**Міністерство освіти і науки України**
**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**
**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

# Звіт

з лабораторної роботи № 6 з дисципліни
«Проектування алгоритмів»

**„Пошук в умовах протидії, ігри з повною інформацією, ігри з елементом випадковості, ігри з неповною інформацією”**

**Виконав(ла)** *ІП-22 Андреєва Уляна Андріївна*
(шифр, прізвище, ім'я, по батькові)

**Перевірив** *Ахаладзе Ілля Елдарійович*
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

# 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи - вивчити основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Ознайомитися з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією, іграх з елементами випадковості та в іграх з неповною інформацією.

# 2 ЗАВДАННЯ

Для ігор з повної інформацією, згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм альфа-бета-відсікань. Реалізувати три рівні складності (легкий, середній, складний).

Для ігор з елементами випадковості, згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток, з користувацьким інтерфейсом, не консольним, для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм мінімакс.

Для карткових ігор, згідно варіанту (таблиця 2.1), реалізувати візуальний ігровий додаток, з користувацьким інтерфейсом, не консольним, для гри користувача з комп'ютерним опонентом. Потрібно реалізувати стратегію комп'ютерного опонента, і звести гру до гри з повною інформацією (див. Лекцію), далі реалізувати стратегію гри комп'ютерного опонента за допомогою алгоритму мінімаксу або альфа-бета-відсікань.

Реалізувати анімацію процесу жеребкування (+1 бал) або реалізувати анімацію ігрових процесів (роздачі карт, анімацію ходів тощо) (+1 бал).

Реалізувати варто тільки одне з бонусних завдань.

Зробити узагальнений висновок лабораторної роботи.

Таблиця 2.1 – Варіанти

| № | Варіант | Тип гри |
|---|---|---|
| 1 | Яцзи https://game-wiki.guru/published/igryi/yaczzyi.html | З елементами випадковості |
| 2 | Лудо http://www.iggamecenter.com/info/ru/ludo.html | З елементами випадковості |
| 3 | Генерал http://www.rules.net.ru/kost.php?id=7 | З елементами випадковості |

| | | |
|---|---|---|
| 4 | Нейтріко http://www.iggamecenter.com/info/ru/neutreeko.html | З повною інформацією |
| 5 | Тринадцять http://www.rules.net.ru/kost.php?id=16 | З елементами випадковості |
| 6 | Індійські кості http://www.rules.net.ru/kost.php?id=9 | З елементами випадковості |
| 7 | Dots and Boxes https://ru.wikipedia.org/wiki/Палочки_(игра) | З повною інформацією |
| 8 | Двадцять одне http://gamerules.ru/igry-v-kosti-part8#dvadtsat-odno | З елементами випадковості |
| 9 | Тіко http://www.iggamecenter.com/info/ru/teeko.html | З повною інформацією |
| 10 | Клоббер http://www.iggamecenter.com/info/ru/clobber.html | З повною інформацією |
| 11 | 101 https://www.durbetsel.ru/2_101.htm | Карткові ігри |
| 12 | Hackenbush http://www.papg.com/show?1TMP | З повною інформацією |
| 13 | Табу https://www.durbetsel.ru/2_taboo.htm | Карткові ігри |
| 14 | Заєць і Вовки (за Зайця) http://www.iggamecenter.com/info/ru/foxh.html | З повною інформацією |
| 15 | Свої козирі https://www.durbetsel.ru/2_svoi-koziri.htm | Карткові ігри |
| 16 | Війна з ботами https://www.durbetsel.ru/2_voina_s_botami.htm | Карткові ігри |
| 17 | Domineering 8x8 http://www.papg.com/show?1TX6 | З повною інформацією |
| 18 | Останній гравець https://www.durbetsel.ru/2_posledny_igrok.htm | Карткові ігри |
| 19 | Заєць и Вовки (за Вовків) http://www.iggamecenter.com/info/ru/foxh.html | З повною інформацією |

| | | |
|---|---|---|
| 20 | Богач https://www.durbetsel.ru/2_bogach.htm | Карткові ігри |
| 21 | Редуду https://www.durbetsel.ru/2_redudu.htm | Карткові ігри |
| 22 | Эльферн https://www.durbetsel.ru/2_elfern.htm | Карткові ігри |
| 23 | Ремінь https://www.durbetsel.ru/2_remen.htm | Карткові ігри |
| 24 | Реверсі https://ru.wikipedia.org/wiki/Реверси | З повною інформацією |
| 25 | Вари http://www.iggamecenter.com/info/ru/oware.html | З повною інформацією |
| 26 | Яцзи https://game-wiki.guru/published/igryi/yaczzyi.html | З елементами випадковості |
| 27 | Лудо http://www.iggamecenter.com/info/ru/ludo.html | З елементами випадковості |
| 28 | Генерал http://www.rules.net.ru/kost.php?id=7 | З елементами випадковості |
| 29 | Сим https://ru.wikipedia.org/wiki/Сим_(игра) | З повною інформацією |
| 30 | Col http://www.papg.com/show?2XLY | З повною інформацією |
| 31 | Snort http://www.papg.com/show?2XM1 | З повною інформацією |
| 32 | Chomp http://www.papg.com/show?3AEA | З повною інформацією |
| 33 | Gale http://www.papg.com/show?1TPI | З повною інформацією |
| 34 | 3D Noughts and Crosses 4 x 4 x 4 http://www.papg.com/show?1TND | З повною інформацією |
| 35 | Snakes http://www.papg.com/show?3AE4 | З повною інформацією |

