**Bulgarian Diploma Thesis**

**Prompt Image Generator**

**Martin Mitev, ID: 200084280**

**Student:**   **, Date:**

*signature*

**Supervisor:**   **, Date:**

*signature*

**Department of Computer Science, AUBG**

**Blagoevgrad, 2023**

**Title**: Prompt Image Generator

**Author**: Martin Mitev

**Abstract:**

In this work, I present the development of a software application designed to transform natural language inputs into visual representations. The program integrates Java, Java Swing, and several external libraries like Deep Learning for Java (DL4J), Word2Vec, ND4J, and Apache Maven to create a versatile and user-friendly graphical user interface (GUI). This GUI serves as the interactive platform for users to input prompts, which are then processed using advanced natural language processing (NLP) techniques. Central to the application's functionality is the use of the Word2Vec model, which facilitates the understanding and translation of complex linguistic patterns into actionable data. Additionally, I implemented the Levenshtein Distance algorithm for correcting typographical errors in user inputs and a variant of K-means clustering to calculate cosine similarities for shape and color recognition. The software showcases a dynamic rendering of various shapes in response to user prompts, underlining its graphical versatility.

**Declaration of authorship:**

“The Senior Project/Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of Prof. Dimitar Christozov. All sources, used in development, are cited in the text and in the Reference section.”

Author: Martin Mitev

**Table of contents:**

Introduction………………………………………………………………...1

Specification of the software requirements and their analysis ………...4

Design of the software solution…………………………………………….15

Implementation…………………………………………………………………55

Results and conclusion……………………………………………………….....60

1. **Introduction**

Throughout the last few years, the world of Computer Science and Information Systems has witnessed substantial developments in the field of Artificial Intelligence (AI). Wind the clock back three short years ago to the year 2020. At that time, the Internet was brimming with articles, such as “A robot wrote this entire article. Are you scared yet, human?” by *The Guardian* and “A new AI language model generates poetry and prose” by *The Economist*, illustrating the capabilities of Natural Language Processing (NLP) models like GPT-3.

While the articles published by *The Guardian* and *The Economist* featured GPT-3’s impressive capacity to understand and express human language, much of what the bot managed to write often sounded clumsy and half-coherent. Such drawbacks meant that the tech industry had to significantly improve these machine learning models to turn them into viable tools.

Fast-forward to 2023 and you will notice that OpenAI, GPT-3’s creator, has already released ChatGPT, a chatbot equipped with GPT-3.5 for regular users and GPT-4 for subscribers to the service. Furthermore, the company recently unveiled DALL-E 3, an advanced text-to-image model capable of understanding complex prompts, generating high-definition (HD) images. What’s more, throughout the past year, a diverse plethora of applications, including AI-powered text editors and grammar checkers, built-in browser chatbots (most notably Microsoft’s Bing chat), as well as Integrated Development Environment (IDE) assistants like GitHub Copilot, have flooded the tech market. With that being said, it is clear Artificial Intelligence has earned its status as a worthy tool, employed in both the professional and academic fields. In fact, machine learning is being so widely integrated into everyday appliances that many researchers and tech leaders are concerned about its safety. Recently, Elon Musk, alongside a thousand other IT experts, signed an open letter urging for a moratorium on the development of large AI systems (Metz and Schmidt).

Considering the notable advancements in and some of the challenges posed by AI over the last decade, I decided to dedicate my Senior Thesis on a computational intelligence-based application. Since I have been especially impressed by DALL-E and other text-to-image models, my intention for this project is to create a simple image generator. In basic terms, the program has the following functionality:

1. It makes use of the NLP technique known as Word2vec. The code first imports text files containing lists of shapes and colors. Then it loads a dictionary that contains the “translation” of a vast set of words into multidimensional vectors consisting of numeric values;
2. The program then forms two clusters containing the vector values of each color and shape. It computes the centroid (the average of the vector values) of each cluster;
3. It employs the two clusters to find the cosine similarity between each word in the user-defined prompt and the centroids.
4. Once it has identified the desired shape and a color, the program draws that shape (colored according to the user’s specifications).

The programming language used for this project is Java, accompanied by Apache Maven, the Eclipse Deeplearning4j (DL4J) machine learning library and its extension N-Dimensional Arrays for Java (ND4J), and Java Swing.

Overall, Java is a widely-used, high-level programming language known for its platform independence, object-oriented structure, and security features. Developed by Sun Microsystems (now part of Oracle Corporation), it operates on the Java Virtual Machine (JVM) that allows Java code to run on any device equipped with a JVM (“What is Java”). Java is commonly used for developing enterprise-level applications, Android apps, web applications, and large systems. Its popularity is supported by a vast ecosystem of libraries, frameworks, and a large developer community.

Maven is a build automation tool, most widely used alongside Java. It was developed by the Apache Software Foundation. It simplifies project configuration and dependency management through an XML-based Project Object Model (POM). Maven emphasizes convention over configuration, offers standardized build lifecycles, and supports plugins for various tasks (“What Is Maven?”). Despite its learning curve and performance concerns, its capacity for consistent, reproducible builds and strong community support make it a popular choice in the Java ecosystem.

Eclipse Deeplearning4j is an open-source, distributed deep learning library tailored for Java and Scala environments, offering seamless integration with big data tools like Apache Hadoop and Spark. It supports a range of neural network types, offers tools for data preprocessing and hyperparameter tuning, and is designed for scalability across multiple GPUs and CPUs. Initially developed by Skymind, DL4J is particularly suited for business applications where integration with existing Java-based systems and big data analytics is essential (“Deeplearning4j Suite Overview”).

Next, ND4J, or N-Dimensional Arrays for Java, plays a vital role in the project as an extension of the Eclipse Deeplearning4j (DL4J) machine learning library. As a powerful scientific computing library for Java, ND4J is akin to NumPy in Python and is specifically optimized for handling high-dimensional numerical data and complex mathematical operations. It provides the backbone for efficient computation within DL4J, especially in the context of neural network training and operations involving large datasets (“Deeplearning4j Suite Overview”).

Finally, Java Swing is a graphical user interface (GUI) toolkit in Java, part of the Java Foundation Classes (JFC). It provides a rich set of widgets and components for building GUI applications, such as buttons, text fields, sliders, and tables. Swing is known for its platform-independent design, allowing developers to create applications that look and feel consistent across different operating systems. It's widely used for developing desktop applications and has been a staple in the Java ecosystem for many years, offering flexibility and customization options for user interfaces (“Introduction to Java Swing”).

Given the synergistic potential of these technologies, my Senior Thesis aims to not only explore the intersection of NLP and image generation but also to contribute to the practical applications of AI in creative domains.

1. **Specification of the software requirements and their analysis**

The software package is designed to integrate various functionalities, including text processing, pattern recognition, color code retrieval, and shape drawing. Its software requirements are shaped by the need to provide efficient, user-friendly, and versatile services.

At the core of the application is the Word2Vec model handling. The software is required to load a Word2Vec model, specifically sourced from the Google News dataset. This feature is pivotal for the application’s ability to perform advanced linguistic processing and semantic analysis. The loading process is accompanied by a performance metric that reports the time taken to load the model, ensuring transparency and providing valuable insights for performance optimization.

In the realm of text processing, the application is equipped to process textual inputs by filtering out unnecessary words and symbols. This includes removing text between braces, non-alphanumeric characters, slashes, semicolons, and stopwords obtained from an external dictionary. Such a feature ensures that the software focuses on meaningful content, enhancing the efficiency and relevance of its text processing capabilities.

The software also boasts a color code retrieval function. It can retrieve the RGB values of a color by its name from an extensive predefined list. This feature is essential for applications where precise color representation is necessary, such as in design or visual applications.

Another significant aspect of the software is its ability to draw colored shapes based on user input. The system can render various shapes, like circles, triangles, squares, and more, each derived from a parent shape data type. The shapes are colored according to user specifications, adding a layer of customization and interactivity to the application.

In addition to these functional requirements, the software has been developed with a strong emphasis on non-functional aspects such as performance, usability, and maintainability. The system is designed to process input prompts swiftly, ensuring that the desired shape and color are determined quickly and rendered without significant latency. Usability is enhanced through features like informing users about the loading time of the Word2Vec model and providing comprehensive runtime logs. These elements not only make the software more transparent but also easier to understand and interact with.

Maintainability is addressed through modular and well-documented code, simplifying future updates and enhancements. The software’s design also allows easy additions of new color names and RGB values, ensuring its adaptability over time.

This application is envisioned to be a utility for processing text and colors, potentially serving a wide range of industries, from design and education to natural language processing. The software package consists of several components with distinct functionalities, detailed as follows:

1. Word2vec Model Loading:

* The program should allow the loading of a Word2Vec model, specifically from the Google News dataset.
* It must measure and report the time taken to load the Word2Vec model.

1. Pattern Recognition and Text Processing:

* The application must identify and cut out irrelevant words and characters that have no effect on the code’s understanding of the user-provided prompt, and would instead clutter the implemented algorithms, slowing down the software’s performance.
* It should be able to read a list of stop words from a file for text processing purposes.
* It ought to provide functionality to check a text against these stop words.

1. Color Code Retrieval and Conversion:

* The application should provide a way to retrieve the RGB values of a color by its name.
* It must support a predefined set of color names and their corresponding RGB codes.
* The system should allow the user to input a color name and receive the correct RGB values.

1. Drawing Shapes Based on User Input:

* The system shall allow a user to input a shape name and color via a prompt.
* The system shall render the specified shape with the given parameters onto the drawing panel.
* The system shall update the drawing in real-time upon receiving new user inputs.

When it comes to satisfying the Word2vec Model Loading functional requirement, the system's capability to handle Word2Vec models, specifically the Google News variant, is a testament to its advanced linguistic processing capabilities. By loading this model from a predetermined location, the system gains access to a vast lexicon and semantic relationships between words. The efficiency of this process is underscored by the system's capacity to track and relay the duration of the model loading phase, which provides transparency and aids in performance tuning.

In the case of the Pattern Recognition requirements, the software package will discern stop words within a body of text through a dedicated pattern recognition function. This function cross-references individual words against the retrieved stop words list, thereby filtering out non-substantive words from the analytical processes. This selective attention to content-rich words ensures that the system's textual analysis is both efficient and pertinent.

Additionally, the color code generation component of the system is pivotal for providing the Color Code Retrieval and Conversion functionality as it converts user-inputted color names into their respective RGB values. This translation is crucial for applications where precise color representation is necessary. The system is designed to accept a string denoting a color and utilize a map of predefined colors to fetch and return the RGB values associated with that color.

Next, the system will fulfill the requirements regarding the Drawing Shapes Based on User Input Analysis functionality due to the usage of a DrawingPanel class, the program’s creative hub where user-specified shapes come to life. Upon receiving inputs—such as shape type and color—the system conducts a rigorous validation process to ensure compatibility with predefined standards. Subsequent to validation, the system harnesses drawing routines tailored to each shape. Furthermore, the DrawingPanel is dynamic, capable of refreshing and updating the visual display in real-time to reflect new user instructions, thus providing an interactive and responsive user experience.

Finally, the code will feature robust error handling and validation are core to the system's stability and reliability. The design includes comprehensive prompt validation, ensuring that the prompt entered is not the same as the placeholder text "Please enter a prompt" and that the shape identified by the model matches any of the shapes available in a preloaded list of available shapes. Additionally, the system is resilient in the face of potential file I/O issues that could arise during the retrieval of stop words or the loading of the Word2Vec model, ensuring that such incidents do not result in system failure.

The pseudocode snippets below illustrate these features:

1. Word2Vec Model Handling:

FUNCTION LoadWord2VecModel()

filePath = "src/main/resources/GoogleNews-vectors-negative300.bin.gz"

startTime = GetCurrentTime()

loadTime = GetCurrentTime() - startTime

PRINT "Time to load: " + loadTime + " ms"

RETURN word2VecModel

END FUNCTION

Fig. 1

The LoadWord2VecModel function is designed to facilitate the loading of a Word2Vec machine learning model into the system, specifically targeting a model provided by Google News. It starts by defining the location of the model file within the project's resources directory, which it expects to find at a given path. With this location pinpointed, the function then records the current time to establish a baseline for performance measurement.

The function first calculates how long the loading process took. It does this by getting the current time again and subtracting the previously recorded start time, yielding the duration of the load in milliseconds. This duration is then announced, typically printed out to a console or a log, providing insight into the performance of the loading process.

In the final act of its operation, the function concludes by returning the loaded Word2Vec model object. This outcome allows the calling process to then utilize the model for various machine learning tasks such as natural language processing, semantic analysis, or other related activities that benefit from the sophisticated linguistic patterns that the Word2Vec model encapsulates.

1. Text Processing and Pattern Recognition:

FUNCTION ProcessText(inputText: STRING, stopWords: LIST<STRING>)

processedText = RemoveSpecialCharacters(inputText)

words = SplitIntoWords(processedText)

FOR EACH word IN words

IF NOT IsStopWord(word, stopWords)

AddToProcessedWords(word)

ENDIF

ENDFOR

RETURN ProcessedWords

END FUNCTION

Fig. 2

The ProcessText function, as outlined in the pseudocode, is a fundamental component in the field of text processing and natural language processing. This function's primary role is to prepare the input text for further analysis by cleansing and filtering it.

At the commencement of the function, the input text, represented by the parameter inputText, undergoes an initial cleaning process where all special characters are removed. This step is crucial as special characters often do not contribute to the semantic analysis of the text. The nature of these special characters is typically inclusive of punctuation marks and other non-alphanumeric symbols, which might otherwise interfere with the processing of meaningful words.

Following the cleansing step, the function breaks down the cleaned text into individual words. This process, termed as splitting, transforms the continuous string of text into a list or an array of words, facilitating word-by-word analysis. The operation of splitting is generally based on predefined delimiters like spaces, enabling the isolation of each word in the text.

The core of the ProcessText function lies in its ability to filter out stopwords from this list of words. Stopwords, which are common words in a language that are often deemed irrelevant in text analysis, are filtered using a conditional check within a loop. Each word in the processed list is checked against the list of stopwords; if a word is not a stopword, it is then added to a separate list of processed words. This selective inclusion ensures that the final output of the function is devoid of unnecessary words, retaining only those words that carry significant meaning for further analysis.

Finally, the function culminates by returning this list of processed words. The returned list is a refined version of the original text, stripped of special characters and stopwords, making it an optimized input for subsequent text analysis tasks such as keyword extraction, sentiment analysis, or other semantic evaluations.

In summary, the ProcessText function plays a pivotal role in preparing text for deeper analysis. By cleansing the text and selectively filtering out irrelevant words, it ensures that further analysis can be performed more efficiently and effectively, focusing solely on the content that contributes meaningfully to the intended analytical objectives.

1. Color Code Generation:

FUNCTION GetRGBValue(colorName: STRING)

colorMap = InitializeColorMap()

rgbValues = colorMap GET colorName

RETURN rgbValues

END FUNCTION

Fig. 3

This snippet describes a function named GetRGBValue which is designed to retrieve the RGB values of a given color name. This function plays a crucial role in applications where color representation and manipulation are important, such as graphic design software, data visualization tools, or any user interface that deals with colors. At the beginning of the function, a colorMap is initialized by invoking the InitializeColorMap method. This colorMap is likely a data structure, such as a dictionary or a map, which holds key-value pairs. In this context, each key is a color name (a string), and the corresponding value is the color's RGB representation, also likely stored as a string or a structured data type encapsulating the RGB components. Once the colorMap is initialized, the function proceeds to check if this map contains the color name provided as an input argument (colorName). This check is crucial as it determines whether the requested color is supported by the system. If the colorMap does contain the color name, the function retrieves the associated RGB values. This retrieval is performed using a GET operation on the colorMap, which fetches the value (RGB data) associated with the key (color name). The RGB values are then returned to the caller of the function. These values can be used for various purposes like setting the color of a graphical object, performing color conversions, or as part of a larger color-related operation within the application.

