**Python - Programmentwurf**

**Team ID:**

**Team Mitglieder (Name + e-mail): Frank Rübenkönig (ruebenkoenig.f-it19@dhbw-ravensburg.de), Sarah Stebich (stebich.sarah-it19@it.dhbw-ravensburg.de), Tobias Spohn (spohn.tobias-it19@it.dhbw-ravensburg.de), Florian Christoph Veit (veit.florian\_c-it19@it.dhbw-ravensburg.de)**

**Git verwendet? (fließt nicht in die Note ein): Ja**

**Buildserver verwendet? (fließt nicht in die Note ein): Ja**

**Verwendete Python Version: 3.8.2**

**Verwendete Pylint Version: 2.4.4**

**Beschreibung des Dateiformats:**

* Wie ist die Datei aufgebaut?

Es wird als Binärdatei gespeichert. Das Modul pickle wird zum Laden und Speichern genutzt. Damit konnten wir eine Einfache und schnelle Umsetzung realisieren. Die Datei ist damit nicht Verschlüsselt. Das hat den Vor- und Nachteil, dass jeder diese Lesen, Schreiben und Nutzen kann.

Die Log Datei wird in UTF-8 codiert und im txt-Format gespeichert damit Sie mit jedem Editor geöffnet werden kann sowie unsere Sonderzeichen geloggt werden können.

**Beschreibung des Benutzerinterfaces:**

* Wie sieht die Ein- und Ausgabe aus?

Es wird immer ein Header und Spielfeld ausgegeben. Die Eingabe erfolgt immer darunter.

Nach einer Eingabe wird die Anzeige im Terminal gelöscht und neu Ausgegeben.

Im Header stehen die Funktionen: Neues Siel, Speichern, Laden und Spiel Beenden.

Das Spielfeld ist von zwei horizontalen Linien umgeben. Es werden schwarze und weiße Quadrate genutzt um das Spielfeld darzustellen (◻ /◼). Die Bauern sind in den richtigen Farben die richtigen Zeichen (♙/♟). Dieses Zeichenkonstrukt wird von Koordinaten (A…H/1…8) umgeben.

Beim wählen von Figuren werden mögliche Züge und die gewählte Figur in einer anderen Farbe dargestellt.

**Beschreibung der Architektur:**

* Änderungen an der Vorlage

Ordnerstruktur geändert. Diese ist nun eine Realisierung von Modulen.

* Grobe Architektur

Diese wurde in der Architektur.graphml geplant, sowie in den einzelnen Modulen eine graphml Datei liegt.

Es wurde in vier Module unterteilt: User Interface, Computer Gegner, Logik (Spielfeld und Figuren) und Dateiverwaltung.

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

* Welcher Ansatz wird verfolgt

***Sorry weiß gerade nichts dazu!***

**Log von einem Spiel:**

* Ein Spieldurchlauf dokumentieren

Dateiname:

**Log von den Tests:**

Test wird in dem Ordner in der die main.py liegt aufgerufen und Ausgeführt, damit werden alle Tests vom Projekt ausgeführt:

**Bewertung der Testergebnisse:**

* Grund für fehlschlagende Tests

Kein Test ist fehlgeschlagen

**Code-Coverage Ausgabe:**

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

* Gewünscht ist eine Coverage von min 75%
  + Wurde dies erreicht? Wenn nein, warum nicht?

Es wurde darauf geachtet, dass bei den Tests jeder mögliche Aufruf getestet wurde. Falsche Übergabe Parameter wurden zum großen Teil durch asserts abgefangen und werden in den Tests nicht mit falschen Parametern aufgerufen.  
Die fehlende Abdeckungen sind try/excepts für die Imports. Sollte dies getestet werden, wird das Programm beendet.  
Es kann sein, dass durch mkdir und removedir falls der Ordner bereits angelegt ist oder gefüllt ist diese Befehle nicht getestet werden.

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

Durch asserts werden die Übergabe Parameter schon ohne Tests überprüft. Durch ein hohen Coverage sowie, dass der Code von Mehreren bearbeitet und besprochen wurde haben wir nun eine hohe Sicherheit, dass wir eine große Abdeckung haben.

* Fehlerfälle in den Tests überprüft?

Durch nutzten von Error-Codes werden die Funktionen falsch aufgerufen und dieser Error-Code überprüft/erwartet. Durch asserts werden die Übergabe Parameter direkt im Code überprüft und werden nicht mehr in den Tests abgeprüft.

**Bewertung der Fehlersicherheit:**

* Werden Fehleingaben korrekt abgefangen?

Es wurde mehrfach durchgespielt und diverse falsche Dinge eingegeben.

Abfangen von falsch eingaben wird in game.py und main.py realisiert (da nur diese Inputs haben).

* + Nenne 2 Beispiele wo ein Fehler abgefangen wird

main.py Zeilen 155 bis 161

* + - decision = int(input())

while consts.FOREVER:

if \_\_decision == 1:

return consts.GAME\_MODE["RESET"]

elif \_\_decision == 2:

return consts.GAME\_MODE["NEW\_GAME"]

else:

print("Falsche eingabe")

main.py Zeilen 183 bis 185

* + - decision = ""

while decision not in [consts.GAME\_MODE["SAVE"], consts.GAME\_MODE["SAVE\_NEW"], consts.GAME\_MODE["QUIT"]]:

decision = str.upper(input("\t\t\t\t\t"))

**Pylint Ausgabe:**

Keine Mängel

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

* Begründet noch vorhandene Pylint Warnungen
* Begründet warum euer Code gut lesbar ist
  + Durch nutzen von Konstanten und eindeutige Namen sowie den Aufbau von Modulen. README-dateien sind in jedem Modul für eine einfache kurze Erklärung.

**Bewertungstabelle:**

* Ausgefüllte Bewertungstabelle