Professor: Offer Wald

Student: Jamoliddinov Dilshodbek.

OS project: “LinGame”.

Uzbek-Israel Joint faculty;
Computer Science.

26.2.2023

Description:

Introduction to the project:

This is a Python script that implements a Kivy app. The app is a word (bash commands) game where the user needs to guess a word by moving letters from a set of shuffled letters to the correct position.

Here is a brief overview of the script:

- The script imports the required modules and loads a .kv file that defines the user interface.
- The script defines a list of letters, a function to read data from a file, and some classes for the app's UI components.
- The script defines a popup for winning and a popup for losing.
- The script defines the main window of the app, which contains the game logic and the UI components.
- The script defines a release method that is called when the user clicks on a letter button.
- The release method updates the UI and checks if the user has guessed the correct word.
- If the user has guessed the correct word, a win popup is displayed, and the game is reset with a new word.

Overall, the script is well-organized and easy to follow. The use of Kivy and its built-in UI components makes it easy to create a polished-looking app.

Why did I choose Kivy framework?

Kivy is a popular open-source framework for developing cross-platform applications in Python. It is a powerful and flexible framework that makes it easy to create cross-platform applications with a beautiful user interface and dynamic graphics. Its open-source nature, Pythonic API, and active community have helped to make it a popular choice among developers. Here are some reasons why it has gained popularity:

Cross-platform development: Kivy allows developers to create applications that can run on multiple platforms, including desktops, mobile devices, and even Raspberry Pi.

User Interface: Kivy provides a fast and easy way to create beautiful user interfaces. It includes a powerful set of widgets and a flexible layout system that makes it easy to create and customize user interfaces.

Graphics: Kivy includes a powerful graphics engine that allows developers to create rich and dynamic graphical applications. It supports hardware-accelerated graphics and can display 2D and 3D graphics.

Open-source: Kivy is an open-source project, which means that it is free to use and modify. This has helped to create a vibrant community of developers who contribute to the framework and share their knowledge and experience.

Pythonic: Kivy is designed to be easy to use for Python developers. Its API is designed to be intuitive and easy to learn, making it a great choice for developers who are new to GUI programming.

Documentation and support: Kivy has a comprehensive documentation that covers all aspects of the framework. It also has an active community of developers who provide support and share their knowledge through forums, chat rooms, and other online resources.

Why is this project useful?

Most importantly, This project fully demonstrates the first subject of syllabus, as it tells the user some story about Linux Kernel every 10 levels. Besides, this program is intended to learn shell commands of Linux, so I think it is already related to the OS Kernel. Enjoy reading and playing this useful application!!!

According to various sources, it's estimated that around 2-3% of desktop computer users use Linux as their primary operating system. While this number may seem relatively small compared to Windows and macOS users, it's worth noting that Linux is widely used in certain industries and communities, such as scientific research, software development, and web hosting. In these areas, it's common for users to have a high level of familiarity with Linux.

The demand for Linux and Bash commands has been steadily increasing in recent years due to several factors. Linux is an open-source operating system that is widely used in many industries, including web hosting, software development, scientific research, and cloud computing. As a result, there is a growing demand for professionals with Linux skills and experience. With the rise of cloud computing, many

organizations are adopting Linux-based systems and technologies such as Docker, Kubernetes, and OpenStack. This has increased the demand for professionals who can manage and administer Linux systems in cloud environments. DevOps practices emphasize collaboration and automation between development and operations teams, with Linux and Bash commands often used to automate tasks and streamline workflows, making them an essential skill for DevOps professionals. Additionally, Linux is known for its security and is often used in security-sensitive environments, further increasing the demand for professionals with Linux skills. In summary, the demand for Linux and Bash commands is driven by the growth of cloud computing, DevOps practices, cybersecurity, and the popularity of Linux as an operating system in many industries.

Documentation

This is a Python program using the Kivy framework to build a word game app. The app displays a word meaning, and the user is tasked with rearranging randomly shuffled letters to form the correct word which is a shell command of Linux. The user can click on buttons with letters to move them to a grid at the top of the screen, where they can arrange the letters to form the correct command. If the user forms the correct word, they win the round and the game advances to the next level, where a new command and meaning are displayed. If the user fails to form the correct word, they lose the round and can try again. Each time when user passes 10 rounds a small information about Linux will appear and saved in the second screen. Now, let's move to the description of the code.

