Modul 122, Dokumentation LB2

SwissAerialGuessr: Konsolenspiel, um Orte in der Schweiz anhand ihres Luftbildes zu erraten

Modul	IET-122 – Praxisarbeit
Eingereicht von	Jonas Vetsch Stammgruppe LeutertHerbstVetsch
Eingereicht bei	Claude Fankhauser
Datum	8. Januar 2026

Änderungsverzeichnis

Datum	Version	Änderung	Autor
17.04.2023	0.5	Vorlage	Urs Dummermuth
30.12.2025	1	Separat geführte Dokumentation und handschriftliche Notizen in diese Dokumentationsvorlage hier übertragen. Reinschrift, Grammatikprüfung und stilistische Schlussüberarbeitung. Bereit zur Abgabe.	Jonas Vetsch

Inhaltsverzeichnis

1 Ziele und Anforderungen

1.1 Einleitung

Die Idee zu diesem Projekt entstand aus dem Wunsch, die im ersten Modul erarbeiteten Grundlagen der API-Anbindung in ein komplexeres, interaktives System zu überführen. Während im vorangegangenen Projekt die reine Automatisierung von Downloads für einen geschäftlichen Zweck im Vordergrund stand, liegt der Fokus von „SwissAerialGuessr“ auf der spielerischen Vermittlung von Geografiekenntnissen und der technischen Anbindung einer selbst erstellten relationalen Datenbank.

Ich verfüge bereits über solide Kenntnisse in Python und dem Umgang mit der GeoAdmin-API, auch wegen der Erfahrungen, die ich im vorherigen Projekt machen durfte. In dieser Arbeit möchte ich einen Schritt weitergehen: Weg von der sequenziellen Abarbeitung einer Textdatei, hin zu einem dynamischen Konsolenspiel, das Daten persistent in einer SQL-Datenbank verwaltet. Ein zentrales Lernziel ist dabei das Verständnis für das Zusammenspiel zwischen Skriptlogik, externen Schnittstellen (APIs) und einem Datenbanksystem (MySQL).

1.2 Rahmenbedingungen (Umfeld)

Das Projekt «SwissAerialGuessr» zielt auf ein Umfeld von Nutzer*innen ab, die entweder bereits mit Geografie-Quizzen vertraut sind oder offen sind, diese auszuprobieren. Als massgebliche Inspiration dient das bekannte Geografie-Quizz «GeoGuessr», das damit begeistert, Orte anhand von Bildmaterial zu identifizieren. «GeoGuessr» nutzt Bildmaterial von Google Street View. Mein Skript versucht ein ähnliches Spiel zu kreieren, das allerdings mit Luftbildern (also der Top-Down-Perspektive) aus der Schweiz arbeitet. Für die Schweiz sind hochauflösende Luftbilder von Swisstopo über deren API einfach zu beziehen und bilden das Fundament für die Ermöglichung dieses Spiels.

Die Zielgruppe für dieses Skript besteht aus Personen, die eine Leidenschaft für Geografie haben und ihr Wissen über die verschiedenen Regionen der Schweiz auf spielerische Weise prüfen und erweitern möchten. Nutzer, die solche Ratespiele schätzen, erwarten eine flüssige Spielerfahrung und eine faire Bewertung ihrer Eingaben. Aus diesem Grund muss das Skript so programmiert sein, dass es technisch stabil läuft und sollte durch eine intelligente Antwortprüfung auch Tippfehler oder alternative Schreibweisen als korrekte Antworten akzeptieren, um Frustration zu vermeiden und den Spielpass zu fördern.

Damit das Spiel im praktischen Einsatz überzeugt, soll ein Fokus auf die Dynamik und den Langzeitwert gelegt. Durch die Anbindung einer MySQL-Datenbank wird sichergestellt, dass die Nutzer von Anfang an eine grosse Auswahl an Orten haben, die abgefragt werden können. Zudem können damit die möglichen Orte sehr einfach von Nutzern selbst erweitert werden und Nutzer können ihre persönlichen Fortschritte über Statistiken (ebenfalls dank der Datenbank) verfolgen.

Die Essenz bei der Entwicklung dieses Skripts besteht darin, die komplexen Hintergrundprozesse (API-Abfragen, Datenverwaltung, Dateimanagement) so zu gestalten, dass für den Endnutzer ein einfaches, motivierendes und lehrreiches Spielerlebnis in der Konsole entsteht.

1.3 Zweck des Skriptes

„SwissAerialGuessr“ ist ein konsolenbasiertes Ratespiel, das den Nutzer herausfordert, Schweizer Orte ausschliesslich anhand von hochauflösenden Luftbildern von Swisstopo zu identifizieren. Das Skript bietet eine vollständige Spiellogik:

- **Datenbankgestützte Auswahl**: Die zu erratenden Orte werden nicht aus einer statischen Textdatei gelesen, sondern es werden für jede Spielrunde zehn zufällige Orte aus einer SQL-Datenbank ausgewählt.
- **Intelligente Antwortprüfung**: Das Skript muss die Antworten des Nutzers prüfen. Dabei stellt sich die Herausforderung, dass ein Nutzer zwar grundsätzlich korrekt antworten kann, jedoch z.B. durch Tippfehler in seiner Antwort oder durch die Verwendung eines anderen Wortes («Olympiamuseum» vs. «olympisches Museum») «falsch» antwortet. Das Skript sollte auch solche leicht abweichenden Antworten als richtig einstufen.
- **Spielerstatistik**: Spielergebnisse werden nach jeder Runde in der Datenbank gespeichert, sodass der Nutzer eine Auswertung über seine durchschnittliche Punktzahl und die Anzahl der gespielten Runden erhält. So sieht der Nutzer, wie er im Vergleich zu seinen vorherigen Leistungen abgeschnitten hat.
- **Caching**: Um die API-Last zu minimieren und die Performance zu steigern, lädt das Skript ein Luftbild nur dann herunter, wenn es notwendig ist.

Das Ziel ist es, ein technisch robustes Konsolenspiel zu schaffen, das zeigt, welches Potenzial in der GeoAdmin API, in den Luftbildern von Swisstopo und letztlich in den vielen spannenden Orten der Schweiz steckt.

1.4 Ziele nach SMART

1. Bis zum Abschluss des Projekts ist eine funktionale MySQL-Datenbank erstellt und angebunden. Sie erhält mindestens 50 verschiedene Schweizer Orte (Name, Adresse und ggf. Koordinaten). Das Spiel soll Orte aus der SQL-Tabelle «ort» laden und nach jeder Spielrunde die erreichte Punktzahl des Spielers in die Tabelle «spiel» speichern.
2. Bis zum Abschluss des Projekts validiert das Skript die Antworten des Benutzers mit einem Algorithmus, der auch Antworten als korrekt erkennt, wenn sie leicht von der Antwort in der Datenbank abweichen, z.B. durch Tippfehler oder durch die Verwendung eines anderen Wortes («Olympiamuseum» vs. «olympisches Museum»). Das System soll Benutzereingaben so validieren, dass bei einer Übereinstimmung von mindestens 78% zum Datenbanknamen (z.B. „Bern HB“ statt „Bern Bahnhof“) die Antwort als korrekt gewertet wird.
3. Bis zum Abschluss des Projekts verfügt das Skript über ein System, das ein Luftbild nur dann herunterlädt, wenn es notwendig ist (z.B. Bild ist nicht im Cache vorhanden oder die Adresse des Ortes wurde in der Datenbank angepasst). Wenn es nicht notwendig ist, das Bild herunterzuladen, wird es aus dem Cache geladen.

1.5 Datenbankstruktur

Die Datenbank für dieses Skript besteht aus zwei Tabellen. Die Tabelle «ort» beinhaltet die für das Spiel zur Verfügung stehenden Orte, zu denen das Spiel ein Lutftbild anzeigen kann. Ein Ort braucht mindestens einen Namen, eine Adresse und einen Wert für die Zoomstufe (legt fest, wie fern/nahe der Bildausschnitt des Luftbildes gewählt wird).

```
create table ort (
    ID_Ort int primary key auto_increment,
    Name varchar(250) not null,
    Adresse varchar(250) not null,
    Nordwert double,
    Ostwert double,
    Zoom int DEFAULT 500,
    UpdateFlag int not null DEFAULT 0
);
```

Zudem braucht das Skript die Tabelle «spiel» in der nach jeder Runde die erreichte Punktzahl des Nutzers und ein Zeitstempel gespeichert wird.

```
CREATE TABLE spiel (
    ID_Spiel int primary key auto_increment,
    Punktzahl int,
    Zeitstempel timestamp DEFAULT CURRENT_TIMESTAMP
);
```

1.5.1 Funktionale Anforderungen

Nr.	Anforderung	Prio
F1	**SQL-Statements**: Das Skript verwendet mindestens drei unterschiedliche SQL-Befehle (z. B. SELECT, INSERT, UPDATE).	Muss
F2	**Datenbank-Abfrage**: Das Skript wählt per Zufall einen Datensatz aus der Tabelle «ort» aus.	Muss
F3	**Interaktivität**: Das Skript speichert das Luftbild des Ortes, der vom Nutzer erraten werden muss unter einem generischen Namen (um nicht den Ort zu verraten) und fordert den Nutzer zur Eingabe seiner Antwort auf.	Soll
F4	**Fehlertoleranter Abgleich**: Das Skript berechnet die Ähnlichkeit der Nutzereingabe zum Zielort mittels eines Algorithmus, der auch geringe Fehler zulässt (z.B. Tippfehler des Nutzers).	Soll
F5	**Ergebnis-Persistenz**: Das Skript schreibt die erreichte Punktzahl nach Spielende in die Tabelle «spiel» der Datenbank.	Muss
F6	**Statistik-Ausgabe**: Das Skript berechnet den historischen Punktedurchschnitt mit einer SQL-Aggregatfunktion und gibt diesen aus.	Muss
F7	**Caching der Bilder**: Das Skript prüft, ob ein Bild bereits lokal vorhanden ist, bevor ein API-Download gemacht wird.	Soll
F8	**Caching der Koordinaten:** Das Skript prüft, ob die Koordinaten zu einem Ort bereits lokal vorhanden sind, bevor eine API-Anfrage zur Auflösung der Adresse in Koordinaten gemacht wird.	Soll
F9	**Externe Steuerung**: Das Skript wird über eine externe Datei (z. B. config.txt oder Datenbank-Einträge) gesteuert oder konfiguriert.	Muss
F10	**Personalisierung**: Das Skript speichert die bevorzugte Konsolenfarbe in einer lokalen Konfigurationsdatei.	Muss
F11	**Nutzerführung:** Das Skript informiert über allfällige Wartezeiten, während im Hintergrund das Bild geladen wird.	Soll

1.5.2 Nicht-Funktionale Anforderungen

Nr.	Anforderung	Prio
NF1	**Nutzerführung:** Das Skript führt den Nutzer mit sinnvollen Anweisungen und Erklärungen auf der Konsole durch das Spiel.	Soll
NF2	**Dateistruktur**: Das Skript ist in einer sauberen Ordnerstruktur organisiert und wird als ein einziges ZIP-Archiv abgegeben.	Muss
NF3	**Modularer Aufbau**: Das Skript ist für die Wart- und Testbarkeit in Module (API, Database, …) unterteilt.	Soll
NF4	**Robustheit**: Das Skript fängt Datenbank-Verbindungsfehler ab und gibt eine verständliche Fehlermeldung aus.	Muss
NF5	**Errorhandling:** Das Skript hat detailliertes Errorhandling und zeigt dabei sinnvolle Fehlermeldungen an.	Soll
NF6	**API-Drosselung**: Das Skript implementiert Logging, um damit das Fair-Use-Limit der API einzuhalten.	Kann
NF7	**Wartbarkeit**: Das Skript ist durchgehend mit Kommentaren versehen.	Muss
NF8	**Performance**: Das Skript validiert die Antwort in weniger als 0.5 Sekunden.	Soll
NF9	**Datenmenge und Datenqualität:** Die Datenbank umfasst mindestens 20 Orte inkl. Adresse und Zoomstufe, die im Spiel erscheinen können. Die Orte haben eine sinnvoll gewählte Zoomstufe, damit der Ort in sinnvoller Grösse abgebildet wird. Die Adressen sind, wenn möglich, präzise gewählt, damit der gesuchte Ort jeweils in der Bildmitte liegt.	Soll
NF10	**Dokumentation:** Die Dokumentation enthält alle geforderten Kapitel und ein korrektes Inhaltsverzeichnis.	Muss

