SEMESTRAL PROJECT
GUESSING GAME IN PYTHON

TERÉZIA MACHAROVÁ

1. Summary

This application is written in the programming language python. It's a guessing game where the user thinks of an animal and the computer is trying to guess it. If the computer is unable to guess the animal correctly the user wins and is asked to teach the computer the new animal. If the computer guesses it correctly, the user loses.

2. User part

This part is for the users who want to play the game on their device. It is a summarized user guide. The goal of this part is to let the user know how to run the application, what data are needed and what to expect.

2.1. How to run the application

To run the application, it's necessary to have a Python installation, the version should be ideally 3.6 or newer. The game is played through command line or terminal, depending on the OS of the user, so it's necessary to have access to this to run the python script. The game also uses library *SpellChecker* for spell checking, This library can be installed by the user through Python's package installer (pip), if the user doesn't have it already.

The game can be either run through command line or terminal by executing the right command or through IDE or with a text editor.

2.2. Input data

The game relies on a JSON file called a*nimaltree.json*. This file stores the decision tree that is used to guess the animal. This file should be in the same directory as the Python script.

This file is modified as needed when the user adds a new animal to the game.

2.3. Output data

The game interacts with the user by asking questions or guessing an animal. We could consider these as output data. When the user starts the game, it will welcome the user and ask the first question right away. Then it will wait for the user's answer and continue asking until the animal is either guessed or not. After finishing the game it will ask the user if they want to play again or not.

2.4. Controlling the application

The application is controlled through terminal or command line. At the beginning, the user answers yes/no questions, while the computer is trying to guess the animal the user is thinking of. When the computer guesses the animal correctly, it will announce that the user lost. If the user wins it means that the computer didn't guess correctly. In this case, the computer asks the user what animal they were thinking of and a question that distinguishes the new animal from the guess. The user will answer and then it will be saved into the json file. Afterwards the user will be given an option of playing again or not. If they choose to play again, the game will start from the beginning.

3. Programming part

This part is for the users who want to look at the source code. It's supposed to help the user understand how the source code works and how it is divided. There are also comments that help with the orientation in the code.

3.1. Division

The code is composed of one class named GuessingGame, which is divided into multiple methods that manage the logic of the game as well as all the work related to the file where the game tree is.

3.2. Libraries

The code uses two external libraries, *json* and *spellchecker*. *Json* library is used to save and load the decision tree. *SpellChecker* is used for the spell checking of the user input.

3.3. Data structures and classes

The *GuessingGame* class consists of multiple methods, the first being the *__init__* method. This one initializes a new instance of the guessing game. It loads the decision tree from json file. If the file doesn't exit, it initializes the game with a default decision tree.

The next to methods are *save_tree* and *load_tree*. These two are responsible for saving the current state of the tree and loading the tree from the JSON file.

The *yes_no_question* method is used when the yes/no question is asked. It loops until a valid answer is received.

The next methods are *guess_animal, new_animal* and *update_tree*. These methods are responsible for the game logic. Method *guess_animal* recursively goes through the decision tree based on the user's answers to guess the animal the user is thinking of. If the guess is wrong it calls the *new_animal* method and lets the user teach the game a new animal. Then the *update_tree* updates the decision tree with the new information.

The method *spell_check* is responsible for checking the spelling of user input when learning a new animal. It uses the *SpellCheck* library.

The method *play* is the main loop of the game. It allows the user to play more rounds of the game, starting from the root of the decision tree and continues until the user decides to stop playing.