Arbeitsprotokoll zum Modulprojekt:

Quordle-CLI

Hugo Meinhof

815220

PRO2-A, SoSe 2023, Jana Götze

Universität Potsdam

Fr. 15.09.2023

Arbeitsprotokoll:

Das Spiel, in seinem Kern, besteht aus drei Klassen, die ähnlich dem Model-View-Controller Konzepts verhalten. Dies liegt nicht etwa daran, dass ich mit aller Gewalt dieses Konzept umsetzen wollte, sondern, dass es einfach so am meisten Sinn ergeben hat.

Jede dieser Klassen liegt in einer Datei mit ähnlichem Namen, entsprechend Konvention. Die Klasse GUI speichert und verarbeitet keine Daten. Mit ihr werden ausschließlich vorbereitete Datenstrukturen für den Spielenden visualisiert. Da Quordle und Sequence sich beinahe genau so verhalten wie wenn man vier mal Wordle gleichzeitig spielen würde, implementiert Wordle große teile der Logik und Datenspeicherung. GameCore verbindet vier Instanzen von Wordle mit der GUI, und reicht so Eingaben und Ausgaben zwischen den Modulen und dem Benutzenden hin und her. Mit diesen drei Klassen ließe sich das Spiel in jegliches Kommandozeilenprogramm einpflegen. Um das Spiel aufrufen und mit den nötigen Daten füttern zu können, gibt es die Dateien main.py, data_install.py und process_wordlist.py. Jede dieser Dateien ist einzeln ausführbar, falls erwünscht. data_install.py nutzt process_wordlist.py um den Standarddatensatz herunterzuladen, zu verarbeiten und bereitzustellen. Dies ist optional und um die Bequemlichkeit der Programmnutzung zu steigern.

main.py liest die erforderlichen Daten ein, startet die Installation des Standarddatensatzes, falls erwünscht, und führt das Spiel aus. Die Idee ist, dass main.py sehr einfach zu bedienen und gleichzeitig sehr flexibel ist. Daher können hier die Daten spezifiziert werden, müssen aber nicht, und auch die Installation der Standarddaten erfordert nicht mehr als eine Bestätigung.

Die Datenspeicherung ist in vier Stufen geteilt. Alles was außerhalb des Spiels, d.h. Model-View-Controller, liegt, speichert keine Daten, oder liefert sie einmalig an den Controller. Innerhalb des Spiels speichert die Liste der vier Models, die vier Wordle Instanzen, alles was spezifisch auf diesen einen Wordle zutrifft. Alles was wordle-übergreifende Daten sind, wird vom Controller, GameCore, gespeichert. Der View, GUI, speichert nichts.

So wird sichergestellt, dass es keine unsynchronisierte doppelte Speicherung von Daten gibt, sondern die Daten immer frisch vom Besitzer abgefragt werden.

Die Unterscheidung zwischen den Spielmodi, Quordle und Sequence, wird

ausschließlich im Controller gemacht. Weder View, noch Model wissen welcher Modus aktiv ist. Dies liegt daran, dass Sequence im Grunde dem Modus Quordle, mit weniger dargestellten Daten entspricht. Daher reicht es, wenn der Controller beim Aufbereiten der Daten der Modelle die richtigen Informationen weglässt, wenn der View aufgerufen wird. Diese Trennung vereinfacht den Code und erhöht die Sicherheit. So bleiben die Schnittstellen der Klassen unabhängig vom Spielmodus gleich.

Mein Arbeitsablauf war im Grunde so:

- vertrautmachen mit der Aufgabe
- erstellen der GUI
- schreiben der Spiellogik
- suche nach einer passenden Wortquelle, Wortverarbeitung und Installation
- säubern des Codes und beschäftigen mit Rückmeldung aus Review

Der erste Schritt, vertrautmachen mit der Aufgabe, mag klingen wie das banale Lesen der Aufgabenstellung, jedoch meine ich damit eher das spielen des zu kopierenden Spiels. Das heißt, ich habe versucht in verschiedene Situationen zu kommen, um zu sehen wie das Spiel reagiert, habe mir Notizen gemacht und Screenshots gespeichert, um das Verhalten und Aussehen später referenzierten zu können.

An dieser Stelle entschied ich mich für die generelle Struktur des Programms und in welcher Reihenfolge ich Code schreiben würde. An diesem Plan hat sich im Laufe des Projekts nicht viel verändert.

