**Python - Programmentwurf**

**Team Mitglieder:**

|  |  |
| --- | --- |
| **Name** | **E-Mail (DHBW)** |
| Carolin Auchter | auchter.caroli-it21@it.dhbw-ravensburg.de |
| Christian Bartsch | bartsch.christ-it21@it.dhbw-ravensburg.de |
| Steven Butz | butz.steven-it21@it.dhbw-ravensburg.de |
| Nicolas Equit | equit.nicolas-it21@it.dhbw-ravensburg.de |

**Verwendete Python Version:**

3.9.7

**Verwendete Pylint Version:**

2.12.2

**Verwendete nachinstallierte Bibliotheken:**

json, random, numpy, sys, weitere für Tests?

**Beschreibung des Dateiformats zum Speichern des Spiels:**

* **Gespeicherter Spielstand einfügen (wenn lesbar):** Dateien nach Spielende des Spiels, das in Log\_Spielablauf\_in\_Konsole.txt dokumentiert ist:

dice.json kniffel\_player.json

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Computer, drinnen enthält.

Automatisch generierte Beschreibung

* **Beschreibung des Formats:**

Wir verwenden in unserem Programm zwei JSON-Dateien, zum einen dice.json zum Zwischenspeichern der Würfelstände in einem eindimensionalen Array und zum anderen kniffel\_player.json zum Speichern der Spieler und des aktuellen Spielstands in einem zweidimensionalen Array. Wir haben uns für dieses Dateiformat entschieden, da es einen einfachen und kompakten Austausch von strukturierten Daten ermöglicht, die in Arrays verschachtelt werden können. Außerdem sind die Dateien, wie oben ersichtlich, menschenlesbar, was den Programmierprozess bzw. die Fehlersuche teilweise deutlich erleichtert. Um die Lesbarkeit der JSON-Dateien zu erhöhen, haben wir sie während des Schreibprozesses mit einem „indent=4“ versehen, sodass die Daten in den Arrays sichtlich strukturiert und nicht in einer langen Zeile abgelegt werden.

**Beschreibung des Benutzerinterfaces: Wie sieht die Ein- und Ausgabe aus?**

Da die Aufgabenstellung grafische Oberflächen nicht zulässt, sprechen wir die Spieler:innen unseres Programms direkt über die Konsole mit Ausgaben (in Form von print-statements) an und fordern sie ebendort an entsprechenden Punkten im Spielverlauf zu notwendigen Eingaben auf. Diese werden in Form von strings entgegengenommen und nach Bedarf weiterverarbeitet (z.B. über split-Funktionen, isdigit-Prüfungen und Umwandlungen in integer-Dateitypen).

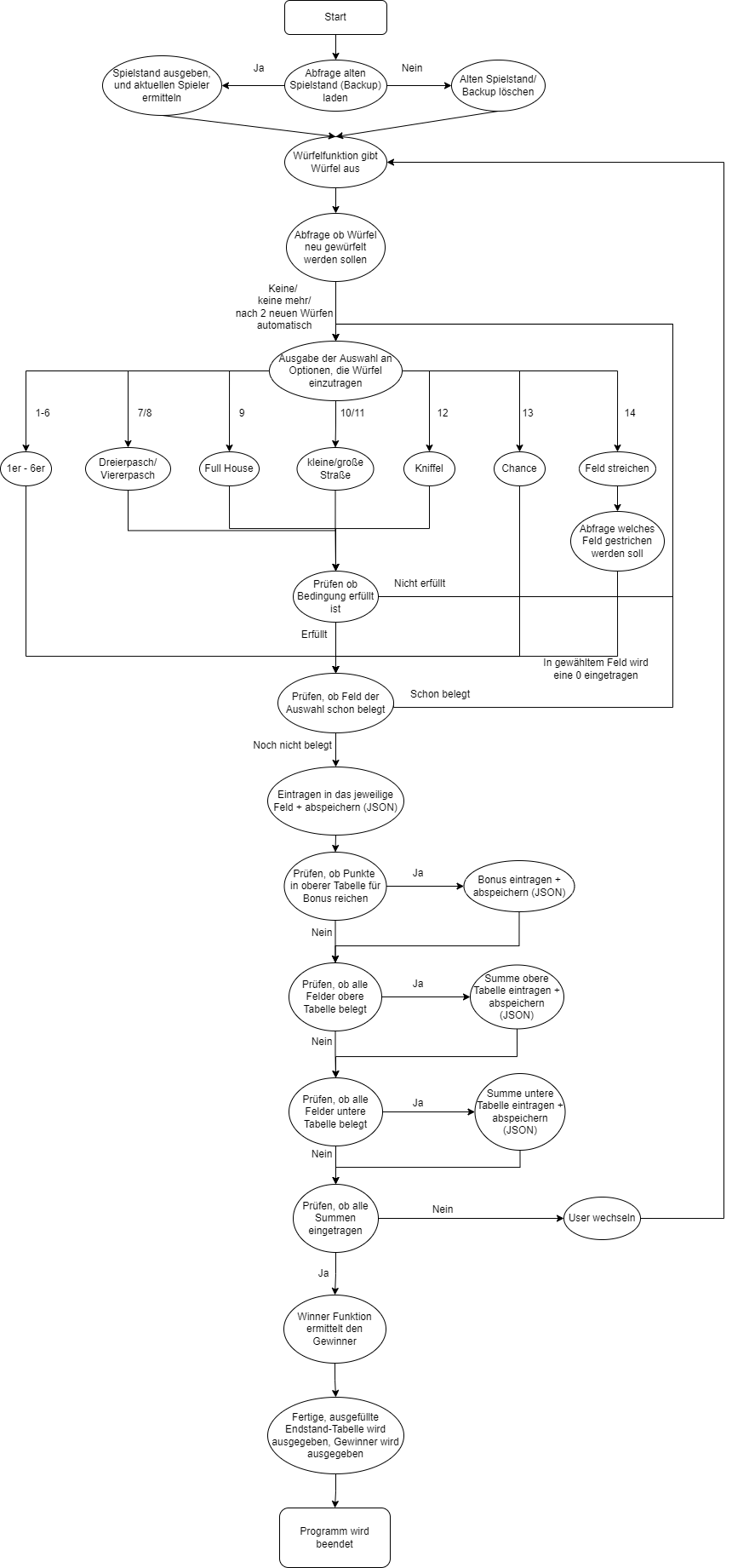
Wir beginnen mit einmaligen Events: einer Begrüßung, einer Kurzanweisung zur Handhabung des Programms (quasi Mini-Spielanleitung) und der Frage, ob der Spielstand des vorherigen Spiels aus dem Backup geladen werden soll. Wird diese Frage bejaht, wechseln wir direkt in das vorherige Spiel, bei Verneinung werden die Spieler:innen nach ihren Namen gefragt und das Programm startet in ein neues Spiel.