In summary, the GetRGBValue function is an efficient and straightforward method for retrieving RGB values corresponding to color names. Its reliance on a pre-initialized colorMap makes it quick and effective for frequent lookups. This function exemplifies a common programming practice in color-related applications, combining data structure utilization with conditional logic to deliver precise and necessary information.

1. Drawing Shapes Based on User Input:

CLASS DrawingPanel

METHOD DrawShape(shapeName: STRING, color: Color)

shape = ShapeFactory.CreateShape(shapeName, color)

DisplayShape(shape)

END METHOD

END CLASS

Fig. 4

The pseudocode snippet presents a class named DrawingPanel, which encapsulates functionality related to drawing shapes. Within this class, there is a defined method DrawShape that serves as the primary interface for rendering shapes based on specified parameters.

The DrawingPanel class is likely a component of a graphical user interface (GUI) or a graphics application where users can draw various shapes. The essence of this class is to provide a canvas or a platform where shapes can be visually represented.

The DrawShape method within the DrawingPanel class is particularly noteworthy. It takes two parameters: shapeName, which is a string representing the type of shape to be drawn, and color, which is an object or a data structure encapsulating the color information for the shape. This method embodies the core functionality of the DrawingPanel class.

The first operation inside the DrawShape method involves invoking the ShapeFactory.CreateShape method. This suggests the presence of a factory design pattern, a common approach in object-oriented programming for creating objects. The ShapeFactory is responsible for creating shape objects based on the provided shapeName and color. The use of a factory here indicates a design choice that abstracts the instantiation logic of shape objects, thereby enhancing the modularity and scalability of the application. It allows the DrawingPanel to remain agnostic of the specific details of each shape type, supporting a wide range of shapes without needing modifications to its code.

Once the shape object is created, the method proceeds to display this shape. The DisplayShape function is called, passing the newly created shape object. This part of the method is responsible for the actual rendering of the shape on the DrawingPanel. It involves graphical operations that take the shape's properties, such as dimensions, color, and style, and visually represent them on the panel.

In summary, the DrawingPanel class, particularly through its DrawShape method, encapsulates the functionality of rendering various shapes in a graphics application. The use of a factory pattern for shape creation exemplifies good design principles, promoting flexibility and scalability. The class design allows for easy extension to accommodate new shapes and colors, making it a versatile component in any graphics or GUI-based application. The clear separation of concerns – shape creation via a factory and shape rendering within the panel – further underscores the robustness of the design approach.

1. Error Handling and Validation:

FUNCTION ValidateInputPrompt(inputPrompt: STRING)

PLACEHOLDER\_TEXT = "Please enter a prompt";

IF inputPrompt EQUALS PLACEHOLDER\_TEXT THEN

RAISE Error("The default placeholder text is not a valid input.");

ENDIF

RETURN TRUE;

END FUNCTION

FUNCTION ValidateShapeName(shapeName: STRING)

KNOWN\_SHAPES = ["circle", "square", "triangle", ..., "pentagon"];

IF NOT KNOWN\_SHAPES CONTAINS shapeName THEN

RAISE Error("Shape not recognized. Please enter a shape from the list of known shapes.");

ENDIF

RETURN TRUE;

END FUNCTION

FUNCTION HandleFileReadError()

PRINT "Error reading the file."

// Additional error handling logic

END FUNCTION

Fig. 5

The code snippet consists of three distinct functions, each integral to the overall robustness of the system by ensuring valid user input and handling potential errors.

The first function, ValidateInputPrompt, is designed to ensure that users do not simply submit the default placeholder text as their input. It compares the user's input prompt with a predefined placeholder string. If the input prompt is identical to this placeholder, the function raises an error, signaling to the user that they need to provide actual input rather than the placeholder text. If the input is not the placeholder, the function proceeds without issue, implicitly confirming the input's validity.

Next, ValidateShapeName plays a critical role in the system's shape recognition capability. It maintains a list of known shapes, which the system can recognize and process. When a shape name is provided, the function checks this list to determine if the name corresponds to one of the shapes that the system is programmed to understand. If the provided shape name is not found within the known shapes, an error is raised, prompting the user to submit a valid shape name from the list.

Lastly, the function HandleFileReadError is a part of the system's error-handling mechanism specific to file operations. When a file-read operation encounters an issue, such as the file being inaccessible, missing, or corrupted, this function is called. It informs the user of the issue by printing an error message. The comment indicates that additional error handling logic should be implemented as well, which might include logging the error details for debugging purposes or attempting to recover from the error by retrying the file read operation or providing alternative solutions.

These functions collectively ensure that the user inputs are as expected and not just placeholders, that they conform to the system's capabilities regarding shape recognition, and that any file reading errors are caught and managed effectively. This approach not only enhances the user experience by providing immediate and clear feedback but also contributes to the system's resilience by preventing unhandled exceptions from causing crashes or other failures.

In terms of non-functional requirements, during the development of the software, I have directed substantial care and effort into adhering to some of the strictest coding guidelines and best practices. Although parts of the program can be subjected to further optimization, I have designed the application to meet at least these nonfunctional requirements in regard to performance, usability, and maintainability:

1. Performance:

* The system shall process the inputted prompts in a timely manner, quickly determining the desired shape and color.
* The system shall efficiently handle the retrieval of RGB values without significant latency.

1. Usability:

* The system shall accurately inform the user about the loading time of the Word2Vec model to guarantee a transparent overview of the delay associated with the size of the model’s source file.
* The system shall provide a thorough log of all major runtime processes to enhance the user’s understanding of the program’s inner workings.
* The system shall provide clear and informative feedback to the user, especially in cases of incorrect input or system errors.
* The system shall ensure ease of use, with a straightforward method for retrieving RGB values and using the Word2Vec model.

1. Maintainability

* The system’s source code shall be modular and well-documented to facilitate easy updates and maintenance.
* The system shall be designed to allow for the easy addition of new color names and their corresponding RGB values to the predefined list.

These requirements delineate the scope and expectations for the application's functionality, as well as the constraints and desired qualities that shape the system's development and operation. They serve as a foundational framework for the design, implementation, and testing phases of the software development lifecycle.

1. **Design of the software solution**

In this section, I will describe the class structure of the software package, demonstrate the capabilities of the user interface, explain in detail the algorithms and data structures employed throughout the program, and finally describe the security and reliability features taken into account by the application.

1. **Class structure and analysis**

The application consists of the abstract and non-abstract abstract data types listed below:

1. Public class Prompter;
2. Public abstract class PatternRecognizer;
3. Public abstract class Word2VecLoader;
4. Public class ClusterInfo;
5. Public abstract class StopWords;
6. Public class DrawingPanel extending Java Swing’s JPanel class;
7. Public class ColorCodeGenerator;
8. Public abstract class Shape;
9. Public class Circle extending the Shape class;
10. Public class Triangle extending the Shape class;
11. Public class Square extending the Shape class;
12. Public class Rectangle extending the Shape class;
13. Public class Arrow extending the Shape class;
14. Public class Crescent extending the Shape class;
15. Public class Cross extending the Shape class;
16. Public class Diamond extending the Shape class;
17. Public class Heart extending the Shape class;
18. Public class Parallelogram extending the Shape class;
19. Public class Rhombus extending the Shape class;
20. Public class Star extending the Shape class;
21. Public class Trapezoid extending the Shape class;
22. Public class Cone extending the Shape class;
23. Public class Cube extending the Shape class;
24. Public class Cylinder extending the Shape class;
25. Public class Prism extending the Shape class;
26. Public class Pyramid extending the Shape class;
27. Public class Sphere extending the Shape class;
28. Public class Ellipse extending the Shape class;
29. Public class Oval extending the Ellipse class;
30. Public class Polygon extending the Shape class;
31. Public class Pentagon extending the Polygon class;
32. Public class Hexagon extending the Polygon class;
33. Public class Octagon extending the Polygon class;
34. **Public Class Prompter**

Laying the foundation of the Prompt Image Generator, the main objectives of the Prompter class are to load the Word2vec model trained on the Google News dictionary of word embeddings, as well as to build and initialize the application's Graphical User Interface. The user interface is built using standard Java Swing components, such as JFrame for the window frame, JTextField for user input, JLabel for labels, and JButton for clickable buttons.

Within the Prompter class, there is a comprehensive integration of various Java libraries essential for GUI creation, such as javax.swing, and tools necessary for event handling, regular expressions, input/output operations, and color processing. This integration signifies the application's capability to handle complex linguistic data and interactive elements efficiently. A regex pattern (charsPunctuationPattern) is thoughtfully defined to filter out unwanted characters from user inputs, a vital step in supporting the NLP tasks.

The constructor of the Prompter class is strategically designed to initialize the GUI, orchestrating the setup of visual components, loading the Word2Vec model, and configuring placeholder text in the input field. This initialization ensures that the interface is not only visually appealing but also ready for user interaction immediately upon execution. An action listener attached to the submit button triggers the processing of the user's text upon being clicked. This listener is responsible for validating the input, sanitizing it by removing punctuation and stop words, and then employing the PatternRecognizer to accurately discern the shape and color specified in the prompt. Upon successful identification, the DrawingPanel takes on the crucial role of rendering the shape in the selected color, effectively translating the user's words into a visual representation.

The separatePromptWords method is an integral part of the input processing pipeline. It cleans the text, which is vital for the subsequent NLP tasks, ensuring the accuracy of shape and color detection.

Finally, the show method and the main method are the gateway to the application's visibility and execution. They ensure that the application is displayed correctly and that it runs in a thread-safe manner, adhering to the conventions of Swing applications.

1. **Public Abstract Class PatternRecognizer**

The second class on the list is PatternRecognizer, which plays a pivotal role in recognizing patterns, especially in identifying specific objects such as shapes or colors, from a collection of words. This class is integral to a system likely designed for processing natural language inputs, suggesting its use in applications where user prompts or textual inputs are analyzed to extract meaningful information.

One of the primary functionalities of the PatternRecognizer data type is the calculation of the Levenshtein distance, as achieved through the levenshteinDistance method. This method calculates the edit distance between two strings, effectively quantifying how dissimilar they are by counting the minimum number of single-character edits (insertions, deletions, or substitutions) needed to change one word into the other. The method employs dynamic programming to efficiently compute these differences, making it a robust tool for spelling correction and similarity assessment in text processing.

Another critical method in the class is findClosestObject, which is adept at determining the closest object to a given word from a list of objects, using the previously mentioned Levenshtein distance. This method not only identifies the most similar object but also considers a maximum distance (maxDistance) as a threshold for comparison, ensuring relevance and accuracy in the recognition process.

The readObjectsFromFile method adds another layer of functionality by reading a list of objects from a specified file. It is designed to handle the file I/O operations, reading each line and adding objects, expected to be separated by commas and spaces, to a list. This method is crucial for populating the system with data necessary for pattern recognition tasks.

Further extending the class's capabilities is the computeTargetObject method. This method identifies the most relevant object from a collection of prompt words. It utilizes a ClusterInfo object and a Word2Vec model (dictVec) to provide additional context, thereby enhancing the accuracy of object recognition. Additionally, it incorporates spell checking with a set maximum allowable edit distance and similarity computation, further refining the process. Once a target object is found, it is removed from the prompt words collection, streamlining the set of words for subsequent processing.

From a design and usage perspective, the PatternRecognizer class is declared abstract, indicating its role as a foundational class providing methods for subclasses rather than being instantiated on its own. The methods within the class are static, allowing for direct invocation without the need for creating an instance of PatternRecognizer. This design choice points towards a utility-like usage of the class, where methods are accessed as tools or services.

1. **Public Abstract Class Word2VecLoader**

Next, the abstract data type Word2VecLoader is tasked with the crucial role of loading the Word2Vec model. The Word2Vec algorithm is well-known for its ability to discern word associations through extensive text analysis.

The class contains a static method named loadDictVec, which furnishes an instance of Word2Vec. On invocation, loadDictVec notes the current time, allowing for the computation of the model loading duration—a feature of importance given the potentially large size and consequent load time of Word2Vec models.

The method proceeds to reference a specific Google News vector file stored in the "src/main/resources/" directory, a file notable for its derivation from a segment of the Google News dataset that encompasses roughly 100 billion words. Utilizing the Deeplearning4j library's WordVectorSerializer.readWord2VecModel, it loads the Word2Vec model from this file.

Upon successfully loading the model, loadDictVec calculates the elapsed time using the earlier recorded start time, presenting the load duration in milliseconds. Such a feature is beneficial for monitoring performance, especially considering the extensive load times associated with substantial models like those trained on Google News vectors. Subsequently, the method culminates by returning the now-loaded Word2Vec model.

In essence, Word2VecLoader operates as a specialized utility class. It streamlines the process of loading Word2Vec models and provides insightful timing metrics for the loading procedure, all facilitated through the Deeplearning4j library's functionality. This class simplifies the integration and application of complex neural network algorithms into systems requiring advanced text analysis and word association capabilities.

1. **Public Class ClusterInfo**

Further, the ClusterInfo class, primarily designed to manage clusters in the context of word embeddings using Word2Vec models, is a vital component for tasks in natural language processing (NLP) and semantic analysis. The data type deals with vector space representations of words.

ClusterInfo encompasses several key fields. The int id field serves as a unique identifier for each cluster. The Word2Vec dictVec is a reference to the Word2Vec dictionary vector model used by the cluster, a crucial element for handling word embeddings. Another significant field is INDArray centroid, which represents the centroid vector of the cluster. This centroid is computed as the average of the vectors of its member objects, thereby serving as a representative point in the multidimensional vector space.

The constructor of the ClusterInfo class plays a critical role. It takes three parameters: an identifier (id), a Word2Vec model (dictVec), and a list of objects, which are presumably words or terms. In the initialization process, the constructor calculates the average vector of the given objects using ND4J's INDArray, a data type for handling numerical operations on high-dimensional arrays. This process effectively establishes the centroid of the cluster. It is noteworthy that if an object's word vector is not found in the dictVec model, the constructor handles this by printing a message and excluding the object from the centroid computation.

The class also includes getter methods like getId and getCentroid, which return the cluster's ID and its centroid, respectively. These methods are essential for accessing the cluster's fundamental properties.

Another critical functionality of the ClusterInfo class is the computation of cosine similarity, as performed by the similarity method. This method computes the cosine similarity between the vector of a given word and the cluster's centroid. Cosine similarity is a measure of how similar a word is to the average context of the cluster, and this computation is executed using ND4J's Transforms.cosineSim method. Such a measure is particularly useful in applications like topic modeling or semantic searching, where understanding the relationship between words and their contextual groups is important.

1. **Public Abstract Class StopWords**

We move onto the abstract class StopWords, designed to streamline the retrieval of a list of stop words within text processing applications. This data type serves as a tool for obtaining words that are frequently encountered in text but generally hold minimal meaningful content, such as "and", "the", and "is". These are often filtered out in various analyses and natural language processing tasks to focus on more substantive elements of the text.

The class houses the static method getStopWords, which enables direct invocation of the class rather than requiring an object instance. The method's primary function is to deliver a collection of stop words in the form of a List<String>. To compile this list, getStopWords employs another static method from the PatternRecognizer class, namely readObjectsFromFile. The latter is fed with a path to a text file—specifically "src/main/resources/stopwords.txt"—which presumably contains the needed stop words, each isolated on a line or separated by commas.

This utilization of PatternRecognizer.readObjectsFromFile is a key step, as it processes the "stopwords.txt" file located in the filesystem, returning the processed content as a list. This list, in turn, becomes the output of the getStopWords method.

The StopWords class is abstract, but despite not having any abstract methods, this designation prevents the instantiation of the class, which aligns with its purpose as it solely exists to provide a static utility method without necessitating any object state or behavior.