2 Ablaufdiagramm

2.1 PAP

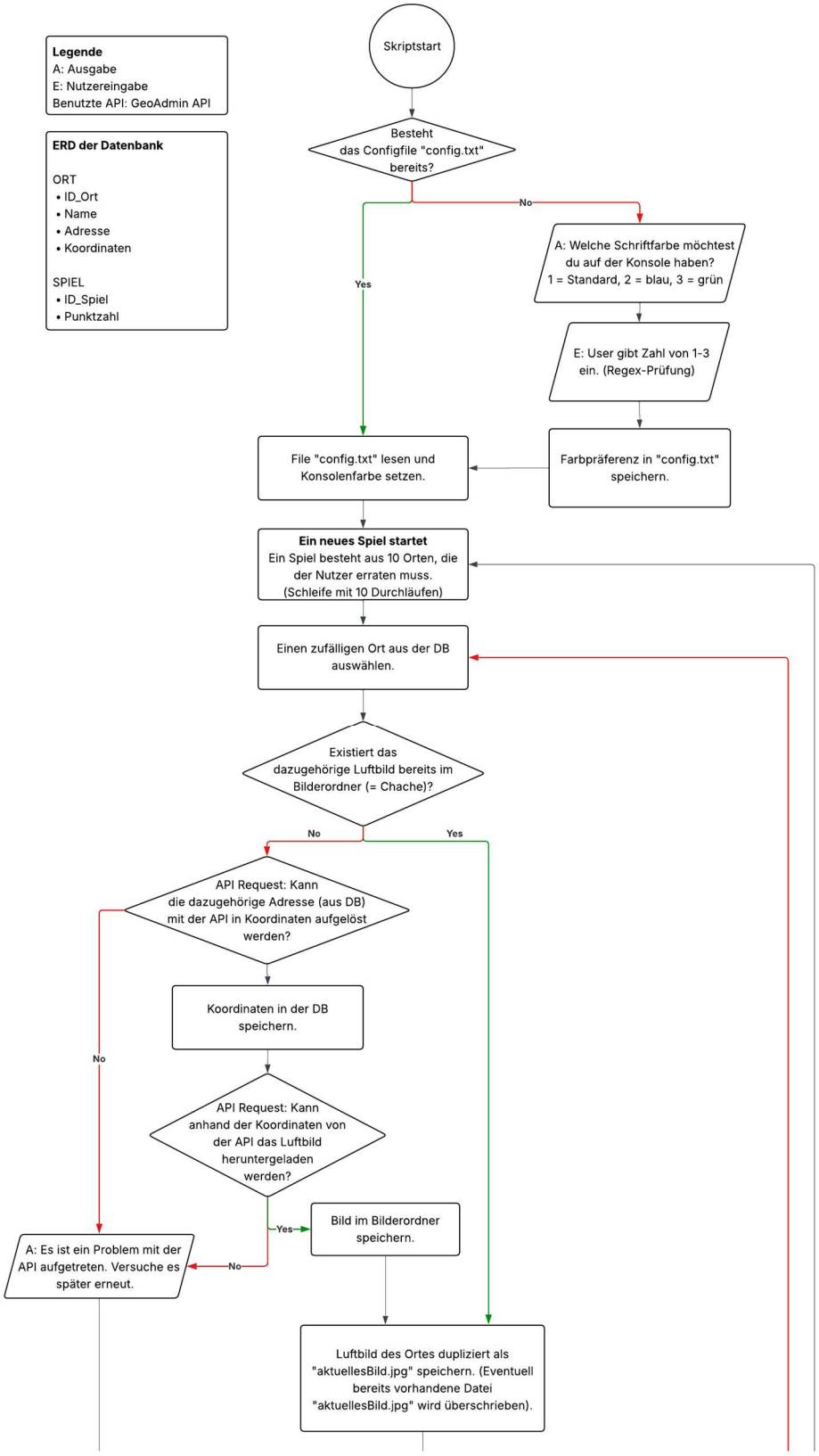

Abbildung 1 PAP des Skripts, Teil 1

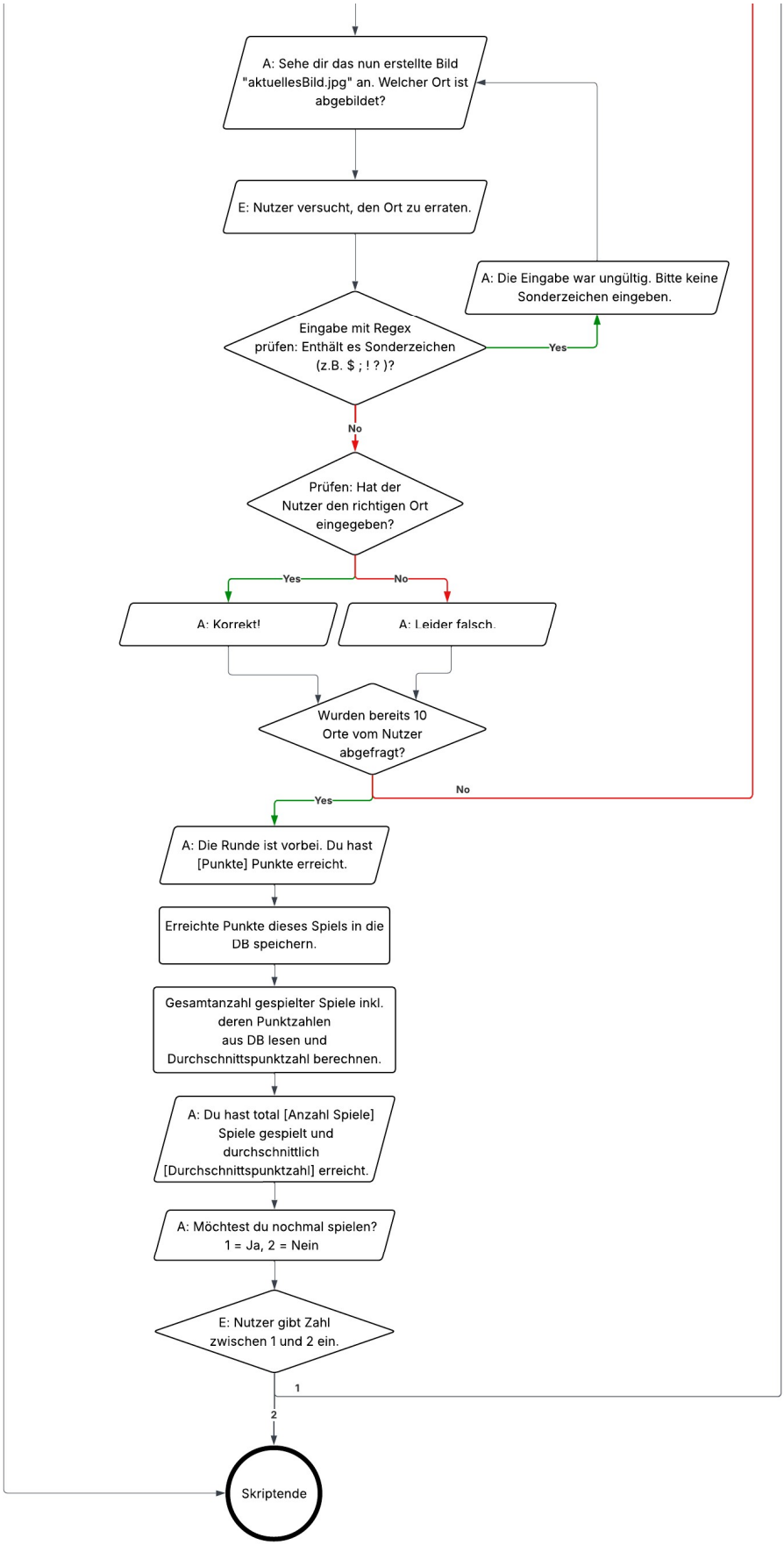

Abbildung 2 PAP des Skripts, Teil 2

2.2 Kommentar / Beschreibung

Das Skript ist modular aufgebaut und startet mit der Konfiguration der Benutzeroberfläche. Zunächst prüft das Skript, ob eine **`config.txt`** mit Farbcodes existiert und liest daraus die Farbe für die Konsole ein. Falls die Datei nicht existiert, wird der Nutzer aufgefordert, eine Konsolenfarbe zu wählen. Die Eingabe des Nutzers wird mittels Regex validiert und für zukünftige Starts dauerhaft gespeichert.

Das eigentliche Spiel läuft in einer Schleife von zehn Runden ab. Pro Runde wählt das Skript via Zufallsprinzip einen Datensatz aus der Datenbank aus. Vor jedem Zugriff auf die Swisstopo-API greift ein Caching-Mechanismus: Das Skript prüft, ob das Luftbild bereits lokal im Bilderordner existiert. Nur wenn nötig werden die Koordinaten über die API aufgelöst und das Bild heruntergeladen. Um den Zielort nicht zu verraten, wird die Datei für die Anzeige als **`aktuellesBild.jpg`** dupliziert.

Als Nächstes gibt der Nutzer seinen Tipp ab, welcher ebenfalls mit Regex auf unzulässige Sonderzeichen geprüft wird. Danach wird die Antwort des Nutzers mit dem Namen des Ortes verglichen. Nach Abschluss aller Runden schreibt das Skript die Gesamtpunktzahl in die MySQL-Tabelle **`spiel`**. Zum Ende berechnet das Skript mittels SQL-Aggregatfunktion den Punktedurchschnitt aller bisherigen Partien und gibt diese Statistik auf der Konsole aus.

3 Skript (Realisierung)

3.1 Technologie Python und verwendete Bibliotheken

Für die Umsetzung wurde die Programmiersprache Python (Version 3.14) gewählt. Python eignet sich besonders durch seine umfangreichen Bibliotheken für die Anbindung von APIs und Datenbanken. Folgende Kern-Bibliotheken kommen in dem Skript zum Einsatz:

- **SQLAlchemy & PyMySQL**: Diese werden verwendet, um die Verbindung zur MySQL-Datenbank herzustellen. SQLAlchemy ermöglicht eine saubere Abwicklung von SQL-Abfragen und sorgt dafür, dass die Datenbankverbindung effizient verwaltet wird.
- **Requests**: Diese Bibliothek dient dazu, die HTTP-Anfragen an die GeoAdmin API (Swisstopo) zu senden, um Koordinaten abzufragen und die Luftbilder herunterzuladen.
- **Fuzzywuzzy**: Um die Benutzerfreundlichkeit zu erhöhen, nutzt das Skript diese Bibliothek für den fehlertoleranten Textvergleich beim Auswerten der Nutzerantworten während dem Spiel. Der verwendete Algorithmus **`token_set_ratio`** basiert auf dem Levenshtein-Abstand und erweitert diesen zusätzlich: Er zerlegt die Eingaben in einzelne Wörter (Tokens) und vergleicht diese unabhängig von ihrer Reihenfolge. Dadurch erkennt das Skript eine Antwort auch dann als korrekt, wenn der Nutzer Wörter vertauscht oder leicht anders schreibt (z.B. wegen eines Tippfehlers).
- **Regex (re)**: Diese Bibliothek wird für die Validierung der Benutzereingaben (z. B. bei der Farbwahl oder den Antworten während dem Spiel) genutzt, um zu prüfen, ob der Nutzer nur erlaubte Zeichen eingibt.

3.2 Ein- und Ausgabe

Die Interaktion mit dem Nutzer findet ausschliesslich über die Konsole statt.

Eingabe: Das Skript erwartet vom Nutzer numerische Eingaben für die Konfiguration (Farbauswahl) und textbasierte Eingaben für das Erraten der Ortsnamen. Die Eingaben werden vor der Verarbeitung bereinigt und mit Regex auf unerlaubte Zeichen geprüft.

Ausgabe: Die Textausgabe auf der Konsole erfolgt in der vom Nutzer gewählten Farbe. Bilder werden nicht direkt in der Konsole ausgegeben. Stattdessen wird das Bild des zu erratenden Ortes immer am selben Ort unter demselben Namen gespeichert. Das Skript versucht jeweils, das Bild auch automatisch zu öffnen mit der für die Bildansicht definierten App des jeweiligen Betriebssystems. Wenn dies nicht funktioniert (je nach Betriebssystem und vorhandenen Programmen kann dies fehlschlagen) öffnet der Nutzer das Bild händisch.

3.3 Kontrollstrukturen

Die Logik von dem Skript basiert auf drei wesentlichen Kontrollstrukturen:

- **Die Hauptschleife (Game-Loop)**: Eine while-Schleife umschliesst das gesamte Programm, sodass der Nutzer nach einer Runde direkt eine weitere starten kann, ohne das Skript neu aufrufen zu müssen.
- **Die Spielrunde**: Eine for-Schleife steuert die 10 Durchgänge pro Spiel.
- **Die UpdateFlag**: In der MySQL-Tabelle `ort` gibt es die Spalte UpdateFlag. Wenn dort eine 1 steht, ignoriert das Skript sowohl die gespeicherten Koordinaten (falls vorhanden), als auch das bereits gecachte Bild (falls vorhanden) und lädt alles (Koordinaten und Bild) frisch von der API herunter. Danach setzt das Skript diese Flag selbst wieder auf 0. Das ist für die Pflege der Datenbank sehr nützlich, wenn ein Datensatz verändert wird (z.B. Adresse anpassen). Damit wird sichergestellt, dass Änderungen in der Datenbank den Cache überschreiben.
- **Bedingte Verzweigungen (Caching)**: Eine zentrale if-else-Struktur prüft vor jedem API-Call zum Download eines Luftbildes zwei Bedingungen: Existiert das Bild bereits lokal? Und ist die UpdateFlag in der MySQL-Datenbank auf 0? Nur wenn der Cache leer ist oder mit der UpdateFlag ein Update erzwungen wird, wird die Download-Logik ausgeführt. Damit werden API-Anfragen gespart und die Infrastruktur der GeoAdmin API geschont.

3.4 Regex

Um die Robustheit des Spiels zu erhöhen, wird die Benutzereingabe in der Funktion `getValidAnswer()` mithilfe von Regulären Ausdrücken (Regex) auf unzulässige Zeichen geprüft. Während das Skript leere Eingaben direkt abweist, stellt die Regex-Prüfung sicher, dass keine Sonderzeichen wie `$`, `%`, `*` oder `;` eingegeben werden. Das verwendete Pattern `r"[!#$%()*+:;<=>?@_{|}§€£¥¡¿«»]"` ist dabei so gewählt, dass es gezielt störende Symbole blockiert, während Zeichen wie Umlaute oder Bindestriche erlaubt bleiben, da diese in Ortsnamen vorkommen können.

Dieser Validierungsschritt ist die erste Schicht vor der eigentlichen Verarbeitung der Eingabe. Diese wird an die Spiellogik übergeben, wo sie zusätzlich normalisiert wird. Dadurch wird der Vergleich mit dem Datenbankeintrag vereinfacht. Durch diese Kombination aus Regex-Filterung und anschliessender Bereinigung wird sichergestellt, dass das Skript auch bei unkonventionellen Eingaben stabil funktioniert und die Nutzereingaben mit hoher Zuverlässigkeit ausgewertet.

3.5 Datenbank

Alle Daten werden in einer relationale MySQL-Datenbank gespeichert. Diese besteht aus zwei Tabellen.

- **Tabelle `ort`**: Speichert Name, Adresse, Koordinaten und UpdateFlag der Orte.
- **Tabelle `spiel`**: Dient der Archivierung der Spielergebnisse mit Zeitstempel.

Die Verbindung wird über das Modul **`engine.py`** mittels SQLAlchemy realisiert. Dies erlaubt es, SQL-Statements sauber vom restlichen Programmcode zu trennen und die Datenbankverbindung effizient zu verwalten. Durch den Einsatz von **`ORDER BY RAND()`** in der Funktion zum Laden eines neuen Bildes stellt das Skript sicher, dass bei jedem Spieldurchgang ein neuer, zufälliger Ort präsentiert wird.

3.6 Modularität

Um die Wartbarkeit zu verbessern, wurde das Skript in mehrere Module aufgeteilt:

- **`main.py`**: Der Einstiegspunkt und die Steuerung der Spielschleife.
- **`api.py`**: Enthält alle Funktionen für die Kommunikation mit Swisstopo.
- **`query.py`** & **`engine.py`**: Verwalten die SQL-Statements und den Datenbank-Handshake.
- **`game.py`**: Beinhaltet die Logik für den Bildabgleich und das Fuzzy-Matching.

4 Integration und Sicherheit

4.1 Implementierung

Die verschiedenen Programmteile arbeiten wie folgend zusammen. Das Hauptprogramm in **`main.py`** verbindet die Datenbank, die GeoAdmin API und die Spiellogik zu einem kohärenten Ablauf. Wenn ein Ort aus der Datenbank geladen wird, prüft das System automatisch, ob bereits ein Bild vorhanden ist. Falls nicht, wird über das Modul **`api.py`** direkt eine Verbindung zur GeoAdmin-API aufgebaut, um die nötigen Koordinaten und das Luftbild zu laden und für später zu speichern.

Für die Sicherheit und Stabilität wurden zwei wichtige Mechanismen eingebaut: Erstens, um die GeoAdmin API nicht zu überlasten, sorgt das Modul **`logs.py`** dafür, dass das Skript bei zu vielen Aufrufen automatisch kurz pausiert. Das verhindert, dass die IP-Adresse des Nutzers bei der API gesperrt wird.

Zweitens wird in **`userInteraction.py`** durch die Eingabeprüfung mit Regex verhindert, dass ungültige Zeichen eingegeben werden, was die Effektivität des Abgleichs der Nutzereingaben mit der Datenbank erhöht.

4.2 Sicherheit und Validierung

Sicherheit bedeutet bei diesem Projekt vor allem den Schutz vor Programmabstürzen durch fehlerhafte Eingaben oder externe Angriffe:

Input-Validierung: Alle Nutzereingaben, sowohl bei der Farbwahl als auch bei der Antwortabgabe, werden mittels Regex auf unerlaubte Eingaben geprüft.

API-Sicherheit: Das Skript nutzt ein selbst gebautes Logging-System, um die Fair-Use-Policy der GeoAdmin API einzuhalten. Durch die Drosselung der Anfragen wird sichergestellt, dass die IP-Adresse des Nutzers nicht aufgrund von zu vielen Anfragen in kurzer Zeit gesperrt wird.

Errorhandling: Das Skript hat ein umfassendes Errorhandling, womit sichergestellt wird, dass das Skript nicht durch zu erwartende Fehler abbricht und dass Fehler verständlich an den Nutzer kommuniziert werden, damit dieser einen Hinweis für seine Fehlersuche bekommt. Zentral bei der Implementierung des Errorhandlings waren vor Allem die Bereiche Ordner- und Pfadvalidierung, API-Requests und Kommunikation mit der Dantenbank.

