Temat: Gra w Wisielca

Dane projektantów

Konrad Krupski *310729*

Julia Polak *310965*

Opis projektu

Tematem projektu jest gra w Wisielca. Polega ona na tym, że zadaniem użytkownika jest zgadnięcie hasła, które jest losowe. Podczas gry użytkownik ma możliwość zgadywania poszczególnych liter, które zagadkowe hasło może zawierać. Wybieranie liter odbywa się poprzez naciskanie odpowiednich przycisków wyświetlanych na interfejsie graficznym.

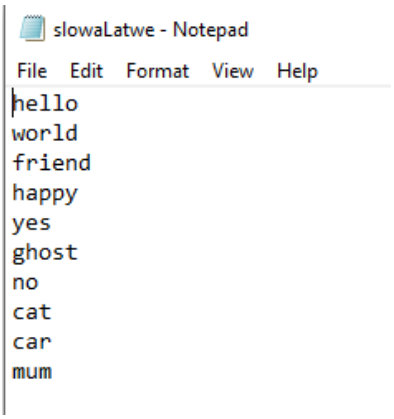
Co więcej, użytkownik ma możliwość zgadywania całego hasła, jeżeli jest mu ono znane. Należy wspomnieć, iż każde niepowodzenie, które może przyjąć formę kliknięcia nieodpowiedniej litery lub zgadnięcia złego słowa jest karane poprzez zredukowanie punktów życia o jeden. Jeżeli punkty życia użytkownika spadną do zera to następuje zakończenie rozgrywki, poinformowanie użytkownika o treści zagadkowego hasła oraz wyłączenie się programu.

Użytkownik ma możliwość rozpoczęcia rozgrywki od początku poprzez kliknięcie przycisku RESTART, który powoduje odnowienie punktów życia oraz wylosowanie nowego hasła z bazy danych.

Rozgrywka może odbywać się na kilku poziomach trudności: łatwym, średnim oraz trudnym. Każdy z tych poziomów trudności charakteryzuje się słowami, które mogą zostać wylosowane z bazy danych.

- słowa na poziome łatwym są długości do maksymalnie 5 liter
- słowa na poziome średnim są długości od 6 do 11 liter
- słowa na poziomie trudnym są długości od 12 do 15 liter.

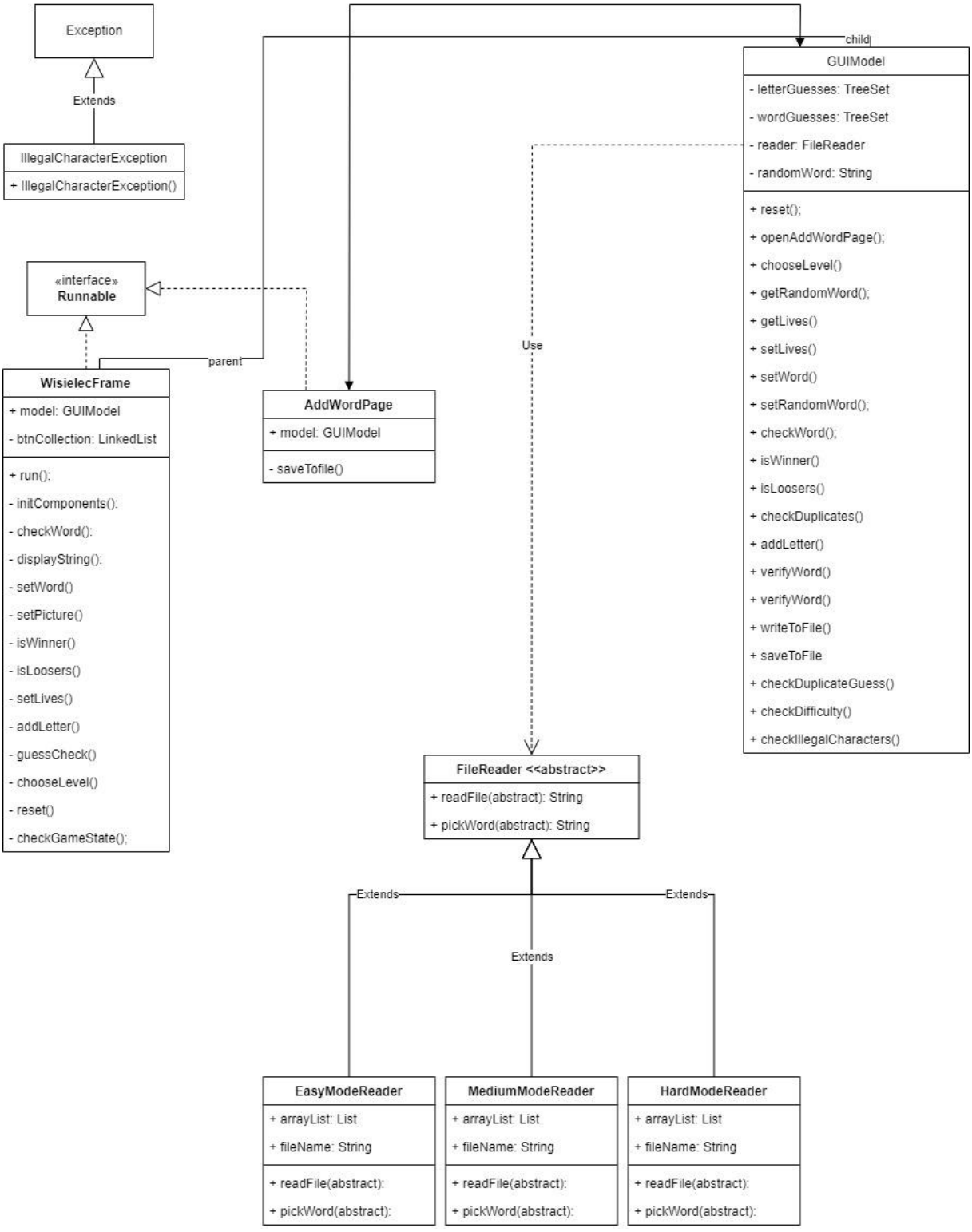
Baza danych została zrealizowana w postaci plików tekstowych, które zawierają poszczególne słowa zapisane w odpowiedni sposób, czyli jedno słowo na jedną linijkę.