Additionally, the getStopWords method depends on a valid path to the stopwords file, and its operation assumes the file's presence in the specified directory. This implies that the calling code should be prepared to handle potential exceptions stemming from file access errors.

1. **Public Class DrawingPanel**

Subsequently, the DrawingPanel, which is an extension of Java Swing’s built-in JPanel class, is adept at handling the drawing of variously colored shapes on the panel, showcasing its utility in graphical user interface (GUI) applications where dynamic and interactive drawing is a key feature.

The DrawingPanel class is equipped with several fields that are central to its functionality. It includes a Color field for storing the color to be used in drawing, and a String field shapeTypeName for storing the name of the shape type to be drawn. Additionally, it holds a Shape object that represents the specific shape to be rendered on the panel, and a currentColor field that keeps track of the current color to be used for drawing.

In terms of initialization, the constructor of the DrawingPanel class sets up the panel with a default color (white) and initializes the shape type name as empty. This setup is crucial for preparing the panel for subsequent drawing operations.

One of the key features of the DrawingPanel class is its setter methods. The setDrawingColor(Color color) method allows for setting the drawing color, catering to the need for color customization in drawing. More interestingly, the setDrawingShape(String targetClass) method sets the drawing shape based on a provided class name. This method utilizes Java Reflection (specifically Class.forName) to dynamically create an instance of the specified shape type. It then calls the constructor of the identified Shape class, passing in the current color. After setting the shape, the method resets the color to white and invokes repaint to refresh the panel, ensuring that the new drawing is rendered correctly.

The DrawingPanel class also overrides the paintComponent method from JPanel. This override is essential for custom rendering behavior. It calls super.paintComponent(g) to ensure standard panel rendering, and if a shape object is present, it invokes the draw method of the Shape object, thereby rendering the shape on the panel.

The primary focus of the DrawingPanel class is to facilitate the drawing of shapes with the selected colors. It allows for the dynamic instantiation of shape objects based on class names, demonstrating the application's capability to handle a variety of shapes defined in separate classes. The use of reflection for creating shape instances provides notable flexibility, enabling the class to instantiate different types of shapes at runtime based on their class names.

From a design and usage perspective, by extending JPanel, the DrawingPanel class becomes a custom panel that can be seamlessly integrated into Swing-based GUI applications. The incorporation of reflection for object creation, though powerful, introduces complexity and necessitates careful handling to ensure security and proper functioning. The class's approach to managing color states for drawing shapes, including resetting the color to white after drawing, reflects a specific design choice that might align with the requirements of the application it is designed for.

1. **Public Class ColorCodeGenerator**

Following DrawingPanel, ColorCodeGenerator serves a specific function: it translates the names of colors into their corresponding RGB color codes. This utility allows users to retrieve the red, green, and blue values associated with common color names, a function that is particularly useful in various applications where color identification is essential.

This data type is structured with three public integer fields—redValue, greenValue, blueValue—which are used to store the individual RGB components of a color. These fields are designed to be readily accessible, ensuring that once a color name is converted to its RGB code, each color component can be easily retrieved and used.

A key feature of the ColorCodeGenerator is the private static colorCode string array, which holds the RGB values as strings. This array works in conjunction with the class's constructor and a static method named generateColorCode. When the constructor is invoked with a specific color name, it calls upon this static method to ascertain the RGB string associated with that name. This method makes use of a HashMap where color names are keyed to their RGB strings, ensuring a quick lookup. Once the RGB code is obtained for the given color name, the constructor parses this code into the individual RGB components and assigns them to their respective fields.

The generateColorCode method, being static, can be utilized independently of class instances. It operates by accessing the predefined HashMap of color names to RGB codes, splitting the found RGB string into separate values, and then returning them as a string array.

Overall, ColorCodeGenerator offers a straightforward way to map color names to their RGB equivalents, allowing for the simple and efficient transformation of color information within software applications.

1. **Public Abstract Class Shape**

The Shape class is an abstract class, serving as a foundational template for various shape subclasses. As such, it establishes a common structure for these shapes but does not allow direct instantiation. The placement of the class in a package dedicated to shapes indicates its role in a larger application focused on graphical representations of different shapes.

A key feature of the Shape class is its protected color field. This field stores the color attribute of a shape and, being protected, is accessible within the class itself and its subclasses. This accessibility allows for a consistent yet flexible approach to color management across various shape types.

The constructor of the Shape class, public Shape(Color color), plays a crucial role in initializing shape objects with a specified color. This design choice ensures that each instance of a shape, created by any subclass, can have its own distinct color, adding to the versatility of the shape's representation.

An essential component of the Shape class is the public Color getColor method. This public getter method enables other classes within the application to access the color of a shape. Such accessibility is important for operations that need to query or utilize the color attribute of shape objects.

Moreover, the class includes an abstract method, public abstract void draw(Graphics g). This method, which must be implemented by all subclasses, is designed to handle the drawing of the shape within a given Graphics context. By making this method abstract, the Shape class enforces that each specific shape, such as a circle or rectangle, provides its own unique implementation for drawing. This requirement is a clear application of the principles of abstraction and polymorphism, ensuring that the drawing logic is appropriately tailored to the specifics of each shape.

In the context of a graphics application, the Shape class is particularly well-suited. It provides a standardized structure and contract for a variety of shapes, thereby facilitating the management and rendering of these shapes in a consistent and organized manner. Subclasses that extend the Shape class are expected to implement specific behaviors and drawing mechanisms for each shape type. This architecture not only simplifies the handling of different shapes in a graphics application but also promotes code reusability and clarity, as each shape encapsulates its own drawing logic while adhering to a common interface defined by the Shape class.

1. **Public Class Circle**

The Circle class is a concrete implementation that extends the abstract Shape class. By extending Shape, the Circle class inherits its properties and methods, effectively tailoring them to represent a specific geometric figure – a circle.

In its constructor, the Circle class calls the constructor of its superclass, Shape, passing along the color parameter. This approach ensures that each instance of Circle is initialized with a specific color, a fundamental attribute inherited from the Shape class. This initialization is crucial as it embeds the circle with its color property, integral to its graphical representation.

A significant aspect of the Circle class is its method override, particularly the draw method which it inherits from Shape. In overriding this method, Circle provides its unique implementation for rendering a circle on a graphical interface. It utilizes the Graphics object g to perform the drawing operation. The method sets the color of the Graphics object to match the circle's color and proceeds to draw the circle at a fixed position (x: 100, y: 25) with a fixed size (width: 100, height: 100). The equal width and height dimensions ensure that the shape rendered is a perfect circle. This override is a clear demonstration of how Circle adapts the generic drawing logic of Shape to suit its particular needs.

1. **Public Class Triangle**

The Triangle type is a concrete implementation that extends the abstract Shape class, inheriting its foundational properties and methods. This class specifically represents the shape of a triangle, adding to the variety of shapes that can be rendered in a graphics application.

Central to the Triangle class is its constructor, which is designed to call the constructor of the superclass, Shape, with a color parameter. This approach ensures that each Triangle instance is initialized with a specific color, adhering to the color attribute defined in the Shape class. Such initialization is crucial for the graphical representation of the triangle, as it establishes the color with which the shape will be rendered.

In thе overridden draw method, the class uses a Graphics object g to draw the triangle. It sets the color of the Graphics object to match the triangle's color, ensuring consistency in its visual presentation. The method then defines the coordinates for the vertices of the triangle (x1, y1, x2, y2, x3, y3) and uses these coordinates to draw a filled triangle on the panel with the g.fillPolygon method. This override demonstrates how the Triangle class provides its specific drawing logic, tailoring the generic drawing functionality of Shape to suit the needs of rendering a triangle.

1. **Public Class Square**

The Square data type is specifically tailored to render a square shape with a given color. This class plays a pivotal role in drawing square shapes, encapsulating the necessary properties and methods to achieve this.

One of the significant features of the Square class is its draw method. In this method, the class employs a Graphics object, referred to as g, to execute the drawing operation. The method first sets the drawing color to match the square's color, ensuring consistency in its appearance. Then, it proceeds to draw the square.

The drawing logic of the Square class involves defining the coordinates (x, y) for the top-left corner of the square and its size, which in this case is hard-coded to be 100x100 pixels. This fixed size ensures that all sides of the square are equal, forming a perfect square. The square is then drawn on the panel using the fillRect method on the Graphics object g, effectively filling the square with the specified color. This approach to drawing uses basic Graphics methods, making the rendering of the square straightforward yet effective.

1. **Public Class Rectangle**

The Rectangle aims to draw rectangle shapes with the designated colors. This class effectively integrates the foundational properties and methods of Shape to cater to the requirements of rendering rectangles.

In the draw method, the class utilizes a Graphics2D object, referred to as g2d, to perform the drawing operation. The method sets the drawing color to align with the rectangle's color, establishing consistency in its visual presentation. Subsequently, the rectangle's position (x, y) and dimensions (width, height) are hard-coded, providing the necessary specifications for the shape. A Rectangle2D.Double object is then created to represent the rectangle accurately. The rectangle is finally drawn and filled with the specified color using the g2d.fill method. The use of Rectangle2D.Double ensures precision in rendering the shape, making it an effective tool for graphical representation.

1. **Public Class Arrow**

The Arrow class is distinctly crafted to draw an arrow shape with a specified color. This class encapsulates the necessary functionalities and attributes to represent and render an arrow in a graphical context.

One of the salient features of the Arrow class is its draw method. In this method, the class employs a Graphics object, designated as g, to execute the drawing of the arrow. The method begins by setting the drawing color to align with the arrow's color, ensuring visual consistency. It then proceeds to draw the two distinct parts of the arrow: the shaft and the arrowhead.

The drawing process for the arrow shaft involves representing it as a horizontal rectangle. The position and size of the shaft are defined by specifying its starting coordinates (startX, startY), length (shaftLength), and width (shaftWidth). The g.fillRect method is used to render the shaft on the panel, effectively creating the base of the arrow.

The arrowhead, designed to be wider than the shaft for visual balance, involves more complex calculations. The dimensions of the arrowhead, including its width (arrowheadWidth) and length (arrowheadLength), are established. The coordinates for the triangle that forms the arrowhead are meticulously calculated to position the arrowhead correctly at the right end of the shaft. The arrowhead is then drawn using the g.fillPolygon method with the calculated xPoints and yPoints, completing the arrow's representation.

The Arrow class serves as a straightforward yet effective means to represent and draw an arrow in graphical applications. It provides the logic for rendering both the shaft and the arrowhead of the arrow, ensuring that the size of the arrowhead is proportionate to the width of the shaft for a visually balanced appearance. The positioning of the arrowhead is carefully aligned with the shaft, contributing to the overall aesthetic of the arrow.

1. **Public Class Crescent**

The goal of the Crescent class is to draw a crescent shape. A key feature of the Crescent class is its draw method. Here, the class sets the drawing color to align with the crescent's color, establishing consistency in its visual presentation. The crescent itself is crafted by overlapping two circles and subtracting the smaller circle from the larger one, creating the distinct crescent shape.

The process of drawing the crescent involves creating two instances of Ellipse2D.Double. The larger circle is created with hard-coded dimensions (diameter1) and positioned at coordinates (x1, y1). In contrast, the smaller circle is slightly offset from the larger one, sharing the same diameter but positioned differently at coordinates (x2, y2). This arrangement of circles, with one partially covering the other, forms the basis of the crescent shape.

To actualize the crescent shape, an Area object is created by subtracting the area of the smaller circle from the larger circle. This complex shape manipulation, made possible through the subtraction operation in the Area object, is ideal for creating the crescent form. The crescent is then filled with the specified color using the g2d.fill method, resulting in a precisely rendered crescent shape.

1. **Public Class Cross**

The Cross data type is an extension of Shape, specifically designed to render a cross shape with a designated color. This class encapsulates the necessary functionalities and attributes to represent and draw a cross in a graphical context.

The class utilizes a Graphics2D object to execute the drawing of the cross. The method sets the drawing color to align with the cross's color, establishing consistency in its visual presentation. The cross itself is constructed using two rectangles—one vertical and one horizontal—that intersect at their centers.

The drawing process for the cross involves creating two instances of Rectangle2D.Double. The vertical rectangle is created with a specified armWidth and armLength, centered at coordinates (centerX, centerY). Similarly, the horizontal rectangle is also centered at the same coordinates but with reversed dimensions (armLength as width and armWidth as height). Both rectangles are then filled with the specified color using the g2d.fill method, completing the representation of the cross.

In terms of design and usage, the Cross class offers a straightforward yet precise way to represent and draw a cross shape using basic geometric shapes. The utilization of Graphics2D for rendering the cross provides a high level of accuracy and quality in the shape's visual output.

1. **Public Class Diamond**

The Diamond class is crafted for drawing a diamond shape with a specific color. This class integrates the foundational properties and methods of Shape to cater to the requirements of rendering diamonds in a graphical context.

The draw method sets the drawing color to match the diamond's color. The diamond's structure is then crafted using hard-coded values for the center coordinates (centerX, centerY) and the half-width (halfWidth) of the diamond.

The drawing process of the diamond involves creating a Path2D.Double object to define the path of the diamond. The path is meticulously plotted, starting at the top vertex and moving sequentially to the right vertex, then to the bottom vertex, and finally to the left vertex, effectively forming the diamond shape. Once the path is closed, completing the shape, the diamond is filled with the specified color using the g2d.fill method. This approach to drawing uses the Path2D.Double object, which ensures precision and clarity in the diamond's shape.

1. **Public Class Heart**

The Heart class is a specialized extension of the Shape class, designed to render a heart shape with a specific color. This class integrates unique methods and properties to represent and draw a heart in a graphical context, utilizing a combination of arcs and a triangle.

The heart's structure is crafted using hard-coded values for the position (x, y) and dimensions (width, height) of the bounding rectangle of the heart.

The drawing process for the heart involves creating two arcs, one for each half of the top of the heart, using Arc2D.Double. These arcs are defined within the bounding rectangle dimensions, with their widths and heights set to half of the bounding rectangle's width and height. Additionally, a triangle is drawn at the bottom part of the heart using Path2D.Double. The triangle's points are strategically placed, starting from the center of each arc and meeting at the bottom of the bounding rectangle.

The Path2D.Double path is constructed by appending the left and right arcs and then drawing the triangle part. This path is then closed to complete the heart shape. Finally, the heart shape is filled with the specified color using g2d.fill. This approach to drawing combines the arcs and triangle seamlessly, forming the recognizable heart shape.

1. **Public Class Parallelogram**

The Parallelogram class is a specialized extension of the Shape class, created to render a parallelogram shape with a specific color. It embodies the functionalities and attributes necessary for representing and drawing a parallelogram in a graphical setting.

The parallelogram's structure is crafted using hard-coded values for the coordinates of its four corners (x1, y1, x2, y2, x3, y3, x4, y4).

The drawing process for the parallelogram involves creating a Path2D.Double object to define its path. The path is meticulously plotted, starting at the first corner and moving sequentially to the second, third, and fourth corners, effectively forming the shape of a parallelogram. Once the path is closed, completing the shape, the parallelogram is filled with the specified color using the g2d.fill method. This approach to drawing uses the Path2D.Double object, which ensures precision and clarity in the parallelogram's shape.

1. **Public Class Rhombus**

The Rhombus class renders a rhombus shape in a specified color. This class encompasses the necessary features and capabilities essential for depicting a rhombus in various graphical contexts.

The construction of the rhombus shape involves using hard-coded values to define the center coordinates (centerX, centerY) and the dimensions (width, height) of the rhombus. The corner points of the rhombus are then calculated based on these central coordinates and dimensions.

For the actual drawing process, a Path2D.Double object is created to outline the path of the rhombus. The path is intricately designed, starting from the top corner and sequentially progressing to the right, bottom, and left corners, effectively forming the rhombus shape. Upon completing the path, the rhombus shape is filled with the designated color using the g2d.fill method. This methodical approach, leveraging the Path2D.Double object, ensures precision and clarity in the rhombus's formation.