SQL-Injection: Ein Schwachpunkt im aktuellen Stand des Skripts ist die Anfälligkeit für SQL-Injection in den Datenbankmodulen (speziell in **`query.py`**). Da die SQL-Befehle mittels F-Strings zusammengesetzt werden, könnten theoretisch schädliche SQL-Kommandos über die Benutzereingabe eingeschleust werden. Da der Fokus dieser Arbeit jedoch primär auf der API-Integration und der Spiellogik lag, wurde auf die aufwendigere Implementierung von Parameterized Queries verzichtet. Dieses Sicherheitsrisiko ist für das vorliegende Projekt jedoch zu vernachlässigen: Da es sich um ein reines Offline-Singleplayer-Spiel handelt, würde ein Angreifer durch eine SQL-Injection lediglich sein eigenes lokales System bzw. seine eigene Datenbank manipulieren. Da der Nutzer die Datenbank zudem selbst hosten und verwalten muss, besitzt er ohnehin vollen Zugriff auf alle Daten. Ein Missbrauch durch Dritte ist ausgeschlossen, da keine Web-Schnittstelle existiert.

4.3 Kompatibilität

Das Skript wurde so entwickelt, dass es auf verschiedenen Systemen lauffähig ist:

Die Wahl «Python»: Mit der Wahl der Sprache «Python» wurde bereits einen wichtigen Schritt in Richtung Kompatibilität gemacht, da Python auf allen Systemen installiert werden kann.

Plattformunabhängigkeit: Durch Abfragen wie os.name erkennt das Skript, ob es auf Windows oder einem UNIX-basierten System (Linux/macOS) läuft, und passt Befehle wie derjenige zum Öffnen des Bildes automatisch an.

Bibliotheken: Die Verwendung von Standard-Bibliotheken in Kombination mit einer requirements.txt stellt sicher, dass die Umgebung auf jedem Rechner mit Python 3.x schnell und identisch reproduziert werden kann.

Datenbank: Der Nutzer muss die Datenbank erstellen, ehe er das Spiel spielen kann. Dafür wird sowohl ein SQL-Skript als auch eine CSV-Datei mitgeliefert, womit der Nutzer mit minimalem Zeitaufwand spielbereit ist. Das SQL-Skript und die CSV-Datei folgen dabei Standardkonventionen, was die Kompatibilität nochmal verbessert.

4.4 Betrieb und Wartung

Für den langfristigen Betrieb und die einfache Wartung wurden spezifische Mechanismen eingebaut:

- **Caching-System:** Um Bandbreite zu sparen und die Ladezeiten zu verkürzen, werden einmal heruntergeladene Bilder lokal gespeichert. Dies reduziert die Abhängigkeit von der API-Verfügbarkeit.
- **Update-Mechanismus:** Über das UpdateFlag in der Datenbank können einzelne Datensätze markiert werden. Das Skript erkennt diese Markierung und aktualisiert diese Daten explizit von der API, ohne dass der Administrator manuell in den Bildordner eingreifen muss.
- **Modularer Aufbau:** Durch die Aufteilung in spezialisierte Dateien (z. B. `api.py` für Schnittstellen, `query.py` für Datenbank) können Änderungen an einer Komponente vorgenommen werden, ohne die Logik der anderen Teile zu gefährden.

5 Usecases und Testfälle

5.1 Usecase: Interaktives Geografie-Quiz „SwissAerialGuessr“

Dieser Usecase beschreibt den Ablauf aus der Sicht eines Nutzers, der spielerisch seine Kenntnisse der Schweizer Geografie testen und verbessern möchte. Im Vordergrund steht dabei die interaktive Abfrage des Nutzers und die Auswertung seiner Eingaben.

Der Ablauf beginnt mit dem Start des Skripts in der Konsole. Falls ich das Skript zum ersten Mal nutze, werde ich durch eine kurze Einführung begrüsst und kann meine bevorzugte Konsolenfarbe wählen. Diese Einstellung merkt sich das Skript für alle zukünftigen Runden.

Sobald das Spiel startet, übernimmt das Skript die gesamte Verwaltung des Spielablaufs:

- **Zufällige Auswahl:** Das Skript wählt aus der MySQL-Datenbank automatisch einen zufälligen Ort aus, den ich erraten muss.
- **Intelligentes Laden**: Das Skript prüft, ob das Bild bereits im Cache liegt. Falls nicht, werden die Koordinaten und das Bild von der Swisstopo-API heruntergeladen.
- **Visuelle Präsentation:** Das Luftbild öffnet sich automatisch in meinem Standard-Bildbetrachter, sofern diese Python-Funktion auf meinem Betriebssystem unterstützt wird. Anderenfalls öffne ich das Bild selbst. Ich sehe das Bild, weiss aber noch nicht, um welchen Ort es sich handelt, dies auch deshalb, weil die Datei neutral benannt ist.
- **Interaktive Eingabe**: Ich gebe meinen Tipp direkt in der Konsole ein. Dabei muss ich nicht auf jedes Detail achten – dank des eingebauten Fuzzy-Matchings erkennt das Skript meine Antwort auch dann, wenn ich einen Tippfehler mache oder die Wörter in vertauschter Reihenfolge eingebe.
- **Direktes Feedback:** Ich erfahre sofort, wie viel Prozent Ähnlichkeit meine Antwort mit dem echten Namen hatte, welches der gesuchte Ort war und ob der Punkt gezählt wurde.

Nach genau zehn Runden präsentiert mir das Skript mein Endergebnis. Besonders spannend ist hier der Datenbank-Aspekt: Das Skript zeigt mir nicht nur meine aktuelle Punktzahl, sondern vergleicht diese auch mit meinem historischen Durchschnitt aus der MySQL-Datenbank. So sehe ich direkt, wie gut ich im Vergleich zu den vergangenen Spielrunden abschneide.

Dieser Usecase zeigt, wie das Skript komplexe Hintergrundprozesse (API-Requests, SQL-Abfragen, Caching) in ein simples und unterhaltsames Benutzererlebnis verwandelt.

Beweis der Durchführung Usecase

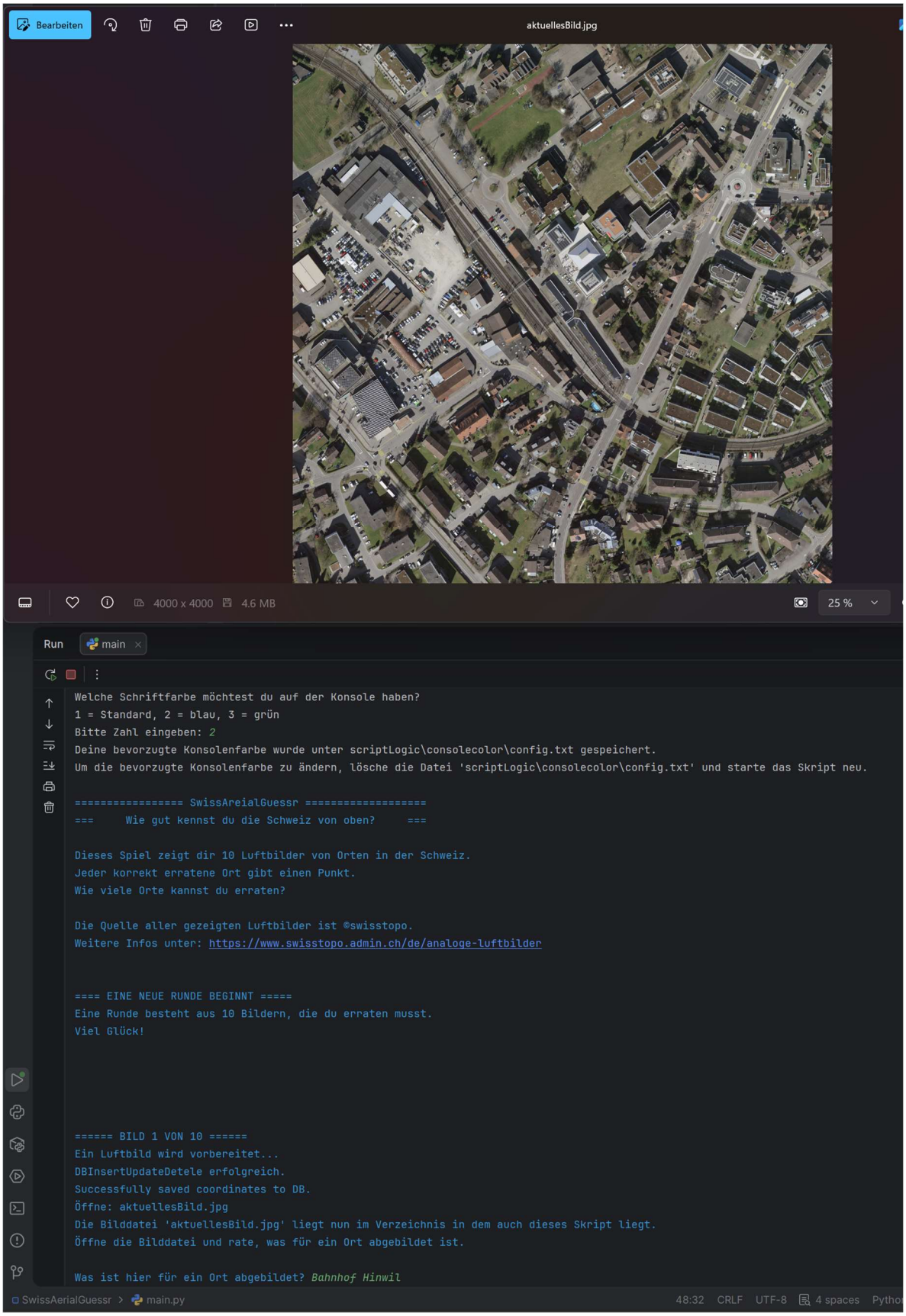

Abbildung 3 Screenshot der Durchführung des Usecases

5.2 Testfälle

Ich habe drei Testfälle ausgewählt, die typische Situationen abdecken und damit die zentralen Funktionen des Skripts testen:

– Standard-Spielablauf & Datenbank-Integration
– Fehlertolerante Antwortprüfung (Fuzzy-Matching)
– Caching-Mechanismus

Alle Testfälle wurden mit dem Skript durchgeführt.

5.2.1 Testfall 1 – Standard-Spielablauf & Datenbank-Integration

Ziel

Überprüfen, ob das Skript erfolgreich Daten aus der Datenbank lädt, die Spielrunde à 10 Orte durchführt und das Ergebnis am Ende speichert.

Vorbedingungen

- MySQL-Datenbank ist online und die Tabellen **`ort`** (mit mind. 10 Einträgen) und spiel sind vorhanden.
- Internetverbindung für den API-Zugriff (GeoAdmin) steht.

Eingabe

- Start des Skripts **`main.py`**.
- Eingabe einer Antwort (z.B. «Bundeshaus» für ein Bild vom Bundeshaus in Bern) für jeden abgefragten Ort.
- Spielen von 10 Spielrunden.

Erwartetes Verhalten

- Skript wählt zufällig Orte aus der Tabelle **`ort`** aus.
- Das Luftbild wird korrekt unter einem neutralen Namen gespeichert und bestenfalls vom Betriebssystem direkt mit dem standardmässigen Bildbetrachtungsprogramm geöffnet.
- Nach der zehnten Runde wird die Gesamtpunktzahl erfolgreich in die Tabelle **`spiel`** geschrieben (zu prüfen über manuelle SQL-Abfrage: **`SELECT * FROM spiel`**).
- Die Konsole gibt am Ende den Punktedurchschnitt aus den bisher gespielten Runden aus.

Ergebnis

Test erfolgreich. Der Spielablauf war wie erwartet. Die Kommunikation mit der Datenbank funktionierte. Das Betriebssystem hat das Bild jeweils selbständig geöffnet, in dieser Testdurchführung war das verwendete Betriebssystem Windows 11 Pro.

Beweis der Durchführung Testfall 1

Das Luftbild wurde korrekt unter dem neutralen Namen «aktuellesBild.jpg» direkt im Skriptordner gespeichert:

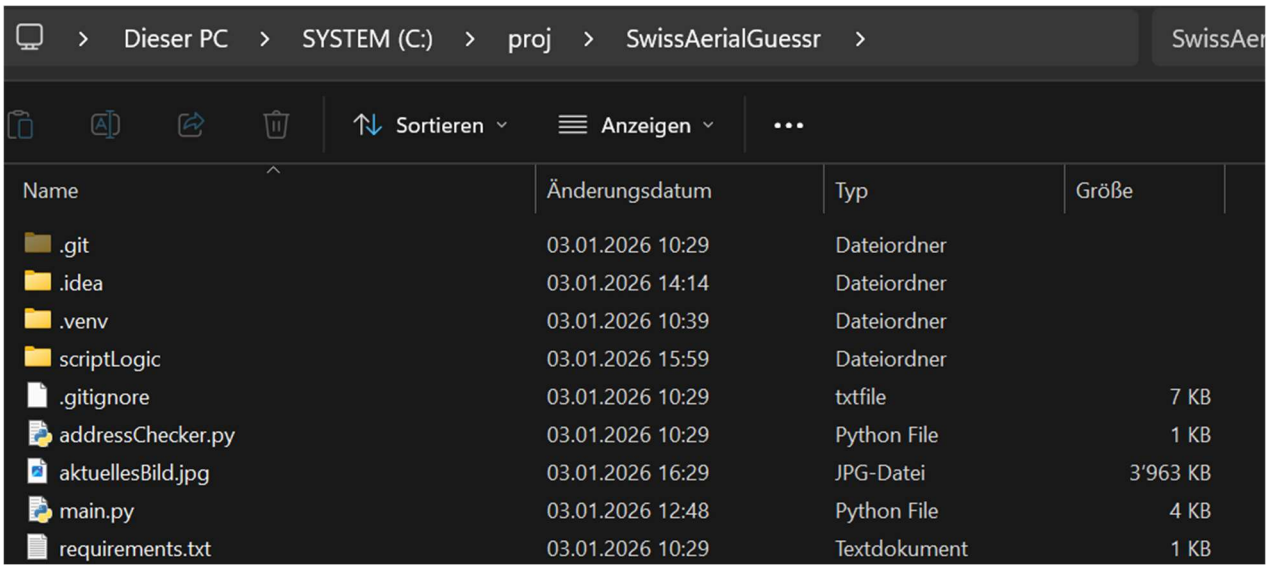

Abbildung 4 Speicherort des Bildes des aktuell gesuchten Ortes.