# 3 ВИКОНАННЯ

## 3.1 Програмна реалізація алгоритму

```
Class HighScore

    Declare a File object named highScore
    Declare an ArrayList of Score objects named scores

    Constructor HighScore()
        If highScore file does not exist
            Try to create a new highScore file
        Load high scores from the file

    Method loadHighScore()
        Clear the scores list
        Try to open the highScore file for reading
            Read each line of the file
            For each line
                Split the line into name and score
                Create a new Score object with name and score
                Add the Score object to the scores list
            Close the file reader
        Sort the scores list in reverse order
        If the list has more than 5 scores
            Keep only the top 5 scores
        If the list has less than 5 scores
            Fill the remaining slots with "Nobody" and score 0
        Trim the highScore file to reflect the current top 5 scores

    Method submitHighScore(name, score)
```

Try to open the highScore file for appending

Write the new score entry to the file

Close the file writer

Method trimHighScore()

Delete the existing highScore file

Try to create a new highScore file

Write the top 5 scores to the new file

Close the file writer

Method getScores()

Return the scores list

End Class

### 3.1.1 Вихідний код

```
package com.example.lab_6;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import java.io.IOException;
import java.util.Objects;

public class App extends Application {
```

```
    private static Scene scene;

    @Override
    public void start(Stage stage) throws IOException {
        scene = new Scene(loadFXML("yatzyScene"));
        stage.setScene(scene);
        stage.setTitle("Yatzy");
        stage.getIcons().add(new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/6.
png"))));
        stage.setResizable(false);
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource("hello-
view.fxml"));
        return fxmlLoader.load();
    }

    public static void main(String[] args) {
        launch();
    }
}

package com.example.lab_6;
```

```
import java.io.*;
import java.util.ArrayList;
import java.util.Collections;

public class HighScore {

    private final File highScore = new File("highScore.dat");
    private final ArrayList<Score> scores = new ArrayList<>();

    public HighScore() {
        if (!highScore.exists()) {
            try {
                highScore.createNewFile();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        loadHighScore();
    }

    public void loadHighScore() {
        scores.clear();
        FileReader fileReader;
        BufferedReader bufferedReader;
        try {
            fileReader = new FileReader(highScore);
            bufferedReader = new BufferedReader(fileReader);
            String data = bufferedReader.readLine();
            while(data != null){
```

```
            String[] s = data.split(":");
            Score score = new Score(s[0], Integer.parseInt(s[1]));
            scores.add(score);
            data = bufferedReader.readLine();
        }
        bufferedReader.close();
        fileReader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    scores.sort(Collections.reverseOrder());
    if (scores.size() > 5)
        scores.subList(5, scores.size()).clear();
    if (scores.size() < 5) {
        int temp = scores.size();
        for (int i = 0; i < 5 - temp; i++) {
            Score score = new Score("Nobody",0);
            scores.add(score);
        }
    }
    trimHighScore();
}

public void submitHighScore(String name, int score) {
    FileWriter fileWriter;
    BufferedWriter bufferedWriter;
    try {
        fileWriter = new FileWriter(highScore, true);
        bufferedWriter = new BufferedWriter(fileWriter);
        bufferedWriter.write(name + ":" + score);
```

```
            bufferedWriter.newLine();
            bufferedWriter.close();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void trimHighScore() {
        FileWriter fileWriter;
        BufferedWriter bufferedWriter;
        highScore.delete();
        try {
            fileWriter = new FileWriter(highScore, true);
            bufferedWriter = new BufferedWriter(fileWriter);
            for (Score score : scores) {
                bufferedWriter.write(score.getName() + ":" + score.getScore());
                bufferedWriter.newLine();
            }
            bufferedWriter.close();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public ArrayList<Score> getScores() {
        return scores;
    }
```

```
}
package com.example.lab_6;

public class Main {

    public static void main(String[] args) {
        App.main(args);
    }
}
package com.example.lab_6;

public class Score implements Comparable<Score> {

    private final String name;
    private final int score;

    public Score(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getScore() {
        return score;
    }

    @Override
```

```
    public String toString() {
        if (score > 376)
            return name.substring(0, Math.min(name.length(),10)) + ": CHEAT";
        else
            return name.substring(0,Math.min(name.length(),10)) + ": " + score;
    }

    @Override
    public int compareTo(Score score) {
        return Integer.compare(this.score, score.score);
    }

}
package com.example.lab_6;