1. **Public Class Star**

The Star class is an extension of the Shape class, specifically designed to represent and render a star shape. A vital feature of the Star class is its implementation of the draw method. The method employs Graphics2D to render the star. The creation of the star relies on hard-coded values for its center coordinates, outer radius (radius), and inner radius (innerRadius). The star's shape is intricately crafted using a Path2D.Double object. This process involves adding points in a loop, where the points are calculated using trigonometric functions. The alternation between the outer and inner radii during this loop is what gives the star its distinct shape. Once the path is closed, the star is filled with the predetermined color, completing its visual formation.

1. **Public Class Trapezoid**

The Trapezoid data type extends the Shape class and is designed to facilitate the rendering of a trapezoid shape in a specified color. This class incorporates specific functionalities and attributes necessary for portraying a trapezoid in various graphical contexts.

The class features an overridden draw method, employing a Graphics2D object to execute the drawing of the trapezoid. The drawing color is set to align with the trapezoid's color, establishing a consistent and appealing visual display. The construction of the trapezoid involves using hard-coded values to define the coordinates of its four vertices (x1, y1, x2, y2, x3, y3, x4, y4), which determine the trapezoid's shape.

For the actual drawing process, a Path2D.Double object is created to outline the path of the trapezoid. The path is intricately plotted, beginning at the top left vertex and proceeding sequentially to the top right, bottom right, and bottom left vertices, effectively forming the trapezoid shape. Upon completing the path, the trapezoid shape is filled with the predetermined color using the g2d.fill method. This methodical approach, leveraging the Path2D.Double object, ensures precision and clarity in the trapezoid's formation.

1. **Public Class Cone**

The Cone class is uniquely designed to render a cone shape in a specific color. This class integrates specialized functionalities and attributes, making it ideal for portraying a cone in various graphical contexts.

The base of the cone, an oval shape, is rendered using Ellipse2D.Double. The coordinates (baseCenterX, baseCenterY) and dimensions (baseWidth, baseHeight) for the oval base are defined, and the base is positioned and filled with the specified color.

In addition to the base, the sides of the cone are drawn using a Path2D.Double object. The path initiates from the left edge of the base, extends to the apex of the cone (apexY), and then connects to the right edge of the base, forming a triangular shape. These sides of the cone are then filled with color, completing the cone's structure.

Moreover, the base of the cone is depicted as an ellipse, with its size and position controlled by the specified variables. The sides of the cone are represented as a triangle that connects the base's edges to the apex, ensuring accurate alignment with the base.

1. **Public Class Cube**

The Cube data type draws a cube shape in a specified color, providing a simulated 3D effect in a 2D space. This class embodies specific functionalities and attributes crucial for creating a cube in various graphical scenarios.

Employing a Graphics2D object, the draw method depicts the cube’s structure using hard-coded values for its starting position (x, y) and size (size). Additionally, a depth projection factor is calculated to simulate a 3D depth effect, giving the cube a sense of perspective.

1. **Public Class Cylinder**

The Cylinder class depicts a cylinder shape in a chosen color. This class incorporates specialized functionalities and attributes essential for creating a cylinder in various graphical contexts.

The cylinder's structure is depicted using hard-coded values for its position (x, y) and dimensions (width, height). Additionally, a GradientPaint is employed to create a 3D effect on the cylinder, with the gradient transitioning from a brighter version of the color at the top to a darker version at the bottom.

In the drawing process, the top and bottom bases of the cylinder are rendered using Ellipse2D.Double. The top ellipse is positioned at the top and filled with a brighter version of the color, while the bottom ellipse is positioned at the bottom and filled with a darker version. The sides of the cylinder are drawn with lines that connect the top and bottom ellipses, using the gradient paint to enhance the 3D effect.

1. **Public Class Prism**

The Prism data type depicts a prism shape in a chosen color, offering a simulated 3D effect within a 2D space. The class utilizes Path2D.Double for its drawing process, showcasing its capability to handle complex geometric forms.

A notable aspect of the Prism class is its adaptation of the draw method. The prism's structure is portrayed using specific hard-coded values for its coordinates (x, y), base width (baseWidth), height (height), top offset (topOffset), and depth (depth).

In the drawing process, the front face of the prism is depicted as a parallelogram, beginning from the bottom-left corner and progressing clockwise. The rear face is similarly illustrated as a parallelogram, offset by the depth factor to simulate the 3D effect. Lines are then drawn to connect the corresponding corners of the front and rear faces, effectively completing the 3D representation of the prism. The path for the prism is created using Path2D.Double, and the shape is outlined using g2d.draw.

The drawing logic of the Prism class incorporates depth and offset to simulate a 3D effect, creating two parallelograms (front and rear faces) and connecting them with lines. The top and bottom edges of the front face are horizontally offset, producing an illusion of perspective.

1. **Public Class Pyramid**

The Pyramid class attempts to portray a pyramid shape in a chosen color. This class is distinct for its ability to represent a pyramid in a 2D space while simulating a 3D effect, primarily utilizing Path2D.Double for its drawing process.

The pyramid's structure is delineated using adjusted values for its position (x, y), dimensions (baseWidth, baseHeight, slant, pyramidHeight), and the tip coordinates (tipX, tipY), calculated from the base dimensions.

In the drawing process, the base of the pyramid is depicted as a parallelogram, crafted using Path2D.Double. The sides of the pyramid are illustrated by connecting the tip to each corner of the base parallelogram. The path of the pyramid is then established, and the shape is outlined using g2d.draw.

The drawing logic of the Pyramid class incorporates a parallelogram base and a raised tip to simulate a 3D effect, creating an illusion of a pyramid extending upwards. The pyramid's sides are represented as triangles that connect the base to the tip.

1. **Public Class Sphere**

The Sphere class renders a sphere shape, incorporating a specific color and an impressive simulated 3D effect. This class integrates sophisticated drawing techniques to achieve its visual goals.

The Sphere’s draw method not only sets the drawing color but also carefully configures the gradient to be used for the sphere, playing a crucial role in the sphere's visual representation.

In drawing the sphere, the dimensions and position are set using hard-coded values, specifically the center coordinates (centerX, centerY) and the radius (radius) of the sphere. A pivotal feature in creating the 3D effect is the use of RadialGradientPaint. This technique varies the color from a brighter shade at the center to a darker one towards the edges, effectively mimicking the way light interacts with a spherical surface and creating an illusion of depth and curvature. The sphere itself is represented by an Ellipse2D.Double object. Given the nature of a sphere, the width and height of this ellipse are equal, each being twice the radius, ensuring a perfect circular shape. The sphere is then filled with this gradient paint using g2d.fill.

1. **Public Class Ellipse**

The Ellipse data type effectively captures the essence of an ellipse shape. This concrete class enjoys the inherited properties and methods of its parent Shape, which forms the foundation of its functionality.

Within the draw method, the Graphics object is cast to Graphics2D, a necessary step to leverage the more advanced graphics capabilities provided by Graphics2D. This includes the ability to draw complex shapes like ellipses with greater precision and flexibility. The method then sets the drawing color for the ellipse and defines fixed coordinates and dimensions, which represent the bounding rectangle for the ellipse. The final stroke of this method is the drawing of a filled ellipse.

1. **Public Class Oval**

The Oval class is a concrete representation that extends the Ellipse class, seamlessly inheriting all its properties and methods. This class is tailored to represent an oval shape, which is a specialized form of an ellipse.

At the core of the Oval class is its constructor. This constructor thoughtfully delegates to the superclass, Ellipse, with a color parameter. This approach ensures that each instance of Oval is depicted with a designated color upon initialization. The simplicity of this constructor reflects the class's direct reliance on the functionality provided by its parent class.

One of the defining aspects of the Oval class is its inherent inheritance from Ellipse. It does not introduce any new methods or fields; rather, it fully embraces all the behaviors encapsulated within Ellipse. This includes the crucial draw method, which implies that an Oval object, when rendered, will follow the same drawing logic as an Ellipse.

1. **Public Class Polygon**

The Polygon class includes properties for the radius and the number of sides, which are instrumental in defining the size and specific form of the polygon.

Integral to the Polygon class are its setter methods: setRadius(int radius) and setSides(int sides). These methods are pivotal in defining the polygon's characteristics. setRadius adjusts the radius of the polygon, directly influencing its size. Meanwhile, setSides determines the number of sides, shaping the polygon's form. Together, these methods offer substantial control over the polygon's appearance.

Leveraging the capabilities of Graphics2D, the draw method is adept at drawing the polygon with precision. It employs trigonometric functions to calculate the vertices of the polygon, ensuring that each side is equidistant from a central point (defined as centerX and centerY). This approach allows for the creation of regular polygons that are symmetrical and aesthetically pleasing.

1. **Public Class Pentagon**

The Pentagon class, as an extension of the Polygon class, inherits all the properties and functionalities of its parent class. As the name implies, this class is specifically tailored to represent a pentagon, which is a polygon with five sides.

The distinctiveness of the Pentagon class is highlighted by its setting of a default radius of 50 and a fixed number of sides at five. These hardcoded values ensure that each instance of the Pentagon class will maintain a consistent size and shape, though the color can vary across different instances.

1. **Public Class Hexagon**

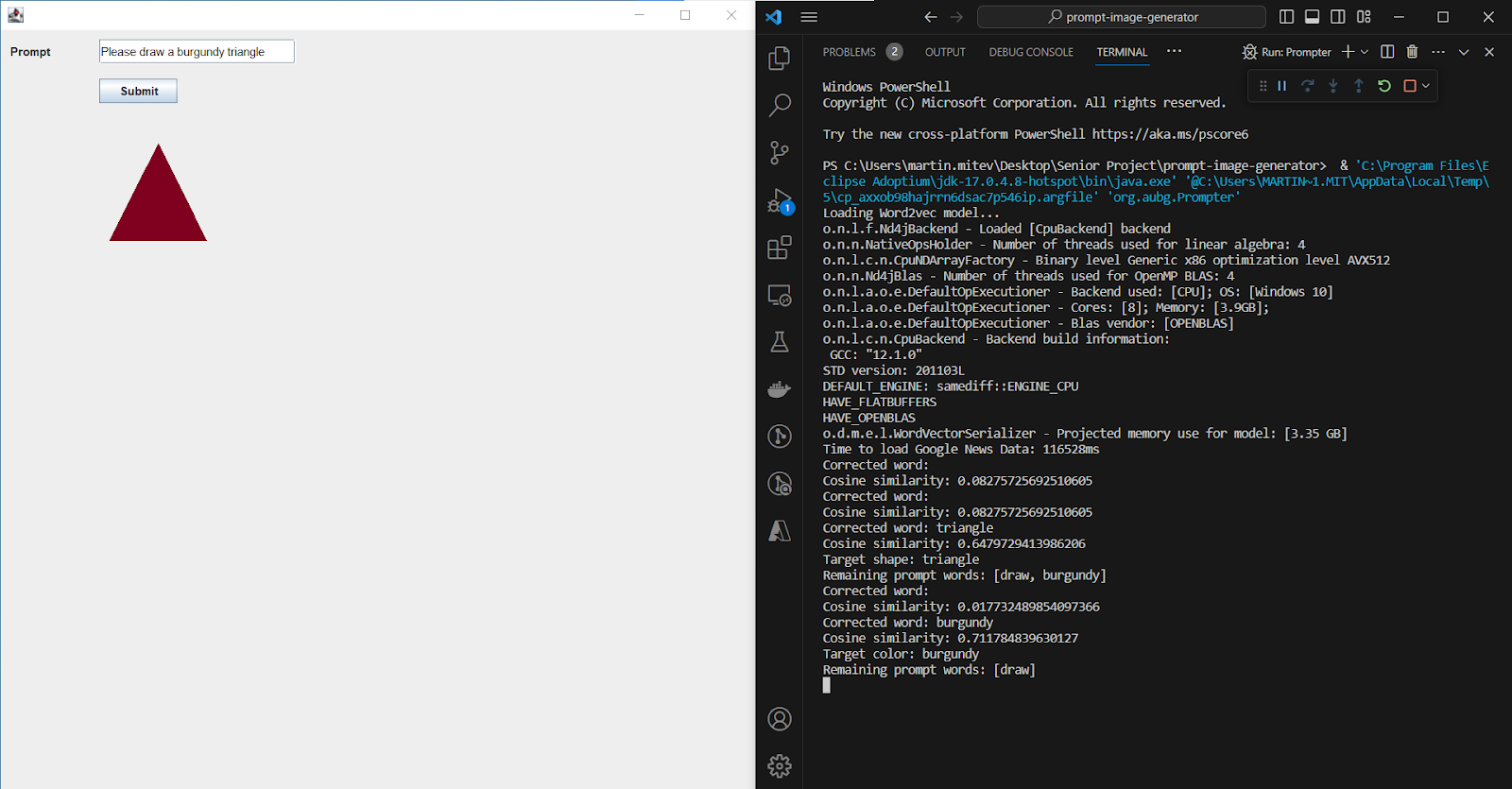
This class is tailored to represent a hexagon, a six-sided polygon, making it a specific and distinct shape within the polygon family. The hexagon data type invokes the Polygon superclass and sets the shape’s radius to 40 and its sides to six.

1. **Public Class Octagon**

The goal of the octagon class is to draw an eight-sided polygon. The constructor of the octagon data type invokes the Polygon superclass, then sets the shape’s radius to 40 and its sides to eight.

