Projektarbeit Python 2025 Dokumentation Verlassen Raumstation

- **Spiel:** Verlassene Raumstation
- **Autor:** Yannik Angeli
- **Abgabe:** 07.04.2025
- **Sprache:** Python 3.13.1
- **System:** Ubuntu 20.04 LTS, macOS 15.2
- **Dozent:** Herr Wiedemann

---

1. Spielbeschreibung

Bei Verlassene Raumstation handelt es sich um ein Konsolenspiel (Man spielt es im Terminal spielen). Es verfolgt ein ähnliches Prinzip wie das klassische „Minesweeper“. Zu Beginn wird dem Spieler ein verdecktes 5 mal 5 Spielfeld angezeigt. Der Spieler kann nun Koordinaten im Terminal eingeben um ein Feld aufzudecken bzw. zu „scannen“. Deckt der Spieler eine der zufällig verteilten Fallen auf ist das Spiel vorbei. Tut er das nicht, so wird das Spielfeld frei und es wird angezeigt wie viele Fallen an das Spielfeld grenzen (Benachbarte Falle). Nun kann der Spieler erneut Koordinaten angeben und ein weiteres Feld aufdecken. Sind alle Felder ohne Fallen aufgedeckt hat der Spieler gewonnen. Deckt er vorher eine Falle auf ist das Spiel vorbei und der Spieler verliert.

---

2. Projektstruktur

```
├── mypy.ini
├──.pylintrc
├── README.md
├── requirements.txt
├── doku/
│   └── documentation.pdf
├── source/
│   ├── __init__.py
│   ├── example.py
│   └── main.py
└── tests/
    ├── __init__.py
    ├── test_example.py
    └── test_main.py
```

Überverzeichnis

- mypy.ini: Beinhaltet strikte Regeln für die Typüberprüfung
- .pylintrc: Beinhaltet 4 Anpassungen für die statische Codeanalyse

- requirements.txt: Beihnaltet aktuelle Version von genutzten Biblotheken
- README.md: Leere Datei um Vorgaben zu erfüllen
- /doku: Enthält die Dokumentation
- /source: Enthält den Spielcode
- /tests: Enthält die Unittest Datei

Source Verzeichnis

- main.py: Enthält den Spielcode (Einstiegspunkt)

Tests Verzeichnis

- test_main.py: Tests für Spiellogik und Hauptfunktionen
- init.py: Kennzeichnet den tests-Ordner als Python-Paket (Testdateien können importiert und vom Test-Runner (unittest, coverage) korrekt erkannt werden)

3. Funktionen des Spiels

3.1 `create_board(size: int, num_traps: int) -> list[list[int]]`

Erstellt ein Spielfeld der angegebenen Größe, wobei zufällig Fallen platziert werden. Jede Falle wird mit `-1` markiert, und für jedes benachbarte Feld wird die Anzahl benachbarter Fallen berechnet und angezeigt.

Parameter:

- `size`: Die Größe des Spielfelds (eine quadratische Matrix).
- `num_traps`: Die Anzahl der Fallen, die auf dem Spielfeld platziert werden.

Rückgabewert:

- Ein 2D-Array (Liste von Listen), das das Spielfeld mit den berechneten benachbarten Fallen darstellt.

3.2 `check_win(board: list[list[int]], revealed: list[list[bool]]) -> bool`

Überprüft, ob das Spiel gewonnen wurde, d.h., ob alle sicheren Felder aufgedeckt sind, ohne dass eine Falle aktiviert wurde.

Parameter:

- `board`: Das Spielfeld mit den Fallen.
- `revealed`: Ein 2D-Array, das den Aufdeckstatus jedes Feldes verfolgt (True für aufgedeckt, False für nicht aufgedeckt).

Rückgabewert:

- `True`, wenn alle sicheren Felder aufgedeckt sind, andernfalls `False`.

3.3 `count_remaining_traps(board: list[list[int]], revealed: list[list[bool]]) -> int`

Zählt die verbleibenden Fallen auf dem Spielfeld, die noch nicht aufgedeckt wurden. Diese Funktion hilft, den Spieler über die verbleibenden Fallen zu informieren.

Parameter:

- `board`: Das Spielfeld mit den Fallen.
- `revealed`: Ein 2D-Array, das den Aufdeckstatus jedes Feldes verfolgt.

Rückgabewert:

- Die Anzahl der nicht aufgedeckten Fallen.

3.4 `reveal_board(board: list[list[int]], revealed: list[list[bool]], x: int, y: int) -> None`

Deckt ein Feld auf, basierend auf den angegebenen Koordinaten. Wenn das Feld leer ist (d.h. keine benachbarten Fallen), werden auch die angrenzenden Felder rekursiv aufgedeckt.

Parameter:

- `board`: Das Spielfeld mit den Fallen.
- `revealed`: Ein 2D-Array, das den Aufdeckstatus jedes Feldes verfolgt.
- `x`, `y`: Die Koordinaten des Feldes, das aufgedeckt werden soll.

Rückgabewert:

- Keine Rückgabe (ändert den `revealed`-Status direkt).

3.5 `display_board(board: list[list[int]], revealed: list[list[bool]]) -> None`

Gibt das Spielfeld auf der Konsole aus, wobei aufgedeckte Felder die Anzahl der benachbarten Fallen anzeigen und nicht aufgedeckte Felder als - angezeigt werden. Zudem wird die Anzahl der verbleibenden Fallen angezeigt.

Parameter:

- `board`: Das Spielfeld mit den Fallen.
- `revealed`: Ein 2D-Array, das den Aufdeckstatus jedes Feldes verfolgt.

Rückgabewert:

- Keine Rückgabe (gibt das Spielfeld direkt auf der Konsole aus).

3.6 `run_game() -> None`

Startet das Spiel „Verlassene Raumstation“. Der Spieler wird aufgefordert, Koordinaten für Felder einzugeben, die er aufdecken möchte. Das Spiel endet, wenn eine Falle aufgedeckt wird oder alle sicheren Felder entdeckt sind.

Rückgabewert:

- Keine Rückgabe (steuert den gesamten Ablauf des Spiels).

4. Spielablauf

1. **Spielfeld erstellen:** Zu Beginn wird das Spielfeld mit zufälligen Fallen und den entsprechenden benachbarten Fallenwerten generiert.
2. **Spieleraktionen:** Der Spieler wird nacheinander aufgefordert, Felder auszuwählen, die er aufdecken möchte. Das Spielfeld zeigt den aktuellen Status jedes Feldes (aufgedeckt oder nicht) und die Anzahl der verbleibenden Fallen an.
3. **Win-/Lose-Bedingungen:** Das Spiel endet, wenn der Spieler eine Falle aufdeckt oder alle sicheren Felder aufgedeckt hat. Ein Gewinn tritt ein, wenn alle sicheren Felder entdeckt wurden, ohne dass eine Falle aktiviert wurde.

5. Nutzerinterface

Das Spiel läuft im Konsolenmodus und zeigt das Spielfeld nach jeder Eingabe des Spielers an. Der Spieler gibt x- und y-Koordinaten für das Feld ein, das er aufdecken möchte. Die Eingaben werden geprüft, und das Spielfeld wird entsprechend aktualisiert.

Pylint (statische Codeanalyse)

- Tool: **pylint**

Aufgerufen mit:

```
pylint main.py
```

Im /source Verzeichnis

Ergebnis:

```
--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

Selbes Ergebnis für test_main.py

Meldungen

MyPy (Typprüfung)

- Tool: `mypy`

Aufgerufen mit:

```
mypy main.py
```

Im /source Verzeichnis

Ergebnis:

```
Success: no issues found in 1 source file
```

Verwendete Konfiguration:

```
[mypy]
warn_return_any = True
warn_unused_configs = True
disallow_untyped_defs = True
disallow_untyped_calls = True
disallow_incomplete_defs = True
```

Selbes Ergebnis für test_main.py

Unitest + Coverage (dynamische Tests)

- Tool: `unitest` + `coverage`

Tests ausgeführt mit:

```
coverage run -m unittest discover
coverage report -m
```

Ergebnis:

```
...
....Feld (0,0) aufgedeckt, Wert: 0
--> Versuche Nachbarfeld (1,0) aufzudecken
Feld (1,0) aufgedeckt, Wert: 0
--> Versuche Nachbarfeld (2,0) aufzudecken
Feld (2,0) aufgedeckt, Wert: 0
--> Versuche Nachbarfeld (2,1) aufzudecken
Feld (2,1) aufgedeckt, Wert: 0
--> Versuche Nachbarfeld (2,2) aufzudecken
Feld (2,2) aufgedeckt, Wert: 0
....
----------------------------------------------------------------------
Ran 8 tests in 0.003s

OK
Name                    Stmts   Miss  Cover   Missing
-------------------------------------------------------
source/__init__.py          0      0   100%
source/main.py             77     12    84%   64, 70, 111-112, 115-116, 119-120, 124-126, 128, 137
tests/__init__.py           0      0   100%
tests/test_example.py       0      0   100%
tests/test_main.py         70      5    93%   90-91, 99-100, 105
-------------------------------------------------------
TOTAL                     147     17    88%
```

6. Verbesserungsvorschläge

- **Multiplayer-Modus:** Es könnte ein alternativer Spielmodus hinzugefügt werden, in dem zwei Spieler abwechselnd Felder aufdecken.
- **Grafische Oberfläche:** Eine grafische Benutzeroberfläche (GUI) könnte entwickelt werden, um das Spiel visuell ansprechender zu gestalten.