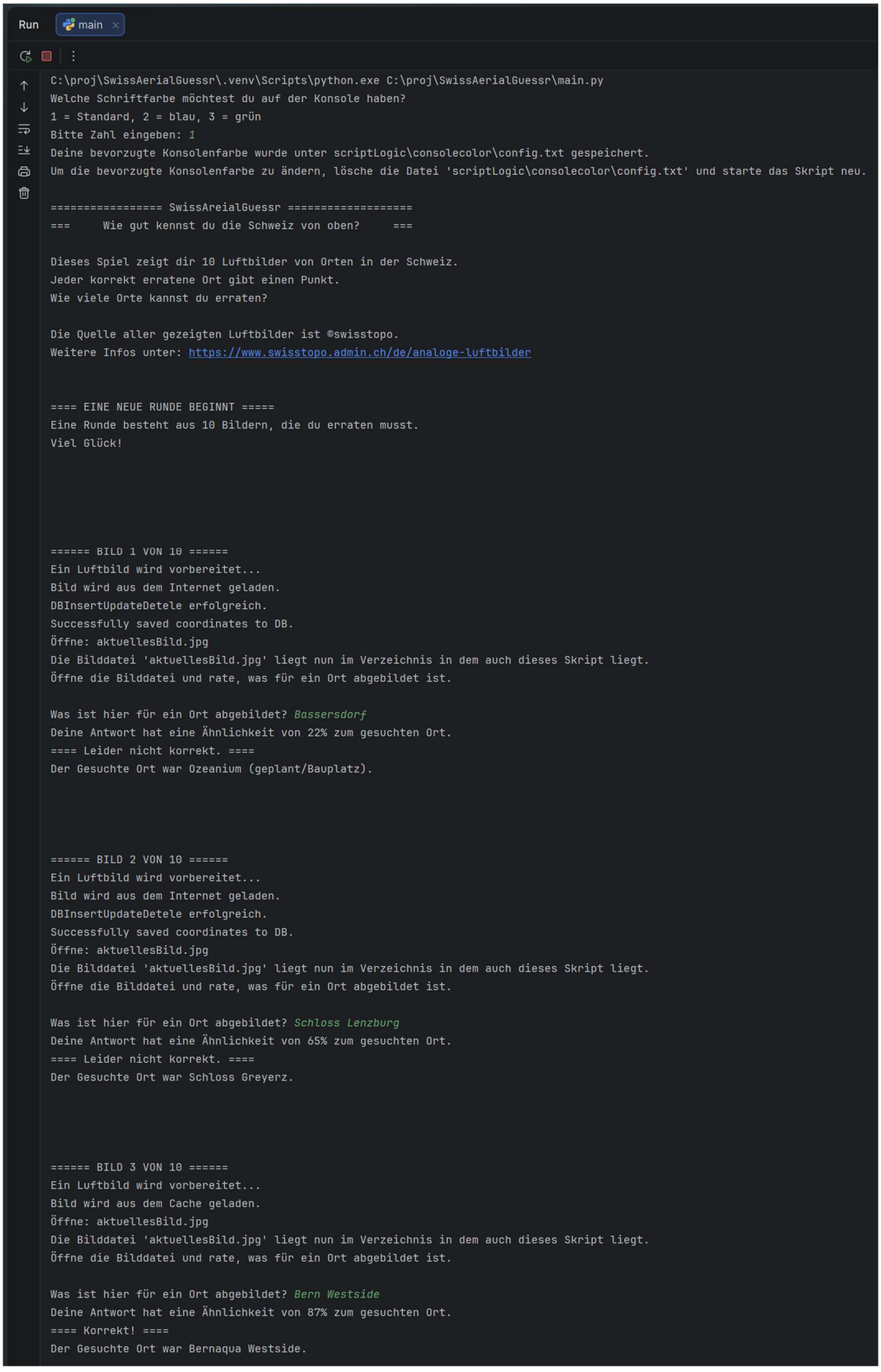

Abbildung 5 Screenshot 1 der Durchführung des ersten Testfalls

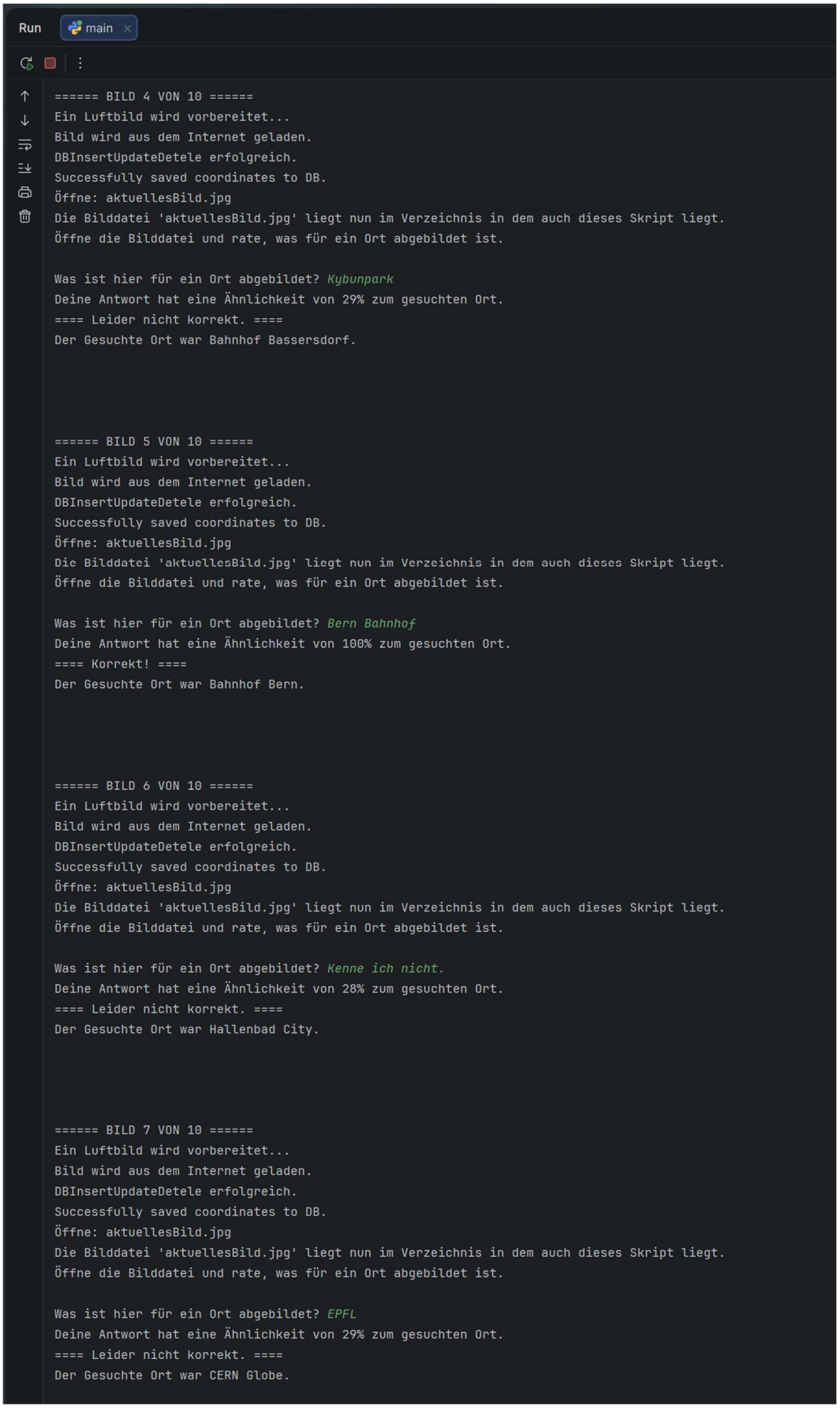

Abbildung 6 Screenshot 2 der Durchführung des ersten Testfalls

```
====== BILD 8 VON 10 ======
Ein Luftbild wird vorbereitet...
Bild wird aus dem Internet geladen.
DBInsertUpdateDetele erfolgreich.
Successfully saved coordinates to DB.
Öffne: aktuellesBild.jpg
Die Bilddatei 'aktuellesBild.jpg' liegt nun im Verzeichnis in dem auch dieses Skript liegt.
Öffne die Bilddatei und rate, was für ein Ort abgebildet ist.

Was ist hier für ein Ort abgebildet? Regensdorf
Deine Antwort hat eine Ähnlichkeit von 19% zum gesuchten Ort.
==== Leider nicht korrekt. ====
Der Gesuchte Ort war Bahnhof Pfäffikon SZ.




====== BILD 9 VON 10 ======
Ein Luftbild wird vorbereitet...
Bild wird aus dem Cache geladen.
Öffne: aktuellesBild.jpg
Die Bilddatei 'aktuellesBild.jpg' liegt nun im Verzeichnis in dem auch dieses Skript liegt.
Öffne die Bilddatei und rate, was für ein Ort abgebildet ist.

Was ist hier für ein Ort abgebildet? Davos Platz
Deine Antwort hat eine Ähnlichkeit von 100% zum gesuchten Ort.
==== Korrekt! ====
Der Gesuchte Ort war Bahnhof Davos Platz.




====== BILD 10 VON 10 ======
Ein Luftbild wird vorbereitet...
Bild wird aus dem Internet geladen.
DBInsertUpdateDetele erfolgreich.
Successfully saved coordinates to DB.
Öffne: aktuellesBild.jpg
Die Bilddatei 'aktuellesBild.jpg' liegt nun im Verzeichnis in dem auch dieses Skript liegt.
Öffne die Bilddatei und rate, was für ein Ort abgebildet ist.

Was ist hier für ein Ort abgebildet? Bahnhof Zollikon
Deine Antwort hat eine Ähnlichkeit von 100% zum gesuchten Ort.
==== Korrekt! ====
Der Gesuchte Ort war Bahnhof Zollikon.




===== DIE RUNDE IST VORBEI =======
Du hast 4 von 10 Punkten erzielt.




DBInsertUpdateDetele erfolgreich.
Du hast total 3 Spiele gespielt.
Du hast durchschnittlich 5.00 Punkte erreicht.




Um nochmal eine Runde zu spielen drücke 'Enter'.
Um das Skript zu beenden, tippe 'Ende'.
```

Abbildung 7 Screenshot 3 der Durchführung des ersten Testfalls

5.2.2 Testfall 2 – Fehlertolerante Antwortprüfung (Fuzzy-Matching)

Ziel

Sicherstellen, dass das Skript eine Antwort auch als korrekt einstuft, wenn sie einen Tippfehler enthält oder die Wörter in einer anderen Reihenfolge eingetippt wurden.

Vorbedingungen

Die Inhalte der Tabelle **`ort`** werden bis auf ID 34 und ID 57 gelöscht:

```
delete from ort where ID_Ort not in (34, 57);
```

Damit blieben nur die folgenden zwei Orte zurück:

`34,Bern Westside,"Riedbachstrasse 98, 3027 Bern"`

`57,Berner Münster,"Münsterplatz 1, 3011 Bern"`

Danach wird das Spiel gestartet.

Eingabe

Das Spiel wird nun so lange gespielt, bis beide oben genannten Orte abgefragt wurden. Da die Datenbank nur noch diese beiden Orte beinhaltet sollte es kaum mehr als 2 Fragen dauern, bis beide Orte als Frage ausgewählt wurden.

- Wird das Bild von «Bern Westside» angezeigt, so wird als Antwort «Westside Bern» eingegeben.
- Wird das Bild vom «Berner Münster» angezeigt, so wird als Antwort «Berner Munster» (mit «u» statt «ü») eingegeben.

Erwartetes Verhalten

Die Bibliothek Fuzzywuzzy berechnet eine Ähnlichkeit der Antworten. Bei beiden Antworten wird die Aufgabe als korrekt bewertet und der Nutzer erhält den Punkt.

Ergebnis

Test erfolgreich: Beide Antworten wurden vom Skript als korrekt gewertet. Bei «Westside Bern» wurde die Ähnlichkeit immer noch mit 100% gewertet. Bei «Berner Munster» hat die Ähnlichkeit aufgrund des fehlenden «ü»-Umlautes nur 3% eingebüsst. Der Algorithmus fängt diese Art von menschlichem Eingabefehler ab.

```
Was ist hier für ein Ort abgebildet? Westside Bern
Deine Antwort hat eine Ähnlichkeit von 100% zum gesuchten Ort.
==== Korrekt! ====
Der Gesuchte Ort war Bern Westside.
```

Abbildung 8 Konsolenausgabe nach Nutzereingabe "Westside Bern"

```
Was ist hier für ein Ort abgebildet? Berner Munster
Deine Antwort hat eine Ähnlichkeit von 97% zum gesuchten Ort.
==== Korrekt! ====
Der Gesuchte Ort war Berner Münster.
```

Abbildung 9 Konsolenausgabe nach Nutzereingabe "Berner Munster"

Beweis der Durchführung Testfall 2

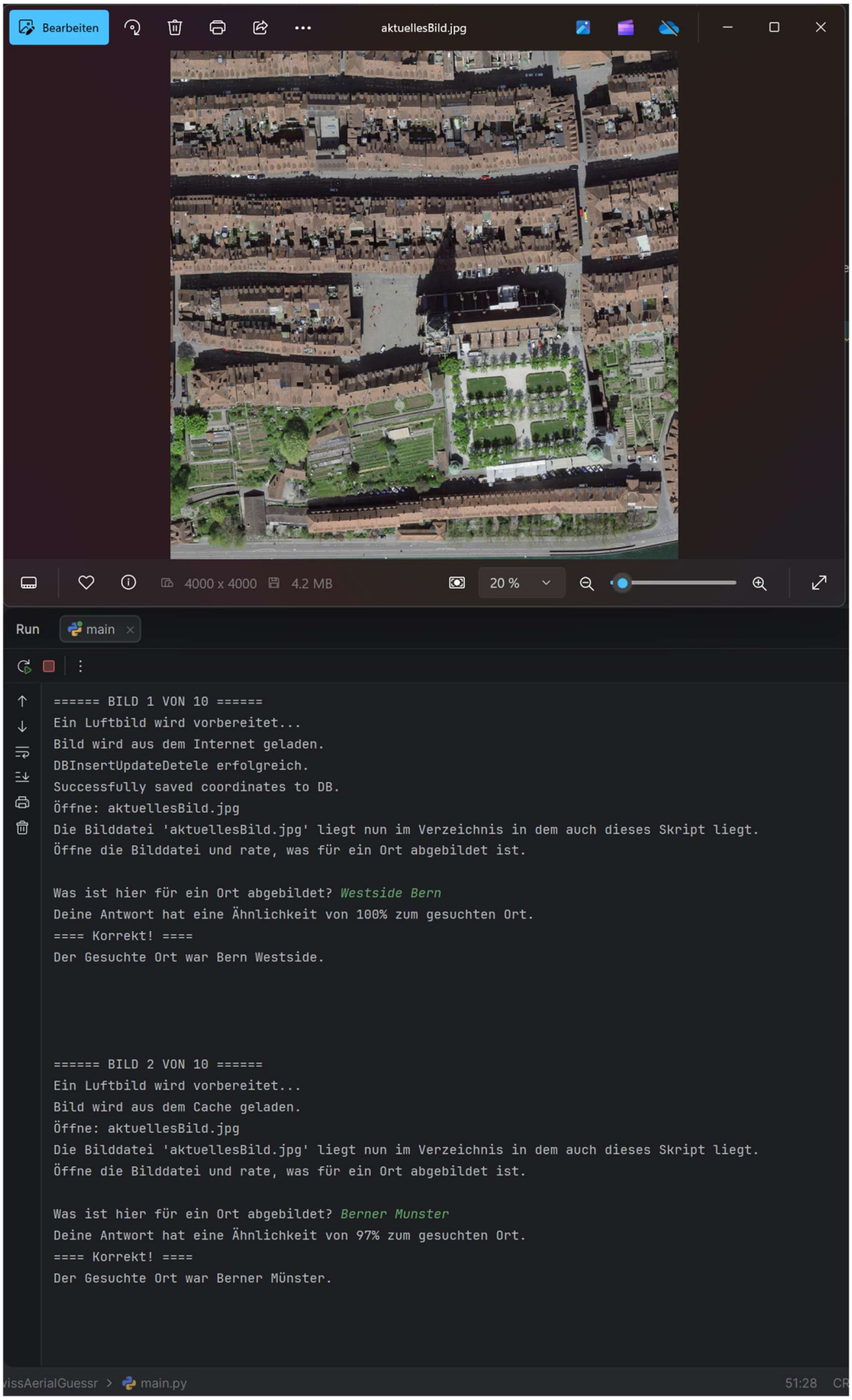

Abbildung 10 Durchführung des zweiten Testfalls

5.2.3 Testfall 3 – Caching-Mechanismus

Ziel

Überprüfen, ob das Skript unnötige API-Downloads vermeidet, wenn ein Bild bereits im Cache existiert.

Vorbedingungen

- Die Datenbank enthält nur genau einen spezifischen Ort (z. B. «Berner Münster»).
- Das Luftbild dieses Ortes befindet sich bereits im lokalen Cache-Ordner.

Eingabe

Start des Spiels. Da die Datenbank nur noch einen Ort enthält, wählt das Skript diesen Ort aus.