1. **Graphical User Interface demonstration and explanation**

The images below exhibit how the GUI works, alongside with the information outputted on the developer console at runtime:

Fig. 6

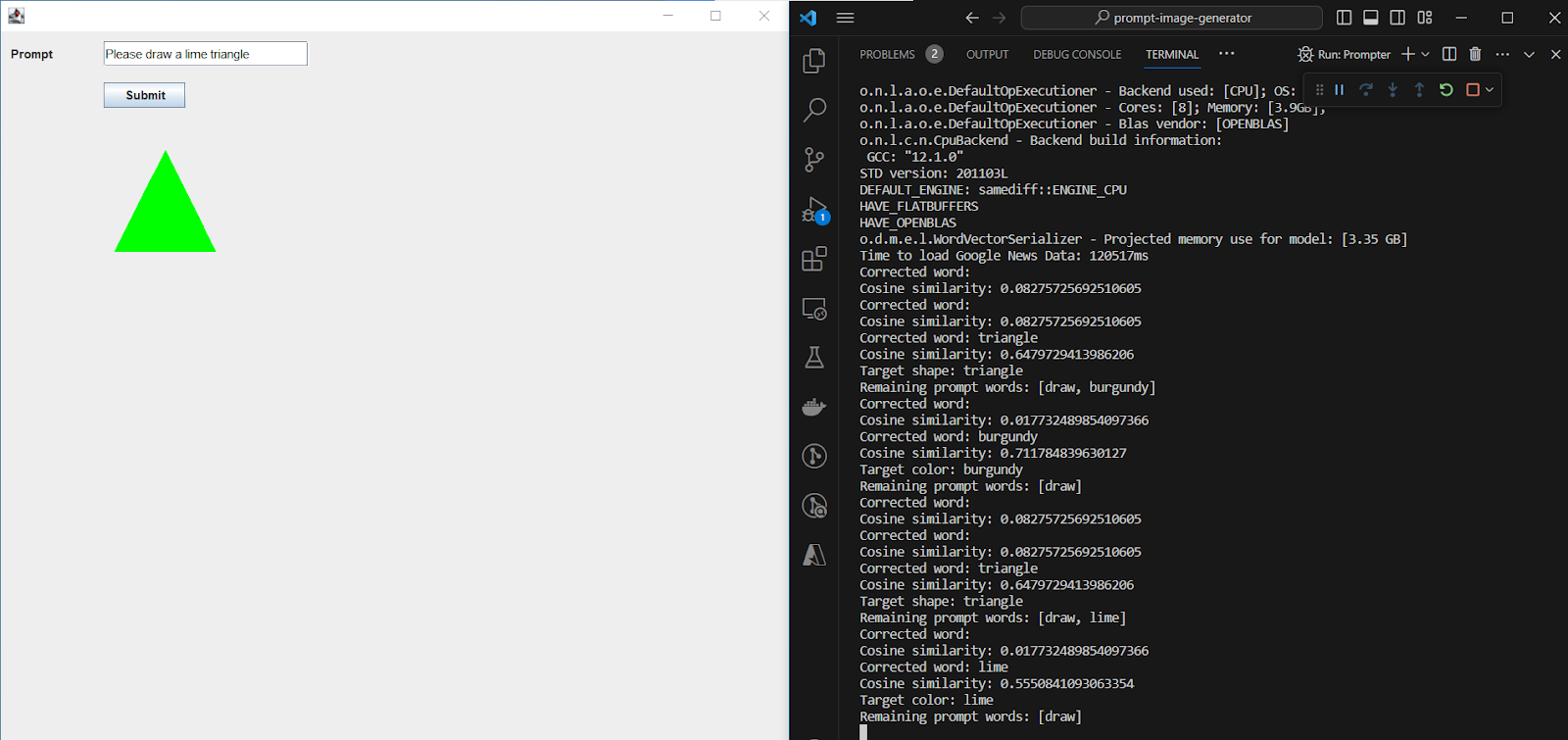
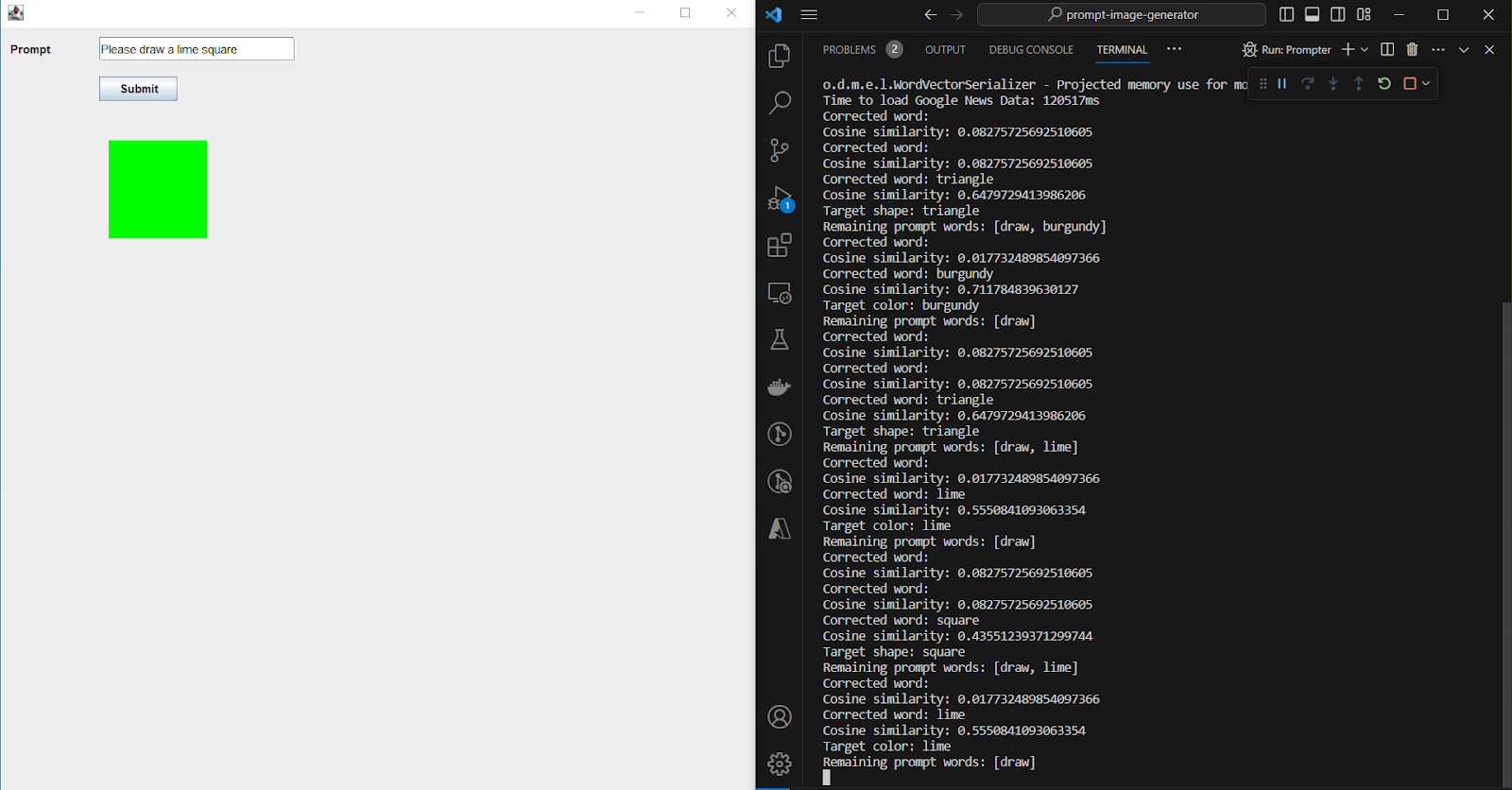
This figure showcases the Prompt Image Generator’s GUI, alongside the developer console displaying the information being logged during runtime. On the right-hand side, the logs illustrate that the loading of the Word2vec model has begun, outputting some information about the operating system, some data about the build, and a projected memory usage for the model, amounting to 3.35 GB. Considering the size of the Google News vector dictionary file is 1.53 GB, this significant memory usage is expected. Additionally, given the size of the vector file, the model’s loading time can take up to several minutes. This considerable lag necessitates thorough logging into the terminal in an attempt to guide the users, as well as help developers debug any potential problems further exacerbating the waiting time. Once the model loads, the GUI opens up and the program displays a message, reporting the number of milliseconds it took for the entire Google News dataset to be processed - 116528 ms in this case, or approximately two minutes. The user interface is quite simplistic, consisting only of a prompt field and a submit button. In this particular example, the user has requested that the program drew a burgundy triangle. The system responds instantaneously with a drawing of the requested shape. The terminal displays a series of corrected words, cosine similarity computations, target objects, and arrays of remaining prompt words. The next subsection explains in detail how the Levenshtein Distance algorithm and a custom variant of K-means Clustering find corrected words and calculate cosine similarities between prompt words and clusters of shapes and colors. The computeTargetObject in the PatternRecognizer class is responsible for outputting the target shape or color, removing prompt words that have already been targeted and printing out the ones remaining afterwards. 

Fig. 7

Figure 7 demonstrates a situation where the user has requested a change in the triangle’s color from burgundy to lime (refer to Figure 1). Once the request is submitted, the system goes through the process of figuring out the correct shape and color, finally portraying the desired drawing onto the panel. In the logs, we see the changes in the cosine similarity and target object values reflecting the result on the screen.

Fig. 8

The prompt in figure 8 only differs in the desired shape, a square rather than a triangle. Following the computations in the terminal, a lime square appears on the drawing panel as the user solicited.

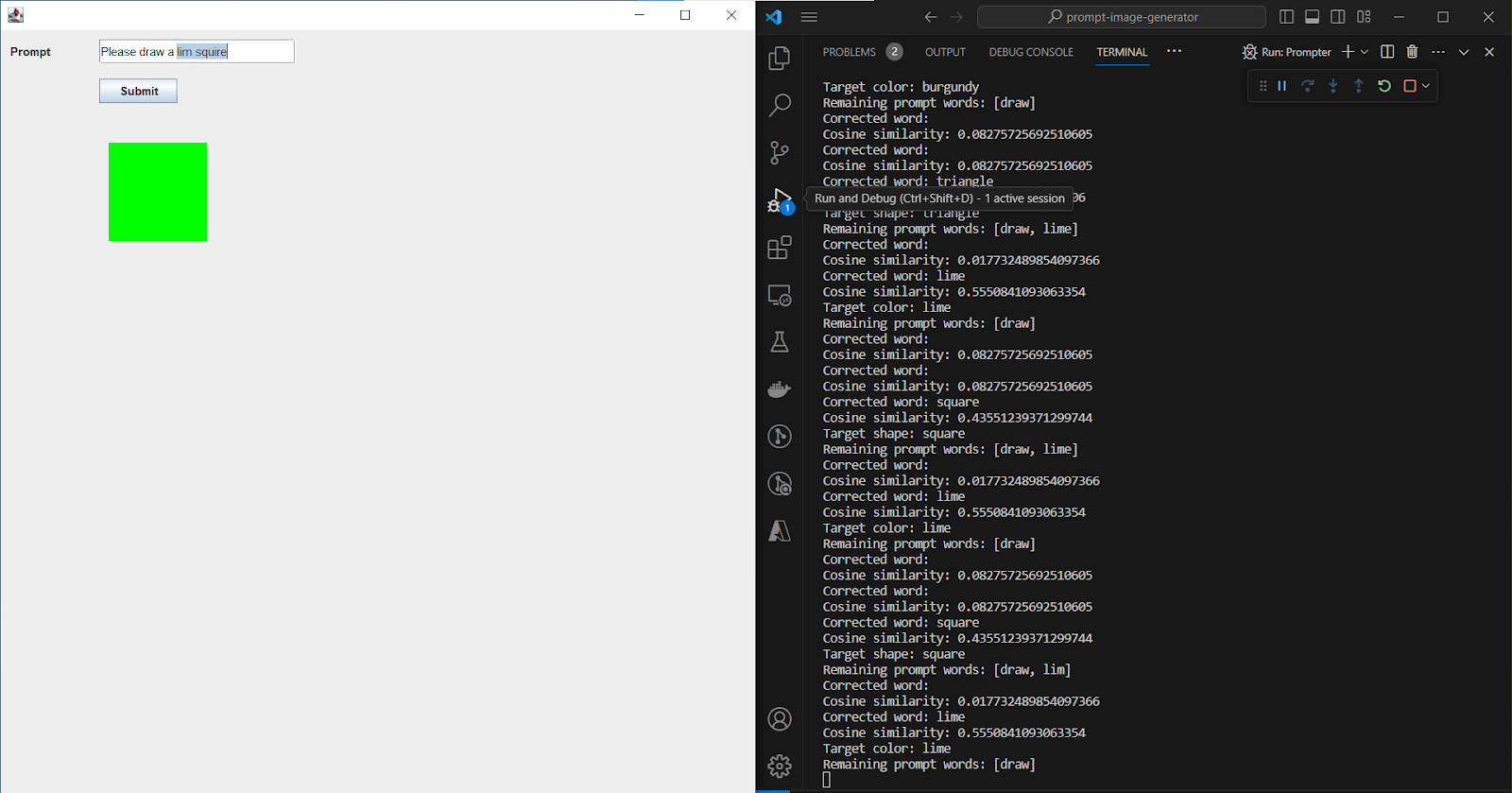
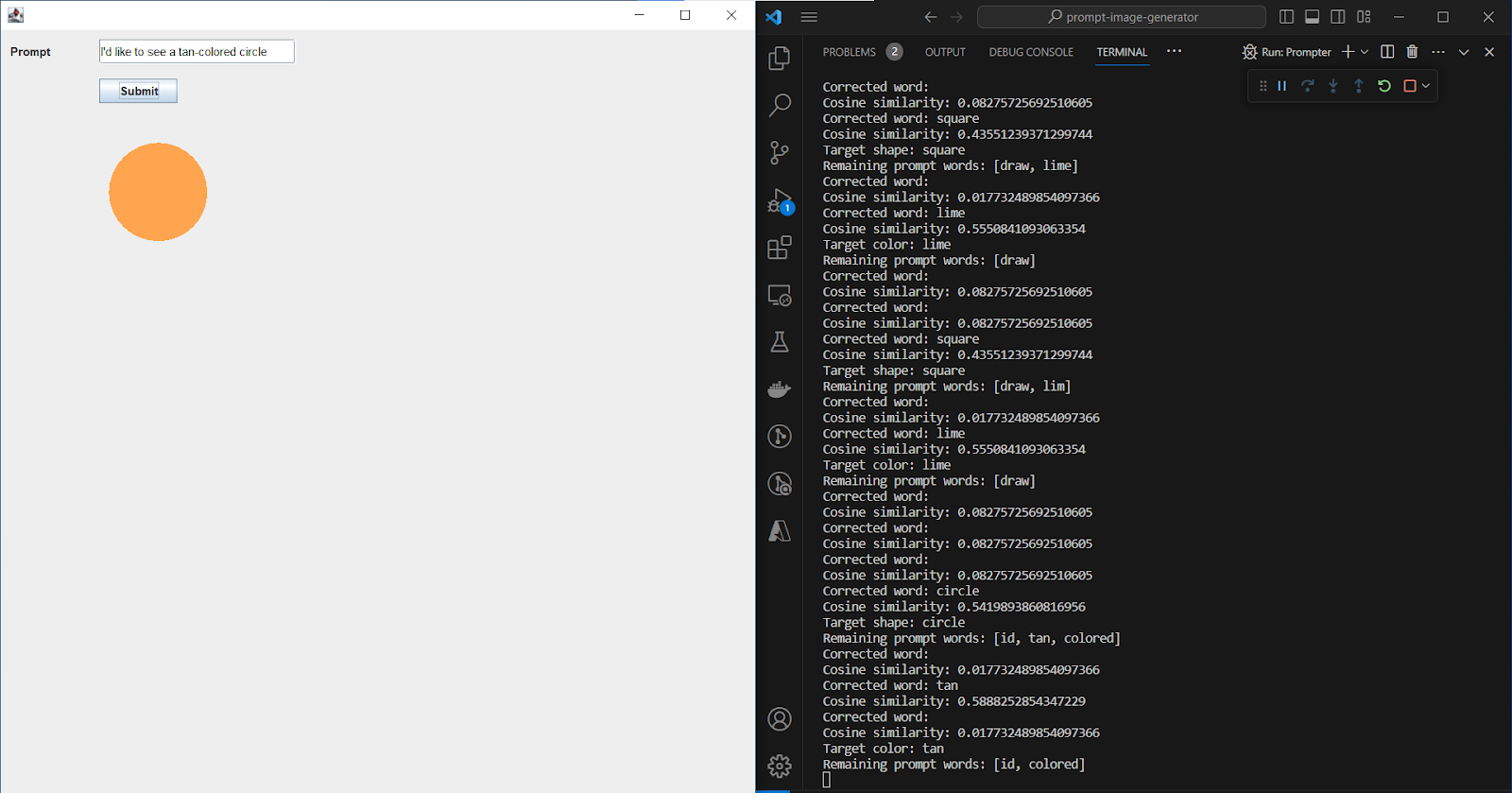
Fig. 9

Figure 9 highlights the program’s capability of correcting mistyped words with the assistance of the Levenshtein Distance technique that edits each prompt phrase up to a maximum of 2 edit distances. In this scenario, the user entered “lim squire” instead of “lime square”. After correcting the typographic errors in the prompt input, as evidenced by the result in the console, the algorithm accurately depicts the requested shape.

Fig. 10

The tenth figure represents two noteworthy alterations other than the adjustments in the demanded shape and color. Firstly, the user chose a distinct word order and diction that the one seen in the previous images. With that being said, it is clear the machine learning model can recognize colors and shapes regardless of how the prompt is structured. Secondly, the user entered the phrase “tan-colored” instead of specifying the color as just “tan”. It is likely that the AI would be unable to recognize “tan-colored” as a valid color since no such color exists in the vector dictionary. However, because separatePromptWords in the Prompter class removes non-alphanumeric characters, including slashes, hyphens, and underscores, the program treats “tan” and “colored’ as separate words, thus allowing the AI to correctly identify the color.