import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.fxml.FXML;
import javafx.scene.Cursor;
import javafx.scene.control.*;
import javafx.scene.effect.ColorAdjust;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Background;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
```

```
import java.util.Arrays;
import java.util.Objects;
import java.util.Optional;
import java.util.Random;

public class YatzyController {

   private final Image diceImageOne = new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/1.
png")));
   private final Image diceImageTwo = new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/2.
png")));
   private final Image diceImageThree = new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/3.
png")));
   private final Image diceImageFour = new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/4.
png")));
   private final Image diceImageFive = new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/5.
png")));
   private final Image diceImageSix = new
Image(Objects.requireNonNull(YatzyController.class.getResourceAsStream("dice/6.
png")));
   private final Image[] diceImages = {diceImageOne, diceImageTwo,
diceImageThree, diceImageFour, diceImageFive, diceImageSix};

   private final ImageView dice0 = new ImageView(diceImageOne),
         dice1 = new ImageView(diceImageTwo),
```

```
        dice2 = new ImageView(diceImageThree),
        dice3 = new ImageView(diceImageFour),
        dice4 = new ImageView(diceImageFive);
    private final ImageView[] diceImageViews = {dice0, dice1, dice2, dice3, dice4};

    private final Text scoreOnes = new Text("0"), scoreTwos = new Text("0"), scoreThrees = new Text("0"),
        scoreFours = new Text("0"), scoreFives = new Text("0"), scoreSixes = new Text("0"), scoreSumUpper = new Text("0"),
        scoreBonus = new Text("50"), scorePair = new Text("0"), scoreTwoPairs = new Text("0"), scoreThreeKind = new Text("0"),
        scoreFourKind = new Text("0"), scoreLowStraight = new Text("0"), scoreHighStraight = new Text("0"),
        scoreFullHouse = new Text("0"), scoreChance = new Text("0"), scoreYatzy = new Text("0"), scoreTotalSum = new Text("0"),
        highScoreOne = new Text(), highScoreTwo = new Text(), highScoreThree = new Text(), highScoreFour = new Text(), highScoreFive = new Text();
    private final Text[] scoreUpperText = {scoreOnes, scoreTwos, scoreThrees, scoreFours, scoreFives, scoreSixes};
    private final Text[] scoreLowerText = {scorePair, scoreTwoPairs, scoreThreeKind, scoreFourKind, scoreLowStraight, scoreHighStraight, scoreFullHouse, scoreChance, scoreYatzy};
    private final Text[] topScoresText = {highScoreOne, highScoreTwo, highScoreThree, highScoreFour, highScoreFive};

    private final Pane paneOnes = new Pane(), paneTwos = new Pane(), paneThrees = new Pane(), paneFours = new Pane(),
        paneFives = new Pane(), paneSixes = new Pane(), panePair = new Pane(), paneTwoPairs = new Pane(),
```

```
        paneThreeKind = new Pane(), paneFourKind = new Pane(), paneLowStraight = new Pane(), paneHighStraight = new Pane(),
        paneFullHouse = new Pane(), paneChance = new Pane(), paneYatzy = new Pane();
    private final Pane[] scorePanes = {paneOnes, paneTwos, paneThrees, paneFours, paneFives, paneSixes, panePair, paneTwoPairs, paneThreeKind, paneFourKind, paneLowStraight, paneHighStraight, paneFullHouse, paneChance, paneYatzy};

    @FXML
    private HBox diceBox;
    @FXML
    private GridPane scorePane, highScoreGrid;
    @FXML
    private Text rollCountLabel;
    @FXML
    public Button rollButton;
    @FXML
    private Button submitButton;


    private final boolean[] isClicked = new boolean[scoreUpperText.length + scoreLowerText.length];
    private final boolean[] isSubmitted = new boolean[scoreUpperText.length + scoreLowerText.length];
    private final int[] scoreUpper = new int[scoreUpperText.length];
    private final int[] scoreLower = new int[scoreLowerText.length];

    private final Die die1 = new Die(), die2 = new Die(), die3 = new Die(), die4 = new Die(), die5 = new Die();
    private final Die[] dice = {die1, die2, die3, die4, die5};
```

```
private int rollCount = 0, nPlayers = 1, sumUpper = 0, sumTotal = 0;
private boolean hasCheats = false;
private final HighScore highScore = new HighScore();

public void initialize() {
    for (int i = 0; i < dice.length; i++) {
        diceImageViews[i].setFitHeight(40.0);
        diceImageViews[i].setFitWidth(40.0);
        int finalI = i;
        diceImageViews[i].setOnMouseClicked(event -> lockDice(finalI));
    }
    diceBox.getChildren().addAll(diceImageViews);

    for (int i = 0; i < scoreUpperText.length; i++) {
        scorePanes[i].setMaxSize(28, 14);
        int finalI = i;
        scorePanes[i].setOnMouseClicked(event -> scoreClicked(finalI));
        scorePanes[i].setOnMouseEntered(event -> scoreHovered(finalI));
        scorePanes[i].setOnMouseExited(event -> scoreExited(finalI));
        scorePane.add(scorePanes[i], 1, i);
    }
    for (int i = 6; i < scoreLowerText.length + 6; i++) {
        scorePanes[i].setMaxSize(28, 14);
        int finalI = i;
        scorePanes[i].setOnMouseClicked(event -> scoreClicked(finalI));
        scorePanes[i].setOnMouseEntered(event -> scoreHovered(finalI));
        scorePanes[i].setOnMouseExited(event -> scoreExited(finalI));
        scorePane.add(scorePanes[i], 1, i + 2);
    }
```

```
    for (int i = 0; i < scoreUpperText.length; i++) {