Erwartetes Verhalten

- Das Skript stellt fest, dass die Datei lokal existiert und die UpdateFlag auf 0 steht.
- Die Konsole gibt explizit aus: «**`Bild wird aus dem Cache geladen.`**»
- Es erfolgt kein Netzwerkzugriff auf die GeoAdmin-API.
- Die Wartezeit beträgt weniger als eine Sekunde bis das Bild sich von selbst öffnet.

Ergebnis

Die Konsolenausgabe ist wie erwartet und gab aus «**`Bild wird aus dem Cache geladen.`**». Zudem war die Wartezeit kürzer als eine Sekunde.

```
====== BILD 1 VON 10 ======
Ein Luftbild wird vorbereitet...
Bild wird aus dem Cache geladen.
Öffne: aktuellesBild.jpg
Die Bilddatei 'aktuellesBild.jpg' liegt nun im Verzeichnis in dem auch dieses Skript liegt.
Öffne die Bilddatei und rate, was für ein Ort abgebildet ist.
```

Abbildung 11 Konsolenausgabe zeigt, dass das Bild aus dem Cache geladen wurde.

Beweis der Durchführung Testfall 3

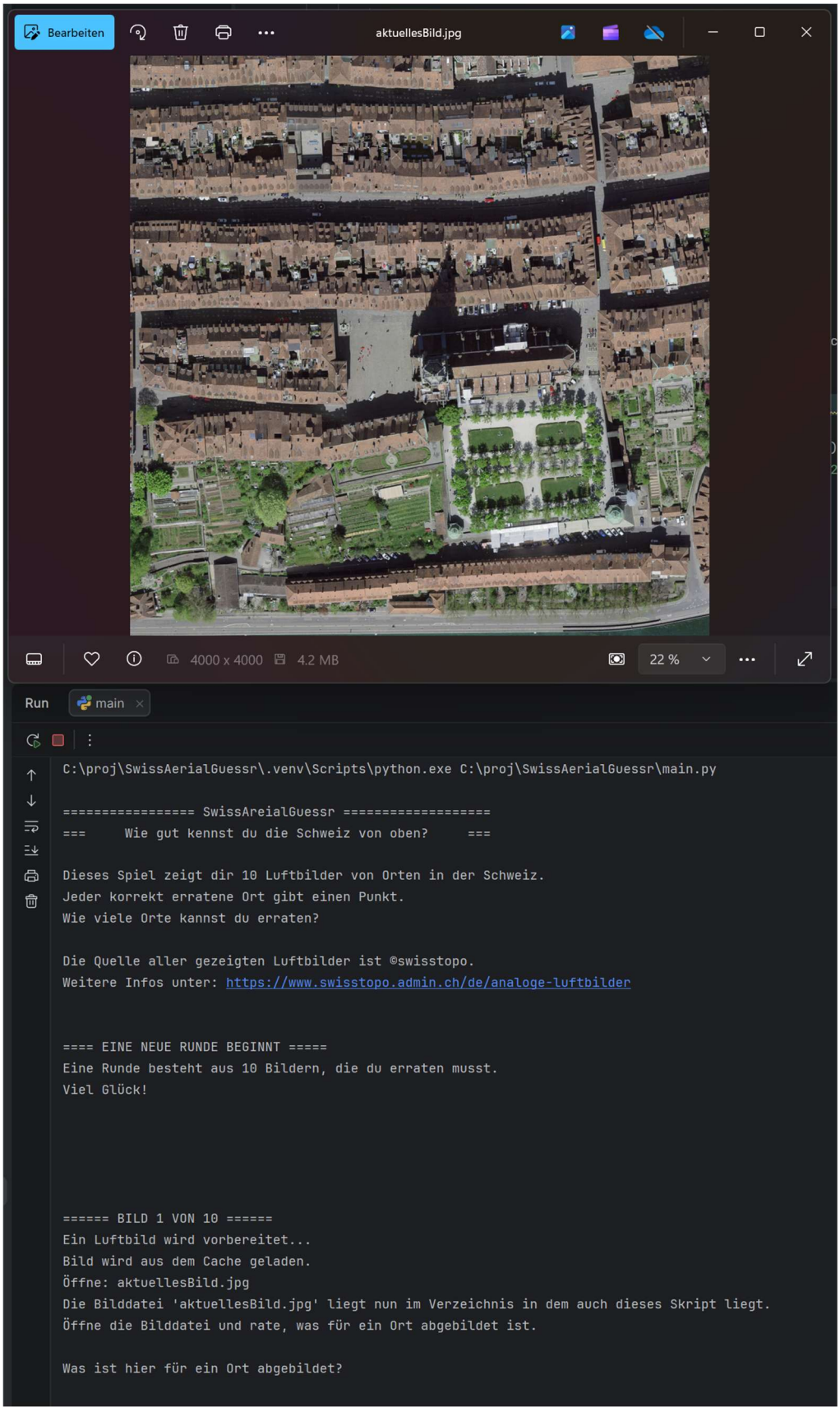

Abbildung 12 Durchführung des dritten Testfalls

6 Präsentation (Video)

Ein Demovideo von «SwissAerialGuessr» zusammen mit einer technischen Erklärung des Skripts kann unter folgendem Link angesehen werden.

https://www.youtube.com/watch?v=Zr19BZf11eE

7 Reflexion

7.1 Arbeitsjournal Jonas Vetsch

Datum	Tätigkeit	Lessons Learnt
15.12.25	**Aufbau der Datenbank & SQL-Anbindung**	Meine grösste Frage heute war: Wie binde ich eine Datenbank in ein Python-Skript ein? Das habe ich vorher noch nie gemacht. Es war weniger schwierig als ich es erwartet hatte. Die Arbeit mit der Bibliothek SQLAlchemy hat mir gezeigt, wie effizient die Brücke zwischen Python und SQL geschlagen werden kann, wenn man eine geeignete Bibliothek findet. Besonders wertvoll war, dass ich mir heute Zeit genommen habe, mich in die Rückgabewerte einer solchen Datenbankabfrage in Python einzudenken: Ein SELECT-Statement liefert Datenobjekte, die man entweder in OOP-Klassen parsen oder als Tupel verarbeiten kann. Da ich noch kein OOP hatte, habe ich mich für die Verarbeitung der Daten als Tupel entschieden. Durch Iteration (for row in result) erhält man direkten Zugriff auf die Zeilen, was die Weiterverarbeitung im Skript enorm erleichtert.
18.12.25	**Implementierung des Errorhandlings**	Zu Beginn empfand ich spezifisches Errorhandling als unnötige Komplexität. Doch gutes Errorhandling gehört einfach zu gutem Code dazu. Ein besonderes Learning heute war, dass es gefährlich ist, wenn ich einfach alle Fälle mit einem allgemeinen `except Exception:` abfange, da es auch Syntax-Fehler (Tippfehler) verschluckt. Man muss so spezifisch wie möglich Fehler fangen und loggen. Es ist wichtig, dass kritische Fehler das Programm eben auch wirklich zum Absturz bringen, damit man während dem Entwickeln auf seine eigenen z.B. Syntaxfehler aufmerksam wird. Das Programm würde ansonsten im schlimmsten Fall fehlerhaft ausgeliefert werden! Zudem: Funktionen mit Rückgabewerten sollten im Fehlerfall explizit `None` returnen, damit der Aufrufer den Status prüfen kann.
22.12.25	**Datenbeschaffung**	Bislang habe ich mit einigen selbst generierten Testdaten in der Datenbank gearbeitet. Um das Skript ausführlich zu testen (und

	& KI-Generierung	damit das Spiel auch wirklich Spass macht beim Spielen) muss eine grosse Datenmenge her. Da der Fokus dieses Projekts auf der Skriptentwicklung lag und nicht auf der manuellen Generierung von dutzenden Datensätzen (würde mehrere Tage in Anspruch nehmen), habe ich mich entschieden, die Daten für die Datenbank dieses Spiels mit KI zu generieren. Die Nutzung von KI für ein Datenset von 200 Schweizer Orten war effizient, barg aber Tücken in der Datenqualität. Die ZoomStufe wurde sehr zuverlässig generiert. Bei der Adresse ist mein generiertes (und händisch nachgepflegtes) Datensetz nach wie vor etwas mangelhaft. Der Grund liegt darin, dass die KI oftmals nicht die genaue Adresse eines Ortes kennt. Besonders bei Bahnhöfen (z.B. Bahnhof Langenthal) sind die Adressen daher leider oft etwas vom eigentlichen Bahnhof entfernt. Dies ist darauf zurückzuführen, dass Bahnhöfe in der Schweiz meistens keine Adresse haben und die KI eine nahegelegene Adresse annehmen muss. Meistens wurde dann beispielsweise «Bahnhofstrasse, Langenthal» eingetragen. Hierfür liefert die API von GeoAdmin allerdings einfach die Koordinate des Ortes, der in der Mitte der Bahnhofstrasse liegt. Ist die Bahnhofstrasse des gesuchten Ortes sehr lange, so liegt die Mitte dieser Strasse weit weg vom Bahnhof und dieser ist nicht mehr auf dem Luftbild sichtbar. Ich habe viele Datensätze händisch nachkorrigiert, was sehr aufwendig ist. Es sind immer noch einige Datensätze von etwas geringer Qualität vorhanden. Die Erhöhung der Datenqualität der Spieldatenbank könnte Gegenstand von einem weiteren Projekt werden. Erkenntnis: KI-Daten sind eine gute Basis, aber ohne massive manuelle Nachbearbeitung der Adressen und Koordinaten erreicht man keine perfekte User Experience.
29.12.25	**Logik der Update-Flags**	Dank der Korrektur vieler Adressen in der Datenbank stiess ich auf ein Logik-Fehler: Wenn ich die UpdateFlag auf 1 setzte, wurden die Koordinaten nicht neu von der API angefragt. Die Skriptlogik muss zwingend auch die Koordinaten neu abrufen, wenn die Adresse geändert wurde. Ohne diesen Schritt bleibt die Korrektur im Spiel unsichtbar, da das System noch mit den alten Geodaten arbeitet. Dies war zum Glück schnell behoben, da es einfach ein Logikfehler in meinem Code war. Die Funktion an sich hatte ich bereits korrekt implementiert, einfach falsch geprüft.

30.12.25	**Entwicklung der Antwort-Validierung**	Die für die Auswertung der Nutzereingaben wollte ich einen Weg finden, um auch leicht abweichende Antworten als korrekt zu erkennen. Ich habe mich heute in diese Thematik eingelesen. Der Levenshtein-Abstand ist hier oft ungeeignet, da kleine Abweichungen (z.B. «St. Moritz» vs. «St. Moritz Dorf») zu hohe Distanzwerte liefern. Ich habe mich dann mit Token-Matching beschäftigt. Hierzu gibt es eine für mein Problem gut geeignete Bibliothek «fuzzywuzzy». Während **`token_sort_ratio`** (alphabetische Sortierung) gut ist, hat sich **`token_set_ratio`** schliesslich als am besten geeignet erwiesen, da es Gemeinsamkeiten in den Wortmengen stärker gewichtet und meinem gewünschten Resultat sehr nahekommt. Diese Lösung ist nicht perfekt, es gibt einige Fälle, die ich gerne abdecken würde, aber leider noch nicht kann. In einem weiterführenden Projekt könnte man sich dieser Thematik annehmen: Ein grosses Problem ist das «Austricksen» des Systems, das nun als Kehrseite leider möglich geworden ist: Nutzer könnten bei Bahnhöfen nur das Wort «Bahnhof» eingeben, was aktuell fälschlicherweise als korrekt eingestuft wird, da z.B. «Bahnhof» und «Bahnhof Lenzburg» von meinem System als ähnlich genug eingestuft werden. Das zeigt mir, dass Validierung immer eine Balance zwischen Fehlertoleranz und strikter Korrektheit sein muss. Hier ist für die Zukunft noch zusätzliche Logik nötig, um solches «Austricksen» zu unterbinden. Für die aktuelle Version und den aktuellen Usecase des Skriptes bin ich dennoch sehr zufrieden, denn aktuell ist es ein Spiel, das offline, allein gespielt wird: So betrügt sich der Nutzer höchstens selbst.

7.2 Auswertung

Die Realisierung von «SwissAerialGuessr» zeigt, dass mit der Kombination aus einer relationalen Datenbank (MySQL), externen Schnittstellen (GeoAdmin API) und einer fehlertoleranten Logik ein stabiles und unterhaltsames Konsolenspiel entstehen kann.

Durch den modularen Aufbau des Skripts in Dateien wie **`api.py`**, **`game.py`** und **`query.py`** konnte eine klare Trennung zwischen Datenhaltung, API-Kommunikation und Spielmechanik erreicht werden. Besonders effizient erwies sich das implementierte Caching-System: Da Bilder nur bei Bedarf oder gesetzter UpdateFlag heruntergeladen werden, wird die Last auf die API minimiert und als Nebeneffekt sogar noch die Spielgeschwindigkeit deutlich erhöht. Die Verwendung der fuzzywuzzy-Bibliothek stellte sicher, dass die Antwortprüfung trotz der Einschränkungen einer Konsoleneingabe fair bleibt, indem sie Tippfehler und leichte Namensabweichungen abfängt.

7.3 Zielprüfung

Ziel 1: Datenbank-Anbindung und Datenhaltung

Es wurde eine relationale MySQL-Datenbank aufgesetzt, die über das Modul engine.py (mithilfe der Bibliothek SQLAlchemy) angebunden ist. Die Tabelle **`ort`** enthält die erforderlichen Datensätze. Das Skript wählt erfolgreich per Zufallsprinzip (**`ORDER BY RAND()`**) Orte aus der Datenbank aus. Nach Abschluss einer Spielrunde wird die erreichte Punktzahl zuverlässig mittels eines INSERT-Statements in die Tabelle **`spiel`** geschrieben, wie im Code in der **`main.py`** ersichtlich ist.

Ziel 2: Validierung mittels Fuzzy-Matching (Fehlertoleranz)

Die Implementierung der Bibliothek fuzzywuzzy im Modul **`game.py`** erfüllt diese Anforderung präzise. Durch die Funktion **`checkAnswerUsingTokenSet`** und den Einsatz des **`token_set_ratio`** wird die Ähnlichkeitsschwelle von 78% angewendet. Tests haben gezeigt, dass dadurch sowohl Tippfehler als auch leicht unterschiedliche Schreibweisen korrekt erkannt werden. Die Vorverarbeitung der Eingaben (Kleinschreibung, Entfernung von Sonderzeichen) unterstützt die Robustheit dieses Ziels zusätzlich.

Ziel 3: Effizientes Caching-System und API-Management

Das Skript verfügt über eine ausgereifte Caching-Logik in der Funktion **`displayImage`**. Bevor ein Bild über die GeoAdmin API angefordert wird, prüft das System, ob die Datei bereits lokal im Ordner **`./scriptLogic/images`** existiert. Ein erneuter Download wird nur gemacht, wenn das Bild fehlt oder die UpdateFlag in der Datenbank auf **`1`** gesetzt wurde (z. B. nach einer Adressänderung). Dies schont die Ressourcen der API und minimiert die Ladezeiten während des Spiels erheblich.

7.4 Fazit

Das Projekt «SwissAerialGuessr» demonstriert erfolgreich den Übergang von einfacher Skriptautomatisierung hin zu einer dynamischen, datenbankgestützten Applikation. Die grösste Herausforderung lag in der Balance zwischen Fehlertoleranz und strikter Korrektheit bei der Antwortvalidierung und in der Qualität der Daten in der Datenbank. Während das System für den aktuellen Usecase als offline Singleplayer-Spiel hervorragend funktioniert, bietet die Lösung noch Raum für Erweiterungen, etwa durch eine Web-Anbindung oder die Einbindung weiterer Geodaten (z. B. Berge statt nur Adressen). Insgesamt ist das Skript ein robustes Beispiel dafür, wie offene Behördendaten der Swisstopo kreativ und technisch sauber in einer Python-Umgebung genutzt werden können.