Ab diesem Zeitpunkt befinden wir uns in sich wiederholenden Events, in denen sich die Anzeige der bisher gewürfelten Augenzahlen mit der Abfrage des nächsten Zuges abwechseln, bis die drei Würfelmöglichkeiten erreicht sind und der gewünschte Eintrag in die Spielstands-Tabelle abgefragt wird. Für eine gute Spielstands-Übersicht lassen wir nach jedem beendeten Spielzug bzw. nach jeder Eintragung in die Spielstands-Tabelle die gesamte Tabelle mit allen bisherigen Einträgen ausgeben, damit die Spieler:innen erkennen können, welche Kombinationen ihnen noch fehlen. Die Ausgabe erfolgt in formatierter Tabellenform, ähnlich dem Aufbau eines realen Spielblocks.

Wir haben uns bemüht die Eingabeaufforderungen eindeutig zu formulieren und sie für eine einfache Handhabung (bis auf die Eingabe der Spieler:innennamen) auf Zahlen einzuschränken, die mit bestimmten Menüpunkten einhergehen. So wird die Wahrscheinlichkeit von Fehleingaben eingeschränkt. Kommt es trotzdem zu falschen User-Inputs, werden die Spieler:innen höflich auf richtige Eingabeformen hingewiesen.

Eine Anleitung zum Pausieren des Spiels bekommen die Spieler:innen zu Beginn des Spiels per Kommentar geliefert. Am Spielende wird der/die Gewinner:in bekanntgegeben, der finale Spielstand angezeigt und das Programm endet automatisch.

**Beschreibung der Architektur:**



**Beschreibung des Computergegners (5er Gruppen):** entfällt, da 4er-Gruppe

**Dokumentation von einem kompletten Spielablauf:**

siehe Datei Spielablauf\_in\_Konsole.txt

Pfad angeben?

**Log von den Tests: Alle Tests über Konsole ausführen und Ausgabe hier einfügen (oder extra Datei)**

**Bewertung der Testergebnisse: Grund für fehlschlagende Tests**

**Code-Coverage Ausgabe: Tests mit Coverage ausführen und Ausgabe hier einfügen**

**Bewertung der Coverage und Sinnvollheit der Tests:**

* **Gewünscht ist eine Coverage von min 75% (pro Datei)**

**Wurde dies erreicht? Wenn nein, warum nicht?**

* **Begründet warum ihr euch sicher seid alles mit Tests abgedeckt zu haben**

Fangfrage!? Testing potentiell unendlich

* **Fehlerfälle in den Tests überprüft?**

**Bewertung der Fehlersicherheit:**

* **Werden Fehleingaben korrekt abgefangen?**

Unser Programm prüft nach jedem User-Input, ob die Eingabe valide ist. Hierfür nutzen wir if/elif/else-Bedingungen oder try/except-Statements, die das Error und Exception Handling vereinfachen.