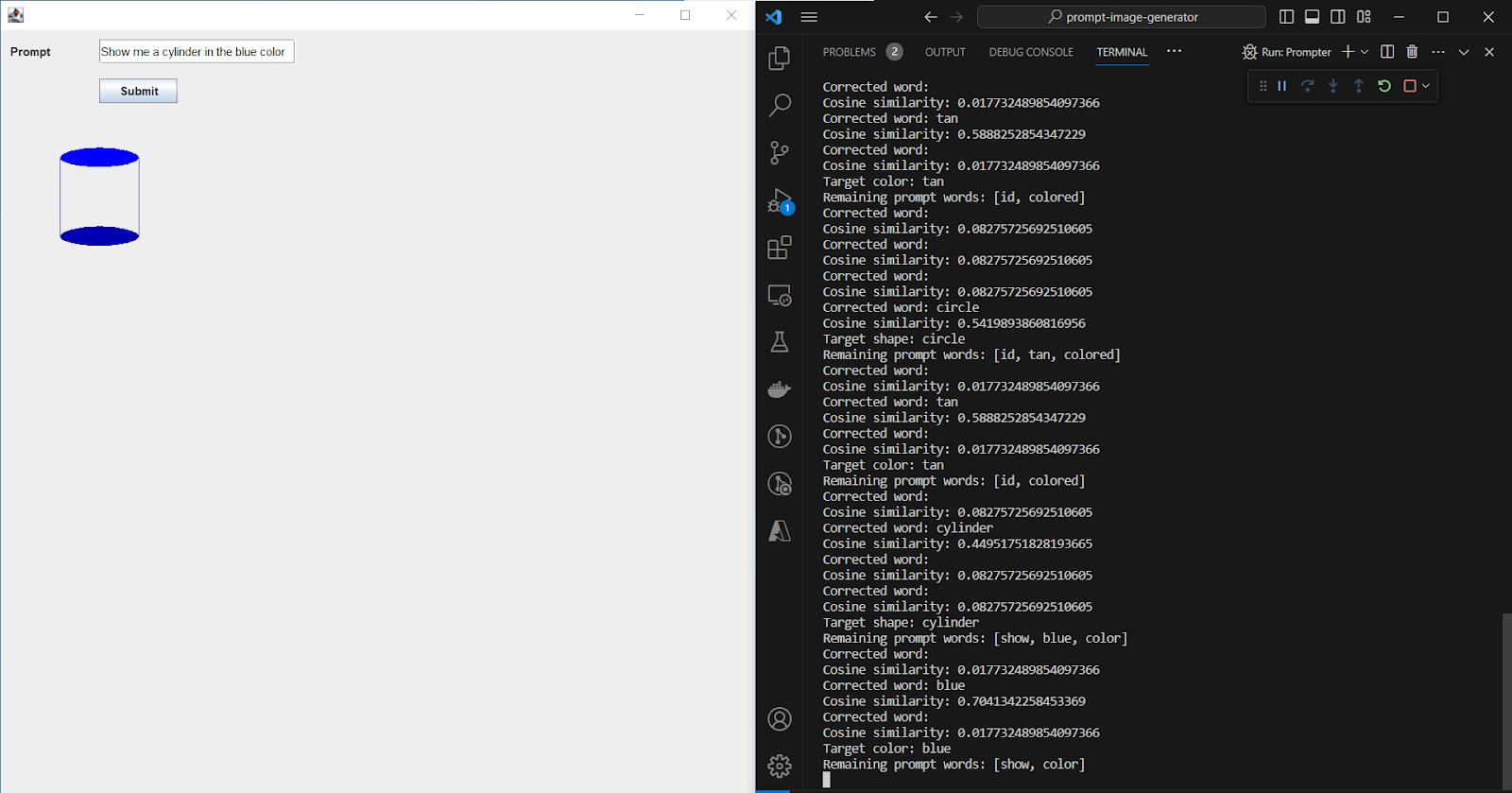
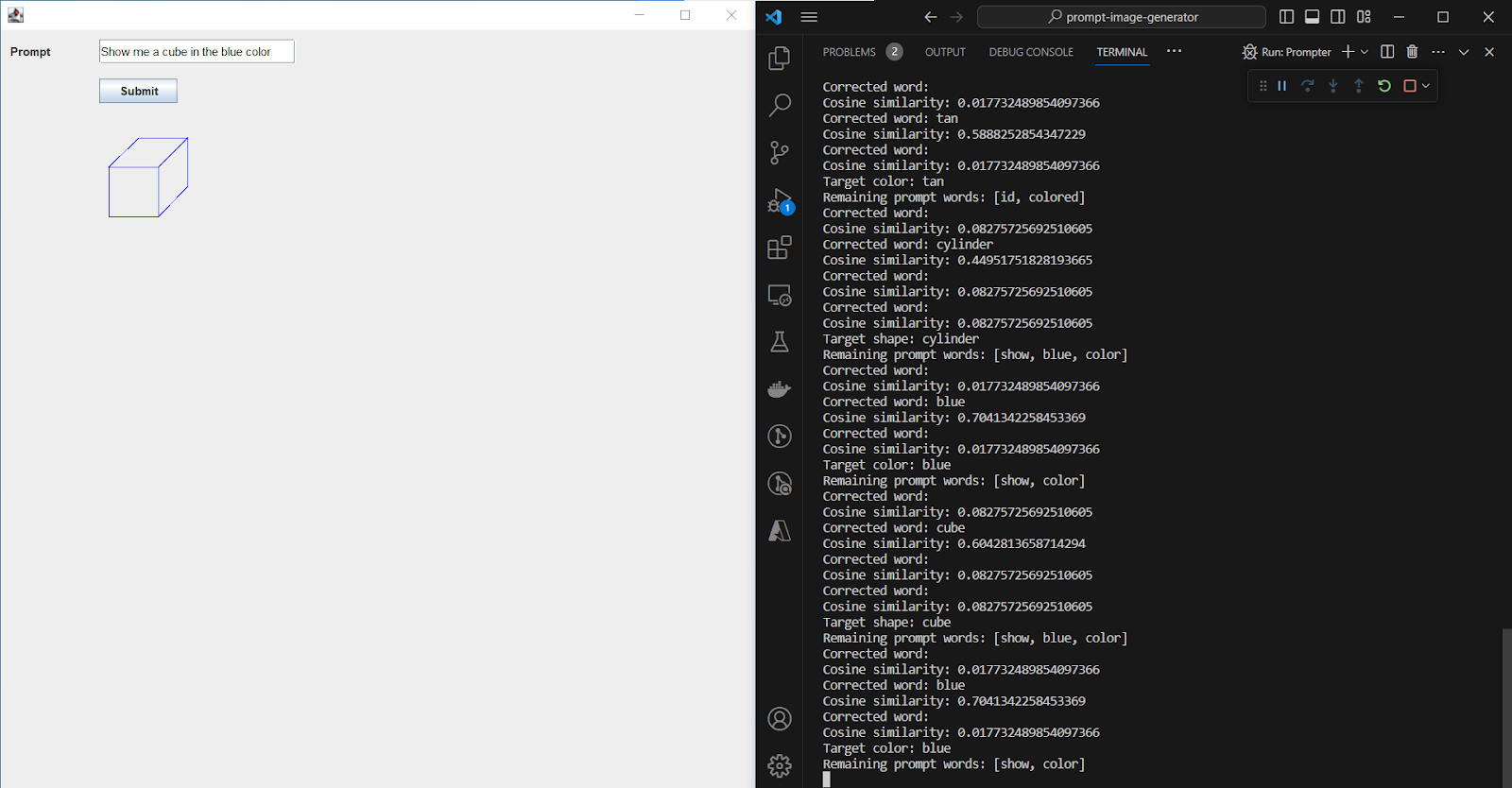
Fig. 11

Figure 11 aims to illustrate two important functionality of the natural language processor and the drawing panel:

1. The language processor is able to identify the requested shape and color despite a yet another alteration of the word order - “Show me a <shape> in the <color> color” rather than “I’d like to see a <color> <shape>”. This change in order indicates that the program works correctly independently of word order, or the shape appearing before the color in the sentence.
2. The drawing panel manages to represent three-dimensional figures in a two-dimensional space by using shading and drawing perpendicular or parallel lines. In this case, the panel applies shading to the bottom of the cylinder to create the impression of depth. The top is also connected to the bottom with two perpendicular lines.

Fig. 12

In figure 12, we see that the drawing panel employs perpendicular and parallel lines to display 3D shapes in situations where filling in the sides with color would make the shape incomprehensible on a 2D plane. This is the case for the cube, as well as for the prism and pyramid.

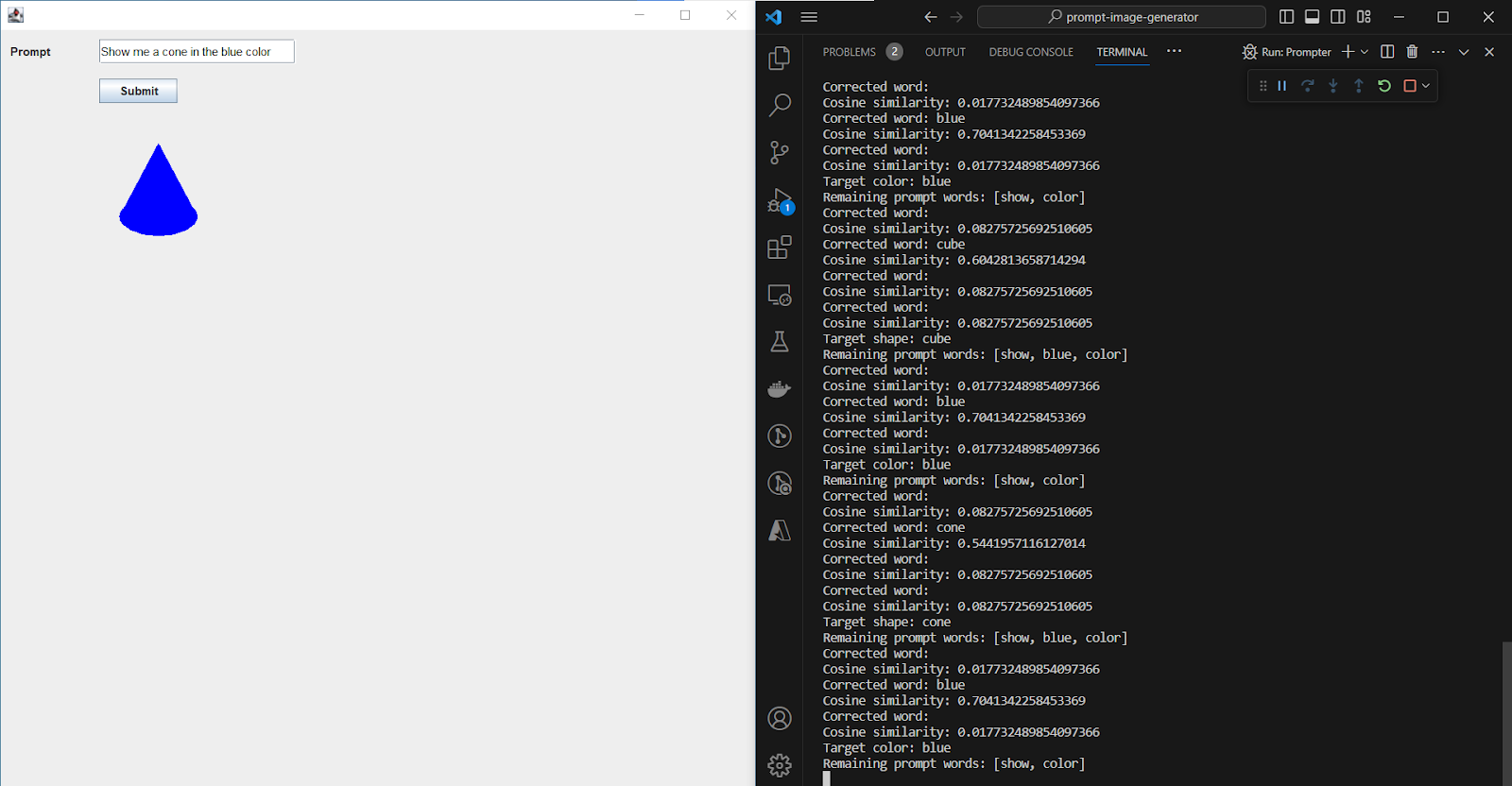
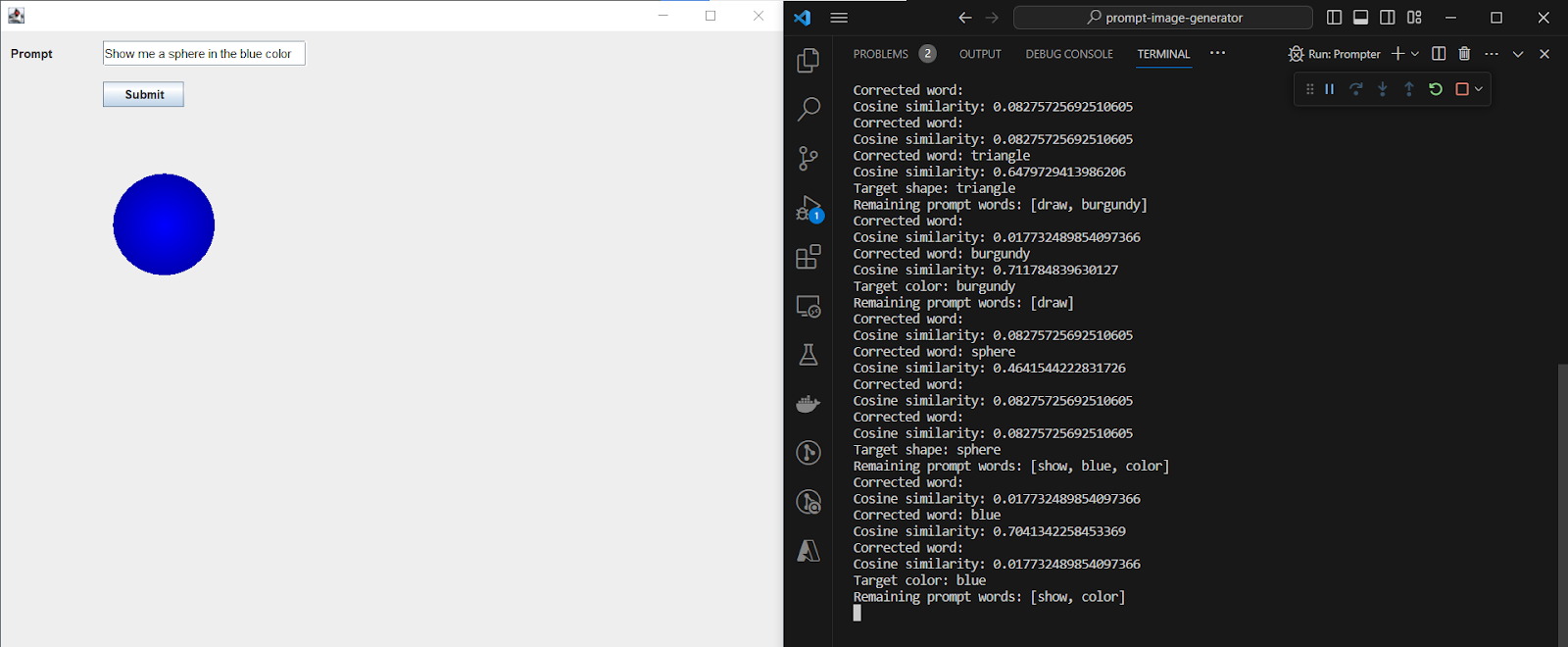
Fig. 13

Figure 13 represents a drawing of a cone. Here, the cone is made up of a triangle and an ellipse instead of being drawn with lines. I have selected this design for the cone since the combination of a triangle on top of an ellipse adequately creates the perception of a three-dimensional shape despite the solid texture of the sides. Shading the figure is inappropriate in the given scenario because it makes the triangle stand out from the ellipse, weakening the 3D effect, considering the figure no longer appears as a single structure.

