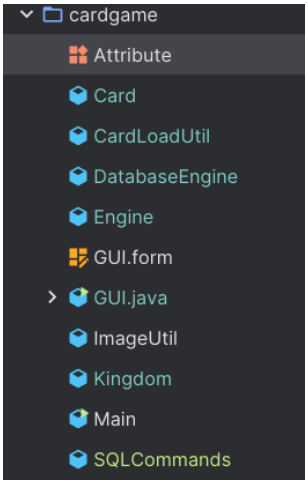
Card game – dokumentacja

Justyna Puz 263469

Link do github: https://github.com/JustynaPuz/.Net-and-Java.git

Projekt to gra karciana, która symuluje zarządzenie królestwem. Poprzez akceptacje lub odrzucenie karty wpływamy na parametry królestwa takie jak jedzenie, wojsko, ekonomia i religia. Gra jest utrzymana w tematyce jamników, wszystkie karty nawiązują do nich.

Struktura programu

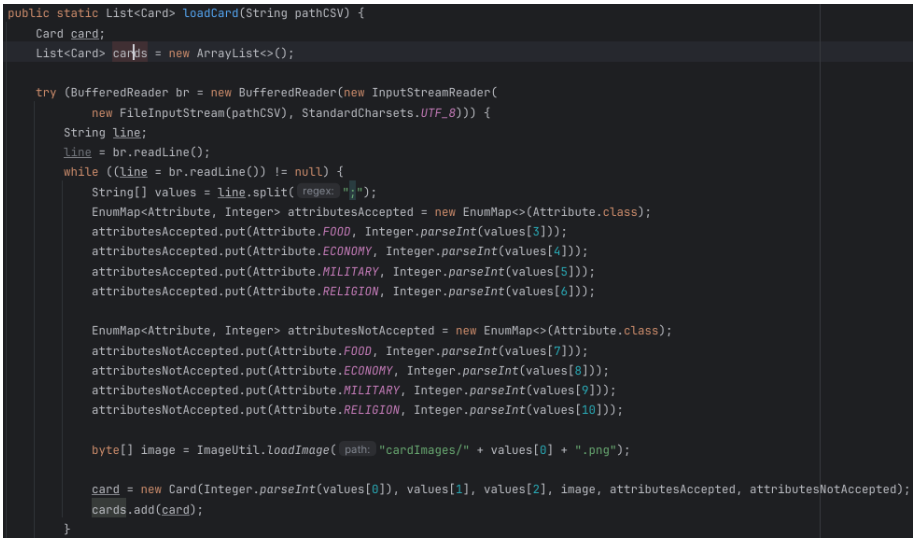


Klasa Attribute to klasa enum, w której znajdują się parametry

Klasa Card to klasa odpowiadająca instancji jednej karty. Oprócz pól opisujących instancje takich jak id, tytuł, opis, zdjęcie posiada EnumMapy w których zapisane są wartości po zaakceptowani karty i po odrzuceniu. Metoda generateRandomCardImpact służy do wprowadzenia większej losowości wpływu karty. Wartości są losowane z przedziału -5 wartość początkowa do +5 wartość początkowa.

```
public class Card {

    private final int id;
    private final String title;
    private final String description;
    private final byte[] image;
    private final EnumMap<Attribute, Integer> attributesAccepted;
    private final EnumMap<Attribute, Integer> attributesNotAccepted;

    4 usages  Justyna Puz *
    public EnumMap<Attribute, Integer> generateRandomCardImpact(Boolean isAccepted) {
        Random random = new Random();
        int value;
        EnumMap<Attribute, Integer> attributes =
                isAccepted ? attributesAccepted : attributesNotAccepted;
        EnumMap<Attribute, Integer> newAttributes = new EnumMap<>(Attribute.class);

        for (Attribute attribute : attributes.keySet()) {
            if (attributes.get(attribute) != 0) {
                int min = attributes.get(attribute) - 6;
                int max = attributes.get(attribute) + 7;
                value = random.nextInt( bound: max - min) + min;

                value = Math.max(-100, Math.min(100, value));
            } else {
                value = 0;
                newAttributes.put(attribute, value);
                continue;
            }
```