        scoreUpperText[i].setOpacity(0.5);
        int finalI = i;
        scoreUpperText[i].setOnMouseClicked(event -> scoreClicked(finalI));
        scoreUpperText[i].setOnMouseEntered(event -> scoreHovered(finalI));
        scoreUpperText[i].setOnMouseExited(event -> scoreExited(finalI));
        scorePane.add(scoreUpperText[i], 1, i);
    }
    for (int i = 0; i < scoreLowerText.length; i++) {
        scoreLowerText[i].setOpacity(0.5);
        int finalI = i + 6;
        scoreLowerText[i].setOnMouseClicked(event -> scoreClicked(finalI));
        scoreLowerText[i].setOnMouseEntered(event -> scoreHovered(finalI));
        scoreLowerText[i].setOnMouseExited(event -> scoreExited(finalI));
        scorePane.add(scoreLowerText[i], 1, i + 8);
    }
    for (int i = 0; i < topScoresText.length; i++) {
        topScoresText[i].setText(highScore.getScores().get(i).toString());
        topScoresText[i].setWrappingWidth(80);
        highScoreGrid.add(topScoresText[i], 0, i + 1);
    }
    scoreBonus.setOpacity(0.2);
    scorePane.add(scoreSumUpper, 1, 6);
    scorePane.add(scoreBonus, 1, 7);
    scorePane.add(scoreTotalSum, 1, 17);

}

private void updateScorePanes() {
    if (rollCount == 0) {
```

```
            for (Pane pane : scorePanes)
                pane.setCursor(Cursor.DEFAULT);
        } else {
            for (int i = 0; i < isSubmitted.length; i++)
                if (!isSubmitted[i])
                    scorePanes[i].setCursor(Cursor.HAND);
        }
        for (int i = 0; i < scorePanes.length; i++) {
            if (!isClicked[i] && !isSubmitted[i]) {
                scorePanes[i].setBackground(Background.fill(null));
                scorePanes[i].setOpacity(1);
            }
            if (isClicked[i]) {
                if (calculatePoints(i) > 0 && validateSubmit(i))
                    scorePanes[i].setBackground(Background.fill(Color.LIGHTGREEN));
                else
                    scorePanes[i].setBackground(Background.fill(Color.INDIANRED));
            }
        }
    }

    private void scoreExited(int i) {
        if (!isClicked[i] && rollCount > 0 && !isSubmitted[i])
            scorePanes[i].setBackground(Background.fill(null));
    }

    private void scoreHovered(int i) {
        if (!isClicked[i] && rollCount > 0 && !isSubmitted[i])
```

```
scorePanes[i].setBackground(Background.fill(Color.LIGHTGOLDENRODYELLOW));
    }

    public void roll() {
        Arrays.stream(dice).filter(die -> !die.isLocked).forEach(Die::roll);
        rollCount++;
        if (rollCount >= 3)
            rollButton.setDisable(true);
        updateImage();
        updateText();
        updateScorePanes();
    }

    private void lockDice(int i) {
        if (rollCount > 0) {
            dice[i].setLocked();
            updateContrast(i);
        }
    }
    private void scoreClicked(int i) {
        if (validateClick(i)) {
            if (i <= 5) {
                scoreUpperText[i].setOpacity(1.0);
                if (validateSubmit(i))
                    scoreUpperText[i].setText(""+calculatePoints(i));
            } else {
                scoreLowerText[i - 6].setOpacity(1.0);
                if (validateSubmit(i))
```

```
                scoreLowerText[i - 6].setText(""+calculatePoints(i));
            }
            if (calculatePoints(i) > 0 && validateSubmit(i))
                scorePanes[i].setBackground(Background.fill(Color.LIGHTGREEN));
            else
                scorePanes[i].setBackground(Background.fill(Color.INDIANRED));
            scorePanes[i].setOpacity(0.5);
        } else if (!isSubmitted[i]) {
            if (i <= 5)
                scoreUpperText[i].setOpacity(0.5);
            else
                scoreLowerText[i - 6].setOpacity(0.5);
            scorePanes[i].setOpacity(1);
            scorePanes[i].setBackground(Background.fill(null));
        }
        updateText();
        updateScorePanes();
    }

    public void submit() {
        for (int i = 0; i < scoreUpperText.length + scoreLowerText.length; i++) {
            if (isClicked[i]) {
                int points = calculatePoints(i);
                if (i <= 5) {
                    scoreUpperText[i].setText("" + points);
                    scoreUpper[i] = points;
                    scoreUpperText[i].setCursor(Cursor.DEFAULT);
                } else if (validateSubmit(i)) {
                    scoreLowerText[i - 6].setText("" + points);
                    scoreLowerText[i - 6].setCursor(Cursor.DEFAULT);
```

```
            scoreLower[i - 6] = points;
        }
        isSubmitted[i] = true;
        reset();
    }
}
boolean isFinished = true;
for (boolean b : isSubmitted)
    if (!b) {
        isFinished = false;
        break;
    }
if (isFinished) {
    rollButton.setDisable(true);
    if (!hasCheats && sumTotal > highScore.getScores().get(4).getScore()) {
        TextInputDialog dialog = new TextInputDialog();
        dialog.setTitle("Game Finished");
        dialog.setHeaderText("You are in the top 5!");
        dialog.setContentText("Please enter your name:");
        Optional<String> result = dialog.showAndWait();
        if (result.isPresent()) {
            if (result.get().equals("")) {
                highScore.submitHighScore("Anon", sumTotal);
            } else {
                highScore.submitHighScore(result.get(), sumTotal);
            }
            highScore.loadHighScore();
            for (int i = 0; i < topScoresText.length; i++) {
