Bericht zum Programmierprojekt – Eszter Bukovszky, 824911

30.09.2024.

In diesem Projekt habe ich ein Spiel implementiert, bei dem der Benutzer die Rolle eines Tutors an einer Universität übernimmt. Der Tutor muss die Arbeiten von seinen 8 Studenten in zwei Runden bewerten, und zwei Feedback-Sitzungen durchführen. Ziel des Spiels ist es, durch erfolgreiches Lösen von Aufgaben und das richtige Vergeben von Noten das Spiel zu gewinnen. Der Tutor kann verschiedene Befehle eingeben, die abhängig vom Spielfortschritt unterschiedliche Auswirkungen haben.

Während des Projekts bin ich auf verschiedene Herausforderungen gestoßen, die sorgfältige Prüfungen und Tests notwendig gemacht haben. Eine der größten Herausforderungen war, wie ich mein Programm strukturiere. Ich habe beschlossen, das Programm in verschiedene Module aufzuteilen, weil dies den Code übersichtlicher gemacht und beim Testen der Funktionalität geholfen hat.

Eines der Hauptprobleme war die Entwicklung der ‘Feedback’-Teile. Schon am Anfang habe ich diese in separaten Python-Dateien behandelt, weil ich gedacht habe, dass sie am komplexesten sind. Die Implementierung von ‘Feedback 2’ war das größte Problem, weil ich nicht genau wusste, welche Codesätze gültig sein sollen, ich habe die Definition nicht richtig verstanden. Aber nachdem es mir endlich klar geworden ist, welche Regel gelten, war es viel einfacher, diesen Teil zu implementieren.

Das andere große Problem bei dem Projekt lag in der Eingabe der Noten. Ursprünglich hatte ich es so kodiert, dass jeder Student nur eine Note bekommen muss. Über die zweiten Noteneintragung habe ich mir zu diesem Zeitpunkt noch nicht so viele Gedanken gemacht. Ich habe jedoch bald gemerkt, als ich mich nochmal bisschen ausführlicher mit diesem Teil beschäftigt habe, dass diese Lösung nicht angemessen ist, weil sie nicht den tatsächlichen Bewertungsprozess widerspiegelt, und der Spieler für jeden Studenten 2 Noten eintragen muss. Nach dieser Erkenntnis habe ich die Funktion ‘leistungsuebersicht_erstellen’ ergänzt, damit für jeden Studenten zwei Noten eingetragen werden können.

Diese Änderung hat neue Herausforderungen mit sich gebracht, da ich die anderen Funktionen und die Struktur des Programms anpassen musste, um das Zwei-Noten-System richtig zu handhaben. Die Neugestaltung der Notenverwaltung wurde recht komplex, da ich sicherstellen musste, dass der Bewertungsprozess logisch geblieben ist und bestimmte Befehle nur dann erlaubt sind, wenn die erste oder die zweite Noteneintragung schon fertig geworden ist.

Darüber hinaus hatte ich das Problem, dass ich den Status der Variable `schokolade_gegessen` innerhalb einer verschachtelten Funktion ändern musste. Dabei musste ich entscheiden, wie ich diese Variable verwalte, um sicherzustellen, dass Änderungen korrekt und ohne Konflikte vorgenommen werden.

Ich habe mich für `nonlocal` entschieden, um dieses Problem zu lösen. Mit der ‘nonlocal’-Anweisung konnte ich den Wert von ‘schokolade_gegessen’ in der inneren Funktion ändern,

ohne auf die globale Ebene zugreifen zu müssen. Dadurch konnte ich den Status lokal verwalten und Konflikte im globalen Namensraum vermeiden. Die Anweisung `global` wäre daher weniger angemessen gewesen.

Die Verwendung von `nonlocal` war effektiver als die Erstellung einer Klasse, weil ich nur eine einfache Datenverwaltung gebraucht habe. Die Nutzung einer Klasse wäre komplexer gewesen und hätte eine zusätzliche Struktur erfordert, die in diesem Fall nicht notwendig war. “Nonlocal“ hat eine effiziente Lösung für mein konkretes Problem geboten, ohne die Implementierung unnötig zu komplizieren.

Bei der Implementierung gab es ein Problem, bei dem die `Evaluationspunkte` in der `Leistungsübersicht` doppelt angezeigt wurden. Dies hat zu einer falschen Evaluation geführt, da die Punkte fälschlicherweise zweimal addiert wurden.

Der Fehler ist entstanden, weil ich bei der Implementierung der Dateien `main.py` und `beschreibungen.py` die Dateien `feedback_1.py` und `feedback_2.py` schon fast fertig geschrieben hatte. Ich hatte die `Evaluationspunkte` bereits in den Feedbackdateien gespeichert, deswegen waren die Punkte bereits in der `Leistungsübersicht` vorhanden, bevor die Evaluation angezeigt wurde.

Beim Kodieren der `main.py` und `beschreibungen.py`-Dateien habe ich nicht mehr daran gedacht, dass ich die Punkte schon in den Feedback-Dateien gespeichert hatte. Das hat dazu geführt, dass die `Evaluationspunkte` durch die Befehle `give feedback` und `delay grade` in der `main.py` wieder addiert wurden.

Ich habe deswegen die Punktzuweisungen zur Leistungsübersicht aus den Feedback-Dateien entfernt. Durch das Entfernen der doppelten Punktezuweisungen konnte ich das Problem beheben, und die Evaluation wird nun korrekt angezeigt.