Fig. 14

The purpose of figure 14 is to show how the drawing panel deals with shading in shapes like a sphere, where the only distinction from its 2D counterpart, a circle, is the vivid coloring and shading. Here, the 3D effect is achieved entirely through the application of a gradient across the circle’s area, giving off the impression of a concavity that makes the circle protrude above the surface of the panel.

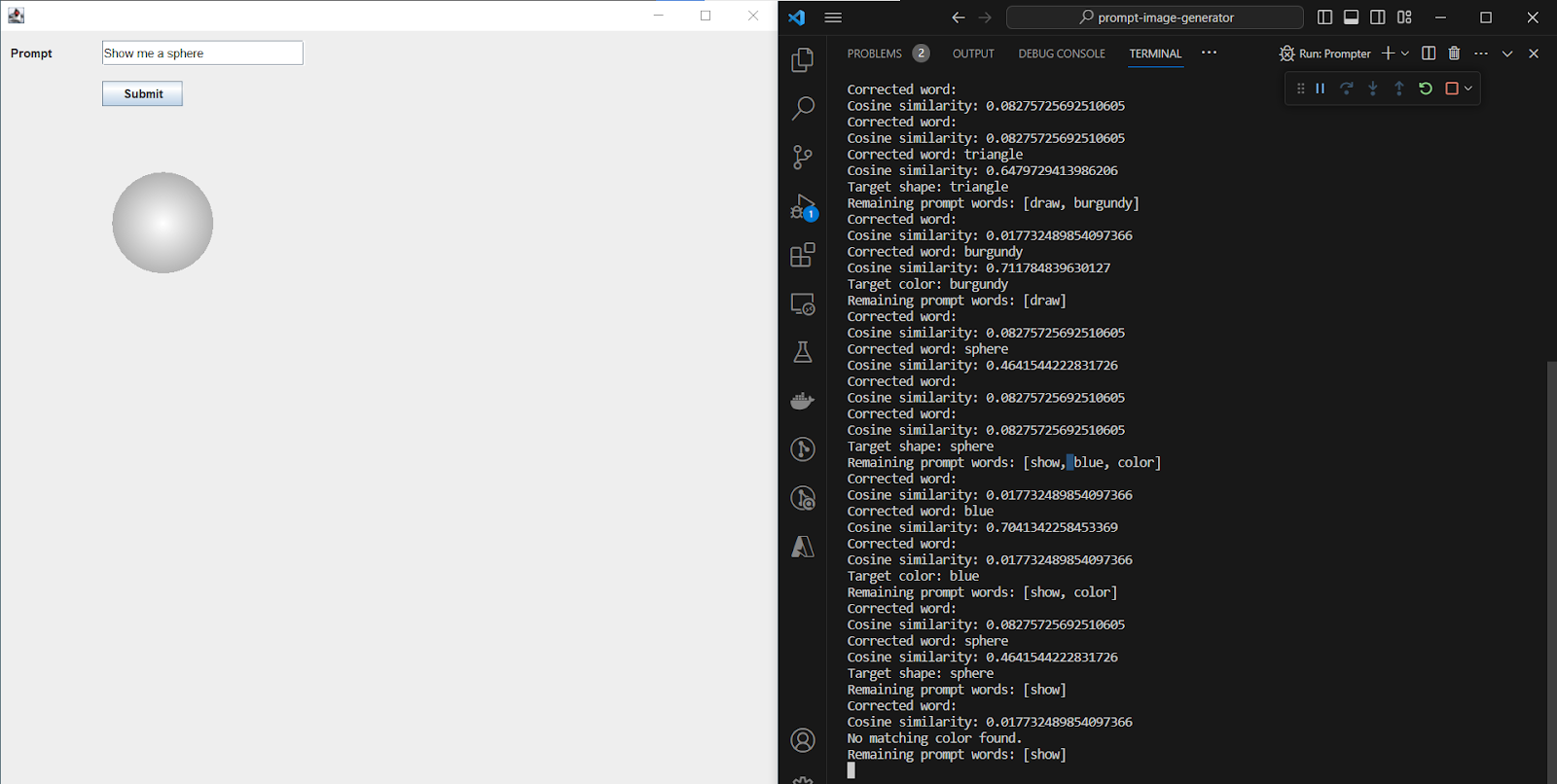
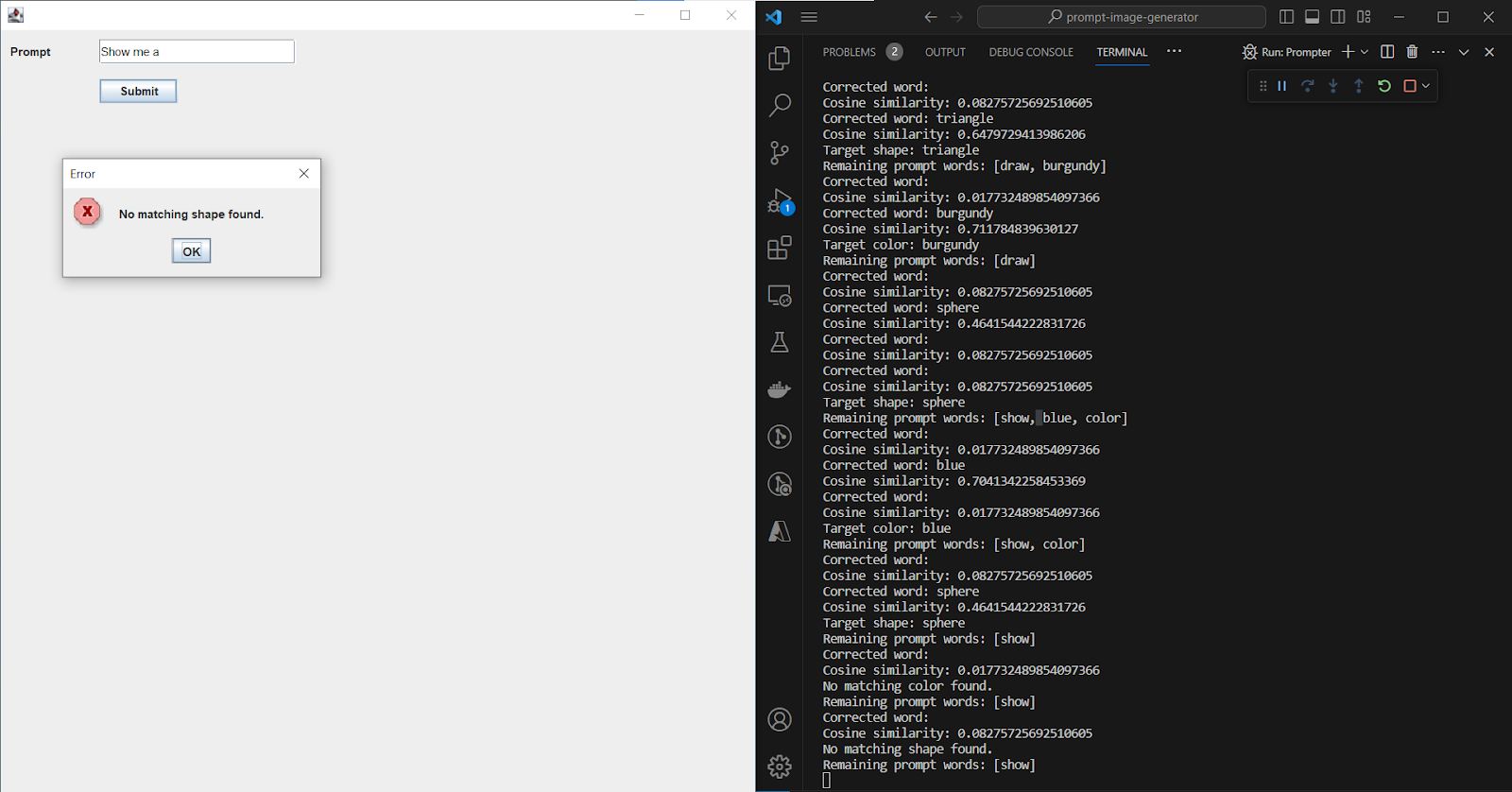
Fig. 15

Figure 15 exemplifies one of the edge cases handled by the application - a missing or unrecognized color. In case the user enters a prompt containing no identifiable color (i.e. requests a color that does not exist in the colors.txt file or does not specify a color at all) the system defaults to white, as a shape was still selected but the color was left blank. Although in the image, the corners of the sphere appear gray, that appearance only specifically applies to gradient-filled shapes like the sphere. In all other cases, the shape will appear in the RGB 255, 255, 255 hue, with either its sides filled or its lines drawn in white. Furthermore, a message reading “No matching color found” can be seen outputted in the console to the user’s assistance.

Fig. 16

The sixteenth figure portrays an error message indicating that no matching shape was found. This popup can come up either if the prompt contains no shape at all or if the shape specified could not be recognized by the algorithm because it does not match any of the shapes listed in the shapes.txt file. As seen on the screen’s right-hand side, the terminal displays both a “No matching color found” and a “No matching shape found” warnings since neither a color nor a shape were entered into the prompt. While it is possible to enter a prompt with no identifiable color as seen in figure 10, failing to specify a valid shape results in an error since there is nothing for the program to show on the screen.

1. **Analysis of the featured algorithms and data structures**

The development of the software package necessitated the usage of the Word2vec natural language processing technique as a means of analyzing the user-provided prompt and discovering the desired shape and color. The program also makes use of the Levenshtein Distance algorithm to first correct any mistyped words. Then, with the help of predefined object clusters, determined through the partial implementation of the K-means Clustering algorithm, the application calculates the cosine similarity between each prompt phrase and the centroid value of each cluster to determine whether a given word corresponds to a given color or shape. This section explains the inner-workings of the Word2vec model and the two underlying algorithms.

1. **Word2vec NLP Model**

The Word2vec model, co-authored in 2013 by Tomáš Mikolov, is a neural network machine learning model that takes in a large corpus of text to detect semantically and syntactically similar words on a basis of multidimensional word embeddings, or words translated into vector values. According to Rong, Word2vec utilizes either of two architectures encapsulating the meaning of different words, the continuous bag-of-words (CBOW) and the skip-gram (SG) architectures.

CBOW, as described by Rong, is designed for one-word and multi-word contexts. In the one-word context, it uses a basic neural network structure with matrices representing the weights between layers. The model predicts target words based on context words using softmax and updates weights according to prediction errors, gradually achieving equilibrium in vector positions. For multi-word contexts, the model averages input context word vectors, using similar loss functions and weight update equations as the one-word context model, adjusting weights for each context word.

On the other hand, the SG model is essentially the reverse of the CBOW model with the target word at the input layer and context words at the output. It generates multiple multinomial distributions for context words using shared weights across the output layer. The loss function and parameter update equations are adapted from the CBOW model, with specific changes to address the skip-gram context. This involves summarizing the prediction error across all context words and modifying the update logic for both hidden-output and input-hidden weights to suit the skip-gram model's structure (Rong).

For the purposes of this project, I imported a pre-trained Word2vec, readily available in the DL4J library. To ensure that the algorithm produces correct results, I had to introduce Word2vec to a substantial word corpus. For this reason, I fed the model a source file containing all words appearing in Google News. This source file is free-to-use and can be downloaded from an online cloud. Utilizing an external library and an outside source file is justified by the fact that developing and training a neural network model requires a significant outlay of development work, which is out of scope for this relatively simple implementation. Moreover, to function properly, the architecture employs a vector dictionary consisting of around 100 billion embeddings. Rather than trying to recreate a well-working neural network, the purpose of this program is to apply a pre-trained one in an attempt to identify and display a shape drawn in the specified color.

1. **Levenshtein Distance**

According to Haldar and Mukhopadhyay, the Levenshtein Distance (LD) is a metric used to quantify the similarity between two strings, specifically a source string and a target string. This distance is calculated by counting the number of deletions, insertions, or substitutions needed to change the source string into the target string. The larger the Levenshtein distance, the more dissimilar the two strings are. For instance, to change "GUMBO" to "GAMBOL", it requires one substitution (replacing "U" with "A") and one insertion (adding "L"), resulting in a Levenshtein distance of two. The algorithm for calculating LD involves three primary steps. In the initialization step, the lengths of the source (s) and target (t) strings are set as n and m respectively, and a matrix is constructed with dimensions based on these lengths. The first row and column of the matrix are initialized with sequential values from 0 to n and m. During the processing phase, each character of the strings s and t are compared. The cost is set to 0 if the characters match, and 1 if they do not. The value of each cell in the matrix, d[i,j], is calculated as the minimum of the values from the cell directly above, to the left, and diagonally above-left, adjusted by the cost. This process is repeated until the final value d[n,m] is determined, providing the LD.

Finding the distance between GUMBO and GAMBOL can be illustrated with the following example:

Table 1 (Step 1, iteration (i) = 0)

|  |  | G | U | M | B | O |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| G | 1 |  |  |  |  |  |
| A | 2 |  |  |  |  |  |
| M | 3 |  |  |  |  |  |
| B | 4 |  |  |  |  |  |
| O | 5 |  |  |  |  |  |
| L | 6 |  |  |  |  |  |

Table 2 (Step 2, iteration (i) = 1)

|  |  | G | U | M | B | O |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| G | 1 | 0 |  |  |  |  |
| A | 2 | 1 |  |  |  |  |
| M | 3 | 2 |  |  |  |  |
| B | 4 | 3 |  |  |  |  |
| O | 5 | 4 |  |  |  |  |
| L | 6 | 5 |  |  |  |  |

Table 3 (Step 3, iteration (i) = 2)

|  |  | G | U | M | B | O |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| G | 1 | 0 | 1 |  |  |  |
| A | 2 | 1 | 1 |  |  |  |
| M | 3 | 2 | 2 |  |  |  |
| B | 4 | 3 | 3 |  |  |  |
| O | 5 | 4 | 4 |  |  |  |
| L | 6 | 5 | 5 |  |  |  |

Table 4 (Step 4, iteration (i) = n, j=m)

|  |  | G | U | M | B | O |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| G | 1 | 0 | 1 | 2 | 3 | 4 |
| A | 2 | 1 | 1 | 2 | 3 | 4 |
| M | 3 | 2 | 2 | 1 | 2 | 3 |
| B | 4 | 3 | 3 | 2 | 1 | 2 |
| O | 5 | 4 | 4 | 3 | 2 | 1 |
| L | 6 | 5 | 5 | 4 | 3 | 2 |

The number in the lower right-hand corner indicates the distance between these two words, in this case 2 (Haldar and Mukhopadhyay).

This algorithm’s implementation is represented by the levenshteinDistance method in the PatternRecognizer class. Considering the method incorporates a nested for-loop, where the inner loop runs n + 1 times for each iteration of the outer loop, the total numbers of iterations is (m + 1) \* (n + 1) which simplifies to O(m \* n). Therefore, the overall time complexity of the function is O(m \* n) where m is the length of string a and n is the length of string b. This means the time complexity is proportional to the product of the lengths of the two input strings.

1. **K-means Clustering**

K-means clustering is a crucial clustering problem, particularly relevant in areas like data mining, data compression, and pattern recognition. It involves the process of identifying k centers in a d-dimensional space to minimize the mean squared distance from each data point to its nearest center, a concept known as squared-error distortion under variance-based clustering.