                topScoresText[i].setText(highScore.getScores().get(i).toString());
            }
```

```
            }
        }
    }
}

public void reset() {
    rollCount = 0;
    for (int i = 0; i < dice.length; i++)
        if (dice[i].isLocked) {
            dice[i].setLocked();
            updateContrast(i);
        }
    Arrays.fill(isClicked, false);
    submitButton.setDisable(true);
    rollButton.setDisable(false);
    updateText();
    updateScorePanes();
}

public void restart() {
    Arrays.fill(isClicked, false);
    Arrays.fill(isSubmitted, false);
    Arrays.fill(scoreUpper, 0);
    Arrays.fill(scoreLower, 0);
    for (Text score : scoreUpperText) {
        score.setText("0");
        score.setOpacity(0.5);
        score.setCursor(Cursor.HAND);
    }
    for (Text score : scoreLowerText) {
```

```
            score.setText("0");
            score.setOpacity(0.5);
            score.setCursor(Cursor.HAND);
        }
        scoreSumUpper.setText("0");
        scoreBonus.setOpacity(0.2);
        scoreTotalSum.setText("0");
        submitButton.setDisable(true);
        rollButton.setDisable(false);
        rollCount = 0;
        for (int i = 0; i < dice.length; i++) {
            dice[i].faceValue = i + 1;
            dice[i].isLocked = false;
            updateContrast(i);
        }
        updateImage();
        updateText();
        updateScorePanes();

    }

    private boolean validateClick(int n) {
        boolean isValid = false;
        if (!isSubmitted[n] && rollCount != 0) {
            for (int i = 0; i < scoreUpperText.length; i++) {
                if (isClicked[i] && i != n) {
                    if (!isSubmitted[i])
                        scoreUpperText[i].setOpacity(0.5);
                    isClicked[i] = false;
                }
```

```
        }
        for (int i = 0; i < scoreLowerText.length; i++) {
            if (isClicked[i + 6] && i + 6 != n) {
                if (!isSubmitted[i + 6])
                    scoreLowerText[i].setOpacity(0.5);
                isClicked[i + 6] = false;
            }
        }
        if (!isClicked[n]) {
            isValid = true;
            submitButton.setDisable(false);
            isClicked[n] = true;
        } else {
            submitButton.setDisable(true);
            isClicked[n] = false;
        }
    } else {
        Arrays.fill(isClicked, false);
        for (int i = 0; i < scoreUpperText.length; i++) {
            if (!isSubmitted[i])
                scoreUpperText[i].setOpacity(0.5);
        }
        for (int i = 0; i < scoreLowerText.length; i++) {
            if (!isSubmitted[i + 6])
                scoreLowerText[i].setOpacity(0.5);
        }
        submitButton.setDisable(true);
    }
    return isValid;
}
```

```
private boolean validateSubmit(int n) {
    boolean isValid = true;
    switch (n) {
        case 6:
            boolean hasPair = false;
            for (int i = 0; i < dice.length - 1; i++)
                for (int j = i + 1; j < dice.length; j++)
                    if (dice[i].faceValue == dice[j].faceValue) {
                        hasPair = true;
                        break;
                    }
            isValid = hasPair;
            break;
        case 7:
            boolean hasTwoPairs = false;
            int numberOfPairs = 0;
            int[] twoPairsBucket = new int[6];
            for (Die die : dice)
                twoPairsBucket[die.faceValue - 1]++;
            for (int j : twoPairsBucket)
                if (j >= 2)
                    numberOfPairs += j / 2;
            if (numberOfPairs >= 2)
                hasTwoPairs = true;
            isValid = hasTwoPairs;
            break;
        case 8:
            boolean hasThreeKind = false;
            int[] threeKindBucket = new int[6];
```

```
    for (Die die : dice)
        threeKindBucket[die.faceValue - 1]++;
    for (int i : threeKindBucket)
        if (i >= 3) {
            hasThreeKind = true;
            break;
        }
    isValid = hasThreeKind;
    break;
case 9:
    boolean hasFourKind = false;
    int[] fourKindBucket = new int[6];
    for (Die die : dice)
        fourKindBucket[die.faceValue - 1]++;
    for (int i : fourKindBucket)
        if (i >= 4) {
            hasFourKind = true;
            break;
        }
    isValid = hasFourKind;
    break;
case 10:
    int[] lowStraight = {1, 2, 3, 4, 5};
    int[] tempLowStraight = new int[dice.length];
    for (int i = 0; i < tempLowStraight.length; i++)
        tempLowStraight[i] = dice[i].faceValue;
    Arrays.sort(tempLowStraight);
    isValid = Arrays.equals(lowStraight, tempLowStraight);
    break;
case 11:
```

```
            int[] highStraight = {2, 3, 4, 5, 6};
            int[] tempHighStraight = new int[dice.length];
            for (int i = 0; i < tempHighStraight.length; i++)
               tempHighStraight[i] = dice[i].faceValue;
            Arrays.sort(tempHighStraight);
            isValid = Arrays.equals(highStraight, tempHighStraight);
            break;
         case 12:
            int[] houseBucket = new int[6];
            for (Die die : dice)
               houseBucket[die.faceValue - 1]++;
            Arrays.sort(houseBucket);
            if (houseBucket[4] != 2 || houseBucket[5] != 3)
               isValid = false;
            break;
         case 14:
            for (int i = 1; i < dice.length; i++) {
               if (dice[0].faceValue != dice[i].faceValue) {