Klasa CardLoadUtil to klasa która wczytuje dane o karcie z pliku CSV. Posiada jedną metodę loadCard (ładuje karty do listy kart)

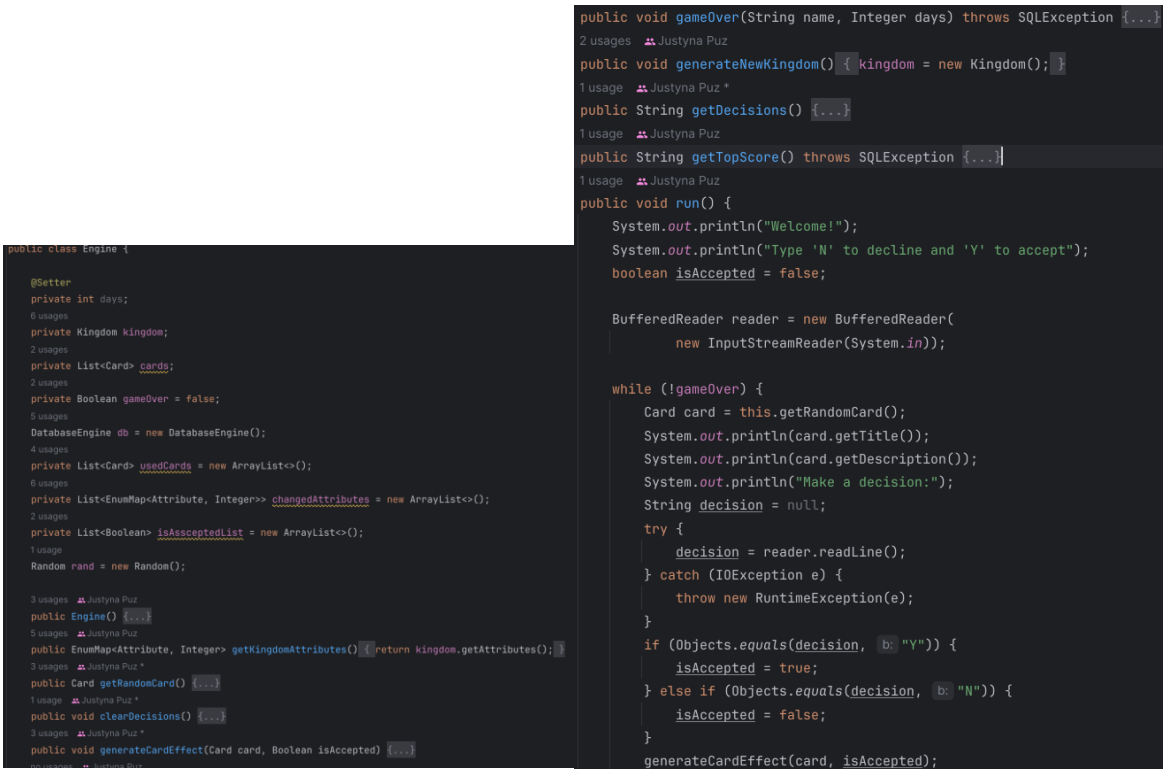


```
public static List<Card> loadCard(String pathCSV) {
    Card card;
    List<Card> cards = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new InputStreamReader(
            new FileInputStream(pathCSV), StandardCharsets.UTF_8))) {
        String line;
        line = br.readLine();
        while ((line = br.readLine()) != null) {
            String[] values = line.split( regex: ";");
            EnumMap<Attribute, Integer> attributesAccepted = new EnumMap<>(Attribute.class);
            attributesAccepted.put(Attribute.FOOD, Integer.parseInt(values[3]));
            attributesAccepted.put(Attribute.ECONOMY, Integer.parseInt(values[4]));
            attributesAccepted.put(Attribute.MILITARY, Integer.parseInt(values[5]));
            attributesAccepted.put(Attribute.RELIGION, Integer.parseInt(values[6]));

            EnumMap<Attribute, Integer> attributesNotAccepted = new EnumMap<>(Attribute.class);
            attributesNotAccepted.put(Attribute.FOOD, Integer.parseInt(values[7]));
            attributesNotAccepted.put(Attribute.ECONOMY, Integer.parseInt(values[8]));
            attributesNotAccepted.put(Attribute.MILITARY, Integer.parseInt(values[9]));
            attributesNotAccepted.put(Attribute.RELIGION, Integer.parseInt(values[10]));

            byte[] image = ImageUtil.loadImage( path: "cardImages/" + values[0] + ".png");

            card = new Card(Integer.parseInt(values[0]), values[1], values[2], image, attributesAccepted, attributesNotAccepted);
            cards.add(card);
        }
```