The program uses the following classes and functions:

First Part:

```
def linux():
    with open('Linux.json') as json_file:
```

```
        data = json.load(json_file)

    with open("Level.txt", "r") as level_file:
        level = level_file.read()

    buts = data[level][0]
    mean = data[level][1]
    return buts, mean, int(level)
```

1. The **linux**() function: This function reads data from two text files, Linux.json and Level.txt, and returns a tuple containing a jumbled word, its meaning, and the level of the game.

```
class WinPopup(Popup):

    def close_pop(self):
        self.dismiss()

    def new_win(self):
        return linux()

    def leveling(self):
        with open("Level.txt", "r") as lev:
            level = lev.read()
        self.title = "Level: " + str(int(level)+1)
        self.open()
```

2. The **WinPopup** class: This class is a popup window that is displayed when the user wins the game. The popup has three methods: close_pop(), new_win(), and leveling(). The close_pop() method dismisses the popup window. The new_win() method calls the linux() function to get a new word and its meaning. The leveling() method updates the level of the game in the popup title.

```
class LoosePopup(Popup):
    def close_pop(self):
        self.dismiss()
```

3. The **LoosePopup** class: This class is a popup window that is displayed when the user loses the game. The popup has one method, close_pop(), which dismisses the popup window.

4. The **DownButtons** and **UpButtons** classes: These classes are used to design the buttons that are used in the game. The DownButtons class is used to create the jumbled letters, while the UpButtons class is used to create the boxes for the correct letters.
5. The **MeanLabel** class: This class is used to display the meaning of the jumbled word.

```
class MainWindow(Screen):
    number = NumericProperty(0)
    downButs = ObjectProperty(None)
    upButs = ObjectProperty(None)
    firstLayout = ObjectProperty(None)
    meanLayout = ObjectProperty(None)
    topBar = ObjectProperty(None)

    theList = {
        0: 'Welcome to the history of  Linux',
        1: 'What Inspired Creating Linux',
        2: 'The Story Behind the Name Linux',
        3: 'Torvalds' Linux Development',
        4: 'Role of GNU GPL in Linux',
        5: 'Linux Mascot',
        6: 'Linux Distributions',
        7: '10 Facts about Linux',
        8: 'List of Linux Distributions',
        9: 'The Conclusion',
    }

    def __init__(self, **kwargs):
        super(MainWindow, self).__init__(**kwargs)

        self.data = linux()
        self.meaning = self.data[1]
        self.word = self.data[0]
        self.level = self.data[2]
        self.check = ""

        self.lab = MeanLabel(text=self.meaning)
        self.ids["task"] = self.lab
        self.meanLayout.add_widget(self.lab)

        self.ran_word = list(self.word) + [random.choice(alphabet) for _ in
range(16 - len(self.word))]
        random.shuffle(self.ran_word)
        for el in self.ran_word:
            self.but = DownButtons(text=el)
            self.but.bind(on_release=self.release)
            self.ids[self.number] = self.but
            self.downButs.add_widget(self.but)
            self.number += 1
        self.number = 16
```

```
        for i in self.word:
            self.ubut = UpButtons(text="")
            self.ubut.bind(on_release=self.release)
            self.ids[self.number] = self.ubut
            self.upButs.add_widget(self.ubut)
            self.number += 1

        self.upButs.cols = 8

        if self.number - 16 > 8:
            self.upButs.rows = 2
        else:
            self.upButs.rows = 1

        self.new = 16

    def release(self, instance):
        self.check += instance.text
        self.ids[self.new].text = instance.text
        self.new += 1
        if len(self.check) == len(self.word):
            if self.word == self.check:
                self.level += 1
                self.check = ""
                pop = WinPopup()
                if self.level == 89:
                    with open("Level.txt", "w") as newLev:
                        newLev.write("0")
                        self.level = 0
                else:
                    with open("Level.txt", "w") as newLev:
                        newLev.write(str(self.level))

                if self.level % 10 == 0:
                    kl = self.level // 10
                    stp = StoryPopup()
                    stp.info(self.theList[kl-1])

                pop.leveling()

                self.data = pop.new_win()
                self.meaning = self.data[1]
                self.word = self.data[0]

                self.ran_word = list(self.word) + [random.choice(alphabet)
for _ in range(16 - len(self.word))]
                random.shuffle(self.ran_word)

                self.ids["task"].text = self.meaning
                for el in enumerate(self.ran_word):
                    self.ids[el[0]].text = el[1]

                self.upButs.clear_widgets()
                self.new = 16
                self.upButs.cols = 20
                for i in self.word:
                    self.ubut = UpButtons(text="")
                    self.ubut.bind(on_release=self.release)
                    self.ids[self.new] = self.ubut
                    self.upButs.add_widget(self.ubut)
                    self.new += 1

                self.upButs.cols = 8
```

```
                if self.new - 16 > 8:
                    self.upButs.rows = 2
                    self.upButs.height = self.downButs.height / 1.25
                else:
                    self.upButs.rows = 1
                    self.upButs.height = self.downButs.height / 1.7
                self.new = 16
            else:
                self.new = 16
                self.check = ""
                print("failed")
                pop = LoosePopup()
                pop.open()
                for el in range(16, 16 + len(self.word)):
                    self.ids[el].text = ""
```