                  isValid = false;
                  break;
               }
            }
            break;
         default:
            break;
      }
      return isValid;
   }

   private int calculatePoints(int i) {
```

```
        int points = 0;
        if (i >= 0 && i <= 5) {
            for (Die die : dice)
                if (i + 1 == die.faceValue)
                    points += die.faceValue;
        } else if (i >= 6 && i <= 9) {
            switch (i) {
                case 6:
                    int pair = 0;
                    int[] pairsBucket = new int[6];
                    for (Die die : dice)
                        pairsBucket[die.faceValue - 1]++;
                    for (int j = 0; j < pairsBucket.length; j++)
                        if (pairsBucket[j] >= 2)
                            pair = (j + 1) * 2;
                    points = pair;
                    break;
                case 7:
                    int twoPairs = 0;
                    int[] twoPairsBucket = new int[6];
                    int[] twoPairsTemp = new int[dice.length];
                    for (int j = 0; j < dice.length; j++)
                        twoPairsTemp[j] = dice[j].faceValue;
                    for (int j = 0; j < dice.length - 1; j++) {
                        for (int k = j + 1; k < dice.length; k++) {
                            if (twoPairsTemp[j] == twoPairsTemp[k] && twoPairsTemp[j] !=
0) {
                                twoPairsBucket[twoPairsTemp[j] - 1]++;
                                twoPairsTemp[j] = 0;
                                twoPairsTemp[k] = 0;
```

```
            }
        }
    }
    for (int j = 0; j < twoPairsBucket.length; j++) {
        if (twoPairsBucket[j] >= 2) {
            twoPairs += ((j + 1) * 2) * 2;
            twoPairsBucket[j]--;
        } else if (twoPairsBucket[j] >= 1) {
            twoPairs += (j + 1) * 2;
        }
    }
    points = twoPairs;
    break;
case 8:
    int threeKind = 0;
    int[] threeKindBucket = new int[6];
    for (Die die : dice)
        threeKindBucket[die.faceValue - 1]++;
    for (int j = 0; j < threeKindBucket.length; j++)
        if (threeKindBucket[j] >= 3)
            threeKind = (j + 1) * 3;
    points = threeKind;
    break;
case 9:
    int fourKind = 0;
    int[] fourKindBucket = new int[6];
    for (Die die : dice)
        fourKindBucket[die.faceValue - 1]++;
    for (int j = 0; j < fourKindBucket.length; j++)
        if (fourKindBucket[j] >= 4)
```

```
                    fourKind = (j + 1) * 4;
                points = fourKind;
                break;
            default:
                break;
        }
    } else if (i == 14) {
        points = 50;
    } else {
        for (Die die : dice)
            points += die.faceValue;
    }
    return points;
}

private void updateImage() {
    for (int i = 0; i < dice.length; i++)
        diceImageViews[i].setImage(diceImages[dice[i].faceValue - 1]);
}

private void updateContrast(int i) {
    ColorAdjust colorAdjust = new ColorAdjust();
    if (dice[i].isLocked)
        colorAdjust.setContrast(-0.5);
    else
        colorAdjust.setContrast(0.0);
    diceImageViews[i].setEffect(colorAdjust);
}

private void updateText() {
```

```
rollCountLabel.setText(rollCount + "/3");
if (rollCount == 0) {
    for (Text score : scoreUpperText)
        score.setCursor(Cursor.DEFAULT);
    for (Text score : scoreLowerText)
        score.setCursor(Cursor.DEFAULT);
} else {
    for (int i = 0; i < isSubmitted.length; i++) {
        if (!isSubmitted[i] && i <= 5) {
            scoreUpperText[i].setCursor(Cursor.HAND);
        } else if (!isSubmitted[i]) {
            scoreLowerText[i - 6].setCursor(Cursor.HAND);
        }
        if (!isSubmitted[i] && !isClicked[i]) {
            if (i <= 5)
                scoreUpperText[i].setText("0");
            else
                scoreLowerText[i - 6].setText("0");
        }
        if (isClicked[i] && validateSubmit(i)) {
            if (i <= 5)
                scoreUpperText[i].setText(""+calculatePoints(i));
            else
                scoreLowerText[i - 6].setText(""+calculatePoints(i));
        }
    }
}
calculateSum();
scoreSumUpper.setText("" + sumUpper);
scoreTotalSum.setText("" + sumTotal);
```

```
}

public void calculateSum() {
    sumUpper = 0;
    for (int i : scoreUpper)
        sumUpper += i;
    sumTotal = sumUpper;
    if (sumUpper >= 63) {
        scoreBonus.setOpacity(1.0);
        sumTotal += 50;
    }
    for (int i : scoreLower)
        sumTotal += i;
}


private static class Die {

    private static final Random rand = new Random();
    private int faceValue;
    private boolean isLocked = false;

    private void setLocked() {
        isLocked = !isLocked;
    }

    private void roll() {
        faceValue = rand.nextInt(6) + 1;
    }
```

```
    }

}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.Cursor?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="420.0" prefWidth="420.0" xmlns="http://javafx.com/javafx/17" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.lab_6.YatzyController">
   <HBox alignment="CENTER" layoutX="260.0" layoutY="45.0" spacing="10.0">
      <Button fx:id="submitButton" disable="true" mnemonicParsing="false" onAction="#submit" text="Submit" />
      <Button fx:id="rollButton" mnemonicParsing="false" onAction="#roll" text="Roll">
         <cursor>
```

```
            <Cursor fx:constant="HAND" />