Klasa DatabaseEngine to klasa odpowiadająca za zarządzanie bazą danych h2. Posiada metody które wpisują do bazy danych dane o karcie, o najlepszym wyniku lub wyciąga z bazy odpowiednie informacje. DatabaseEngine odpowiada za połączenie z bazą danych.

```
public class DatabaseEngine {

    1 usage
    static final String JDBC_DRIVER = "org.h2.Driver";
    1 usage
    static final String DB_URL = "jdbc:h2:tcp://localhost/~/test";
    //  Database credentials
    1 usage
    static final String USER = "sa";
    1 usage
    static final String PASS = "";
    10 usages
    Connection conn = null;
    4 usages
    Statement stmt = null;

    2 usages  Justyna Puz *
    public DatabaseEngine() {...}
    no usages  Justyna Puz
    public void closeDatabase() throws SQLException {...}
    2 usages  Justyna Puz *
    public void insertCards(List<Card> cards) {...}
    1 usage  Justyna Puz *
    public void insertTopScore(String nameStr, Integer days) throws SQLException {...}
    2 usages  Justyna Puz *
    public Card getCardById(int cardId) {...}
    2 usages  Justyna Puz *
    public int getNumberOfCards() {...}
    1 usage  Justyna Puz *
    public Map<String, Integer> getTopScore() throws SQLException {...}

}
```

Klasa Engine to serce całej gry. To tutaj aktualizowane są parametry królestwa, pobierane są karty z bazy danych, tworzona jest instancja królestwa oraz wyświetlane są wyniki najlepszych graczy. Metoda run przedstawia terminalową wersje gry. Gra się kończy jeśli jeden z parametrów będzie równy 0 lub mniej.



```
public class Engine {

    @Setter
    private int days;
    6 usages
    private Kingdom kingdom;
    2 usages
    private List<Card> cards;
    2 usages
    private Boolean gameOver = false;
    5 usages
    DatabaseEngine db = new DatabaseEngine();
    4 usages
    private List<Card> usedCards = new ArrayList<>();
    6 usages
    private List<EnumMap<Attribute, Integer>> changedAttributes = new ArrayList<>();
    2 usages
    private List<Boolean> isAssceptedList = new ArrayList<>();
    1 usage
    Random rand = new Random();

    3 usages  Justyna Puz
    public Engine() {...}
    5 usages  Justyna Puz
    public EnumMap<Attribute, Integer> getKingdomAttributes() { return kingdom.getAttributes(); }
    3 usages  Justyna Puz *
    public Card getRandomCard() {...}
    1 usage  Justyna Puz *
    public void clearDecisions() {...}
    3 usages  Justyna Puz *
    public void generateCardEffect(Card card, Boolean isAccepted) {...}
```

```
public void gameOver(String name, Integer days) throws SQLException {...}
2 usages  Justyna Puz
public void generateNewKingdom() { kingdom = new Kingdom(); }
1 usage  Justyna Puz *
public String getDecisions() {...}
1 usage  Justyna Puz
public String getTopScore() throws SQLException {...}
1 usage  Justyna Puz
public void run() {
    System.out.println("Welcome!");
    System.out.println("Type 'N' to decline and 'Y' to accept");
    boolean isAccepted = false;

    BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));

    while (!gameOver) {
        Card card = this.getRandomCard();
        System.out.println(card.getTitle());
        System.out.println(card.getDescription());
        System.out.println("Make a decision:");
        String decision = null;
        try {
            decision = reader.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        if (Objects.equals(decision, b: "Y")) {
            isAccepted = true;
        } else if (Objects.equals(decision, b: "N")) {
            isAccepted = false;
        }
        generateCardEffect(card, isAccepted);
```

