**Python - Programmentwurf**

**Team Mitglieder:**

|  |  |
| --- | --- |
| **Name** | **e-Mail (DHBW)** |
| Elias Keimer | Keimer.elias-it21@it.dhbw-ravensburg.de |
| Yannic Grafwallner | Grafwallner.ya-it21@it.dhbw-ravensburg.de |
| Jonas Heiß | Heiss.jonas-it21@it.dhbw-ravensburg.de |
| Ali Gündogan | Guendogan.ali-it21@it.dhbw-ravensburg.de |
| Christoph Gründer | Gruender.chris-it21@it.dhbw-ravensburg.de |

**Verwendete Python Version: 3.10**

**Verwendete Pylint Version: 57.0.0**

**Verwendete nachinstallierte Bibliotheken: prettytable, jsonpickle, io, random, datetime, os, TestCase, patch**

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

* Gespeicherter Spielstand einfügen (wenn lesbar)
* "[{\"1\": {\"Einser\": [3, 0]}}, {\"2\": {\"Zweier\": [0, 0]}}, {\"3\": {\"Dreier\": [6, 0]}}, {\"4\": {\"Vierer\": [0, 4]}}, {\"5\": {\"Funfer\": [10, 0]}}, {\"6\": {\"Sechser\": [0, 0]}}, {\"7\": {\"Oben\": [19, 4]}}, {\"8\": {\"Bonus\": [\"-\", \"-\"]}}, {\"9\": {\"GesamtOben\": [19, 4]}}, {\"10\": {\"Dreierpasch\": [0, 0]}}, {\"11\": {\"Viererpasch\": [0, 0]}}, {\"12\": {\"Full-House\": [0, 0]}}, {\"13\": {\"Kleine-Stra\\u00dfe\": [30, 0]}}, {\"14\": {\"Gro\\u00dfe-Stra\\u00dfe\": [0, 0]}}, {\"15\": {\"Kniffel\": [0, 0]}}, {\"16\": {\"Chance\": [17, 19]}}, {\"17\": {\"Unten\": [47, 19]}}, {\"18\": {\"Oben\": [19, 4]}}, {\"19\": {\"Gesamt\": [66, 23]}}]"
* Beschreibung des Formats
  + Speicherung im Standard Format der jsonpickle freeze()-Methode der Tabelle des Spielblocks

**Beschreibung des Benutzerinterfaces:**

* Wie sieht die Ein- und Ausgabe aus?
  + Eingabe erfolgt über Nummerneingabe je nach Aufforderung durch die Konsole

**Beschreibung der Architektur:**

* Aus der main() wird ein Objekt Game initialisiert, und durch dessen Memberfunktion startgame() gestartet.
* Aus game heraus werden die Methoden der Klassen UI, Gameblock, Player und Dice aufgerufen.
  + In der Klasse Game ist die Logik des Spiels umgesetzt. Von hier werden die Funktionen der anderen Klassen aufgerufen um komplexe Methoden wie Ein- und Ausgaben oder das Abspeichern und Berechnen der Werte im Spielblock zu kapseln.
  + Wir haben uns entschieden, alle Ein- und Ausgaben in der Klasse UI zu bündeln, um hier eine gute Übersicht über User-Interaktionen zur Verfügung zu stellen, und Ein- und Ausgabeprobleme kapseln zu können.
  + In der Klasse Gameblock, die eine Abbildung eines realen Kniffel-Blockes darstellt, werden in die durch den User gewählte Zeile die entsprechende Anzahl an Punkten gespeichert. Dieser Block wird durch jsonpickle automatisch gespeichert.
  + Ein Game hat als Attribut unter anderem eine Reihe von Würfel-Objekten, deren Funktionalität darin besteht, einen zufälligen wert zwischen 0 und 6 anzunehmen und durch den boolschen Wert isactivated die Position im Becher oder auf dem Tisch zu simulieren.
  + An einem Spiel nehmen zwei Entitäten der Klasse Spieler teil, deren Bestimmung es ist, zwischen physischer Eingabe eines echten Spielers und fake Eingabe bei PvE zu unterscheiden, und diese durch eventuelle Überladung der Funktion weiterzugeben.
  + In Block.json werden die Einträge des Spielblockes gespeichert.
  + In dem File test.py werden die erforderlichen Tests durch Aufrufen von Methoden und simulierten Ein- du Ausgaben durchgeführt.

**Beschreibung des Computergegners (5er Gruppen):**

* Der Computergegner funktioniert durch überladen der eingabe-Methoden in UI. Sollte die Methode überladen werden, wird der entsprechende übergebene Wert anstelle einer Usereingabe genutzt.
* Der Computergegner wählt keine besonderen Würfel aus, nutzt immer seine 3 Würfe und füllt den Block der Reihe nach von unten nach oben. Sollte er die Bedingungen für eine Zeile nicht erfüllen, so trägt er in diese eine 0 ein.