Rysunek 1 Baza danych z słowami

Dodatkowo użytkownik ma możliwość dodawania nowych słów do bazy danych poprzez odpowiednią funkcję interfejsu graficznego. W celu dodania nowego słowa należy kliknąć przycisk ADD A WORD, który uruchamia nowe okno pozwalające na zrealizowanie tej operacji. Słowa proponowane przez użytkownika są poddawane odpowiedniej weryfikacji, która zostanie opisana w dalszej części dokumentacji.

Diagram Klas



Opis przeznaczenia poszczególnych klas

Klasa Main

Klasa jest odpowiedzialna za uruchomienie programu. W tej klasie zrealizowane zostało utworzenie nowego wątku zajmującego się obsługa interfejsu graficznego oraz uruchomienie go. Klasa Main posiada metodę typu static, dlatego nie wymaga ona tworzenia nowego obiektu tej klasy.

```
1  public class Main {
2      public static void main(String[] args) {
3          Thread thread1 = new Thread(new WisielecFrame());
4          thread1.start();
5      }
```

Rysunek 2 Klasa Main realizacja

Klasa WisielecFrame

Ogólne przeznaczenie

Klasa odpowiada między innymi za wyświetlenie interfejsu graficznego. Jest ona głównym oknem całej aplikacji, w którym prowadzona jest rozgrywka. Klasa odpowiada za odpowiednie rozmieszczenie elementów GUI takich jak: JButton, JPanel, JTextField. Co więcej, klasa realizuje odpowiednie metody potrzebne do poprawnego przeprowadzenia rozgrywki, które są wykonywane przy użyciu klasy GUIModel. Należy wspomnieć, iż w klasie WisielecFrame dodatkowo są przypisywani słuchacze do każdego elementu, który wymagał takiej funkcjonalności.

Należy zauważyć, iż WisielecFrame() implementuje interfejs Runnable, gdyż obiekt tej klasy będzie uruchamiany w osobnym wątku. Oczywiście narzuca to implementację metody run(), która zostanie później opisana.

```
public class WisielecFrame extends javax.swing.JFrame implements Runnable {
```

Rysunek 3 Dziedziczenie i interefej klasy

Szczegóły implementacji

Konstruktor WisielecFrame()

```
public WisielecFrame() {
    this.model = new GUIModel();
    chooseLevel();
}
```

Rysunek 4 Konstruktor WisielecFrame()

Zadaniem konstruktora klasy WisielecFrame, jest utworzenie odpowiedniego powiązania (asocjacji) z klasą modelu poprzez przypisanie jego nowego obiektu do pola **model**. Dodatkowo uruchamiana jest funkcja chooseLevel(), która pozwala użytkownikowi na wybór poziomu trudności. Szczegóły implementacji tej metody będą zawarte w dalszej części dokumentacji.

run()

```
public void run() {
    initComponents();
    setVisible(true);
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setBackground(new java.awt.Color( r: 30,  g: 30,  b: 30));
    setResizable(false);
}
```

Rysunek 5 Metoda run() klasy WisielecFrame()

Zadaniem tej metody jest inicjalizacja komponentów GUI przy użyciu metody initComponents() oraz zapewnienie widoczności głównego okna aplikacji. Następnie ustawiane są odpowiednie parametry okna takie jak, domyśla operacja podczas jego zamknięcia, kolor tła oraz blokada możliwości zmiany rozmiarów okna.

Należy wspomnieć, że metoda run() uruchamia się podczas startu wątku odpowiedzialnego za pracę WisielecFrame().

initComponents()

Zadaniem tej metody, jest inicjalizacja wszystkich komponentów GUI klasy WisielecFrame. Podczas tworzenia interfejsu graficznego zdecydowaliśmy się użyć zewnętrznego oprogramowania Apache NetBeans IDE 13. Komponenty posiadają absolutnego zarządcę rozkładu, oznacza to, że ich rozmieszenie jest definiowane poprzez parametry X oraz Y.

