09.01.2025 – Max Mendgen
2INF2 LK Q1.1 Klausurersatzleistung – Projekt Wissensspiel
Kurze Projektdokumentation: Scrambled v0.3

Einleitung

Ziel war es, ein Wissensspiel zu entwickeln und dabei mein Wissen in Python und GUI Implementierung bzw. Tkinter zu erweitern. Dabei gab es vorgegebene Ideen, welche Wissensspiele, beispielsweise ein Länder-Quiz, entwickelt werden können. Aber ich wollte etwas mehr herausfordernd entwickeln, da ich schon Vorerfahrung in Unity Spielentwicklung habe.

Nach einer intensiven Findung von Ideen habe ich mich für ein selbst gedachtes Spiel entschieden: „Scrambled!", was die Problemlösungsfähigkeiten von Anagrammen fördert.

Projektbeschreibung

Spielkonzept
In diesem Wissensspiel versucht der Spieler, aus einer Mischung von gegebenen Buchstaben ein korrektes Wort zusammenzusetzen. Der Spieler kann als Aktion 2 Buchstaben per Zug verstauchen, jedoch hat er eine Limitierung der Züge.

Zur Einstellung der Schwierigkeit kann die Anzahl der Buchstaben bzw. die Länge des Wortes und die Anzahl der Züge verändert werden. Optional kann noch ein Zeitlimit hinzugefügt werden.

Technische Umsetzung:
- **Verwendete Programmiersprachen:** Python und SQL (Nur bei Datenextraktion zu TXT-Files)
- **Verwendete Bibliotheken:**
  - `tkinter`: Für die Gestaltung der Benutzeroberfläche.
  - `random`: Zum Mischen der Buchstaben.
  - `os`: Zum orten der Dateien als Pfad.
  - `ast`: Zum Parsen von gespeicherten Spielständen.
  - `math`: Für verschiedene mathematische Operationen.
  - `webbrowser`: Zum öffnen von Links
  - `time`: Genauere Zeitverfolgung des Zeitlimits
  - `itertools`: Zur Überprüfung des gemischten Wortes
- **Wörterdatenbank:** Wortschatz Leipzig - Mixed-typical 10k Wörter von 2011
  - Ich habe diese Datenbank verwendet, da ich beim vorigen Versuch die Wörterauswahl von OpenThesaurus nicht alltagstauglich empfand.
- **Datenextraktion aus SQL-Files:** Maria DB
- **Struktur des Codes:** OOP (objektorientierter Programmierung)
  - Eine klare Trennung zwischen GUI-Elementen und Spiel-Logik.
  - Verwendung von feststehende Variablen.
  - Alles auf Englisch gelassen statt eine Mischung für besseren Verständnis.
- **Recherche Hilfen:**
  - Google: Suchmaschine
  - StackOverflow: Forum mit Fragen und Antworten von Leuten über Programmierung und anderes
  - und weiteres...
- **Code Editor:** Visual Studio Code
- **Logo:** Mit Canva.com erschaffen

Implementierung

Da dieses Programm groß ist und mehrere wichtige Teile hat, werde ich nur einige davon aufklären.

Wörterauswahl
Die Kernpunkte der Funktionalität bei der Auswahl von Buchstaben läuft in diesen Schritten ab:
1. Zufälliges Wort auswählen nach gegebene Länge
2. Weitere Anagramm suchen falls vorhanden

3. Buchstaben mischen und herausgeben

Die Wörter sind in mehrere TXT-Dateien gespeichert und nach Wörterlänge sortiert. Hier kurz einen Code-Snippet zur den Kernpunkte 1 und 2 (Z.111):

```
def get_random_word_set(wordLength):
  """Generate a random word set."""
  fileName = f"words{wordLength}.txt"
  with open(DATA_PATH + fileName, "r") as file:
    word = get_random_line(file)
  with open(DATA_PATH + fileName, "r") as file:
    anagramms = get_anagramms(word, file)
  return [word] + anagramms
```

Erstmal finde ich den richtigen Namen der TXT-Datei, dann lese ich bzw. hole mit meiner Funktion `get_random_line()` über Waterman's Reservoir Algorithm eine zufällige Zeile raus, sowie die Anagramme mit meiner Funktion `get_anagramms()`. Genauere Details werde ich hier nicht eingehen, sehe selbst unter Z.75. Zuletzt speichere ich das gefundene Wort zusammen mit den gefundenen Anagramme als „word set“ (Typ liste). Weiter zum Kernpunkt 3 (Z.100):

```
def get_suffled_word(wordSet):
  """Gets a shuffled word that are not in the wordSet."""
  word = list(wordSet[0])
  random_shuffle(word)
  from itertools import combinations
  for charCombi in combinations(list(word), len(wordSet[0])):
    wordCombined = "".join(charCombi)
    if wordCombined not in wordSet:
      return wordCombined
  return ""
```

Ich vermische das Wort indem ich das erste Wort aus dem `wordSet` die `str` als eine `liste` von `chars` mit `random.suffle()` (impotiert als `random_shuffle()`) vermische. Dabei überprüfe ich ob diese Mischung kein Wort aus versehen erschaffen hat. Dafür iteriere ich durch jede Kombination des Wortes über `combinations` von `itertools`. Falls ein Wort doch aus versehen erschaffen wurde, wird dann stattdessen eine leere `str` zurückgegeben. Bei dies fragt mein Programm nochmal nach eine neue Mischung des Wortes (Sehe selbst unter Z.748).

