**Python - Programmentwurf**

**Team Mitglieder:**

|  |  |  |
| --- | --- | --- |
| **Name** | **e-Mail (DHBW)** |  |
| Thomas Gingele | thomas.gingele-it21@it.dhbw-ravensburg.de |  |
| Lukas Eisele | eisele.lukas-it21@it.dhbw-ravensburg.de |  |
| Jannik Künstler |  |  |
| Felix Schladt | schladt.felix-it21@it.dhbw-ravensburg.de |  |
|  |  |  |

**Verwendete Python Version: 3.9.7, 3.10.4**

**Verwendete Pylint Version: 2.13.7**

**Verwendete nachinstallierte Bibliotheken: keine**

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

* {"turns": 2, "player\_0": {"name": "Lukas", "dices": [1, 1, 2, 5, 1], "flags": [false, true, true, true, true, false, false, true, true, false, false, false, true], "scores": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "active": true}, "player\_1": {"name": "Thomas", "dices": [6, 2, 4, 6, 5], "flags": [false, false, true, true, true, true, false, false, true, false, true, false, true], "scores": [0, 0, 0, 6, 0, 10, 12, 28, 0, 0, 0, 25, 0, 40, 0, 13, 106], "active": false}}
* Die Datei wird im json-Format gespeichert. Jeder Spieler besteht aus einem Dictionary. In diesem Dictionary steht jeweils der Name, die geworfenen Augenzahlen der Würfel und der Spielstand.

Die Anzahl der verbleibenden Spielzüge wird in einem eigenen Eintrag namens „turns“ gespeichert.

**Beschreibung des Benutzerinterfaces:**

* Wie sieht die Ein- und Ausgabe aus?

Das Spiel wird an die Terminalgröße angepasst und entsprechend ausgegeben.

Man kann sehen:

* Die Würfel
* Ein Scoreboard
* Ein kleines „how-to-play“
* Die Anzahl der verbleibenden Spielzüge

Ein Screenshot ist zu finden in:

./doc/working\_game\_screenshot.png

Für die Auswahl der Kategorie für die Punktzahl wird eine List ausgegeben

Und mit „input()“ die Auswahl eingelesen.

Möchte man das Spiel frühzeitig mit „q“ verlassen, öffnet sich ein Menu,

welches nach Bestätigung fragt und dem Spieler mitteilt, wo sein Spielstand gespeichert ist.

**Beschreibung der Architektur:**

* Grobe Architektur beschreiben (Textuell, oder Diagramm)

→ ./doc/depency\_diagramm.png

→ ./doc/structure\_diagramm.png

**Dokumentation von einem kompletten Spielablauf:**

* Kopie der kompletten Ausgabe auf der Konsole. Evtl. eigene Datei

→ ./doc/played\_game\_record.log

* + Datei ist mittels tee erstellt → kopie von stdout
  + Ausgabe ohne color codes auf stdout:
    - cat ./doc/played\_game\_record.log > /dev/tty

Da die Ausgabe des Spiels von der Größe des Terminals

Abhängig ist, dann es passieren, dass der Inhalt der Datei schwer ablesbar ist.

**Log von den Tests:**

* Alle Tests über Konsole ausführen und Ausgabe hier einfügen (oder extra Datei)

→ ./doc/unittest.log

**Bewertung der Testergebnisse:**

* Grund für fehlschlagende Tests

Keine fehlschlagenden Tests

**Code-Coverage Ausgabe:**

* Tests mit Coverage ausführen und Ausgabe hier einfügen

→ ./doc/coverage.log

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

* Gewünscht ist eine Coverage von min 75% (pro Datei)
  + Wurde dies erreicht? Wenn nein, warum nicht?
    - Ja wurde es
* Begründet warum ihr euch sicher seid alles mit Tests abgedeckt zu haben

Die verbleibenden Freistellen in ./src/terminal.py sind aufgrund systemabhängiger

Funktionen nicht 100% testbar.

Die Funktionalität wurde hier mit User-Tests auf Windows ergänzt und sichergestellt.

Für ./src/game\_engine.py:

* Z. 79: Durch den Konstruktoraufruf in der Methode selbst ist es nicht möglich,

display\_height und display\_width entsprechend zu patchen.