Im nächsten Schritt erstellte ich die GUI. Dies hatte vor allem den Grund, dass ich dann im weiteren Verlauf immer sehen konnte was in der Spiellogik passiert und was funktioniert. Die GUI hat sich zum testen von work-in-progress-code viel besser geeignet als unit-tests. Gleichzeitig habe ich damit eine klare Kommunikationsschnittstelle festgelegt. Da die GUI Klasse keine Daten speichern, und Werte errechnen soll, legte ich durch die Erstellung des Interfaces fest auf welche Art und Weise die Daten gespeichert und kommuniziert werden. Dies gab mir beim erstellen der Spiellogik eine klare Struktur.

Die Erstellung der Logik der Models war dann auch der nächste Schritt. Hier habe ich von Anfang an kleine translation-layers geschrieben, um Kommunikation zwischen den Modellen und des Views zu ermöglichen. Erst nachdem die Logik

der Modelle gefestigt war, entstand ein richtiger Controller, welcher die translation-layer ersetzte.
Bis zu diesem Zeitpunkt habe ich noch mit Dummy-Daten gearbeitet, da die Daten ersetzbar sind, und am Spiel direkt nichts ändern.
Rückblickend bereue ich nur die Entscheidung meine Testscripte auf der GUI und nicht assert passierten Unittests aufgebaut zu haben. Nicht weil ich Unittests für besser halte, ganz im Gegenteil, sondern weil ich jetzt bei der Abgabe keine vorzuweisen habe.
Ich plane auch im weiteren meine Programme unter Zuhilfenahme von test-driven-development zu schreiben, wie dieses auch, und ich plane in Zukunft mehr traditionelle Unittests zu benutzen. An vielen Stellen finde ich sie jedoch noch wenig aussagekräftig für den Aufwand den sie bedeuten, oft weil sie nicht dicht genug an der tatsächlichen Benutzung liegen. Ob ein Input-Statement Daten richtig verarbeitet und weitergibt lässt sich meist nur schlecht testen, selbiges gilt dafür wie etwas dargestellt wird.
Wenn ich noch mehr Zeit hätte um das Programm aufzupolieren, würde ich die Interfaces der Klassen stärker schließen, um für mehr Sicherheit im Code zu sorgen. Außerdem würde ich versuchen auch allgemeinere Datensätze aufbereiten zu können, und ich würde mir wahrscheinlich meinen eigenen Datensatz erstellen, welcher dann online zur Verfügung stünde. Des weiteren würde ich versuchen noch ein Review zu bekommen, da im Rahmen dieser Prüfungsleistung richtige Kooperation untersagt ist. So abgeschottet arbeiten zu müssen, lässt mich das Arbeiten mit anderen sehr wertschätzen.