8 Anhang: Skriptlisting

8.1 main.py

```
#!/usr/bin/env python3

import os
import time

from scriptLogic.game import displayImage, checkAnswerUsingTokenSet
from scriptLogic.userInteraction import initialExplanationOfScript, get-
ValidAnswer
from scriptLogic.logging.logs import cleanUpLogs
from scriptLogic.consolecolor.consoleColor import setUpConsoleColor
from scriptLogic.database.query import dbInsertUpdateDelete, dbQuery

"""
================================================================================
=====
PROJECT:        SwissAerialGuessr
AUTHOR:         Jonas Vetsch, jve161514@stud.gibb.ch
VERSION:        1.4.0 (2025-12-30)
DESCRIPTION:    Interaktives Ratespiel basierend auf Schweizer Luftbildern
(©swisstopo).
                Nutzt die GeoAdmin API (WMS) und fuzzywuzzy zur Validierung
der Antworten.

USAGE:          1. Requirements im venv installieren: pip install -r re-
quirements.txt
                2. Datenbank erstellen und Daten importieren gemäss
./scriptLogic/database
                3. Mit Python das Skript ./main.py ausführen (Benötigt Py-
thon 3.x und die Abhängigkeiten unter ./requirements.txt)

VERSIONS:
    2025-12-30: v1.4.0 - Optimierung der Validierung & Fuzzy Matching
    2025-12-29: v1.3.1 - Fix der Daten-Synchronisation
    2025-12-22: v1.3.0 - Datenbank vergrössern und bereinigen
    2025-12-18: v1.2.0 - Robustheit & Errorhandling
    2025-12-15: v1.1.0 - Datenbank-Anbindung & Datenverarbeitung
    2025-12-08: v1.0.0 - Initialer Prototyp (Proof of Concept)
================================================================================
=====
"""

# Set console color to user preference
setUpConsoleColor()

# Display basic explanation about script to user
initialExplanationOfScript()
```

```
# Restart a new game until user wants to end script
endScriptCondition = ''
while endScriptCondition == '':
    # Play SwissAerialGuessr
    print("==== EINE NEUE RUNDE BEGINNT =====")
    print("Eine Runde besteht aus 10 Bildern, die du erraten musst.")
    print("Viel Glück!")
    print("\n" * 4)

    score = 0
    numberOfRounds = 10
    for i in range(numberOfRounds):

        print(f'====== BILD {i+1} VON {numberOfRounds} ======')
        # Get image and correct answer
        print("Ein Luftbild wird vorbereitet...")

        placeToGuess = displayImage()
        if placeToGuess is None:
            # Error while getting image and answer
            print("In diesem Durchgang konnte kein Luftbild geladen wer-
den.")
            continue


        # Ask Answer from User
        print("Die Bilddatei 'aktuellesBild.jpg' liegt nun im Verzeichnis
in dem auch dieses Skript liegt.")
        print("Öffne die Bilddatei und rate, was für ein Ort abgebildet
ist.")
        print()
        rawGuess = getValidAnswer()
        if rawGuess == '':
            continue


        # Process answer
        isCorrect = checkAnswerUsingTokenSet(rawGuess, placeToGuess, simi-
larityTreshold=78)
        if isCorrect:
            score += 1
            print("==== Korrekt! ====")
        else:
            print("==== Leider nicht korrekt. ====")

        print(f"Der Gesuchte Ort war {placeToGuess}.")
        print("\n" * 3)
        time.sleep(2)

    # End of Game
    print("===== DIE RUNDE IST VORBEI =======")
```

```
    print(f"Du hast {score} von {numberOfRounds} Punkten erzielt.")
    print("\n" * 3)

    # Save score to DB
    dbInsertUpdateDelete(f"INSERT INTO spiel (Punktzahl) VALUES
({score});")

    # Show user stats (total games played, average score)
    statsTupel = dbQuery("SELECT COUNT(*) AS Anzahl, AVG(Punktzahl) AS
Durchschnitt FROM spiel;").first()
    if statsTupel is not None:
        print(f"Du hast total {statsTupel.Anzahl} Spiele gespielt.")
        print(f"Du hast durchschnittlich {statsTupel.Durchschnitt:.2f}
Punkte erreicht.")
        print("\n" * 3)

    print("Um nochmal eine Runde zu spielen drücke 'Enter'.")
    print("Um das Skript zu beenden, tippe 'Ende'.")
    endScriptCondition = input()


# CLEAN UP
# Delete logging older than a day
cleanUpLogs()

# Reset console color to default
print('\033[0m')
```

8.2 api.py

```
import requests
import os
from scriptLogic.logging.logs import logRequest, safeRequest


## ===== API (LOAD AN IMAGE FOR A GIVEN ADRESS) ======

# Get coordinates and official name of a given location
def requestGeoInformation(address):
    safeRequest() # Check logging to stay within API limits and wait if
necessary

    try:
        # URL of geocoding API endpoint of GeoAdmin
        url = "https://api3.geo.admin.ch/rest/services/api/SearchServer"

        # Parameters for the API Request
        params = {
            'searchText': address,  # Die Adresse, die geocodiert werden
soll
            'type': 'locations',  # Wir suchen nach geokodierten Orten
            'limit': 1,  # Maximale Anzahl der Ergebnisse
```

```
            'returnGeometry': 'true',  # Wir wollen die Geometrie (Koordi-
naten) im Ergebnis
            'sr': '2056'  # Neue Landesvermessung (2...../1....)
        }

        # Send request
        response = requests.get(url, params=params)
        logRequest()

        # Valid answer from API
        if response.status_code == 200:
            try:
                data = response.json()

                # Check if API found the requested location
                if data['results']:
                    # Coordinates, in the swiss coordinate system 'Neue
Landesvermessung'
                    x = float(data['results'][0]['attrs']['x'])
                    y = float(data['results'][0]['attrs']['y'])

                    # Official name of location
                    # locationNameRaw = data['results'][0]['attrs']['la-
bel']
                    # locationName = re.sub(r"<.*?>", "", locationNameRaw)
                    # print(f'Für den Ort "{locationFromUser}" wurde "{lo-
cationName}" gefunden.')
                    return x, y

                else:
                    print(f'Für den Ort "{address}" kennt GeoAdmin leider
keinen Eintrag.')
                    return None
            except Exception as e:
                print(f'Fehler beim Lesen der Koordinaten in der Antwort
der Search-Server-API: {e}')
                return None

        # Status Code from API raises an issue
        else:
            print(f"Für den Ort {address} ist folgender Fehler
aufgetreten:")
            print(f"{response.status_code} - {response.text}")
            return None

    # API can't be reached
    except requests.exceptions.RequestException as e:
        print(f"Die API konnte nicht erreicht werden (z.B. Netzwerkfehler)
um die Koordinaten für {address} aufzulösen: {e}")
        return None
```

```
    # All other errors
    except Exception as e:
        print(f"Es ist ein unerwarteter Fehler während dem Auflösen der
Koordinaten für {address} aufgetreten: {e}")
        return None



# Get aerial image for coordinates and save to given output folder with
given file name
def requestAndSaveImage(x, y, zoomLevel, address, pathToImages):
    safeRequest() # Check logging to stay within API limits and wait if
necessary

    try:

        # Calculate image frame with desired zoom level
        minx = x - zoomLevel / 2
        miny = y - zoomLevel / 2
        maxx = x + zoomLevel / 2
        maxy = y + zoomLevel / 2

        # WMS 1.3.0 with EPSG:2056 requires this weired format: (y,x,y,x)
        bbox = f"{miny},{minx},{maxy},{maxx}"

        # Web Map Service API of Geo Admin
        url = "https://wms.geo.admin.ch/?"

        # Parameters for the API Request
        params = {
            "SERVICE": "WMS",
            "VERSION": "1.3.0",
            "REQUEST": "GetMap",
            "LAYERS": "ch.swisstopo.swissimage",
            "STYLES": "",
            "FORMAT": "image/jpeg",
            "TRANSPARENT": "FALSE",
            "CRS": "EPSG:2056",
            "BBOX": f"{bbox}",
            "WIDTH": "4000",
            "HEIGHT": "4000"
        }

        # Send request
        response = requests.get(url, params=params)
        logRequest()

        # Case: Got results
        if response.status_code == 200:
            try:
                # Create output sub folder in case it doesn't exist yet
```

```
                os.makedirs(pathToImages, exist_ok=True)

                # Save image
                imagePath = f"{pathToImages}/{address}.jpg"
                with open(imagePath, "wb") as f:
                    f.write(response.content)

                # print(f"Luftbild für {address} erfolgreich heruntergela-
den und als {imagePath} gespeichert.")
                return True

            # Error during saving of the received image
            except IOError as e:
                print(f"Fehler beim speichern des Bildes für {address}.")

        # API can be reached but reutrns error code
        else:
            print(f"Die API gab während der Luftbildanfrage für {address}
folgenden Fehler: {response.status_code} - {response.text}")
            return False

    # API can't be reached
    except requests.exceptions.RequestException as e:
        print(f"Die API konnte nicht erreicht werden (z.B. Netzwerkfehler)
für das Bild {address}: {e}")
        return False

    # All other errors
    except Exception as e:
        print(f"Es ist ein unerwarteter Fehler während dem Laden des Bildes
{address} aufgetreten: {e}")
        return False
```

8.3 game.py

```
import os
import re
import shutil
from fuzzywuzzy import fuzz
from scriptLogic.api import requestGeoInformation, requestAndSaveImage
from scriptLogic.database.query import dbInsertUpdateDelete, dbQuery
from scriptLogic.helperFunctions import openImage



# Get a random place from DB,
# get its image from chache or API
# save image as ./aktuellesBild.jpg
def displayImage():

    foundImageToDisplay = False
    while not foundImageToDisplay:
```

```
        # Select a random place from the data base
        try:
            placeTupel = dbQuery("SELECT * FROM ort ORDER BY RAND() LIMIT
1;").first()
            if placeTupel is None:
                # Restart loop
                print("Didn't get a place from DB. Trying again...")
                continue


            # Check if image needs to be loaded from API or from cache (=
./images)
            imagesFolder = './scriptLogic/images'
            imageName = f'{placeTupel.Adresse}.jpg'
            pathToImage = os.path.join(imagesFolder, imageName)

            # Should Image be downloaded through the API?
            if not os.path.exists(pathToImage) or (placeTupel.UpdateFlag ==
1):
                # Image for this place needs to be downloaded (either not
in cache or update flag)
                print("Bild wird aus dem Internet geladen.")

                # Coordinates already in DB?
                if (placeTupel.UpdateFlag == 1) or (placeTupel.Ostwert is
None or placeTupel.Nordwert is None):
                    # No coordinates in DB or need to update: Get coordi-
nates from API
                    x, y = requestGeoInformation(placeTupel.Adresse)
                else:
                    # Coordinates in DB: let's access them here
                    x = placeTupel.Ostwert
                    y = placeTupel.Nordwert

                # Get Image for Coordinates (with zoom factor from DB) and
save to images folder
                foundImageToDisplay = requestAndSaveImage(x, y, pla-
ceTupel.Zoom, placeTupel.Adresse, imagesFolder)
                if foundImageToDisplay:
                    # save coordinates to DB
                    if dbInsertUpdateDelete(f"UPDATE ort SET Ostwert = {x},
Nordwert = {y}, UpdateFlag = 0 WHERE ID_Ort = {placeTupel.ID_Ort}"):
                        print("Successfully saved coordinates to DB.")
                else:
                    print("Image couldn't be downloaded from API. Trying
another place now.")
                    continue
            else:
                # Image to place already exists in cache and doesn't need
to be reloaded
                foundImageToDisplay = True
```

```
                print("Bild wird aus dem Cache geladen.")

            # Duplicate image to 'aktuelles_Bild.jpg'
            destinationPath = './aktuellesBild.jpg'
            try:
                shutil.copy(pathToImage, destinationPath) # duplicate image
to not give away the name of the place
                openImage(destinationPath) # try to open image with the de-
fault image viewer of the users OS
            except FileNotFoundError:
                print(f'Bild an der Quelle {pathToImage} konnte nicht ge-
funden werden.')
                return None
            except Exception as e:
                print(f'Allgemeiner Fehler beim Kopieren des Bildes: {e}')
                return None

        except Exception:
            print("Es konnte kein Ort aus der Datenbank geladen werden.")
            print("Prüfe, ob die Datenbank ausgeführt wird und Daten in der
Tabelle 'ort' vorhanden sind.")
            continue

    return placeTupel.Name


# Clean a text to improve accuracy in comparisons
# Should be used to clean user input and correct place name from DB
def cleanInput(text):
    # Remove leading, trailing spaces and make all lower case
    text = text.strip().lower()

    # Replace Umlaute, accents and other symbols
    # this also helps with location names containing a "." (St.Gallen etc.)
    replacements = {
        'ä': 'ae', 'ö': 'oe', 'ü': 'ue', 'é': 'e', 'è': 'e', 'à': 'a', 'ç':
'c',
        '-': ' ', '/': ' ', '.': ' '
    }
    for old, new in replacements.items():
        text = text.replace(old, new)

    # Replace all types of white spaces with a simple ' '
    text = re.sub(r'\s+', ' ', text)

    return text


# Compare user answer to correct answer: Calculate similarity and rate user
answer
```

```
def checkAnswerUsingTokenSet(correctAnswer, userGuess, similarityTreshold=80):

    # Clean both strings
    correctAnswerClean = cleanInput(correctAnswer)
    correctGuessClean = cleanInput(userGuess)

    # Calculate similarity using 'token set ratio' algorithm by fuzzywuzzy
    similarity = fuzz.token_set_ratio(correctAnswerClean, correctGuessClean)
    print(f"Deine Antwort hat eine Ähnlichkeit von {similarity}% zum gesuchten Ort.")

    # Rate Answer
    if similarity >= similarityTreshold:
        return True
    else:
        return False
```

8.4 helperFunctions.py

```
import platform
import subprocess
import os
from pathlib import Path


# Try to open the file at a given path
def openImage(pathRelative):
    # create path object from relative path
    pathObject = Path(pathRelative)

    # convert to absolute path
    pathAbsolute = pathObject.resolve()

    # check if file exists before trying to open it
    if not pathAbsolute.exists():
        print(f"Fehler beim Öffnen der Datei: Die Datei '{pathAbsolute}' wurde nicht gefunden.")
        return

    # check operating system name
    system = platform.system()

    try:
        if system == 'Windows':
            # Windows
            os.startfile(pathAbsolute)
        elif system == 'Darwin':
            # macOS
            subprocess.run(['open', str(pathAbsolute)], check=True)
        else:
```

```
            # Linux and other os
            subprocess.run(['xdg-open', str(pathAbsolute)], check=True)

        print(f"Öffne: {pathAbsolute.name}")

    except Exception as e:
        print(f"Ein Fehler ist während dem Öffnen der Datei aufgetreten:
{e}")
```

8.5 userInteraction.py

```
import re

## ===== INTERACTIONS WITH USER ON CONSOLE =====

# Display basic explanation about script
def initialExplanationOfScript():
    print("================= SwissAreialGuessr ==================")
    print("===     Wie gut kennst du die Schweiz von oben?     ===")
    print()
    print("Dieses Spiel zeigt dir 10 Luftbilder von Orten in der Schweiz.")
    print("Jeder korrekt erratene Ort gibt einen Punkt.")
    print("Wie viele Orte kannst du erraten?")
    print()
    print('Die Quelle aller gezeigten Luftbilder ist ©swisstopo.')
    print('Weitere Infos unter: https://www.swisstopo.admin.ch/de/analoge-
luftbilder')
    print()
    print()


# Get answer from user and check with Regex
def getValidAnswer():
    rawGuess = ''

    validAnswer = False
    while not validAnswer:
        rawGuess = input("Was ist hier für ein Ort abgebildet? ")

        if rawGuess == '':
            print("Bitte keine leere Eingabe machen.")
            continue

        # validate answer of User (Regex)
        forbiddenChars = re.search(r"[!#$%()*+:;<=>?@_{|}§€£¥¡¿«»]", raw-
Guess)
        if not forbiddenChars:
            validAnswer = True
        else:
            print("Bitte keine Sonderzeichen eingeben.")

    return rawGuess
```