Anzeige der Buchstaben
Sehen wir mal einen Teil der Erschaffung der Buchstaben, bzw. wo eine Reihe von Buchstaben hinzugefügt werden (Z.441):

```
class Table(tk.Frame):
[...]
  def add_new_row(self, turn, currentWord, binder_onBoxClick, selectedBoxSwap=[-1, -1],
hideBoxesOnCreation=False):
    """Adds a new row to the table."""
    self.turn = turn
    self.currentWord = currentWord
    self.binder_onBoxClick = binder_onBoxClick
    row = Row(self, self.turn*60 + 35, self.turn, self.currentWord,
self.binder_onBoxClick, hideBoxesOnCreation)
    self.frameRows.append(row)
    # Selected boxes
    for i in range(2):
      if selectedBoxSwap[i] >= 0:
        row.labelBoxes[selectedBoxSwap[i]].config(background=SELECT_COLOR)
```

Es gibt die Objekte `Table` (Inhärenz von `tk.Frame`) und `Row` (gleiche Inhärenz). Diese Methode von der Tabelle (`Table`), die wir sehen, benötigt Argumente (Variablen) zur Erschaffung einer Reihe (`Row`). Erstmal speichert diese gesehene Methode einige Argumente als Attribute und erzeugt eine neuen Reihen Objekt, wo `self.turn` basierend auf der y-Position der Reihe ist. Diese Reihe wird auch in der Tabelle gespeichert in einer Liste (`self.FrameRows.append(row)`). Nachträglich werden, falls bei Ladung von gespeicherten Spielen, die Boxen (Buchstaben) in der Reihe gewählt.

Datenextraktion
Wie diese TXT-Dateien erschaffen sind, habe ich Wörter aus der SQL-Datei von der Leipziger Wörterdatenbank extrahiert. Dabei habe ich mir in der Konsole über MariaDB diese kompakte SQL-Query verwendet: (Sehe auch unter `data\deu_mixed-typical_2011_10K\ExtractionQuery.txt`)

Einen Beispiel mit einer Wörterlänge von 5:

```
SELECT TRIM(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(word, 'ä', 'ae'),
'Ä', 'Ae'), 'ö', 'oe'), 'Ö', 'Oe'), 'ü', 'ue'), 'Ü', 'Ue'), 'ß', 'ss')) AS word FROM
words
WHERE LENGTH(word) = 5
 AND word REGEXP '^[A-Za-zÄÖÜäöüß]+$'
INTO OUTFILE 'C:/Users/[USERNAME]/Downloads/OUTPUT_FILES/words5.txt';
```

Diese SQL-Query ist nur einzeilig und ist nach der Datenbank angepasst. Dabei werden alle Sonderzeichen bzw. Umlaute und ß mit REPLACE() zu zweier Buchstaben umgewandelt. Es werden nur Wörter gewählt, die eine Länge von 5 haben (`LENTH(word) = 5`) und noch zur Sicherheit aus dem deutschen Alphabet kommen (`word REGEXP '^[A-Za-zÄÖÜäöüß]+$':`). Danach werden die Wörter in der neuen Textdatei words5.txt eingeschrieben.

Herausforderungen

- **Datenbankintegration:**
  - Das Arbeiten mit MariaDB stellte eine Herausforderung dar, da ich keine SQL-Vorkenntnisse hatte und viel Recherche begehen musste. Bei den vorigen Versuchen hatte ich die Daten von OpenThesaurus extrahiert. Jedoch empfand ich, dass die Wörterauswahl nicht alltagstauglich genug sei und habe letztlich eine andere Datenquelle verwendet, wo die SQL-Datenstruktur anders ist.
  - Es gab auch einige Versuche, den Prozess von Datenextraktion zu automatisieren über Python und MySQL, dennoch scheiterte dies. Also habe ich es manuell extrahiert.
- **GUI-Programmierung:** Die erweiterte Verwendung von Tkinter erforderte den Umgang mit Klassen und Konzepten, die nicht im Unterricht behandelt wurden. Es gab auch an einigen Stellen, wo ich Tkinter umgehen musste, bspw. bei Bindung von Labels konnte keine Argumente eingereicht werden. Als Lösung habe ich Lambda-Expressionen als Funktionsdefinition verwendet (Z.70).
- **Zeitmanagement:** Aufgrund von ungeplanten Ansätzen und Zeitüberschreitungen konnte die vollständige Version des Projekts leider erst nach der Frist abgeben. Das Projekt war teils zu groß für die „geplante“ Zeit von einer Woche. Dabei musste ich weitere Ideen wie eine größere Menge an Buchstaben als die Länge des gesuchten Wortes eingehen.

Fazit

Das Projekt „Scrambled!“ hat sein Ziel, ein Wissensspiel zu entwickeln, dass die Problemlösungsfähigkeiten von Anagrammen fördert, größtenteils erreicht. Es ermöglichte mir, mein Wissen in Python, objektorientierter Programmierung und GUI-Design mit Tkinter erheblich zu erweitern. Die Umsetzung der Kernfunktionen, wie bspw. die zufällige Wörterauswahl, das Mischen der Buchstaben und die Darstellung über eine grafische Benutzeroberfläche, war erfolgreich.

Allerdings gab es auch einige Herausforderungen, insbesondere bei der Datenbankintegration und der GUI-Programmierung, was mir wertvolle Lernerfahrungen gab. Es konnten nicht alle geplanten Funktionen implementiert werden, jedoch bietet dieses Programm Potenzial für zukünftige Erweiterungen.

Zusammenfassend habe ich durch dieses Projekt mein Wissen zur Implementierung in Python und Tkinter erweitert und gelernt, für nächstes Mal besseren Zeitmanagement und Planung einzuführen. Das Projekt war herausfordernd, aber es hat sich gelohnt und Spaß gemacht.