Co więcej, zadaniem tej metody jest również przypisanie odpowiednich słuchaczy do komponentów, którzy zapewniają interaktywność interfejsu graficznego.

checkWord()

```
private String checkWord() {
    return model.checkWord();
}
```

Rysunek 6 Metoda checkWord()

Zadaniem tej metody jest uruchomienie odpowiedniej metody modelu, która odpowiada za zwrócenie ciągu znaków, który ma zostać wyświetlony na ekranie. Szczegóły implementacji ten metody zostaną zawarte w dalszej części dokumentacji.

displayString()

```
private void displayString(String word) {
    textField.setText(word);
}
```

Rysunek 7 Metoda displayString()

Zadaniem tej metody jest wyświetlenie na ekranie słowa podanego jako argument. Wyświetlenie jest realizowanie poprzez zmianę zawartości tekstu w odpowiednim komponencie interfejsu graficznego.

isWinner()

```
private boolean isWinner() {
    return model.isWinner();
}
```

Rysunek 8 Metoda isWinner()

Zadanie tej metody jest uruchomienie odpowiedniej metody modelu i zwrócenie wartości true jeżeli rozgrywka zakończyła się wygraną albo wartości false jeżeli wygrana jeszcze nie nastąpiła.

isLooser()

```
private boolean isLooser() {
    return model.isLooser();
}
```

Rysunek 9 Metoda isLooser()

Zadanie tej metody jest uruchomienie odpowiedniej metody modelu i zwrócenie wartości true jeżeli punktu życia użytkownika są równe zero albo wartości false jeżeli punktu życia użytkownika są większe niż zero.

setWord()

```
private String setWord() {
    return model.setWord();
}
```

Rysunek 10 Metoda setWord()

Zadanie tej metody jest uruchomienie odpowiedniej metody modelu. Metoda jest uruchamiana na samym początku działania programu w celu ustawienia odpowiedniej wartości wyświetlanej na ekranie. W zależności od długości słowa zwracana jest inna ilość „podłóg”.

Na przykład dla słowa „cat” zostały zwrócone trzy znaki „_”, które później przy użyciu metody dispalyString() zostały wyświetlone na ekranie.

setPicture()

```
private void setPicture(JLabel label) {
    try{
        BufferedImage newPic = ImageIO.read(new File( pathname: model.getLives() + ".png"));
        label.setIcon(new ImageIcon(newPic));
    } catch (IOException e) {
        JOptionPane.showMessageDialog(mainPanel,
                message: "Could not find a picture",
                title: "ERROR",
                JOptionPane.ERROR_MESSAGE);
    }

}
```

Rysunek 11 Metoda setPicture()

Zadaniem tej metody jest odczytanie odpowiedniego pliku graficznego oraz wyświetlenie go na interfejsie graficznym. Argumentem tej metody jest obiekt JLabel, w którym dany obraz ma zostać wyświetlony. Jeżeli odczyt pliku zakończy się niepowodzeniem, to użytkownik jest informowany poprzez okno dialogowe z wyjaśnieniem błędu.

setLives()

```
private void setLives(char c) {
    model.setLives(c);
}
```

Rysunek 12 Metoda setLives()

Zdaniem tej metody jest ustawienie punktów życia, przy użycie settera, na liczbę podaną jako argument. Ilość punktów życia jest przechowywana w modelu GUIModel.

addLetter()

```
private void addLetter(Character letter) {
    model.addLetter(letter);
}
```

Rysunek 13 Metoda addLetter()

Zadaniem tej metody jest dodanie litery, zgadniętej przez użytkownika i podanej jako argument, do kolekcji przechowywanej w modelu.

disposeWindow()

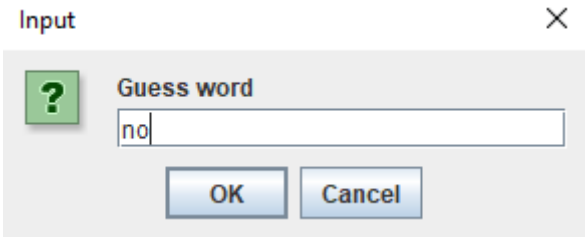
```
private void disposeWindow() {
    this.setVisible(false);
    this.dispose();
    System.exit( status: 0);
}
```

Rysunek 14 Metoda disposeWindow

Zadaniem tej metody jest zakończenie pogramu. Gdy zostanie ona wywołana to główne okno aplikacji jest ukrywane oraz usuwane. Poprzez użycie funkcji exit(), zakańczane jest działanie programu z kodem wyjścia 0.

guessCheck()

Zadaniem tej metody jest obsługa przycisku GuessWord. Metoda ta jest zobowiązana uruchomić się, gdy przycisk zostanie naciśnięty. Realizuje ona uruchomienie okna dialogowego z możliwością wpisania tekstu. Użytkownik ma prawo wpisać tam słowo, które podejrzewa, że może być rozwiązaniem zagadki. Podane przez użytkownika słowo jest następnie przez metodę guessCheck() sprawdzanie i w zależności od jego poprawności odejmowany jest punkt życia przy użyciu metody setLives() albo pokazywana jest informacja o wygranej w nowym oknie dialogowym.



Rysunek 15 Okno dialogowe GuessWord

chooseLevel()

```
private void chooseLevel() {
    String[] options = {"Easy", "Medium", "Hard"};

    int n = JOptionPane.showOptionDialog(mainPanel,
            message: "The difficulty of a word should be...",
            title: "Choose difficulty",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            icon: null,
            options,
            options[2]);
    model.chooseLevel(n);
}
```

Rysunek 16 Metoda chooseLevel

Zadaniem tej metody jest wyświetlenie okna dialogowego pozwalającego użytkownikowi na wybór poziomu trudności rozgrywki. W zależności od wybranego poziomu trudności do zmiennej n jest przypisywana liczba 0, 1 lub 2. Następnie uruchamiana jest odpowiednia metoda GUIModel.

checkGameState()

Zadaniem tej metody jest wyświetlenie okna dialogowego, które informuje użytkownika o jego wygranej bądź porażce. Zawartość okna dialogowego różni się od wartości zwracanych przez metody isWinner() oraz isLooser(). Jeżeli nastąpiła porażka czyli isLooser() zwróciło true to wyświetlane jest okno pokazujące zagadkowe hasło i informujące o sytuacji. Jeżeli nastąpiła wygrana czyli isWinner() zwróciło true to również wyświetlane jest okno informujące użytkownika o sytuacji. W wypadku, gdy rozgrywka cały czas trwa to metoda nie wyświetla żadnej informacji.

GUIModel

Ogólne przeznaczenie

Zadaniem klasy GUIModel jest przechowywanie ważnych parametrów potrzeby do odpowiedniego działania rozgrywki a także realizacji funkcji, które operują na interfejsie graficznym. Funkcje te dotyczą np. sprawdzania czy rozgrywa może być zakończona, śledzenie ilość punktów życia itp..

Szczegóły implementacji

Pola klasy GUIModel()

letterGuesses

Zadaniem tego pola jest realizacja kolekcji, która przechowywuje wszystkie litery, które zostały zgadnięte przez użytkownika. Kolekcję zdecydowaliśmy się zrealizować jako TreeSet. Wszystkie operacje, które są realizowane na tej kolekcji związane są z dodawaniem nowych elementów, których kolejność nie ma znaczenia, natomiast ważne jest to, aby nie pojawiały się w niej duplikaty.

Co więcej, na kolekcji realizowana jest operacja sprawdzania czy element się w niej znajduje. Dana implementacja zapewnia, że te operacje czyli add(), contains() posiadają złożoność log(n), co zapewnia dużą szybkość działania.

wordGuesses

Zadaniem tego pola jest realizacja kolekcji, która przechowuje wszystkie słowa zgadnięte przez użytkownika. Kolekcję zdecydowaliśmy się zrealizować jako TreeSet, z tych samych powodów co kolekcję letterGuesses.

int lives

Zmienna, w która przechowuje liczbę reprezentującą ilość punktów życia użytkownika.

String randomWord

Zmienna, które przechowuje słowo wylosowane z bazy danych. To słowo jest wartością, którą użytkownik musi odgadnąć podczas rozgrywki.

FileReader reader

Pole, które przechowuje obiekt typu FileReader, który jest wykorzystywany do odczytania losowego słowa z bazy danych.

Int countWindows

Zmienna pomocnicza, której wartość zapewnia, że ilość dodatkowych okien służących do dodawania nowych słów do bazy danych nie może zostać przekroczona. Wartość tej zmiennej wynosi domyślnie 1, co oznacza, że możemy otworzyć tylko jedno takie okno.

Metody

openAddWordPage

```
public void openAddWordPage() {
    if (countWidnows > 0) {
        countWidnows--;
        AddWordPage page = new AddWordPage( model: this);
        Thread thread = new Thread(page);
        thread.start();
        page.addWindowListener((WindowAdapter) windowClosing(e) -> {
                thread.interrupt();
                countWidnows++;
        });
    }
}
```

Rysunek 17 Metoda openAddWordPage

Zadaniem tej metody jest otwarcie nowego okna służącego do wpisywania nowych słów do bazy danych. Dzięki zmiennej countWindow upewniliśmy się, że ilość otworzonych okien nie przekracza pewnej wartości, tutaj 1.

Nowe okno jest otwierane poprzez otworzenie obiektu typu AddWordPage i przypisanie go do wątku. W celu upewnienia się, że wątek je odpowiednio zamykany po wyłączeniu okna zrealizowaliśmy przypisanie słuchacza typu WidnowAdapter. Zadaniem tego słuchacza jest przerwanie pracy wątku po zamknięciu okna klasy AddWordPage.

chooseLevel()

```
public void chooseLevel(int n) {
    if (n==0){
        reader = new EasyModeReader();
    }else if (n==1) {
        reader = new MediumModeReader();
    }else if (n==2) {
        reader = new HardModeReader();
    }
    setRandomWord(reader.readFile());
    System.out.println(randomWord);
}
```

Rysunek 18 Metoda chooseLevel()

Zadaniem tej metody jest utworzenie nowego obiektu typu FileReader i przypisanie go do pola reader przechowywanego w klasie GUIModel. Typ tworzonego obiektu zależy od argumentu podanego w momencie wywołania metody. W zależności od typu utworzonego FileReader'a, losowane jest słowo z innej puli, zawierające kolejno słowa łatwe, średnie i trudne. Na końcu przy użyciu settera, wylosowane słowo jest przypisywane do pola randomWord.