(Ein weiterer wichtiger Punkt war, dass ich zu Beginn die in den Feedback-Sitzungen erhaltenen Evaluationspunkte direkt in der Leistungsübersicht gespeichert habe. Es ist mir aber später aufgefallen, dass in der Aufgabenstellung festgelegt war, dass die Punkte erst nach dem erfolgreichen Abschluss von Feedback 2 angezeigt werden dürfen, wenn der Befehl ‘get eval’ eingegeben wird. Daher musste ich die Punkteverwaltung anpassen, um sicherzustellen, dass die Evaluationspunkte erst zu diesem Zeitpunkt hinzugefügt werden. So habe ich die Evaluationspunkte in einer neuen Variable zwischengespeichert.)

Ich habe mich bei der Implementierung des Projekts für die Verwendung von Klassen entschieden, weil ich so die unterschiedlichen Funktionen und Daten in separate, gut strukturierte Einheiten einteilen konnte. Die Klasse hat es mir auch erleichtert, Objekte zu erstellen.

Das Programm wurde dadurch logischer und übersichtlicher, und es war einfacher, während der Programmierung Änderungen vorzunehmen, ohne dass ich das gesamte Programm neu schreiben musste. Diese Struktur hat deutlich gemacht, welche Funktion an welchem Teil arbeitet, und hat es mir ermöglicht, die verschiedenen Bewertungs- und Feedbackprozesse unabhängig voneinander zu verwalten.

Ein anderer Grund, warum ich die Klasse verwendet habe, ist dass ich mich ausführlicher mit ihnen auseinandersetzen wollte, weil ich bisher noch wenig Erfahrung damit hatte.

Eine der wichtigsten Erkenntnissen, die ich aus diesem Projekt gelernt habe, ist die Bedeutung von Planung und Modularität für komplexere Programme.

Am Anfang habe ich die einzelnen Funktionen unabhängig voneinander entwickelt, und dann hat sich die Verknüpfung dieser Funktionen manchmal als schwierig erwiesen.

So habe ich zum Beispiel die Funktionen ‘ausruhen’, ‘schokolade_essen’ und ‘sind_gueltige_noten’ innerhalb der Funktion ‘noteneintragung’ definiert, weil ich beschlossen habe, dass sie nur in der Noteneintragung verfügbar sein sollten, und es hat mehr Sinn gemacht, diese drei Funktionen innerhalb der größeren Funktion zu definieren.

Am Anfang habe ich die Funktionen ‘ausruhen und ‘schokolade_essen’ so implementiert, dass sie keine bestimmten Zustände erfordert haben. Nachdem man diese Befehle ausgeführt hatte, konnte man direkt mit der Noteneintragung fortfahren, ohne einen zusätzlichen Befehl wie ‘grade’ eingeben zu müssen. Das hat jedoch dazu geführt, dass die Logik des Programms weniger stringent war, da der Benutzer nach der Ausruhe- oder Schokoladenfunktion sofort weitermachen konnte, ohne den Prozess bewusst wieder aufzunehmen. Um dies zu verbessern, habe ich die Logik angepasst, sodass nach den Befehlen ‘rest’ oder ‘eat chocolate’ immer der Befehl ‘grade’ erneut eingegeben werden muss, um mit der Noteneintragung fortzufahren. Dadurch wurde sichergestellt, dass der Benutzer nach einer Pause bewusst zur Noteneintragung zurückkehrt, was die Konsistenz des Programms und den Spielfluss verbessert hat.

Die Fehlersuche war während der Entwicklung auch von wesentlicher Bedeutung. Ich habe hauptsächlich mit print()-Statements gearbeitet, um Fehler zu lokalisieren, insbesondere während der Bewertungs- und Feedbacksitzungen. Diese Methode war sehr effektiv und hat mir geholfen, Probleme schnell zu finden und das Spiel dadurch besser zu gestalten.

Ich habe durch dieses Projekt viel gelernt, insbesondere wie man ein komplexeres Programm strukturiert und wie man mit verschiedenen Zuständen und Ereignissen umgeht. In Zukunft würde ich wahrscheinlich mehr daran denken, die Struktur des Programms schon zu Beginn des Entwurfs genauer zu überdenken, damit ich später keine größeren Änderungen vornehmen muss. Ich würde auch viel mehr Zeit auf das Testen verwenden, da bei der Verknüpfung von Funktionen, die zu Beginn nicht sichtbar sind, Fehler auftreten.

Eine weitere Sache, die ich anders machen würde, ist es, wie ich Befehle und Funktionen benenne. Am Anfang hatte ich zum Beispiel getrennte Befehle für ‘Feedback 1’ und ‘Feedback 2’, später habe ich sie dann überarbeitet, um eine bessere Lösung zu verwenden, bei der ‘give feedback’ und ‘delay grade’ die Befehle sind, ohne zwischen Feedback 1 und 2 bewusst zu unterscheiden. Die Überarbeitung dieser Befehle hat dazu beigetragen, dass das Programm logischer geworden ist.

Insgesamt ist die wichtigste Erkenntnis, dass einige der Probleme dadurch entstanden sind, dass ich mich bei der Programmierung in Details verloren habe und bestimmte Teile nicht genau so umgesetzt habe, wie sie in der Aufgabenstellung vorgeschrieben waren. Es war nicht einfach, auf alles gleichzeitig zu achten. In Zukunft würde ich deswegen viel mehr

Gewicht auf eine sehr genaue und gut durchdachte Planung im Voraus legen, um solche Fehler zu vermeiden.

Flowchart über meinen gewählten Programmablauf: