Python - Programmentwurf

Allgemeines

Team ID	1
Team Mitglieder	**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 Note ein)*	✓
Buildserver verwendet *(fließt nicht in Note ein)*	✓
Python Version	3.8.2
Pylint Version	2.4.4

Dateiformat

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.

Benutzerinterface

Zu Beginn des Spiels wird ein Menü (Abbildung 1) ausgegeben. Es wird zur Eingabe aufgefordert, um ein neues Spiel zu erstellen, ein altes zu Laden oder zu Beenden.

Wird ein Spiel erstellt, wird zur Eingabe von Computer Gegner, Spielname und Spieler Namen aufgefordert.

Wenn das geladene oder erstellte Spiel läuft wird ein Header, Spielfeld und Footer ausgegeben. Die Eingabe erfolgt immer darunter.

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

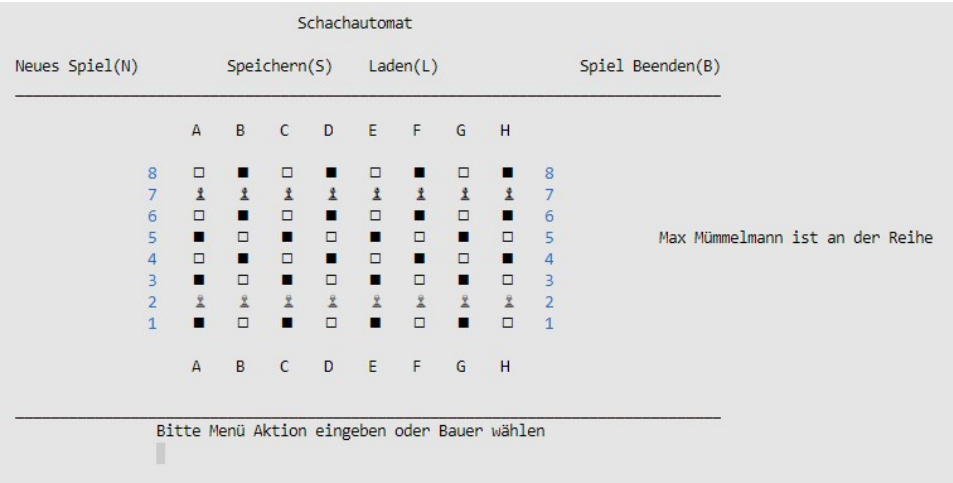

Abbildung 1: Spielfeld

Im Header stehen die Funktionen „Neues Siel“, „Speichern“, „Laden“ und „Spiel Beenden“ (vgl. Abbildung 1).

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 und in den richtigen Zeichen (♙ / ♟). Dieses Zeichenkonstrukt wird von Koordinaten (A…H/1…8) umgeben. Rechts befindet sich eine Infoausgabe, welche den aktiven Spieler anzeigt. Unter der Infoausgabe befindet sich die Fehlerausgabe.

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

Im Footer wird nur eine Abgrenzung angezeigt.

Architektur

Änderungen an der Vorlage

Ordnerstruktur geändert. Diese ist nun eine Realisierung von Modulen. Keine Tests und Pawn übernommen.

Grobe Architektur

Diese wurde in *Modul_Plan.graphml* (Software: yEd) geplant. Des Weiteren stehen die groben Infos in den README-Dateien zur Verfügung.

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

Computergegner

Durch Zufall wird eine der Spielfiguren des Computergegners ausgewählt.

Zunächst wird geprüft, ob geschlagen werden kann. Dabei wird zuerst links geprüft, ist dies nicht möglich, wird die rechte Seite geprüft. Kann weder rechts noch links geschlagen werden, wird überprüft, ob die Figur nach vorn bewegt werden kann. Ist dies auch nicht möglich wird wieder zufällig eine neue Spielfigur ausgewählt. Dieser Prozess wird solang wiederholt, bis ein Spielzug durchgeführt werden kann. Die aufzurufende Methode gibt schlussendlich die alte und neue Position des Bauern als String-Liste zurück.

Ausgabelogs

Log von einem Spiel

Ein Spieldurchlauf wird in der Datei *First_Game_log.txt* dokumentiert

Log von den Tests

Test wird im Ordner der main.py aufgerufen und ausgeführt. Damit werden alle Tests vom Projekt ausgeführt. Die Datei mit der Terminalausgabe heißt *Log_UnitTest_Terminal_Output.txt*.

Bewertung der Testergebnisse

Keiner der Tests ist fehlgeschlagen.

Code-Coverage

Code-Coverage Ausgabe

Name	Stmts	Miss	Cover	Missing
chess_logik__init__.py	0	0	100%	
chess_logik\consts.py	11	0	100%	
chess_logik\field.py	84	3	96%	8-10
chess_logik\figure.py	26	3	88%	7-9
chess_logik\pawn.py	66	3	95%	8-10
chess_logik\position.py	21	3	86%	6-8
chess_logik\test_field.py	192	3	98%	9-11
chess_logik\test_figure.py	25	3	88%	8-10
chess_logik\test_pawn.py	60	3	95%	8-10
chess_logik\test_position.py	32	3	91%	7-9
chess_storage__init__.py	0	0	100%	
chess_storage\chess_storage.py	72	3	96%	9-11
chess_storage\consts.py	4	0	100%	
chess_storage\test_chessstorage.py	134	5	96%	10-12, 156, 167
computer_gegner__init__.py	0	0	100%	
computer_gegner\opponent_move.py	81	3	96%	8-10
computer_gegner\test_opponent.py	150	3	98%	11-13
consts.py	6	0	100%	
game.py	203	3	99%	10-12
main.py	169	7	96%	12-14, 260-262, 266
test_game.py	308	6	98%	18-20, 418-420
test_main.py	151	6	96%	20-22, 289-291
TOTAL	1795	60	97%	

Genauer Infos können in der HTML nachgelesen werden.

Bewertung der Coverage und Sinnhaftigkeit der Tests

Gewünscht ist eine Coverage von mindestens 75%

Es wurde darauf geachtet, dass bei den Tests jeder mögliche Aufruf getestet wurde. Falsche Übergabeparameter wurden zum großen Teil durch asserts abgefangen und werden in den Tests nicht mit falschen Parametern aufgerufen.
Die fehlende Abdeckungen sind try-except-Blöcke für die Imports. Würde dies getestet werden, wird das Programm beendet.
Es besteht die Möglichkeit, dass durch bereits existierende oder gefüllte Ordner die Funktionen zum Erstellen und füllen nicht getestet werden.
Es kann nicht auf beide Betriebssysteme gleichzeitig getestet werden (main.py [264…269]).
Der main Aufruf wird von den Tests nicht ausgeführt, da die Funktionen direkt aufgerufen werden.

Darum ist alles mit den Tests abgedeckt

Durch asserts werden die Übergabeparameter schon ohne Tests überprüft. Durch ein hohen Coverage, sowie dass der Code von mehreren Entwicklern bearbeitet und besprochen wurde, wird eine hohe Sicherheit gewährleistet. Dadurch, dass die *test_main.py* und *test_game.py* nochmals Funktionen der einzelnen Module aufrufen, werden diese teilweise getestet.

Fehlerfälle in Tests überprüfen

Durch nutzten von Error-Codes werden die Funktionen falsch aufgerufen und dieser Error-Code überprüft/erwartet. Durch asserts werden die Übergabeparameter 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, sowie in den Tests überprüft.

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

Beispiele für abgefangene Fehler

main.py Zeilen 162 bis 169

```
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 193 bis 195

```
	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 Pylint Warnungen.

Code-Qualität und Lesbarkeit

Die Lesbarkeit und Qualität werden durch nutzen von Konstanten und eindeutige Namen, sowie den Aufbau von Modulen gewährleistet. In jedem Modul wurden README-Dateien für eine einfache kurze Erklärung eingepflegt, des Weiteren wird eine graphische Darstellung der Architektur verwendet.

Bewertungstabelle

Kategorie	max. Punkte	Einschätzung Studenten	Erreichte Punkte
Erfolgreiche Tests	20	20	
Testabdeckung (Coverage + sinnvolle Tests)	20	20	
Spielbarkeit (ausprobieren von Hand)	10	10	
Fehlersicherheit (Falscheingaben abfangen)	10	10	
Pylint (10 – Warnungen)	10	10	
Lesbarkeit (Analyse von Hand)	20	18	
Dokumentation	10	10	
Summe	100	98	0