         </cursor>
      </Button>
      <Text fx:id="rollCountLabel" strokeType="OUTSIDE" strokeWidth="0.0" text="0/3" textAlignment="CENTER">
         <font>
            <Font size="24.0" />
         </font>
      </Text>
   </HBox>
   <HBox fx:id="diceBox" alignment="CENTER" layoutX="217.0" layoutY="5.0">
   </HBox>
   <GridPane fx:id="scorePane" gridLinesVisible="true" layoutX="14.0" layoutY="14.0">
      <columnConstraints>
         <ColumnConstraints hgrow="SOMETIMES" prefWidth="100.0" />
         <ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" prefWidth="30.0" />
      </columnConstraints>
      <rowConstraints>
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints vgrow="SOMETIMES" />
```

```
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
        </rowConstraints>
        <opaqueInsets>
            <Insets />
        </opaqueInsets>
        <Pane style="-fx-background-color: #dbdbdb;" GridPane.rowIndex="6" />
        <Pane layoutX="10.0" layoutY="110.0" style="-fx-background-color:
#dbdbdb;" GridPane.columnIndex="1" GridPane.rowIndex="6" />
        <Pane layoutX="10.0" layoutY="106.0" style="-fx-background-color:
#dbdbdb;" GridPane.rowIndex="7" />
        <Pane layoutX="10.0" layoutY="129.0" style="-fx-background-color:
#dbdbdb;" GridPane.columnIndex="1" GridPane.rowIndex="7" />
        <Pane layoutX="10.0" layoutY="110.0" style="-fx-background-color:
#dbdbdb;" GridPane.rowIndex="17" />
        <Pane layoutX="10.0" layoutY="299.0" style="-fx-background-color:
#dbdbdb;" GridPane.columnIndex="1" GridPane.rowIndex="17" />
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Ones:"
GridPane.halignment="LEFT">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
```

```
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Twos:"
GridPane.rowIndex="1">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Threes:"
GridPane.rowIndex="2">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Fours:"
GridPane.rowIndex="3">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Fives:"
GridPane.rowIndex="4">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Sixes:"
GridPane.rowIndex="5">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
```

```
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Sum:" GridPane.rowIndex="6">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Bonus:" GridPane.rowIndex="7">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="A Pair:" GridPane.rowIndex="8">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Two Pairs:" GridPane.rowIndex="9">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Three of a kind:" GridPane.rowIndex="10">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
```

```
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Four of a kind:" GridPane.rowIndex="11">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Low Straight:" GridPane.rowIndex="12">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="High Straight:" GridPane.rowIndex="13">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Full House:" GridPane.rowIndex="14">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Chance:" GridPane.rowIndex="15">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
```

```
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Yatzy:" GridPane.rowIndex="16">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Total Sum:" GridPane.rowIndex="17">
            <GridPane.margin>
                <Insets left="5.0" />
            </GridPane.margin>
        </Text>
    </GridPane>
    <Button layoutX="318.0" layoutY="381.0" mnemonicParsing="false" onAction="#restart" text="Restart Game" />
    <HBox alignment="CENTER" layoutX="312.0" layoutY="176.0">


    </HBox>
    <Text layoutX="14.0" layoutY="395.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Yatzy Game">
        <font>
            <Font size="36.0" />
        </font>
    </Text>
    <GridPane fx:id="highScoreGrid" layoutX="230.0" layoutY="295.0">
        <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" />
        </columnConstraints>
        <rowConstraints>
```

```
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
        </rowConstraints>
        <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Top 5">
            <font>
                <Font size="24.0" />
            </font>
        </Text>
    </GridPane>
</AnchorPane>


import static org.junit.Assert.*;

import com.example.lab_6.HighScore;
import com.example.lab_6.Score;
import org.junit.*;
import java.util.ArrayList;

public class Test {

    private HighScore highScore;

    @Before
    public void setUp() {
        highScore = new HighScore();
```

```
}

@org.junit.Test
public void testInitializationWithExistingFile() {
    highScore = new HighScore();
    ArrayList<Score> scores = highScore.getScores();
    assertEquals("Expected number of scores", 5, scores.size());
}

@org.junit.Test
public void testLoadHighScoreWithCorruptedData() {
    highScore = new HighScore();
    ArrayList<Score> scores = highScore.getScores();
}

@org.junit.Test
public void testScoresLimit() {
    for (int i = 1; i <= 10; i++) {
        highScore.submitHighScore("User" + i, i * 10);
    }

    ArrayList<Score> scores = highScore.getScores();

    assertEquals("Scores list should only contain top 5 scores", 5, scores.size());
}

@org.junit.Test
public void testDataPersistence() {
    highScore.submitHighScore("PersistentUser", 200);
```

