 많이 하게 된 프로젝트였다.

처음 제안서를 작성할 때만 해도 이렇게까지 여러 가지 기능을 넣을 생각은 없었지만, 만들면 만들수록 더 욕심이 났다. 사실 이렇게 보고서를 작성하면서도 이 기능, 저 기능 더 넣어보고 싶은 아쉬움이 있다.

프로젝트를 진행하면서 크게 두 가지의 어려움을 겪었다. 첫 번째는 새의 움직임을 구현하는 것이었고, 두 번째는 잡으면 안되는 새를 만드는 것이었다.
먼저, 새의 움직임을 구현하기 위한 아이디어는 교수님과 함께 강의 시간에 풀이했던 간단한 스윙 프로그램에서 아이디어를 얻었다. @로 표시되는 사용자가 M으로 표시되는 괴물을 피해 달아나는 게임이었다. 빠르게 움직이면 생각보다 자연스러운 움직임이 나오는 것을 깨닫고, 새가 날개를 퍼덕이는 듯한 모습을 보이기 위해 위아래를 향한 날개의 모습과 머리 방향이 좌우인 4개의 이미지를 구했다. 레이블의 이미지를 바꾸면서 움직이고, 스레드로 이 행동을 빠르게 하자 꽤 자연스러운 새의 움직임을 구현할 수 있었다.
두 번째는, 잡으면 안되는 새였다. 문득 떠오른 아이디어였고 정말 어려웠지만 재밌을 것 같아서 끝까지 욕심부려 완성했다. 어느정도 완성했을 때, 여러 가지 문제가 생겼었다. 처음 구현했을 때는 스레드 동기화를 이해하지 못하고 그저 따로 작동하는 스레드로 구현했었다. 이렇게 하자, 사냥이 끝나지도 않았는데 사냥하면 안되는 새가 바뀌거나, 사냥하면 안되는 새를 잡아도 점수가 오르거나, 사냥하면 안되는 새의 스레드가 또 작동해 새 레이블이 겹치는 등 정말 많은 문제가 발생했다. 또, 인터럽트를 발생시켜 이미지를 바꾸는 방식으로 구현하니 sleep 시간동안 인터럽트 예외가 발생하지 않는 경우도 생겼다. 이 문제를 해결하기 위해 Flag를 사용하기도 했는데, 사냥이 완료된 시점에 기존의 새들이 채 제거되기도 전에 새로운 새가 추가되어 화면에 너무 많은 새가 있는 문제가 생기기도 했다.
스레드 동기화 강의를 듣고, 스스로 공부하고 고민하기도 하면서 문제를 해결했다. 사냥하면 안되는 새는 한번 실행 후 wait하고, 사냥이 완료되면 게임을 진행하는 메인 스레드가 사냥하면 안되는 새를 설정하는 스레드를 깨운다. notify되는 순간 이 스레드는 기존의 사냥하면 안되는 새를 다른 새로 바꾸게 된다. 또, 기존의 새들이 제거되기 전에 새로운 새들이 추가되던 문제는 join을 통해 해결했다. 사냥하면 안되는 새나, 메인 게임 스레드와는 다르게 각 새들의 스레드는 정답을 맞출 때 마다 인터럽트를 통해 return을 실행하고 종료된다. join을 통해 각 새들의 스레드가 끝날 때 까지 기다려주도록 구현해서 문제를 해결했다.

두 가지 문제 외에도, Bomb의 구현이나, 세련된 디자인, 랭킹 시스템 등 여러 문제를 겪었다. 여러 문제를 겪으면서 세부적으로 알게 된 것이 정말 많았다. 예를 들어, 단어를 저장하는 함수에서 FileWriter의 두 번째 매개변수로 true를 주는 것을 몰라서 단어 소스파일을 통째로 날려버리기도 했다. 소스파일을 어디서 구했었는지 기억이 나지않아 정말 난처했었다. 결국

다시 찾아내고, 두 번째 매개변수가 어떤 기능을 하는지 알게 되었다.

모바일&스마트시스템 강의 때도 느꼈지만, 좋은 프로그래머는 “단순히 코드를 잘 짠다.”, “알고리즘을 잘 짠다.” 와는 좀 다른 것 같다. 하나의 온전한 프로그램을 만들기 위해선 알아야 할 것이 정말 너무너무 많다. 그 모든 것들을 깊이 있게, 정확히 이해해서 활용하는 능력을 갖춘 사람이 좋은 프로그래머가 되는 것 같다. 모바일&스마트시스템 강의 프로젝트에선 통신 프로토콜과 하드웨어적 이해가 필요했듯, 객체지향언어2 강의 프로젝트 완성을 위해선 스레드와 그래픽, 객체지향프로그래밍에 대해 모두 이해하고 있어야 했다.

그래도 꽤 그럴싸한 게임을 완성해서 뿌듯하다. 언젠가 내가 완성한 프로젝트가 포트폴리오가 될 수 있도록, 이대로 그만두지 않고 욕심나던 기능을 이것저것 추가해 볼 생각이다. 지금은 단어 입력을 통해 사냥하지만, 마우스로 클릭을 통해 사냥한다던지, 좀 더 실감나는 효과를 넣어 본다던지, 일정 라운드마다 특별 몬스터가 나온다거나 하는 식이다.

책에 모든 지식을 적어둘 수도 없고, 내 이해와 암기가 완벽할 수 없다는 것도 알았다. 직접 무언가를 만들어 보면서, 부족한 부분을 깨닫고, 공부하고, 궁금하고 답답한 부분을 찾아가며 공부하는 것이 진짜 프로그래밍 공부라는 것을 깨닫게 됐다.