Allerdings ist ein einziges print statement nicht fehleranfällig

und kann deshalb übersprungen werden.

* Z. 189: Die Funktion „get\_options()“ wurde gepatch um nicht zurückzugeben.

Dadurch wird die if-Abfrage nicht ausgeführt.

Ein Test der Funktion „select\_rule()“ findet separat statt.

* Z. 227-228: Würde „input()“ gepatch, um einen Wert zurückzugeben, damit dieses if-Statement aufgerufen wird, würde der unittest in einer Endlosschleife

stecken bleiben.

* Z. 348-354: Ein Unittets würde hier zu einer Endlosschleife führen.

Da das Spiel ohne diese Funktion allerdings gar nicht erst startet,

ist die Funktionalität durch User-Tests sichergestellt.

**Bewertung der Fehlersicherheit:**

* Werden Fehleingaben korrekt abgefangen?
  + Nenne 2 Beispiele wo ein Fehler abgefangen wird
    - ./src/game\_eninge.py:
      * Z. 68-71: Drückt der Spieler eine Taste, die nicht in Ascii codierbar ist,

wird der entprechende Fehler mit systemabhängigen Error-Klassen

abgefangen und ein entsprechender Ersatzwert zurückgegeben.

Z. 349-354: Ist das Terminal zu klein um das Spiel zu zeichnen, wird der

Fehler abgefangen und eine entsprechende Nachricht auf dem

Terminal ausgegeben.

**Pylint Ausgabe:**

* Pylint auf der Konsole Ausführen und Ergebnis einfügen

make pylint | tee doc/pylint.log

pylint ./src/

\*\*\*\*\*\*\*\*\*\*\*\*\* Module src.game\_engine

src/game\_engine.py:40:34: E1120: No value for argument 'self' in unbound method call (no-value-for-parameter)

src/game\_engine.py:62:40: E1120: No value for argument 'self' in unbound method call (no-value-for-parameter)

\*\*\*\*\*\*\*\*\*\*\*\*\* Module src.terminal

src/terminal.py:14:4: E0401: Unable to import 'msvcrt' (import-error)

src/terminal.py:24:0: R0903: Too few public methods (0/2) (too-few-public-methods)

------------------------------------------------------------------

Your code has been rated at 9.72/10 (previous run: 9.72/10, +0.00)

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

* Begründet noch vorhandene Pylint Warnungen
* E1120: Dieser Fehler tritt auf, da die Funktion, die über „self“ aufgerufen wird, eine statische Funktion ist (Funktion ist mit „@staticmethod“ gekennzeichnet).

Da sie sich allerdings in einer Klasse befindet (\_posix oder \_windows, ja nach System),

wird „self“ als Parameter übergeben.

Sollte der Parameter in den Methodenkopf übernommen werden, um diesen Fehler zu vermeiden, meldet pylint alternativ, dass der Parameter „self“ nicht verwendet wird.

* E0401: Da .src/terminal.py systemabhängige import-Statements benötigt,

können manche Module je nach System nicht importiert werden.

Die Imports sind durch entsprechende Abfragen gesichert.

Je nach System kann dieser Fehler in unterschiedlichen Zeilen auftreten.

* R0903: Die Klasse „Colors“ in .src/terminal.py dient dem einfachen Zugang zu

Ascii-Color-Codes. Sie benötigt deshalb keine Methoden.

* Begründet warum euer Code gut lesbar ist

Wir haben uns intern auf einige Regeln für guten Code geeinigt.

1. Guter Code ist prägnanter Code → Kurzer Code ist guter Code und erhöht die Lesbarkeit. Python gibt dem Programmierer dank inline if-Abfragen und List-Comprehensions die Möglichkeit, möglichst gut lesbaren Code zu schreiben.

2. Python ist nicht Java → Objektorientierung ist gut und schön, aber nicht immer sinnvoll.   
 Deshalb haben wir absichtlich auf Vererbung und unnötigen Boilerplate-Code verzichtet.

1. Segmentierung → Wir haben versucht unseren Code in möglichst logische

Unterteilungen zu splitten.

1. Consistency → Wir haben versucht, unsere libraries möglichst konsistent zu halten, um eine

einfache Benutzung durch alle Teammitglieder zu ermöglichen.

**Bewertungstabelle:**

* Ausgefüllte Bewertungstabelle