getRandomWord()

Metoda zwracająca wartość pola randomWord.

getLives()

Metoda zwracająca wartość pola lives.

setLives()

Metoda ustawiająca wartość pola lives w zależności od podanego argumentu.

setRandomWord()

Metoda ustawiająca wartość pola randomWord w zależności od podanego argumentu.

checkWord()

```
public String checkWord() {
    StringBuilder newWord = new StringBuilder();
    for(int i=0; i<randomWord.length(); i++) {
        if(letterGuesses.contains(randomWord.charAt(i))) {
            newWord.append(randomWord.charAt(i)).append(" ");
        }else{
            newWord.append("_ ");
        }
    }
    return newWord.toString();
}
```

Rysunek 19 Metoda checkWord

Zadaniem tej metody jest zwrócenie ciągu znaków, który składa się z liter i symboli „_”. Ciąg znaków tworzony jest w oparciu o zawartość kolekcji letterGuesses. Jeżeli w kolekcji letterGuesses znajduje się litera, którą zawiera słowo zagadka to jest ona dodawana do wyjściowego ciągu znaków. W przeciwnym wypadku dodawany jest symbol „_”.

isWinner()

```
public boolean isWinner() {
    int count = 0;
    for (int i = 0; i < randomWord.length(); i++) {
        if (letterGuesses.contains(randomWord.charAt(i))) {
            count++;
        }
    }
    return count == randomWord.length();
}
```

Rysunek 20 Metoda isWinner()

Zadaniem tej metody jest zwrócenie wartości true jeżeli użytkownik zgadł wszystkie litery zawarte w słowie zagadce(randomWord) lub zwrócenie wartości false jeżeli taka sytuacja nie nastąpiła. Obliczanie wartości wynikowej tej metody opiera się o sprawdzanie zawartości kolekcji letterGuesses.

isLooser()

Zadaniem tej metody jest zwrócenie wartości true albo fale w zależności czy użytkownikowi skończyły się punkty zdrowia.

```
public boolean isLooser() { return this.lives <= 0; }
```

Rysunek 21 Metoda isLooser()

setWord()

```
public String setWord() {
    return "_ ".repeat(randomWord.length());
}
```

Zadaniem tej funkcji jest zwrócenie ciągu znaków składających się z symbolu „_” w zależności od długości słowa wylosowanego z bazy danych.

Przykład

Dla słowa „kot” zostałby zwrócony ciąg znaków składający się z „ _ _ _” trzech dashy.

setLives()

```
public void setLives(char c) {
    char[] word = randomWord.toCharArray();
    for(char letter : word) {
        if (letter == c){
            return;
        }
    }
    this.lives--;
}
```

Rysunek 22 Metoda setLives()

Zadaniem tej metody jest odjęcie pojedynczego punktu zdrowa w przypadku, gdy litera podana jako argument nie zawiera się w słowie, który użytkownik musi zgadnąć.

addLetter()

Metoda dodająca literę podaną jako argument do kolekcji letterGuesses. Litera podana jako argument jest literą zgadniętą przez użytkownika.

checkDuplicates()

```java
boolean checkDuplicates(String fileName, String word) {
    try {
        var in = new Scanner(new InputStreamReader(new FileInputStream(fileName)));
        while (in.hasNext()){
            var nextWord :String = in.next();
            if (nextWord.equals(word)) return false;
        }
        in.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return true;
}
```

Rysunek 23 Metoda checkDuplicates()

Metoda, której zadaniem jest zapewnienie, że słowo, które użytkownik próbuje dodać do bazy danych już się tam nie znajduje. W przypadku gdy propozycja nowego słowa znajduje się już w bazie danych słowo nie jest dodawane.

Przykład

Mamy bazę danych składającą się z słów: kot, pies, samochód

Jeżeli użytkownik będzie chciał dodać słowo „kot”, to ta operacja zakończy się niepowodzeniem.

Jeżeli użytkownik będzie chciał dodać słowo „panda”, to nowe słowo zostanie dodane do bazy danych.

verifyWord()

```java
public boolean verifyWord(String word) {
    if(word.trim().isEmpty()){
        return false;
    }else{
        for(int i = 0; i < word.length(); i++){
            if(Character.isWhitespace(word.charAt(i))){
                return false;
            }
        }
    }
    return true;
}
```

Rysunek 24 Metoda verifyWord()

Zadaniem tej metody jest zapewnienie, że ciąg znaków podany jako jej argument nie jest pusty. Dodatkowo metoda zapewnia, że słowo nie zawiera pustych znaków i zwraca false jeżeli takie wykryje.

Przykład

Wywołujemy metodę verifyWord() z argumentem „Hello World” -> metoda zwraca wartość false, ponieważ argument zawiera „spację”.