Lloyd's algorithm, known as “naïve k-means” due to the existence of much faster alternatives, employs an iterative scheme to achieve a locally minimal solution in the k-means clustering problem. It operates by continuously updating the centers to the centroid of the nearest data points until convergence criteria are met. However, it's noted that Lloyd's algorithm, while popular for its simplicity and flexibility, only guarantees convergence to a local minimum, not necessarily the global minimum. This distinction underlines the challenges in achieving optimal solutions in k-means clustering but also highlights the practical applicability and the widespread adoption of Lloyd's algorithm in various analytical contexts (Kanungo et al.).

The computeTargetObject method and the ClusterInfo class together reflect many key principles of the k-means clustering algorithm, albeit within a narrowly defined scope. Their functionality draws parallels with certain steps in k-means clustering, while also highlighting some notable deviations from the standard algorithm.

In k-means clustering, the concept of a cluster centroid is central. It represents the average position of all the points within a cluster. The ClusterInfo class embodies this concept by maintaining a centroid for its cluster. The crucial task of this class is to compute the cosine similarity between a given word and the cluster's centroid, echoing the k-means step where points are typically assigned to the nearest centroid based on a distance measure.

The computeTargetObject method takes this a step further by iterating over each word in the promptWords collection. It performs spell checking on each word and then calculates its cosine similarity to the centroid. The word that exhibits the highest similarity is identified as the targetObject. This mirrors the assignment phase in k-means clustering, where points are allocated to the nearest cluster center.

Once a targetObject is identified, the computeTargetObject method removes the closest word from the promptWords collection. This action parallels the reassignment process in k-means clustering, wherein points might be reassigned to different clusters following the recalculation of centroids.

Unlike the complete k-means process, which involves repeatedly recalculating the centroids of clusters after the assignment of points and reassigning points until the centroids stabilize or a maximum number of iterations is reached, the computeTargetObject method lacks a recalibration and reassignment process. This modification is justified by the system’s specific needs. For this project, calculating the centroid value of the cluster is the only essential computation, considering the program’s goal is to compare the cosine similarities of different prompt words and the cluster’s centroid value. Additionally, the clusters in the program act as completely separate entities with distinct centroids representing the average of the color and shape vectors rather than a collection of clusters or cluster points that necessitate a recalibration process.

Furthermore, while traditional k-means clustering typically uses Euclidean distance to measure the proximity of points to centroids, this implementation employs cosine similarity. This choice is more suited for comparing word vectors in a high-dimensional space, as is common with Word2Vec models.

Among the various data structures present in the software package, 2D arrays, lists, and hash maps are the most prominent. The implementation of the Levenshtein distance algorithm utilizes a 2D array, specifically an integer array (int[][] dp), as part of its dynamic programming approach. This array is pivotal in storing the minimum number of operations required to transform one string into another. By leveraging this 2D array, the algorithm can efficiently compute the Levenshtein distance, which is a measure of the difference between two sequences.

In certain parts of the system, lists, which are part of Java's collection framework, are extensively used. For example, the StopWords class employs a list to retrieve and store stopwords from a file. Additionally, lists are utilized in methods like computeTargetObject to store and process words. The use of lists in these contexts highlights their versatility and efficiency in handling collections of data, particularly when the data size is variable or when the application demands dynamic data manipulation.

The ColorCodeGenerator class demonstrates the use of a HashMap, specifically HashMap<String, String>, to create a mapping between color names and their corresponding RGB values. This hash map is a critical component for the efficient lookup of RGB codes based on color names. The choice of a hash map in this scenario is particularly advantageous due to its O(1) average time complexity for operations like insertions, deletions, and lookups. This efficiency makes hash maps an ideal choice for applications that require frequent and fast data retrieval, as seen in the ColorCodeGenerator class where quick access to color codes is essential.

1. **Security features and reliability considerations**

While the software package lacks any specific security measures, such as encryption for passwords, anti-hacking precautions, or other typical cybersecurity instruments, the system places an emphasis on reliability. The primary reason for the absence of sophisticated security features could be attributed to the nature of the application itself. As a graphical user interface (GUI)-based application, it is primarily concerned with drawing shapes and processing language input rather than handling sensitive data or user accounts. Typically, applications dealing with graphics and basic language processing do not involve aspects like user authentication, financial transactions, or storage of sensitive personal data, where one would expect advanced security. However, to ensure that the codebase works consistently and adequately handles edge cases, the system includes robust error handling and input validation, as well as data handling and processing, described below.

1. Error Handling and Input Validation:

The application properly validates user inputs, especially when using them to instantiate objects via reflection (as seen in the DrawingPanel class) in order to prevent errors or potential security vulnerabilities. Furthermore, the code logs any relevant information to the console, enhancing the user’s ability to understand the software’s inner-workings and enabling developers to debug any potential flaws. For instance, the system prints out information about the Word2vec model and the Deeplearning4j through the Google News dictionary’s loading process. Further, the program catches any I/O Exceptions that might occur when opening the shapes and colors text files, most notably within the readObjectsFromFile method in the PatternRecognizer class.

1. Data Handling and Processing:

The application processes textual data and requires careful handling of textual input to avoid issues like buffer overflows or improper data sanitization, which could potentially lead to stability issues or vulnerabilities.

1. **Implementation**
2. **Analysis of the chosen programming language and external libraries**

The software solution described throughout this paper leverages a variety of technologies, each chosen for its specific strengths and suitability to the tasks at hand. Here is an overview of these choices:

1. **Programming Language**
2. Java: The programming language used is Java, a versatile and widely-used language known for its portability, object-oriented features, and extensive standard libraries. Java's platform independence makes it a suitable choice for applications that need to run across different operating systems without modification. The use of Java also benefits from strong community support and a rich ecosystem of libraries and tools.
3. Java Swing for GUI: For the graphical user interface, the Java Swing library is utilized. Swing is a part of Java's standard libraries and provides a rich set of components for building cross-platform window-based applications. It was chosen for its integration with Java, ease of use, and the ability to create customizable and responsive GUIs.
4. **External Frameworks and Libraries**
5. Java I/O and Utility Libraries: Standard libraries for file input/output and collection manipulation are utilized across various classes. These libraries are integral to Java and provide reliable and efficient means to handle file operations and data structures. The choice to use these built-in libraries ensures compatibility, reduces the need for external dependencies, and leverages Java's native capabilities for handling data.
6. Apache Maven: Maven is a powerful project management and comprehension tool widely used in Java projects. It plays a pivotal role in managing project builds, dependencies, documentation, and more. For this project, Maven is utilized for simplifying the process of including external libraries (like DL4J and ND4J). It automatically handles the downloading, updating, and integrating of these libraries, ensuring that the correct versions are used and reducing the risk of conflicts or incompatibilities. Furthermore, Maven streamlines the build process by handling tasks such as compilation, testing, and packaging. This automation enhances the efficiency of the development process, especially in complex projects with multiple components and dependencies. Maven's standard project layout and conventions, the project gains a structured and standardized format. This standardization is particularly beneficial for maintenance and scalability, as it makes the codebase more understandable and manageable for developers.
7. Deep Learning for Java (DL4J): This is used particularly for the Word2Vec model. DL4J is a comprehensive deep learning library for Java, chosen for its native support for Java, ease of integration, and robust capabilities in handling complex natural language processing tasks.
8. ND4J (N-Dimensional Arrays for Java) as an extension of DL4J: This library is employed for handling high-dimensional numerical data, particularly in the ClusterInfo class. ND4J is essentially Java's answer to Python's NumPy, providing a platform for scientific computing and a wide array of linear algebra and numerical operations. Its selection is justified by the need for efficient numerical computations, which is crucial in processing and manipulating word embeddings and other mathematical models in the application.

The technologies for this software solution are meticulously selected to optimize its functionality, usability, and maintainability across various platforms.

Java stands out for its cross-platform capabilities, ensuring the application's compatibility with different operating systems. This universality makes the application widely accessible and usable, catering to a diverse user base irrespective of their operating system. This feature is particularly vital for an application designed to process natural language and visualize data, as it broadens the reach and applicability of the software.

For the graphical user interface, Swing is utilized for its flexibility and robustness in GUI design. Swing's extensive toolkit allows for the creation of a user-friendly and interactive interface, a critical aspect for applications that heavily involve user interaction and data visualization. The ability to design intuitive and responsive interfaces with Swing enhances the overall user experience, making complex functionalities more accessible and easier to navigate.

Java’s I/O and utility libraries offer robust, well-tested methods for file handling and data manipulation, essential for the application's needs in effectively reading, processing, and storing data. These libraries ensure that data is handled securely and efficiently, maintaining the integrity and reliability of the application.

Maven is instrumental in managing dependencies, particularly for external libraries like DL4J and ND4J. Its seamless integration of these libraries into the project minimizes potential conflicts and compatibility issues, ensuring a smooth development process. Maven’s ability to automate and standardize build processes is vital for maintaining consistency, especially in collaborative environments or when scaling the project. It guarantees uniform project setups across different development environments, reducing the likelihood of errors related to build configurations. Furthermore, Maven's standardized project structure and documentation capabilities greatly enhance the accessibility and comprehensibility of the project for new developers, fostering better collaboration and easier maintenance.

The software integrates DL4J for its natural language processing

capabilities. DL4J is adept at handling complex NLP tasks, which are central to the application's core functionality of interpreting user inputs. This integration enables sophisticated processing and understanding of natural language data, a cornerstone in delivering accurate and contextually relevant responses to user queries.

For numerical computations, ND4J provides efficient and powerful computational abilities. Its optimized performance is crucial for handling the computationally intensive tasks inherent in clustering and vector calculations, especially relevant in the context of word embeddings and natural language data processing. ND4J's capacity to handle high-dimensional data and perform intricate mathematical operations seamlessly integrates with the application's requirements.

In summary, the selection of Java, Swing, Java’s I/O and utility libraries, Maven DL4J, and ND4J is a strategic decision, aligning with the project's goals of cross-platform operability, user-friendly interface design, advanced natural language processing, efficient numerical computations, and streamlined project management. These technologies collectively form a robust framework, ensuring the application is not only functional and efficient but also scalable and maintainable.

1. **Software installation requirements and dependencies**

Usage of the software package requires specific installation prerequisites to be met, particularly concerning the libraries essential for its functionality. These range from GUI creation and natural language processing to numerical computations. The core of the application is built in Java, necessitating the installation of the Java Development Kit (JDK). The JDK is pivotal as it provides the environment needed to develop and run Java applications.

For the graphical user interface, the application leverages Swing, a part of the standard Java library. Therefore, the Java Runtime Environment (JRE) must include Swing components. Usually, a standard installation of the JDK or JRE would suffice since Swing is commonly bundled in these packages.

One essential aspect of the application is its natural language processing capabilities, powered by Deep Learning for Java. Installing DL4J typically requires a dependency management tool like Maven because DL4J comes with several of its own dependencies. The specific DL4J version needed should be specified in the project’s build file, such as the pom.xml for Maven. This setup ensures automatic handling of the library download and setup process.

ND4J, essential for numerical computation, especially for handling high-dimensional data arrays, is another critical component. The installation process for ND4J mirrors that of DL4J, favoring the use of a dependency manager like Maven. This approach guarantees that all required components and dependencies of ND4J are properly installed and configured.

Standard Java I/O and utility libraries, integral for file handling and data manipulation, are part of the JDK. Therefore, no additional installation steps are typically needed beyond the JDK itself.

While not mandatory, using Maven is highly recommended for managing the project’s dependencies, particularly for libraries like DL4J and ND4J. Maven streamlines the project setup and maintenance, effectively managing library versions and ensuring consistent resolution of dependencies. For Maven to function, it should be installed and configured on the development machine, and the project must include a pom.xml file that details all dependencies. Before running the application, it is imperative to ensure that all these libraries and tools are correctly installed and configured. Failure to do so can result in runtime errors and impede the application's functionality. Additionally, checking the system requirements of each library, particularly regarding operating system compatibility and memory demands, is advisable to guarantee the application's optimal performance.

1. **Results and conclusion**

The development journey of this project has been a testament to growth and learning, marked by significant progress from the original concept to the final product. The project set out with a vision to create a software application capable of transforming natural language inputs into visual representations, effectively bridging the gap between words and images.

Central to the project's success was the seamless integration of a variety of technologies culminating in a unified application. A key achievement was the establishment of an intuitive graphical user interface using Java Swing, which became the interactive platform for users to input prompts. These prompts were then processed using natural language processing techniques, thanks to the capabilities of the Deep Learning for Java (DL4J) library and the Word2Vec model. This NLP integration was not merely a technical milestone but also a functional breakthrough, as it allowed the translation of complex linguistic patterns into actionable data for the system.

An important aspect of the project was the creation of a hierarchy of shape classes in Java that enabled the dynamic rendering of various shapes in response to user prompts. This functionality highlighted the application's versatility in graphical representation, able to depict anything from simple circles to complex polygons.

When reflecting on the initial objectives, the system largely met expectations, particularly in its proficiency in parsing text and producing corresponding shapes. Nonetheless, the system's capabilities to understand and interpret more abstract or intricate prompts remained elementary. The application's NLP component worked efficiently within the bounds of predefined shapes, but the interpretation of more subtle or context-heavy instructions was limited.

The project proved to be a rich learning experience, particularly regarding the technical aspects of integrating NLP with graphical rendering in Java. It required a deep understanding of Java's class structures and drawing capabilities and a foray into NLP and machine learning principles. Beyond the technicalities, the project emphasized the importance of defining a clear scope and managing expectations, serving as a practical example of the iterative nature of software development where visions adapt in light of emerging challenges and opportunities.

In summary, the project has been a remarkable exercise in interdisciplinary software development, combining language processing with graphic design and user experience. While there are opportunities for further improvement—especially in terms of broadening the NLP's comprehension and the system's flexibility—the project has laid down a robust foundation. It paves the way for future enhancements, potentially transforming the application into an even more dynamic and powerful tool for converting linguistic expressions into visual art.

Works Cited

“A New AI Language Model Generates Poetry and Prose.” *The Economist*, The Economist Newspaper, 6 Aug. 2020, www.economist.com/science-and-technology/2020/08/06/a-new-ai-language-model-generates-poetry-and-prose.

“A Robot Wrote This Entire Article. Are You Scared Yet, Human?” *The Guardian*,

Guardian News and Media, 8 Sept. 2020,

www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3.

Deeplearning4j Suite Overview.” *Deeplearning4j*, Eclipse Foundation,

deeplearning4j.konduit.ai/. Accessed 28 Nov. 2023.

Haldar, Rishin, and Debajyoti Mukhopadhyay. “Levenshtein Distance Technique in

Dictionary Lookup Methods: An Improved Approach.” *arXiv*, Cornell Tech, 6 Jan.

2016, doi.org/10.48550/arXiv.1411.2738.

“Introduction to Java Swing.” *GeeksforGeeks*, 29 Mar. 2023,

www.geeksforgeeks.org/introduction-to-java-swing.

Kanungo, Tapas, et al. “The analysis of a simple k-means clustering algorithm.” *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, 1 May 2000, https://doi.org/10.1145/336154.336189.

Metz, Cade, and Gregory Schmidt. “Elon Musk and Others Call for Pause on A.I., Citing ‘Profound Risks to Society.’” *The New York Times*, The New York Times, 29 Mar. 2023, www.nytimes.com/2023/03/29/technology/ai-artificial-intelligence-musk-risks.html.

Rong, Xin. “Word2vec Parameter Learning Explained.” *arXiv*, Cornell Tech, 5 June

2016, doi.org/10.48550/arXiv.1411.2738.

“What Is Java Technology and Why Do I Need It?” *Java*, Oracle,

www.java.com/en/download/help/whatis\_java.html. Accessed 28 Nov. 2023.

“What Is Maven?” *Maven*, The Apache Software Foundation,

maven.apache.org/what-is-maven.html. Accessed 28 Nov. 2023.