6. The **MainWindow** class: This class is the main class of the program, and it defines the logic of the game. The class has several properties and methods that are used to implement the game. The __init__() method initializes the game by creating the jumbled letters, the boxes for the correct letters, and the MeanLabel object. The release() method is called when the user clicks a button, and it updates the user's input. If the user successfully unscrambles the word, the WinPopup popup is displayed. If the user fails to unscramble the word, the LoosePopup popup is displayed.

Second Part:

```
class StoryPopup(Popup):
    message = ObjectProperty(None)
    pop_story = ObjectProperty(None)

    def info(self, data):
        with open(f"history/{data}.txt", "r") as f:
            dd = f.read()
        self.title = data
        self.message.text = dd
        self.open()
```

1. This piece of Kivy code defines a class named **StoryPopup**, which inherits from Popup. It contains two ObjectProperty attributes, message and pop_story. The info() method is defined which takes data as an argument. Inside the info() method, it opens a JSON file named history.json, loads its content into a variable dd, and uses the data parameter to

retrieve the title and message for a specific story from the JSON file. The retrieved title is assigned to the title attribute of the StoryPopup object, and the retrieved message is assigned to the text attribute of the message object. Finally, the open() method is called to display the popup.

```
class Story(Screen):
    theList = {
        'Welcome to the History of Linux': 0,
        'What Inspired Creating Linux': 1,
        'The Story Behind the Name Linux': 2,
        'Torvalds' Linux Development': 3,
        'Role of GNU GPL in Linux': 4,
        'Linux Mascot': 5,
        'Linux Distributions': 6,
        '10 Facts about Linux': 7,
        'List of Linux Distributions': 8,
        'The Conclusion': 9,
    }
    topBar = ObjectProperty(None)
    boxList = ObjectProperty(None)

    def __init__(self, **kwargs):
        super(Story, self).__init__(**kwargs)

    def thereader(self):
        with open("Level.txt", "r") as f:
            d = f.read()
        data = int(d) // 10
        self.boxList.clear_widgets()

        container = BoxLayout(orientation='vertical', size_hint_y=None)
        container.bind(minimum_height=container.setter('height'))

        with open(f"history.json", "r") as f:
            dd = json.load(f)

        for el in range(data + 1):
            bt = Topics(text=dd[str(el)][0])
            self.ids[el] = bt
            bt.bind(on_release=self.release)
            container.add_widget(bt)

        self.boxList.add_widget(container)

    def release(self, instance):
        popper = StoryPopup()
        popper.info(str(self.theList[instance.text]))
```

2. The **Story** class is a Screen class that represents the second screen of the application. It has a dictionary theList which maps integers to strings representing the topics of the

Linux operating system. The class has two attributes, topBar and boxList, which are both ObjectProperty objects. The thereader() method reads the content of a file named "Level.txt" and calculates the number of buttons to display on the screen. It then creates buttons for each topic and adds them to the boxList widget. The release() method creates an instance of the StoryPopup class and calls its info() method, passing the text(position) of the clicked button as an argument.

```
class MyGame(App):
    st = Story(name='story')
    mw = MainWindow(name='main')

    def build(self):
        self.sm = ScreenManager()
        self.sm.add_widget(self.mw)
        self.sm.add_widget(self.st)

        return self.sm
```

Finally, the **MyGame** class is a subclass of the App class and has two attributes st and mw, which are instances of the Story and MainWindow classes, respectively. Its build() method creates an instance of the ScreenManager class and adds both the mw and st screens to it. The method then returns the ScreenManager instance.

Short explanation of how it works on mobile phone

In order to make this app run on mobile devices we have turned our app into the apk file. It was possible by buildozer.spec file. So, here is a step by step explanation of the program usage:

1. We have to install the apk file on our mobile phone;
2. When entering the game we face the first screen, where the gaming is done (MainWindow class);
3. If we click on the button which is at the top corner, the screen is changed to the second one;
4. At the 2nd screen will be listed all the topics related to the Kernel Linux;
5. When user click on one of them there will be a popup window with full explanation of this particular topic;
6. Every 10 levels a new topic will be added to the second sceen.
7. Overall there are 10 topics

Some Photos of demonstration: