**Python - Programmentwurf**

**Team Mitglieder:**

|  |  |
| --- | --- |
| **Name** | **e-Mail (DHBW)** |
| Tobias Traiser | traiser.tobias-it20@it.dhbw-ravensburg.de |
| Emanuel Avadani | avadani.emanue-it20@it.dhbw-ravensburg.de |
| Lukas Zwaller | zwaller.lukas-it20@it.dhbw-ravensburg.de |
| Pascal Wildermuth | wildermuth.pas-it20@it.dhbw-ravensburg.de |

**Verwendete Python Version:** 3.9.1 & 3.9.2

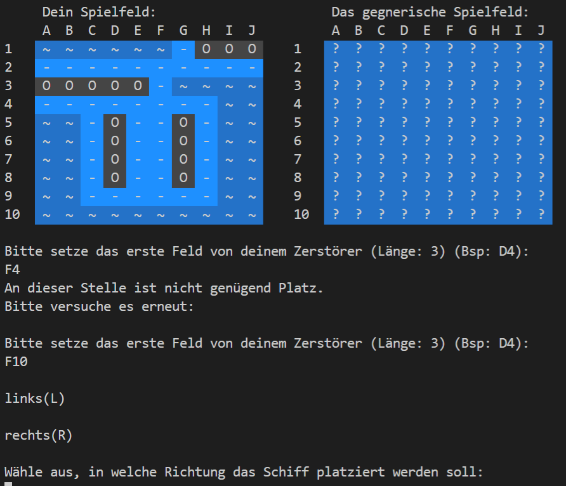
**Verwendete Pylint Version:** 2.6.0

**Verwendete nachinstallierte Bibliotheken: -**

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

* Wir haben JSON zur Spielstandspeicherung verwendet, da sich dieses Format sehr einfach bearbeiten, speichern und auslesen lässt.

Gespeichert werden die Namen der Spieler, deren Zustand der Spielfelder und die aktuelle Rundenzahl, aus der sich berechnen lässt, welcher Spieler an der Reihe ist.

**Beschreibung des Benutzerinterfaces:**

* Die Ein- und Ausgabe erfolgt über das Terminal. Jeder Spieler erhält für seinen Zug sein eigenes Spielfeld und eine verdeckte Ansicht des Gegnerfeldes, auf dem alle Schüsse verzeichnet werden. Die Spielfelder werden für eine bessere Lesbarkeit eingefärbt.
* Das Spiel beginnt mit einem Startbildschirm, auf dem die Auswahl zwischen Spielerspiel, Botspiel und Spielstand laden getroffen wird.
* Danach setzten alle Spieler ihre Schiffe über die Eingabe eines Startfeldes und der platzierten Richtung.
* Sind die Schiffe gesetzt, so wird abwechselnd auf ein Gegnerfeld geschossen. Bei einem Treffer darf erneut geschossen werden. Die Treffer werden farbig im Spielfeld vermerkt.

**Beschreibung der Architektur:**

Das Programm lässt sich gut in fünf verschiedene Abschnitte unterteilen:

* **Schnittstelle Mensch/Computer**. Die Formatierung des Terminals, die farbliche Darstellung der Felder und die Inputs des Users werden von den Funktionen spielfeld\_erstellen, cls, startbildschirm und farbiges\_spielfeld gehandelt.
* **Schiffe setzen**. Das Platzieren erfolgt über einen Algorithmus, der anhand eines eingegebenen Startfeldes auf alle vier möglichen Setzrichtungen auf dem Spielfeld überprüft und dem Nutzer alle funktionierenden Möglichkeiten aufzeigt. Dieser entscheidet sich für eine und lässt das Schiff vollständig generieren. Um ein Schiff werden Sperrfelder reserviert, die verhindern, dass sich zwei Schiffe berühren. Die Funktionen heißen schiffe\_setzen, finde\_zahl und setz\_richtung\_generieren.
* **Spielfeld beschießen**. Nachdem beide Spielfelder vollständig sind, wählen die Spieler abwechselnd Felder, die mit dem Gegnerfeld auf ein Schiff geprüft werden. Die Spielerreihenfolge wird über die Rundenzahl berechnet. Wird ein Treffer erkannt, so darf der Spieler erneut schießen. Ebenso unterscheidet die Funktion per Name zwischen einem Mensch und dem Bot und erwartet somit andere Kommunikation. Nach jedem Treffer überprüft die Funktion gewinner\_suchen auf einen Sieg und die Funktion schiff\_versenkt, ob das getroffene Schiff vollständig versenkt wurde. Schiff\_abschießen ruft sich selbst auf und beendet erst, wenn ein Gewinner erkannt wurde.
* **Spielstand speichern/laden**. Ein Spielstand wird automatisch nach jeder Runde gespeichert, sobald beide Spielfelder vollständig besetzt wurden. Das Speichern erfolgt über eine JSON Datei, in die alle Spielfelder, die Spielernamen sowie die aktuelle Runde geschrieben werden. Somit kann das Spiel jederzeit verlustfrei geschlossen werden. Beim Spielstart überprüft die Funktion spielstand\_auslesen, ob dem Spieler die Option des „Spiel ladens“ ermöglicht werden kann.
* **Computergegner**. Die Klasse ComputerGegner generiert das Spielfeld über die zufällige Auswahl der Startposition und Richtung. Falls die Position nicht möglich ist, wird ein neuer Wert generiert, bis alle Schiffe gesetzt sind. Beschossen werden das Spielfeld ebenfalls zufällig. Wird allerdings ein Schiff getroffen, so zielt der Bot beim nächsten Schuss auf ein benachbartes Feld.

**Beschreibung des Computergegners (4er Gruppen):**

Der Computergegner wählt mit der Funktion random.choice aus einer Liste ein zufälliges Feld.

* Beim Schiffe setzen wird überprüft, ob dieses Zufallsfeld noch frei ist und wenn ja, in welche Richtungen das Schiff platziert werden kann. Aus dieser Auswahl wählt der Computer wieder zufällig. Konnte das Feld nicht besetzt werden, so wählt er zufällig ein neues Feld. Dieser Vorgang wird wiederholt, bis alle Schiffe gesetzt wurden.
* Beim Angreifen wird auf ein Zufallsfeld geschossen. Der Computer erhält den Hinweis, auf welches Feld er beim letzten Mal geschossen hat und ob dieses Feld ein Treffer war. Ist das der Fall, so schießt er zufällig auf eines der vier benachbarten Felder.

**Dokumentation von einem kompletten Spielablauf:**

* Kompletter Spielverlauf einer Runde Spieler/Bot in der Datei „Spielverlauf.txt“. Leider ohne farbiges Spielfeld.

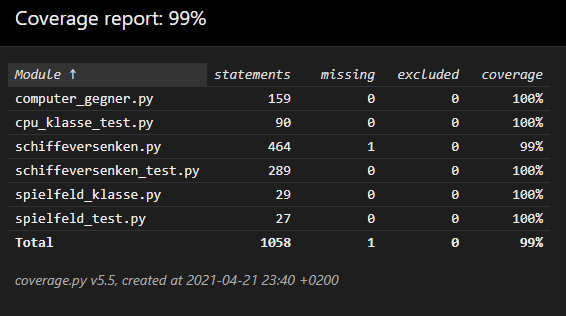
**Log von den Tests:**

* Komplette Logdatei finden Sie als „TestLog.txt“.

**Bewertung der Testergebnisse:**

* Unsere Tests besitzen eine Abdeckung von 99% und laufen fehlerfrei. Grenzfälle werden gezielt abgefragt

**Code-Coverage Ausgabe:**



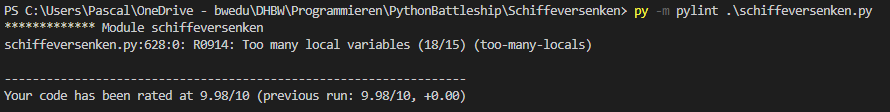
**Bewertung der Coverage und Sinnvollheit der Tests:**

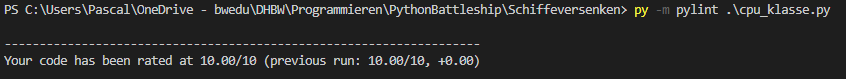
* Gewünscht ist eine Coverage von min 75%
  + Erreicht wurden 99%
* Wir haben alle Fälle getestet und einen kompletten Spielablauf inkludiert.
* Zu jeder Funktion wurden sich entsprechende Fehlerfälle überlegt und gezielt abgefragt.

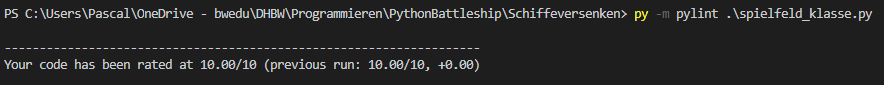
**Bewertung der Fehlersicherheit:**

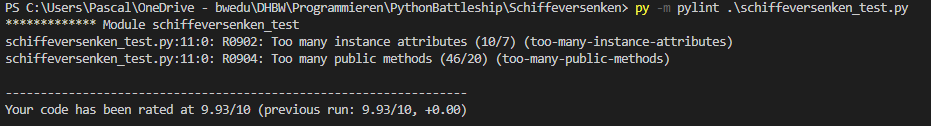
* Auf die Fehlersicherheit haben wir großen Wert gelegt. Jede Nutzerinteraktion wird auf ihre Richtigkeit geprüft und bei Fehleingabe wiederholt.
  + Jede Eingabe der Feldauswahl, die dem Format (D4) abweicht, wird abgefangen und wiederholt.
  + Da das Spielfeld in einer 2D-Liste gespeichert wird, müssen viele IndexErrors beim Überschreiten der Spielfeldgröße abgefangen werden. In den meisten Fällen wird schon im Voraus das Auftreten eines IndexError verhindert.

**Pylint Ausgabe:**









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

* **Pylint Warnung:**

*„schiffeversenken.py:587:0: R0914: Too many local variables (18/15) (too-many-locals)“*

Diese Warnung lässt sich ohne Überarbeitung der kompletten Funktionalität nicht beheben, da viele der Variablen als Zwischenspeicher dienen oder per bool-Wert einen Zustand abfragen. Eventuell ist die Funktionalität für eine Funktion zu umfangreich, allerdings ist „schiff\_abschießen“ das Bindeglied und der Mittelpunkt unseres Codes.

*„schiffeversenken\_test.py:11:0: R0904: Too many public methods (46/20) (too-many-public-methods)“*

Diese Warnung ist notwendig, da wir kompakt die Hauptdatei mit einer Testdatei abdecken wollten. Somit wurden viele Methoden intern wiederverwendet.

*“schiffeversenken\_test.py:11:0: R0902: Too many instance attributes (10/7) (too-many-instance-attributes)”*

Diese Warnung muss erhalten bleiben, da wir im setup eine Mindestzahl an self Aufrufen benötigen.

* **Lesbarkeit:**

Unsere Funktionen sind nach Ablauf und Themen sortiert, um umständliches Scrollen zu vermeiden. Die Variablennamen sind sprechend und auf Deutsch formuliert. Alle Docstrings sind gewissenhaft ausgefüllt. Zwei Klassen erleichtern die Übersichtlichkeit der Funktionen.

**Bewertungstabelle:**