* **Zwei Beispiele**:

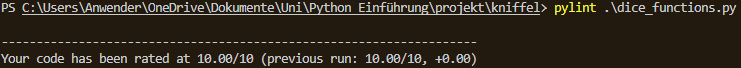
1. Zu Beginn des Spiels werden Spieler:innen gefragt, ob sie das vorherige Spiel aus dem Backup laden möchten (Funktion main in Modul kniffel\_main), als mögliche Antworten werden „j/n“ angeboten. Neben einem if-Statement für Antwortmöglichkeit „n“ und einem elif-Statement für Antwortmöglichkeit „j“, verfügt das Programm über ein else-Statement, das für alle Eingaben außerhalb von „j/n“ auffordert mit „j“ oder „n“ fortzufahren.
2. Bei der Spielzug-Auswahl (Funktion combine in Modul function\_decision) können Spieler:innen aus einer nummerierten Liste wählen, welcher Spielzug in die Spielstands-Tabelle eingetragen werden soll. Der User-Input wird in der Variable action\_combine in einen Integer umgewandelt, d.h. das Programm erwartet eine Zahl als Eingabe. Fehleingaben in Form von Buchstaben oder Sonderzeichen statt Zahlen werden innerhalb eines try/except-Statements mit einem ValueError abgefangen. Die/Der Spieler:in wird per print-Statement aufgefordert Zahlen einzugeben. Negative Zahlen werden innerhalb eines if/elif/else-Statements abgefangen und die/der Spieler:in wird aufgefordert einen Zahlenwert zwischen 1 und 14 einzugeben.

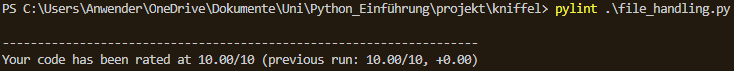
**Pylint Ausgabe:** 10.00/10 in allen Methodenblocks

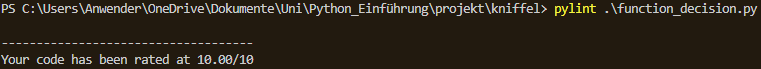


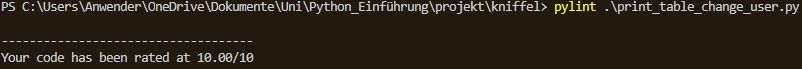


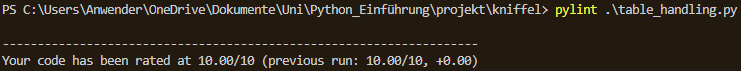












**Bewertung der Code-Qualität/Lesbarkeit:**

* **Begründet noch vorhandene Pylint Warnungen:**

Keine Pylint-Warnungen vorhanden

* **Begründet warum euer Code gut lesbar ist**
* Wir haben und, wie im Programmablaufdiagramm oben ersichtlich, für eine prozedurale bzw. funktionale Programmierweise entschieden. Um trotzdem eine gute Übersicht und Lesbarkeit des Codes zu gewährleisten haben wir die Funktionen in acht sinnhaft zusammenhängende Module aufgeteilt und jeweils in eigenen Dateien gespeichert, die dann in der main-Funktion oder anderen Modulen aufgerufen werden. Die einzelnen Module sind sinnvoll und selbstsprechend benannt und spiegeln so wider, welche Funktionen sie beinhalten.
* Wir sind von der Annahme ausgegangen in einem deutschen Entwicklerteam und für den deutschen Markt zu produzieren. Daher sind alle Ausgaben in deutscher Sprache gestaltet, dasselbe gilt für die Docstrings. Alle Variablennamen haben wir auf Englisch gewählt, um die englischsprachige Syntax von Python einheitlich zu gestalten. Jeder Variablenname ist dabei sinnvoll und selbstsprechend gewählt, sodass abgesehen von den Docstrings kaum weitere Kommentare nötig sind. Das einzige Wort, das wir durchgängig in deutscher Sprache verwendet haben ist das programmnamensgebende „Kniffel“. Wir hätten es ins internationale „Yahtzee“ übersetzen können, allerdings sind die Spielregeln für Yahtzee leicht abweichend von unseren Vorgaben, sodass wir es für sinnvoll erachten, in der Namensgebung bei „Kniffel“ zu bleiben.
* Wie in den nicht vorhandenen Pylint-Warnungen ersichtlich, halten wir uns an alle Python-Konventionen für gut lesbaren Code (Leerzeichen, Leerzeilen, keine ungenutzten Variablen, etc.).
* Für die Übertragung von einem Windows-System auf ein Linux-System waren keine Anpassungen nötig.
* Im Modul print\_table\_change\_user befindet sich die Funktion ausgabe, die die aktuelle Spielstands-Tabelle ausgibt. Wir hätten, wie an anderen Stellen im Code geschehen, die print-Statements in ein Statement zusammenfassen können, haben uns aber der Übersichtlichkeit halber bewusst dagegen entschieden. Da die Formatierung aufgrund zahlreicher Tabs und Zeilenumbrüche sehr umfangreich ist, erschien uns die listenhafte Darstellung mit mehreren print-Statements dem gewünschten Output ähnlicher und daher übersichtlicher.

**Bewertungstabelle:**

**Ausgefüllte Bewertungstabelle (Vorschlag an alle über Discord raus, bei Fehlersicherheit nachgebessert, jetzt 5/5)**

**Pfad angeben?**