Wywołujemy metodę verifyWord() z argumentem „HelloWorld” -> metoda zwraca wartość true, ponieważ argument nie zawiera „spacji”.

writeToFile()

```
public void writeToFile(String filename, JTextField textField) {
    try {
        FileOutputStream file = new FileOutputStream(filename,  append: true);
        file.write("\n".getBytes(StandardCharsets.UTF_8));
        file.write(textField.getText().toLowerCase().getBytes(StandardCharsets.UTF_8));
        file.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Rysunek 25 Metoda writeToFile()

Zadaniem tej metody jest zapisanie zawartość pola tekstowego podanego jako argument do odpowiedniego pliku.

saveTofile()

Zadaniem tej metody jest uruchomienie metody writeToFile() z odpowiednimi argumentami w zależności od zaznaczonego pola w oknie służącym do dodawania nowych słów do bazy danych. Jeżeli żadne z tych pól nie było zaznaczone to metoda nie wykonuje żadnej operacji.

checkduplicatesGuess()

```
public boolean checkDuplicateGuess(String word) {
    if (wordGuesses.contains(word)) return false;
    wordGuesses.add(word);
    return true;
}
```

Rysunek 26 Metoda checkDuplicateGuess()

Zadaniem tej metody jest sprawdzenie czy użytkownik próbuje zgadnąć to samo słowo po raz kolejny. Użytkownik może spróbować zgadnąć całe słowo za pomocą przycisku Guess Word. Jeżeli w przypadku takiej operacji pojawi się duplikat to wyświetlane jest odpowiednie okno dialogowe.

checkIllegalCharacters()

```
public void checkIllegalCharacters(String text) throws IllegalCharacterException{
    for(int i=0; i< text.length(); i++){
        int c = text.charAt(i);
        if(!((c>=65 && c<=90)||(c>=97 && c<=122)||((char)c == ' ')))
            throw new IllegalCharacterException("Non English Character");
    }
}
```

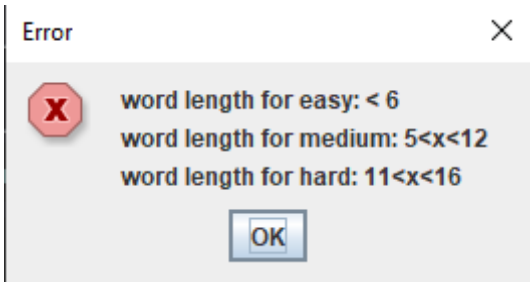
Rysunek 27 Metoda checkIllegalCharacters()

Zadaniem tej metody jest sprawdzanie czy w słowie podanym jako argument znajdują się niedozwolone znaki. Niedozwolonymi znakami są wszystkie znaki nie znajdujące się w alfabecie a także

znaki specjalne np. „ś, ć, %” itp. W przypadku znalezienia nielegalnego znaku metoda wywołuje nowy wyjątek.

checkDifficulty()

Zadaniem tej metody jest sprawdzenie długości słowa, które ma zostać wpisane do bazy danych. Metoda zwraca false jeżeli warunki dla danej trudności nie są spełnione. Metoda zwraca true jeżeli warunki są spełnione.



Rysunek 28 Wymagania słów

reset()

```
public void reset() {
    setLives(6);
    letterGuesses.clear();
    wordGuesses.clear();
}
```

Rysunek 29 Metoda reset()

Zadaniem tej metody jest zrestartowanie modelu, oznacza to wyczyszczenie kolekcji oraz zrestartowanie ilości żyć.

Klasa AddWordPage

Ogólne przeznaczenie

Zadaniem tej klasy jest wyświetlenie okna odpowiedzialnego za interfejs służąc do dodawania nowych słów do bazy danych. W klasie tworzone są odpowiednie komponenty oraz przypisywane są obiektu słuchaczy do tych komponentów. W klasie dodatkowo są zaimplementowane metody odpowiedzialne za zapisywanie proponowanego słowa do pliku oraz zapewnienie jego weryfikacji.

Dodatkowo klasa AddWordPage posiada odpowiednie powiązanie(asocjacja) z klasą GUImodel, który realizuje funkcje związane z zapisałem oraz weryfikacją słów zaproponowanych przez użytkownika.

Należy również wspomnieć, że klasa AddWordPage implementuje interfejs Runnable, który pozwala na uruchomienie jej w nowym wątku.

Szczegóły implementacji

run()

```
public void run() {
    initComponents();
    setVisible(true);
    setResizable(false);
    setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
}
```

Rysunek 30 Metoda run()

Implementacja tej metody jest wymagana przez interfejs Runnable. Uruchomiana jest ona przy starcie nowego wątku obsługującego klasę AddWordPage. Metoda ta odpowiada za uruchomienie metody initComponents() inicjalizującej wszystkie komponenty interfejsu graficznego i decydującej o ich rozmieszczeniu. Co więcej, metoda run() ustawia odpowiednie parametry JFrame po którym dziedziczy klasa AddWordPage.

initComponents()

Zadaniem tej metody, jest inicjalizacja wszystkich komponentów GUI klasy AddWordPage. Podczas tworzenia interfejsu graficznego zdecydowaliśmy się użyć zewnętrznego oprogramowania Apache NetBeans IDE 13. Komponenty posiadają absolutnego zarządcę rozkładu, oznacza to, że ich rozmieszenie jest definiowane poprzez parametry X oraz Y.

Co więcej, zadaniem tej metody jest również przypisanie odpowiednich słuchaczy do komponentów, którzy zapewniają interaktywność interfejsu graficznego.

saveTofile()

Zadaniem tej metody jest uruchomienie metody modelu odpowiadającej za zapisanie propozycji słowa, które ma być wpisane do bazy danych.

checkDifficulty()

Zadaniem tej metody jest uruchomienie metody modelu odpowiadającej za sprawdzenie czy propozycja słowa spełnia wymagania dotyczące słów łatwych, średnich oraz trudnych.

Klasa abstrakcyjna FileReader