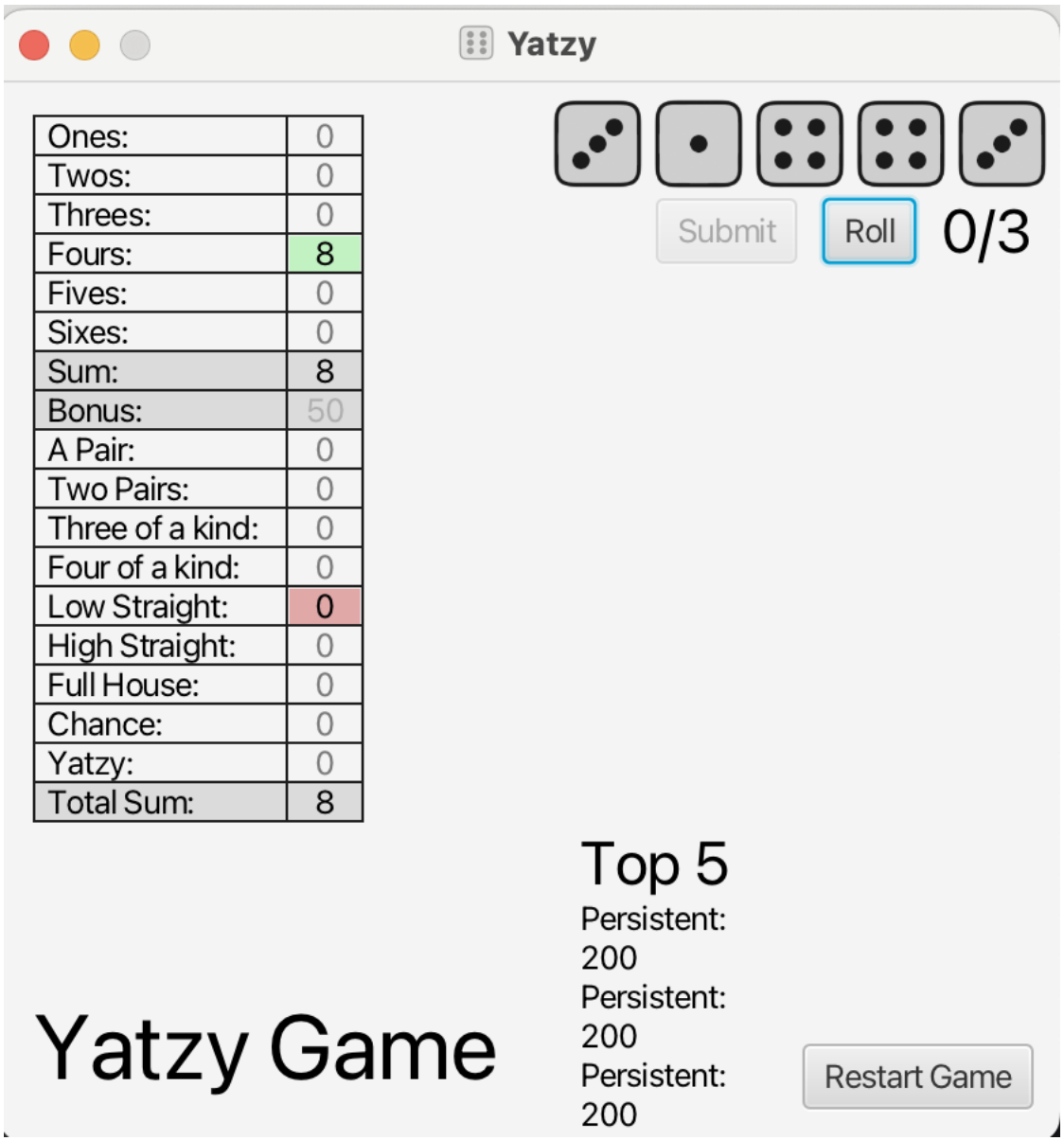
```
        HighScore newHighScore = new HighScore();
        ArrayList<Score> scores = newHighScore.getScores();

        assertEquals("PersistentUser", scores.get(0).getName());
        assertEquals(200, scores.get(0).getScore());
    }
}
```

### 3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 – **random rolling of 5 dices**

Рисунок 3.2 – **Combinations and scores**



# ВИСНОВОК

Підводячи підсумок вивчення теми „Пошук в умовах протидії, ігри з повною і неповною інформацією та ігри з елементом випадковості”, я можу сказати, що це був вражаючий досвід, який допоміг мені зрозуміти глибину та складність стратегічного мислення. Ці концепції є ключовими в галузі теорії ігор та штучного інтелекту, оскільки вони відображають реальні ситуації, в яких учасники приймають рішення, керуючись різними ступенями інформації та невизначеності.

# КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 31.12.2023 включно максимальний бал дорівнює – 5. Після 31.12.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація – 75%;
- робота з гіт – 20%;
- висновок – 5%.

+1 додатковий бал можна отримати за реалізацію анімації ігрових процесів (жеребкування, роздачі карт, анімацію ходів тощо).

+1 додатковий бал можна отримати за виконання та захист роботи до 24.12.2023.