8.6 logs.py

```
import time
from datetime import datetime, timedelta


## ===== LOGGING in order to stay within API limits =====

logsInstructions = '''# NOTE: Do not change the contents or name of this
file,
# unless you know what you're doing.

# File keeps track of API requests so that the script can
# slow itself down to make sure it doesn't go over the
# fair use limit of 20 requests per minute.


'''

pathToLogFile = "./scriptLogic/logging/logs.txt"

# Log the timestamp of a request to logfile
def logRequest():
    with open(pathToLogFile, "a", encoding="utf-8") as logFile:
        timestamp = datetime.now()
        logFile.write(f'\n{timestamp}')


# Read and return logging from log file
def getLogs():
    # Read the logging
    with open(pathToLogFile, "r", encoding="utf-8") as logFile:
        logData = logFile.read()

    cleanLogs = []
    logsRaw = logData.splitlines()

    # [] Remove the instruction text
    for line in logsRaw[8:]:
        line = line.strip()
        if line == "":
            continue
        try:
            datetime.fromisoformat(line)
            cleanLogs.append(line)
        except:
            continue
    return cleanLogs


# Read logging and check if more than 20 requests have been made in the
past minute
```

```
def requestAllowed():
    # API Rate limit in requests per Minute
    requestsPerMinuteAllowed = 20

    logs = getLogs()

    oneMinuteAgo = datetime.now() - timedelta(minutes=1)
    pastMinute = 0

    for log in logs:
        dateTimeObject = datetime.fromisoformat(log)
        if dateTimeObject >= oneMinuteAgo:
            pastMinute += 1

    if pastMinute >= requestsPerMinuteAllowed:
        return False
    else:
        return True


# Stops execution until an API call can be safely made
# without going over the api limit
def safeRequest():
    while not requestAllowed():
        print("Das Skript wartet, um den API-Dienst nicht zu überlas-
ten...")
        time.sleep(3)


def cleanUpLogs():
    logs = getLogs()
    oneDayAgo = datetime.now() - timedelta(days=1)

    recentLogs = [log for log in logs if datetime.fromisoformat(log) >=
oneDayAgo]

    with open(pathToLogFile, "w", encoding="utf-8") as logFile:
        logFile.write(logsInstructions)

        for log in recentLogs:
            print(log, file=logFile)
```

8.7 query.py

```
from sqlalchemy import text
from scriptLogic.database.engine import engine

# Test db connection
def test_db_connection():
    try:
        with engine.connect() as connection:
            # Führe einen einfachen SQL-Befehl aus
```

```
            result = connection.execute(text("SELECT VERSION();"))
            print("Verbindung erfolgreich!")
            print(f"Datenbankversion: {result.scalar()}")
    except Exception as e:
        print(f"Fehler bei der Verbindung: {e}")

# Custom SELCT query
def dbQuery(sqlQuery):
    try:
        with engine.connect() as connection:
            query = text(sqlQuery)
            return connection.execute(query)
    except Exception as e:
        print(f"Fehler bei der SQL-Abfrage: {e}")
        return None

# Custom INSERT query
def dbInsertUpdateDelete(sqlQuery):
    try:
        with engine.connect() as connection:
            query = text(sqlQuery)
            connection.execute(query)
            connection.commit()
            print("DBInsertUpdateDetele erfolgreich.")
            return True
    except Exception as e:
        print(f"Fehler bei insert, update oder delete in der DB: {e}")
        return False
```

8.8 ort.csv

```
ID_Ort;Name;Adresse;Nordwert;Ostwert;Zoom;UpdateFlag
1;Bundeshaus;Bundesplatz 3, 3003 Bern;;;400;0
2;Prime Tower;Hardstrasse 201, 8005 Zürich;;;400;0
3;Verkehrshaus der Schweiz;Lidostrasse 5, 6006 Luzern;;;600;0
4;Letzigrund Stadion;Badenerstrasse 500, 8048 Zürich;;;500;0
5;Bahnhof Olten;Bahnhofstrasse 22, 4600 Olten;2635441.25;1244664.125;600;1
6;Rheinfall;Rheinfallquai, 8212 Neuhausen am Rhein-
fall;2688048.75;1281511.5;800;1
7;Schloss Chillon;Avenue de Chillon 21, 1820 Veytaux;;;400;0
8;Zürich Hauptbahnhof;Bahnhofplatz, 8001 Zü-
rich;2683260.75;1247919.125;800;0
9;Flughafen Zürich Terminal 1;8058 Zürich-Flughafen;;;4000;0
10;St. Jakob-Park;St. Jakob-Strasse 395, 4052 Ba-
sel;2613584.5;1265612.875;600;0
11;KKL Luzern;Europaplatz 1, 6005 Luzern;2666356.25;1211434;400;0
12;Bahnhof Bern;Bahnhofplatz 10, 3011 Bern;2600107.5;1199722;700;0
13;Palais des Nations;Avenue de la Paix 14, 1211 Genève;;;1000;0
14;Stiftsbibliothek St. Gallen;Klosterhof 6, 9000 St. Gallen;;;300;0
15;Jet d'Eau;Quai Gustave-Ador, 1207 Genève;;;800;0
16;Oerlikon Offene Rennbahn;Wallisellenstrasse 4, 8050 Zürich;;;400;0
18;Schloss Thun;Schlossberg 1, 3600 Thun;2614617.5;1178778.75;300;1
```

```
19;Tissot Arena;Boulevard des Sports 18, 2504 Biel;;;600;0
20;Maison Cailler;Rue Jules Bellet 7, 1636 Broc;2574717.25;1161805.5;400;0
21;Technorama;Technoramastrasse 1, 8404 Winter-
thur;2699833.75;1263377.625;500;0
22;Bahnhof Frauenfeld;Bahnhofplatz, 8500 Frauenfeld;2709628;1268393;500;1
23;Kybunpark;Zürcher Strasse 464, 9015 St. Gallen;2740801.25;1252460;500;0
24;Schloss Lenzburg;Schloss, 5600 Lenzburg;2656372.25;1248785;400;1
25;Papiliorama;Moosmatte 1, 3210 Kerzers;;;400;0
26;Bahnhof Chur;Bahnhofplatz 1, 7000 Chur;;;600;1
27;Kirche San Giovanni Battista;6696 Mogno;;;200;0
28;Landesmuseum Zürich;Museumstrasse 2, 8001 Zürich;;;400;0
30;Goetheanum;Rüttiweg 45, 4143 Dornach;;;500;0
31;Stadion Wankdorf;Papiermühlestrasse 71, 3014
Bern;2601995.75;1201234;600;0
32;Kloster Einsiedeln;Klosterplatz 2, 8840 Einsied-
eln;2699590.25;1220453.75;600;0
34;Bern Westside;Riedbachstrasse 98, 3027 Bern;;;800;0
35;Zoo Basel;Binningerstrasse 40, 4054 Basel;;;800;0
39;Hallenstadion;Wallisellenstrasse 45, 8050 Zürich;;;400;0
40;Kloster Königsfelden;Zürcherstrasse, 5210 Windisch;;;400;0
41;Messe Basel (Halle 1);Messeplatz 10, 4058 Ba-
sel;2612287;1268205.375;600;0
42;Bahnhof St. Gallen;St. Gallen, Poststr. 28;;;600;1
43;Schloss Rapperswil;Eichfeldstrasse, 8640 Rap-
perswil;2705572.75;1231800.75;300;1
45;Glattzentrum;Neue Winterthurerstrasse 99, 8304 Wallisellen;;;800;0
46;Shoppi Tivoli;Hochstrasse 1, 8957 Spreiten-
bach;2670107.5;1252671.75;1000;1
48;Castelgrande;Salita al Castelgrande 18, 6500 Bellin-
zona;2722219.5;1116895.125;400;1
50;KVA Josefstrasse (Turm);Josefstrasse 205, 8005 Zürich;;;300;0
52;FIFA Museum;Seestrasse 27, 8002 Zürich;;;200;0
53;CERN Globe;Route de Meyrin 385, 1217 Meyrin;2493089;1121223.5;400;0
55;Siky Park;Chemin de la Forêt 1, 2743 Eschert;2598694.5;1234657.375;400;1
57;Münster Bern;Münsterplatz 1, 3011 Bern;;;300;0
59;Forum Fribourg;Route du Lac 12, 1763 Granges-Pac-
cot;2578229.5;1185964.125;500;1
61;Swissminiatur;Melide, Via Cantonale 11;2716867.5;1090180.125;300;1
63;Fondation Beyeler;Baselstrasse 101, 4125 Riehen;;;400;0
65;Knies Kinderzoo;Oberseestrasse 36, 8640 Rap-
perswil;2704800;1231265.75;400;0
67;Ballenberg West;Museumsstrasse 131, 3858 Hofstetten;;;1200;0
69;Chaplin's World;Route de Saint-Légier 2, 1804 Corsier-sur-Vevey;;;500;0
71;Alpamare;Gwattstrasse 12, 8808 Pfäffikon SZ;;;500;0
73;Autobahnraststätte Würenlos;A1, 5436 Würenlos;2668589;1254630.875;600;0
75;Zentralbibliothek Zürich;Zähringerplatz 6, 8001 Zü-
rich;2683551.5;1247618.125;200;1
77;Flugplatz Emmen;Emmen, Seetalstr. 175.97;2667257;1218115;4000;1
79;Hafen Romanshorn;Romanshorn, Friedrichshafnerstr.
55;2745436.25;1269097.75;600;1
```

```
81;Landhaus Solothurn;Landhausquai 4, 4500 Solo-
thurn;2607488.75;1228403.25;200;0
83;Dreispitzareal;Münchensteinerstrasse, 4053 Basel;;;1200;0
85;Schloss Laufen;Dachsen, Laufen am Rheinfall
2.1;2687700.5;1280421.5;700;1
87;Olympisches Museum, Olympiamuseum;Quai d'Ouchy 1, 1006 Lau-
sanne;2538276.75;1151058.75;400;1
89;Rolex Learning Center;Route Cantonale, 1024 Ecublens;;;400;0
91;Marmorbrücke (Grand Pont);Place de la Riponne, 1005 Lausanne;;;400;0
93;Kaserne Basel;Klybeckstrasse 1b, 4057 Basel;;;400;0
95;Vögele Kultur Zentrum;Gwattstrasse 14, 8808 Pfäffikon
SZ;2702574.75;1228481.5;300;1
97;Schokoladenmuseum Lindt;Schokoladenplatz 1, 8802 Kilchberg;;;400;0
99;Sauriermuseum Aathal;Zürichstrasse 69, 8607 Aathal;;;400;0
101;Tierpark Goldau;Parkweg 30, 6410 Goldau;;;800;0
103;Rathaus Basel;Marktplatz 9, 4001 Basel;;;200;0
105;Sechseläutenplatz;Sechseläutenplatz, 8001 Zürich;;;400;0
107;Conny-Land;Connylandstrasse 1, 8564 Lipperswil;;;500;0
109;Sitterviadukt BT;Sittertalstrasse, 9014 St. Gallen;;;600;0
111;Kyburg;Schloss Kyburg, 8314 Kyburg;2698404.75;1257229.625;300;0
113;Abtei Payerne;Place du Marché, 1530 Payerne;2561675;1185628.75;300;0
115;Schloss Greyerz;Rue du Château 8, 1663 Gru-
yères;2572781.5;1159326.125;300;0
117;Lac des Dix Staumauer;Hérémence, Rte du Barrage
69;2604222.75;1103422.5;1000;0
119;Bahnhof Brig;Brig, Bahnhofstr. 2;;;600;1
120;Stockalperschloss;Alte Vereina-Strasse, 3900 Brig;;;400;0
121;Bahnhof Locarno;Muralto, Via Scazziga Vittore 8;2704454.5;1113517;500;1
122;Piazza Grande;Piazza Grande 7, 6600 Locarno;;;400;0
123;Bahnhof Biasca;Biasca, Via Bellinzona 5;2718173;1134584.375;400;1
124;Schloss Sargans;Schlossstrasse, 7320 Sargans;;;300;0
126;Tamina Therme;Hans-Albrecht-Strasse, 7310 Bad
Ragaz;2757140.25;1207415.75;400;1
129;Hallenbad City Zürich;Sihlstrasse 72, 8001 Zü-
rich;2682651.75;1247403.75;200;0
132;Eisstadion Davos (Vaillant Arena);Talstrasse 41, 7270 Davos;;;400;0
133;Bahnhof St. Moritz;St. Moritz, Via Grevas 65;;;500;1
134;Schanze Einsiedeln;Schnabelsbergstrasse, 8840 Einsiedeln;;;500;0
135;Bahnhof Samedan;Samedan, Via Retica 22;2786751.75;1156529.875;500;1
136;Flugplatz Samedan;Plazza Aviatica 11, 7503 Same-
dan;2786897.25;1156100.375;1500;1
137;Bahnhof Scuol-Tarasp;Scuol, Via da la Staziun
40;2817367.5;1186444.125;400;1
138;Bärenpark Bern;Grosser Muristalden 6, 3006
Bern;2601603.5;1199624.875;400;1
140;Zentrum Paul Klee;Monument im Fruchtland 3, 3006 Bern;;;600;0
146;Internationales Uhrenmuseum;Rue des Musées 29, 2300 La Chaux-de-
Fonds;;;300;0
159;Bahnhof Lenzburg;Lenzburg, Bahnhofstr. 50;2655839.75;1248802.625;500;1
174;Bahnhof Thalwil;Gotthardstr. 24, 8800 Thal-
wil;2685113.75;1238918.25;500;1
```

```
176;Bahnhof Wädenswil;Wädenswil, Bahnhofstr. 9;2693609;1231703.75;500;1
178;Bahnhof Küsnacht ZH;Küsnacht, Kohlrainstr. 16;;;400;1
182;Bahnhof Hinwil;Hinwil, Bahnhofplatz 1.5;2706085.75;1239753.875;500;1
185;Bahnhof Bassersdorf;Bassersdorf, Bahnhofstr.
40.1;2689739.5;1255290.75;400;1
186;Bahnhof Kloten;Lindenstrasse 10, 8302
Kloten;2686265.25;1256007.625;400;1
```

8.9 engine.py

```python
from sqlalchemy import create_engine

# Parameters for the pymysql DB-API Driver
DB_USER = "root"
DB_PASS = "1234"
DB_HOST = "localhost"
DB_NAME = "swissaerialguessr"

DATABASE_URL = (
    f"mysql+pymysql://{DB_USER}:{DB_PASS}@{DB_HOST}/{DB_NAME}"
)

# Create DB engine to interact with DB
engine = create_engine(DATABASE_URL)
```