```
abstract public class FileReader {
    abstract String readFile();
    abstract String pickWord();
}
```

Rysunek 31 Klasa FileReader

Zadaniem tej klasy jest zapewnienie, że wszystkie dziedziczące po niej inne klasy implementują metody readFile() oraz pickWord(). Klasy typu FileReader są odpowiedzialne za odczytanie wszystkich słów z pliku tekstowego zawierającego i wybranie z nich jednego losowego, które będzie celem użytkownika.

Odczytywanie zawartości plików zostało zrealizowane za pomocą strumieni.

Klasy EasyModeReader, MediumModeReader, HardModeReader

Ogólne przeznaczenie

Zadaniem tych klas jest odczytanie losowanego słowa z plików tekstowych, które traktujemy jako bazy danych. W danym momencie posiadamy trzy pliki tekstowe odpowiednie dla każdego poziomu trudności. Dodatkowo klasy zawierają poszczególne kolekcje, które są wykorzystywane do zapisu słów oraz wykonywania na nich operacji losowania.

Szczegóły implementacji

Szczegóły implementacji zostaną omówione na postawie tylko jednej klasy, ponieważ ogólne metody realizowane przez każdą z nich są wykonywane w podobny sposób. Należy w wspomnieć , że każda z tych klas dziedziczy po klasie abstrakcyjnej FileReader().

Pola klas

arrayList

Kolekcja, która służy do przechowywania odczytanych słów z pliku tekstowego. Zdecydowaliśmy się zrealizować tą kolekcję jako obiekt klasy ArrayList, ponieważ w tym przypadku potrzebna była nam funkcjonalność związana z indeksowaniem elementów. Co prawda, elementy znajdujące się w bazie danych nie mogą się powtarzać, jednak kolekcja TreeSet nie pozwala nam na zrealizowanie wybrania losowego elementu w łatwy i dogodny sposób. Rozważaliśmy również opcję użycia kolekcji HashSet jednak w dokumentacji napisane jest, że ta kolekcja nie zapewnia zachowania kolejności wraz z upływem czasu, co mogło spowodować problemy z losowaniem słowa w skończonym czasie. Zaimplementowanie ArrayList, która zapewnia metodę get() pozwoliło nam na wylosowanie słowa w dogodny i prosty sposób.

```
public String pickWord() { return arrayList.get((int) (Math.random() * arrayList.size())); }
```

String fileName

Pole służące do przechowywania nazwy pliku, z którego mają być odczytywane słowa do losowania.

Metody

readFile()

```
public String readFile() {
    try {
        var in = new Scanner(new InputStreamReader(new FileInputStream(fileName)));
        while (in.hasNext()){
            var word :String = in.next();
            arrayList.add(word);
        }
        in.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        System.out.println("File not found!");
    }
    return pickWord();
}
```

Rysunek 32 Metoda ReadFile

Zadaniem tej metody jest odczytanie wszystkich słów z pliku tekstowego oraz uruchomienie metody pickWord(), która zwraca jedno losowe słowo z tych odczytanych.

pickWord()

Zadaniem tej metody jest zwrócenie losowanego elementu z kolekcji arrayList.

Klasa IllegalCharacterException