Klasa ImageUtil odpowiad za odpowiednie wczytanie obrazka z pliku i stworzenie przeskalowanego odpowiednika. Obrz zapisywany jest w postaci byte[]

```
public class ImageUtil {
    1 usage  Justyna Puz
    public static byte[] loadImage(String path) throws IOException {
        try (InputStream is = ImageUtil.class.getClassLoader().getResourceAsStream(path)) {
            if (is == null) {
                throw new IOException("File not found: " + path);
            }
            BufferedImage bImage = ImageIO.read(is);
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            ImageIO.write(bImage, formatName: "png", bos);
            return bos.toByteArray();
        }
    }
    2 usages  Justyna Puz
    public static JLabel createScaledImage(byte[] imageData, int width, int height) throws IOException
        ByteArrayInputStream bais = new ByteArrayInputStream(imageData);
        BufferedImage bImage = ImageIO.read(bais);
        ImageIcon imageIcon = new ImageIcon(bImage.getScaledInstance(width, height, Image.SCALE_SMOOTH))
        return new JLabel(imageIcon);
    }
}
```

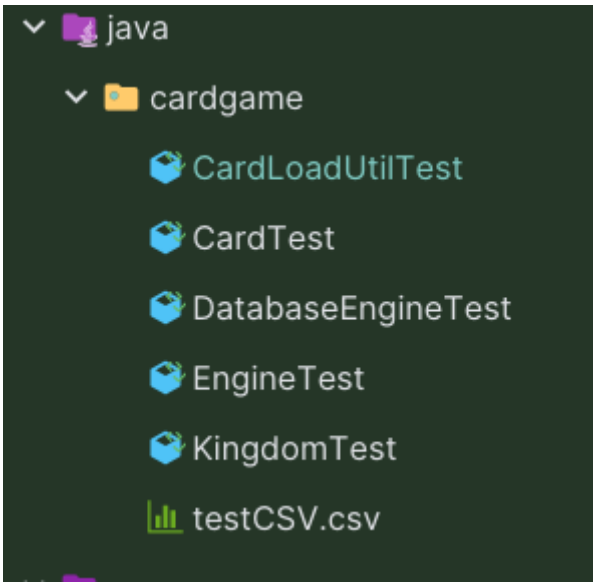
Klasa Kingdom to klasa odpowiadająca za stworzenie instancji królestwa z wygenerowanymi wartościami dla atrybutów.