8.10 databaseSetup.sql

```sql
create database swissaerialguessr;
use swissaerialguessr;

create table ort (
    ID_Ort int primary key auto_increment,
    Name varchar(250) not null,
    Adresse varchar(250) not null,
    Nordwert double,
    Ostwert double,
    Zoom int DEFAULT 500,
    UpdateFlag int not null DEFAULT 0
);

CREATE TABLE spiel (
    ID_Spiel int primary key auto_increment,
    Punktzahl int,
    Zeitstempel timestamp DEFAULT CURRENT_TIMESTAMP
);

# Import Game Data:
LOAD DATA local INFILE '.\\database\\ort.csv'
    INTO TABLE ort
    FIELDS TERMINATED BY ';'
    LINES TERMINATED BY '\n'
    IGNORE 1 LINES
```

```
    (ID_Ort, Name, Adresse, Nordwert, Ostwert, Zoom, UpdateFlag);

# If Error "[42000][3948] Loading local data is disabled;
# this must be enabled on both the client and server sides"
# then try the following:
SET GLOBAL local_infile = 1;
# Then: Retry the command "LOAD DATA" above

# Else try to import with the MANUAL DATA IMPORT below.


# Test your Import
select * from ort;

# MANUAL DATA IMPORT with insert command (Fallback-Solution)
INSERT INTO ort (ID_Ort, Name, Adresse, Nordwert, Ostwert, Zoom, Up-
dateFlag) VALUES

(1, 'Bundeshaus', 'Bundesplatz 3, 3003 Bern', NULL, NULL, 400, 0),

(2, 'Prime Tower', 'Hardstrasse 201, 8005 Zürich', NULL, NULL, 400, 0),

(3, 'Verkehrshaus der Schweiz', 'Lidostrasse 5, 6006 Luzern', NULL, NULL,
600, 0),

(4, 'Letzigrund Stadion', 'Badenerstrasse 500, 8048 Zürich', NULL, NULL,
500, 0),

(5, 'Bahnhof Olten', 'Bahnhofstrasse 22, 4600 Olten', 2635441.25,
1244664.125, 600, 1),

(6, 'Rheinfall', 'Rheinfallquai, 8212 Neuhausen am Rheinfall', 2688048.75,
1281511.5, 800, 1),

(7, 'Schloss Chillon', 'Avenue de Chillon 21, 1820 Veytaux', NULL, NULL,
400, 0),

(8, 'Zürich Hauptbahnhof', 'Bahnhofplatz, 8001 Zürich', 2683260.75,
1247919.125, 800, 0),

(9, 'Flughafen Zürich Terminal 1', '8058 Zürich-Flughafen', NULL, NULL,
4000, 0),

(10, 'St. Jakob-Park', 'St. Jakob-Strasse 395, 4052 Basel', 2613584.5,
1265612.875, 600, 0),

(11, 'KKL Luzern', 'Europaplatz 1, 6005 Luzern', 2666356.25, 1211434, 400,
0),

(12, 'Bahnhof Bern', 'Bahnhofplatz 10, 3011 Bern', 2600107.5, 1199722, 700,
0),
```

```
(13, 'Palais des Nations', 'Avenue de la Paix 14, 1211 Genève', NULL, NULL,
1000, 0),

(14, 'Stiftsbibliothek St. Gallen', 'Klosterhof 6, 9000 St. Gallen', NULL,
NULL, 300, 0),

(15, 'Jet d''Eau', 'Quai Gustave-Ador, 1207 Genève', NULL, NULL, 800, 0),

(16, 'Oerlikon Offene Rennbahn', 'Wallisellenstrasse 4, 8050 Zürich', NULL,
NULL, 400, 0),

(18, 'Schloss Thun', 'Schlossberg 1, 3600 Thun', 2614617.5, 1178778.75,
300, 1),

(19, 'Tissot Arena', 'Boulevard des Sports 18, 2504 Biel', NULL, NULL, 600,
0),

(20, 'Maison Cailler', 'Rue Jules Bellet 7, 1636 Broc', 2574717.25,
1161805.5, 400, 0),

(21, 'Technorama', 'Technoramastrasse 1, 8404 Winterthur', 2699833.75,
1263377.625, 500, 0),

(22, 'Bahnhof Frauenfeld', 'Bahnhofplatz, 8500 Frauenfeld', 2709628,
1268393, 500, 1),

(23, 'Kybunpark', 'Zürcher Strasse 464, 9015 St. Gallen', 2740801.25,
1252460, 500, 0),

(24, 'Schloss Lenzburg', 'Schloss, 5600 Lenzburg', 2656372.25, 1248785,
400, 1),

(25, 'Papiliorama', 'Moosmatte 1, 3210 Kerzers', NULL, NULL, 400, 0),

(26, 'Bahnhof Chur', 'Bahnhofplatz 1, 7000 Chur', NULL, NULL, 600, 1),

(27, 'Kirche San Giovanni Battista', '6696 Mogno', NULL, NULL, 200, 0),

(28, 'Landesmuseum Zürich', 'Museumstrasse 2, 8001 Zürich', NULL, NULL,
400, 0),

(30, 'Goetheanum', 'Rüttiweg 45, 4143 Dornach', NULL, NULL, 500, 0),

(31, 'Stadion Wankdorf', 'Papiermühlestrasse 71, 3014 Bern', 2601995.75,
1201234, 600, 0),

(32, 'Kloster Einsiedeln', 'Klosterplatz 2, 8840 Einsiedeln', 2699590.25,
1220453.75, 600, 0),

(34, 'Bern Westside', 'Riedbachstrasse 98, 3027 Bern', NULL, NULL, 800, 0),
```

```
(35, 'Zoo Basel', 'Binningerstrasse 40, 4054 Basel', NULL, NULL, 800, 0),

(39, 'Hallenstadion', 'Wallisellenstrasse 45, 8050 Zürich', NULL, NULL,
400, 0),

(40, 'Kloster Königsfelden', 'Zürcherstrasse, 5210 Windisch', NULL, NULL,
400, 0),

(41, 'Messe Basel (Halle 1)', 'Messeplatz 10, 4058 Basel', 2612287,
1268205.375, 600, 0),

(42, 'Bahnhof St. Gallen', 'St. Gallen, Poststr. 28', NULL, NULL, 600, 1),

(43, 'Schloss Rapperswil', 'Eichfeldstrasse, 8640 Rapperswil', 2705572.75,
1231800.75, 300, 1),

(45, 'Glattzentrum', 'Neue Winterthurerstrasse 99, 8304 Wallisellen', NULL,
NULL, 800, 0),

(46, 'Shoppi Tivoli', 'Hochstrasse 1, 8957 Spreitenbach', 2670107.5,
1252671.75, 1000, 1),

(48, 'Castelgrande', 'Salita al Castelgrande 18, 6500 Bellinzona',
2722219.5, 1116895.125, 400, 1),

(50, 'KVA Josefstrasse (Turm)', 'Josefstrasse 205, 8005 Zürich', NULL,
NULL, 300, 0),

(52, 'FIFA Museum', 'Seestrasse 27, 8002 Zürich', NULL, NULL, 200, 0),

(53, 'CERN Globe', 'Route de Meyrin 385, 1217 Meyrin', 2493089, 1121223.5,
400, 0),

(55, 'Siky Park', 'Chemin de la Forêt 1, 2743 Eschert', 2598694.5,
1234657.375, 400, 1),

(57, 'Münster Bern', 'Münsterplatz 1, 3011 Bern', NULL, NULL, 300, 0),

(59, 'Forum Fribourg', 'Route du Lac 12, 1763 Granges-Paccot', 2578229.5,
1185964.125, 500, 1),

(61, 'Swissminiatur', 'Melide, Via Cantonale 11', 2716867.5, 1090180.125,
300, 1),

(63, 'Fondation Beyeler', 'Baselstrasse 101, 4125 Riehen', NULL, NULL, 400,
0),

(65, 'Knies Kinderzoo', 'Oberseestrasse 36, 8640 Rapperswil', 2704800,
1231265.75, 400, 0),
```

```
(67, 'Ballenberg West', 'Museumsstrasse 131, 3858 Hofstetten', NULL, NULL,
1200, 0),

(69, 'Chaplin''s World', 'Route de Saint-Légier 2, 1804 Corsier-sur-Vevey',
NULL, NULL, 500, 0),

(71, 'Alpamare', 'Gwattstrasse 12, 8808 Pfäffikon SZ', NULL, NULL, 500, 0),

(73, 'Autobahnraststätte Würenlos', 'A1, 5436 Würenlos', 2668589,
1254630.875, 600, 0),

(75, 'Zentralbibliothek Zürich', 'Zähringerplatz 6, 8001 Zürich',
2683551.5, 1247618.125, 200, 1),

(77, 'Flugplatz Emmen', 'Emmen, Seetalstr. 175.97', 2667257, 1218115, 4000,
1),

(79, 'Hafen Romanshorn', 'Romanshorn, Friedrichshafnerstr. 55', 2745436.25,
1269097.75, 600, 1),

(81, 'Landhaus Solothurn', 'Landhausquai 4, 4500 Solothurn', 2607488.75,
1228403.25, 200, 0),

(83, 'Dreispitzareal', 'Münchensteinerstrasse, 4053 Basel', NULL, NULL,
1200, 0),

(85, 'Schloss Laufen', 'Dachsen, Laufen am Rheinfall 2.1', 2687700.5,
1280421.5, 700, 1),

(87, 'Olympisches Museum, Olympiamuseum', 'Quai d''Ouchy 1, 1006 Lausanne',
2538276.75, 1151058.75, 400, 1),

(89, 'Rolex Learning Center', 'Route Cantonale, 1024 Ecublens', NULL, NULL,
400, 0),

(91, 'Marmorbrücke (Grand Pont)', 'Place de la Riponne, 1005 Lausanne',
NULL, NULL, 400, 0),

(93, 'Kaserne Basel', 'Klybeckstrasse 1b, 4057 Basel', NULL, NULL, 400, 0),

(95, 'Vögele Kultur Zentrum', 'Gwattstrasse 14, 8808 Pfäffikon SZ',
2702574.75, 1228481.5, 300, 1),

(97, 'Schokoladenmuseum Lindt', 'Schokoladenplatz 1, 8802 Kilchberg', NULL,
NULL, 400, 0),

(99, 'Sauriermuseum Aathal', 'Zürichstrasse 69, 8607 Aathal', NULL, NULL,
400, 0),

(101, 'Tierpark Goldau', 'Parkweg 30, 6410 Goldau', NULL, NULL, 800, 0),
```

```
(103, 'Rathaus Basel', 'Marktplatz 9, 4001 Basel', NULL, NULL, 200, 0),

(105, 'Sechseläutenplatz', 'Sechseläutenplatz, 8001 Zürich', NULL, NULL,
400, 0),

(107, 'Conny-Land', 'Connylandstrasse 1, 8564 Lipperswil', NULL, NULL, 500,
0),

(109, 'Sitterviadukt BT', 'Sittertalstrasse, 9014 St. Gallen', NULL, NULL,
600, 0),

(111, 'Kyburg', 'Schloss Kyburg, 8314 Kyburg', 2698404.75, 1257229.625,
300, 0),

(113, 'Abtei Payerne', 'Place du Marché, 1530 Payerne', 2561675,
1185628.75, 300, 0),

(115, 'Schloss Greyerz', 'Rue du Château 8, 1663 Gruyères', 2572781.5,
1159326.125, 300, 0),

(117, 'Lac des Dix Staumauer', 'Hérémence, Rte du Barrage 69', 2604222.75,
1103422.5, 1000, 0),

(119, 'Bahnhof Brig', 'Brig, Bahnhofstr. 2', NULL, NULL, 600, 1),

(120, 'Stockalperschloss', 'Alte Vereina-Strasse, 3900 Brig', NULL, NULL,
400, 0),

(121, 'Bahnhof Locarno', 'Muralto, Via Scazziga Vittore 8', 2704454.5,
1113517, 500, 1),

(122, 'Piazza Grande', 'Piazza Grande 7, 6600 Locarno', NULL, NULL, 400,
0),

(123, 'Bahnhof Biasca', 'Biasca, Via Bellinzona 5', 2718173, 1134584.375,
400, 1),

(124, 'Schloss Sargans', 'Schlossstrasse, 7320 Sargans', NULL, NULL, 300,
0),

(126, 'Tamina Therme', 'Hans-Albrecht-Strasse, 7310 Bad Ragaz', 2757140.25,
1207415.75, 400, 1),

(129, 'Hallenbad City Zürich', 'Sihlstrasse 72, 8001 Zürich', 2682651.75,
1247403.75, 200, 0),

(132, 'Eisstadion Davos (Vaillant Arena)', 'Talstrasse 41, 7270 Davos',
NULL, NULL, 400, 0),

(133, 'Bahnhof St. Moritz', 'St. Moritz, Via Grevas 65', NULL, NULL, 500,
1),
```

```
(134, 'Schanze Einsiedeln', 'Schnabelsbergstrasse, 8840 Einsiedeln', NULL,
NULL, 500, 0),

(135, 'Bahnhof Samedan', 'Samedan, Via Retica 22', 2786751.75, 1156529.875,
500, 1),

(136, 'Flugplatz Samedan', 'Plazza Aviatica 11, 7503 Samedan', 2786897.25,
1156100.375, 1500, 1),

(137, 'Bahnhof Scuol-Tarasp', 'Scuol, Via da la Staziun 40', 2817367.5,
1186444.125, 400, 1),

(138, 'Bärenpark Bern', 'Grosser Muristalden 6, 3006 Bern', 2601603.5,
1199624.875, 400, 1),

(140, 'Zentrum Paul Klee', 'Monument im Fruchtland 3, 3006 Bern', NULL,
NULL, 600, 0),

(146, 'Internationales Uhrenmuseum', 'Rue des Musées 29, 2300 La Chaux-de-
Fonds', NULL, NULL, 300, 0),

(159, 'Bahnhof Lenzburg', 'Lenzburg, Bahnhofstr. 50', 2655839.75,
1248802.625, 500, 1),

(174, 'Bahnhof Thalwil', 'Gotthardstr. 24, 8800 Thalwil', 2685113.75,
1238918.25, 500, 1),

(176, 'Bahnhof Wädenswil', 'Wädenswil, Bahnhofstr. 9', 2693609, 1231703.75,
500, 1),

(178, 'Bahnhof Küsnacht ZH', 'Küsnacht, Kohlrainstr. 16', NULL, NULL, 400,
1),

(182, 'Bahnhof Hinwil', 'Hinwil, Bahnhofplatz 1.5', 2706085.75,
1239753.875, 500, 1),

(185, 'Bahnhof Bassersdorf', 'Bassersdorf, Bahnhofstr. 40.1', 2689739.5,
1255290.75, 400, 1),

(186, 'Bahnhof Kloten', 'Lindenstrasse 10, 8302 Kloten', 2686265.25,
1256007.625, 400, 1);
```

consoleColor.py

```
import re
from pathlib import Path

## ===== CONSOLE COLOR LOGIC ======

PathToConfigFile = Path("./scriptLogic/consolecolor/config.txt")
```

```
def writeConsoleColorConfig(colorCode):
    with open(PathToConfigFile, "w", encoding="utf-8") as configFile:
        configFile.write(colorCode)
        print(f"Deine bevorzugte Konsolenfarbe wurde unter {PathToConfig-
File} gespeichert.")
        print(f"Um die bevorzugte Konsolenfarbe zu ändern, lösche die Datei
'{PathToConfigFile}' und starte das Skript neu.")

def setUpConsoleColor():
    # No config file found: Ask prefered console color from user and write
config file
    if not PathToConfigFile.exists():
        print("Welche Schriftfarbe möchtest du auf der Konsole haben?")
        print("1 = Standard, 2 = blau, 3 = grün")

        userInputAccepted = False
        while not userInputAccepted:
            userInput = input('Bitte Zahl eingeben: ')
            if re.fullmatch(r"^[1-3]$", userInput):
                match int(userInput):
                    case 1:
                        # Standard console color
                        writeConsoleColorConfig('\033[0m')
                    case 2:
                        # Blue
                        writeConsoleColorConfig('\033[34m')
                    case 3:
                        # Green
                        writeConsoleColorConfig('\033[32m')
                    case _:
                        print("Die Eingabe konnte nicht zugeordnet
werden.")

                userInputAccepted = True


    # Set console color to user preference
    with open(PathToConfigFile, "r", encoding="utf-8") as configFile:
        colorCode = configFile.read().strip()
        # Set conosle to desired color
        print(colorCode)
```