**Dokumentation von einem kompletten Spielablauf:**

* Siehe Anhang bsp\_Ablauf.txt

**Log von den Tests:**

* Siehe testres.txt

**Bewertung der Testergebnisse:**

**Code-Coverage Ausgabe:**

* A screenshot of a computer

  Description automatically generated with medium confidence

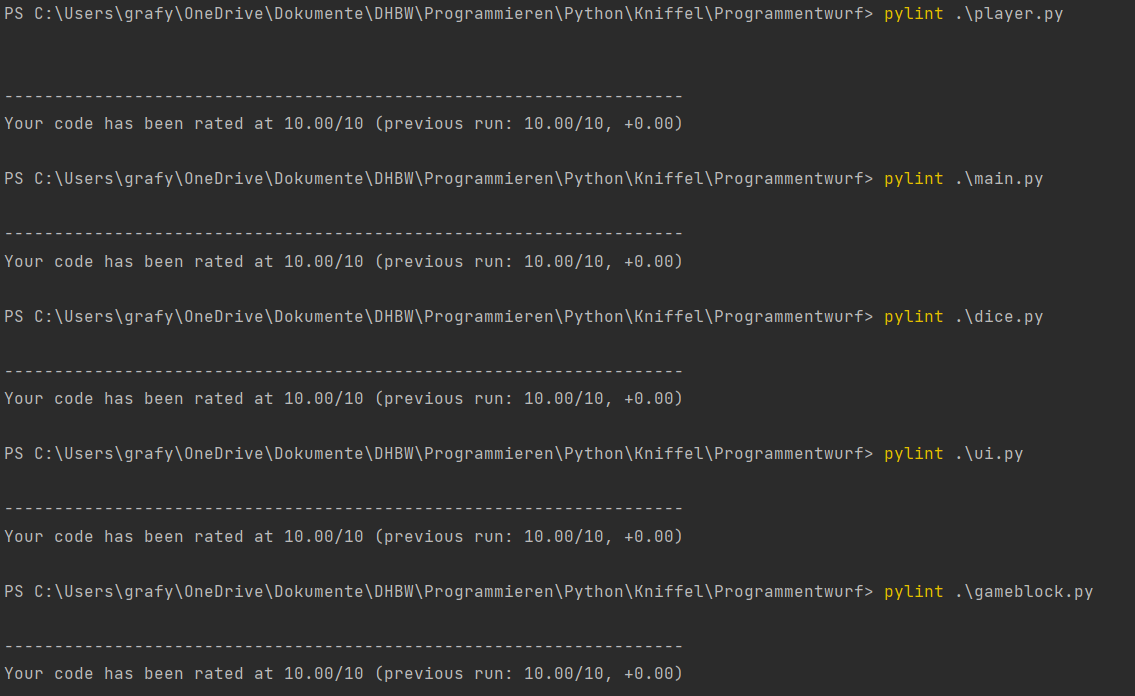
**Bewertung der Coverage und Sinnvollheit der Tests:**

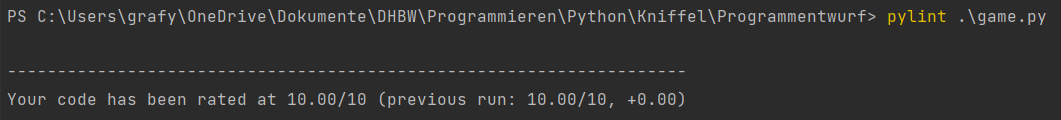
* Gewünscht ist eine Coverage von min 75% (pro Datei)
  + Wurde dies erreicht? Wenn nein, warum nicht?
    - Ja
    - Eine höhere Test Coverage ist im aktuellen Stadium nicht zuverlässig möglich, da gewisse Codeteile nur bei z.B. einem Kniffel durchlaufen werden
    - Des weiteren haben wir falsche eingaben nicht explizit automatisch abgetestet, da es hier zu recht komplexen test cases commen würde.
* Begründet warum ihr euch sicher seid alles mit Tests abgedeckt zu haben
  + In den wichtigsten Tests (testPVP/testPVE) wird das ganze Programm durchlaufen. Damit das Ergebnis den String „Der Gewinner ist“ enthält, müssen alle Funktionen des Programmes zumindest rudimentär funktionieren

**Bewertung der Fehlersicherheit:**

* Werden Fehleingaben korrekt abgefangen?
  + Nenne 2 Beispiele wo ein Fehler abgefangen wird
    - Wenn der Nutzer versucht die Zeile einzugeben, um eine große Straße einzutragen, obwohl er keine große Straße hat, so wird er gefragt, ob er 0 Punkte eintragen möchte, oder sich verwählt hat.
    - Immer wenn eine Zahl erwartet wird, werden Buchstaben als Eingaben nicht gewertet, sondern abgefangen

**Pylint Ausgabe:**





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

* Begründet warum euer Code gut lesbar ist
  + Aussagekräftige Variablen- und Methodennamen
  + Sinnvolle Methoden
  + Strukturierung durch Objektorientierung
    - Bsp. Alle Asugaben in Klasse ui

**Bewertungstabelle:**

* Ausgefüllte Bewertungstabelle
  + Siehe Anhang Bewertung.xsls