Klasa SQLCommands zawiera zapytania do bazy danych h2

```java
@Getter
public class Kingdom {
    @Setter
    private EnumMap<Attribute, Integer> attributes = new EnumMap<>(Attribute.class);

    3 usages  Justyna Puz
    public Kingdom() {
        Random rand = new Random();
        for (Attribute attribute : Attribute.values()) {
            int randomValue = rand.nextInt( origin: 40,  bound: 91);
            attributes.put(attribute, randomValue);
        }
    }

    Justyna Puz *
    @Override
    public String toString() {
        StringBuilder output = new StringBuilder();
        for (Attribute atr : attributes.keySet()) {
            output.append(atr.toString()).append(" ").append(attributes.get(atr)).append("\n");
        }
        return output.toString();
    }
```

```java
public class SQLCommands {
    1 usage
    public static final String createTableCard = "CREATE TABLE  IF NOT EXISTS  CARD " +
            "(id INTEGER not NULL, " +
            " title VARCHAR(255), " +
            " description TEXT, " +
            "image BLOB," +
            " foodYes INTEGER, " +
            " economyYes INTEGER, " +
            " militaryYes INTEGER, " +
            " religionYes INTEGER, " +
            " foodNo INTEGER, " +
            " economyNo INTEGER, " +
            " militaryNo INTEGER, " +
            " religionNo INTEGER, " +
            " PRIMARY KEY ( id ))";
    1 usage
    public static final String createTableTopScore = "CREATE TABLE IF NOT EXISTS TOPSCORE" +
            "(name VARCHAR(50)," +
            "score INTEGER)";
    1 usage
    public static final String insertCard = "INSERT INTO CARD (id, title, description, image, foodYes, economyYe
    1 usage
    public static final String getTopSoreByName = "SELECT score FROM TOPSCORE WHERE name = ?";
    1 usage
    public static final String updateTopScore = "UPDATE TOPSCORE SET score = ? WHERE name = ?";
    1 usage
    public static final String insertTopScore = "INSERT INTO TOPSCORE (name, score) VALUES (?, ?)";
    1 usage
    public static final String getCardById = "SELECT * FROM CARD WHERE id = ?";
    1 usage
    public static final String getNumberOfCards = "SELECT MAX(id) AS max_id FROM CARD";
    1 usage
    public static final String getTopScores  = "SELECT name, score FROM TOPSCORE ORDER BY score DESC";
}
```

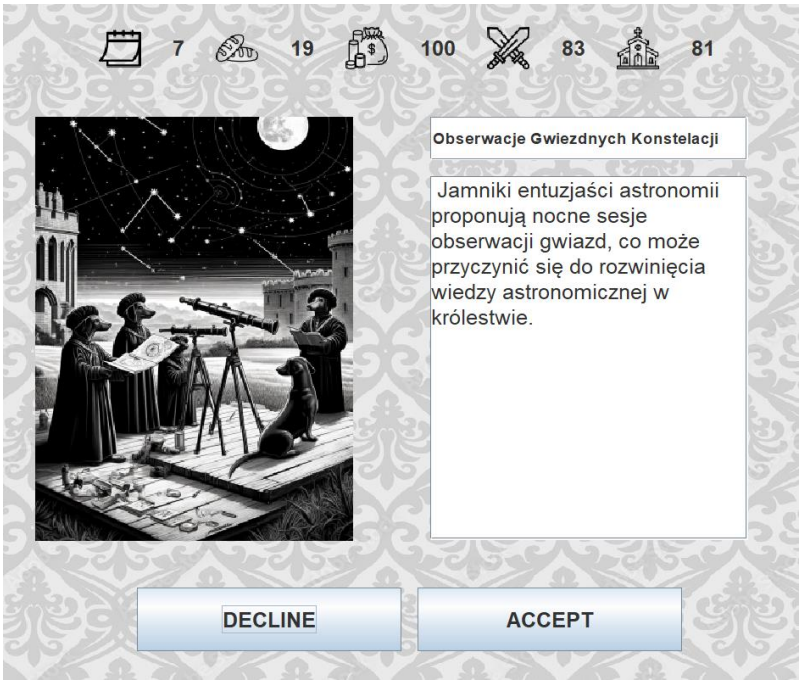
Dla każdej klasy są stworzone również testy jednostkowe sprawdzające poprawność działania kodu.



GUI

Ekran startowy (można wybrać rozpoczęcie gry albo zobaczenie najlepszych wyników

Ekran gry (tutaj pojawiają się nowe karty, które możemy odrzucić albo przyjąć. Automatycznie aktualizowane są parametry królestwa i ilość dni władania królestwem (wynik).



Ekran końca gry (można tutaj dodać swój nik i zapisać go w bazie danych z wynikami graczy, można zobaczyć podejmowane decyzje w trakcie trwania rozgrywki oraz przejść do ekranu starowego.