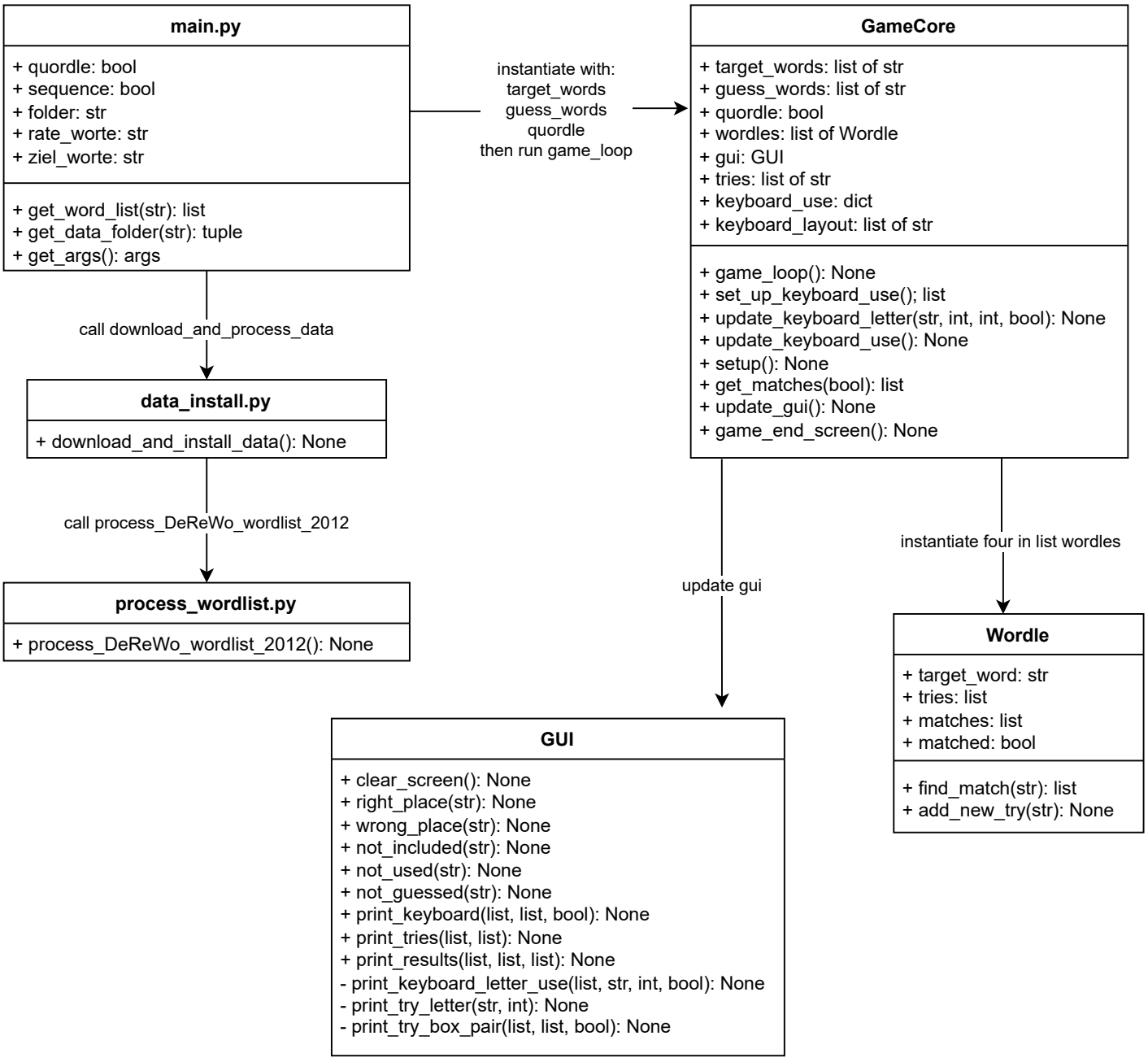
Nun noch:

1. Link zum Repo: https://github.com/Theoreticallyhugo/quordle_CLI

2. Letzter relevanter Commit: cbad10d

3. Mein Review für Angelina:
   https://github.com/Angelina-L12/PRO_II_Modulprojekt/pull/1

4. Angelinas Review für mich:
   https://github.com/Theoreticallyhugo/quordle_CLI/pull/3


main.py
+ quordle: bool
+ sequence: bool
+ folder: str
+ rate_worte: str
+ ziel_worte: str
+ get_word_list(str): list
+ get_data_folder(str): tuple
+ get_args(): args
instantiate with:
target_words
guess_words
quordle
then run game_loop
GameCore
+ target_words: list of str
+ guess_words: list of str
+ quordle: bool
+ wordles: list of Wordle
+ gui: GUI
+ tries: list of str
+ keyboard_use: dict
+ keyboard_layout: list of str
+ game_loop(): None
+ set_up_keyboard_use(); list
+ update_keyboard_letter(str, int, int, bool): None
+ update_keyboard_use(): None
+ setup(): None
+ get_matches(bool): list
+ update_gui(): None
+ game_end_screen(): None
call download_and_process_data
data_install.py
+ download_and_install_data(): None
call process_DeReWo_wordlist_2012
process_wordlist.py
+ process_DeReWo_wordlist_2012(): None
update gui
instantiate four in list wordles
Wordle
+ target_word: str
+ tries: list
+ matches: list
+ matched: bool
+ find_match(str): list
+ add_new_try(str): None
GUI
+ clear_screen(): None
+ right_place(str): None
+ wrong_place(str): None
+ not_included(str): None
+ not_used(str): None
+ not_guessed(str): None
+ print_keyboard(list, list, bool): None
+ print_tries(list, list): None
+ print_results(list, list, list): None
- print_keyboard_letter_use(list, str, int, bool): None
- print_try_letter(str, int): None
- print_try_box_pair(list, list, bool): None