```
public class IllegalCharacterException extends Exception{
    IllegalCharacterException(String errorMessage) { super(errorMessage); }
}
```

Rysunek 33 Klasa IllegalCharacterException

Klasa dziedziczy po klasie Exception. Zadaniem jej jest zgłoszenie odpowiedniej sytuacji wyjątkowej związanej z wystąpieniem nieodpowiedniego znaku podczas wykonywania operacji pobierania ciągu znaków od użytkownika.

Zaimplementowane kolekcje

TreeSet

Kolekcja została wykorzystana do realizacji pola letterGuesses i wordGuesses klasy GUIModel(). Uzasadnienie wykorzystania tych kolekcji zostało napisane w tym miejscu.

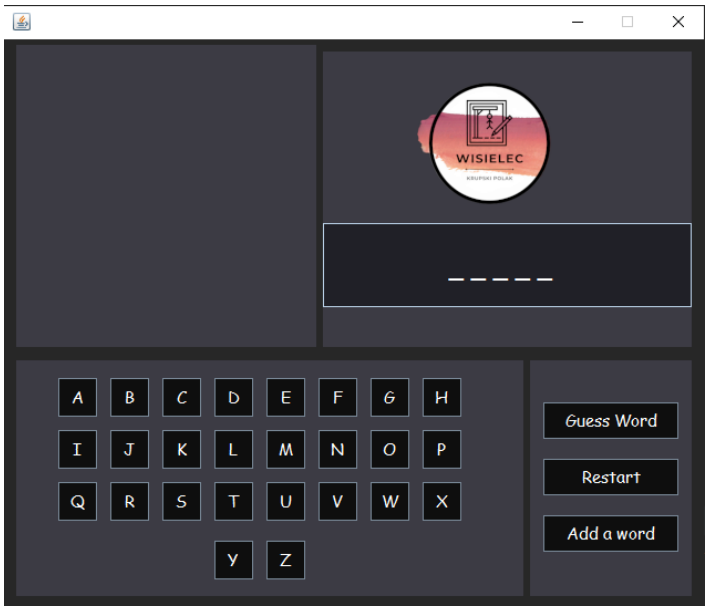
ArrayList

Kolekcja została wykorzystana do realizacji pola arrayList w klasach EasyModeReader, MediumModeReader, HardModeReader. Uzasadnienie wykorzystania tych kolekcji zostało napisane w tym miejscu.

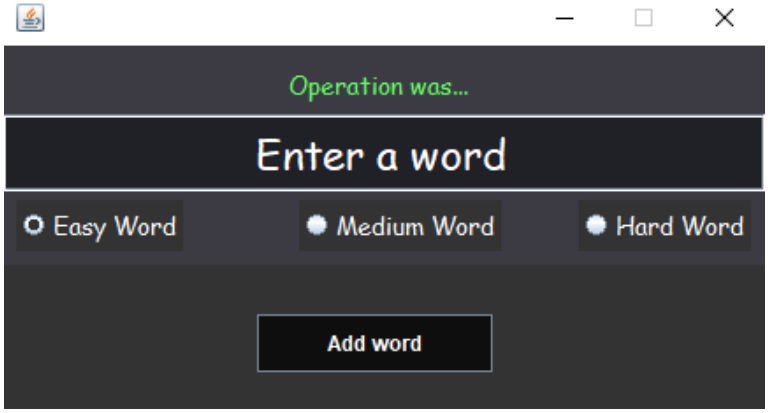
Graficzny interfejs użytkownika

W tej części zostanie zaprezentowany obecny wygląd GUI.

Okno główne aplikacji



Okno AddWordPage



Zapis i odczyt informacji za pomocą strumieni

Zapis do pliku z wykorzystaniem strumieni został zrealizowany w metodzie writeToFile() klasy GUIModel().

```
public void writeToFile(String filename, JTextField textField) {
    try {
        FileOutputStream file = new FileOutputStream(filename, append: true);
        file.write("\n".getBytes(StandardCharsets.UTF_8));
        file.write(textField.getText().toLowerCase().getBytes(StandardCharsets.UTF_8));
        file.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Rysunek 34 Zapis za pomocą strumieni

W tej klasie otwieramy odpowiedni plik tekstowego, do którego będziemy zapisywać słowo znajdujące się w obiekcie JTextField podanym jako argument.

Odczyt z pliku za pomocą strumieni został zrealizowany w metodzie readFile() klasy FileReader().

```java
public String readFile() {
    try {
        var in = new Scanner(new InputStreamReader(new FileInputStream(fileName)));
        while (in.hasNext()){
            var word :String = in.next();
            arrayList.add(word);
        }
        in.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        System.out.println("File not found!");
    }
    return pickWord();
}
```

Rysunek 35 Odczyt z pliku za pomocą strumieni

Wielowątkowość

Wielowątkowość programu została zrealizowana poprzez uruchomienie okna WisielecFrame oraz AddWordPage w osobnych wątkach.

Osobny wątek dla okna głównego aplikacji WisielecFrame, został zrealizowany, ponieważ obsługa interfejsu graficznego jest najbardziej efektywna i najmniej problemowa w ten sposób. Jeżeli GUI potrzebuje wykonać różanego rodzaju operacje związane z animacjami i interaktywnością interfejsu nie zakłóca to pracy innych klas, które np. muszą odczytywać różne informację z plików czy obliczać wartości pewnych zmiennych.

Osobny wątek dla okna AddWordPage został zrealizowany, ponieważ w tym oknie realizowane są operacje odczytywania i zapisywania do plików różnych informacji. Operacje te mogą być kosztowne czasowo, jeżeli pliki tekstowe zawierają dużą ilość danych. Dodatkowo chcieliśmy, aby restartowanie rozgrywki nie miało wpływu na stan tego okna. Dzięki realizacji w osobnym wątku użytkownik ma możliwość restartowania rozgrywki i dodawania nowych słów w tym samym momencie. Co więcej, jeżeli chcielibyśmy otworzyć okno AddWordPage bez uruchamiania go w nowym wątku to operacje na nim wykonywane mogłyby kolidować z oknem WisielecFrame powodując brak płynności pracy.

Wykorzystanie wyjątków kontrolowanych

Wyjątki kontrolowane zostały wykorzystane np. przy weryfikacji słów wpisywanych przez użytkownika podczas wprowadzania ich do bazy danych. W tym celu została zrealizowana osobna klasa dziedzicząco po klasie Extension.

```java
public void checkIllegalCharacters(String text) throws IllegalCharacterException{
    for(int i=0; i< text.length(); i++){
        int c = text.charAt(i);
        if(!((c>=65 && c<=90)||(c>=97 && c<=122)||((char)c == ' ')))
            throw new IllegalCharacterException("Non English Character");
    }
}
```

Rysunek 36 Przykład wyjątków kontrolowanych