Inhaltsverzeichnis

1. Einleitung

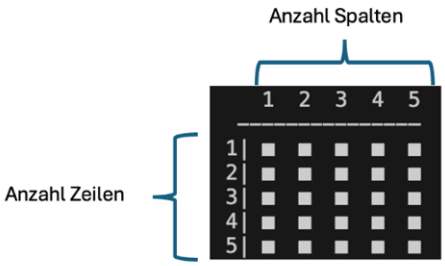
In dieser Dokumentation wird der grundlegende Aufbau des Spiels Verlassene Raumstation sowie die Architektur detailliert anhand von Diagrammen erklärt. Es wird außerdem die Ordnerstruktur sowie die Ergebnisse von dazu passenden Unittests dokumentiert. Um das Spiel zu starten, muss man sich im Hauptordner des Spiels PYTHON_ABGABE befinden und in der Konsole folgenden Befehl eingeben: PYTHONPATH=$(pwd) python src/main.py

2. Grundlegender Aufbau des Spiels

2.1 Benutzeroberfläche

```
-----------------Welcome-----------------
Please enter the size of the game table: a
Please enter a valid number.
Please enter the size of the game table: 4
4 is out of the Possible Range (5-15)
Please enter the size of the game table: 5
```

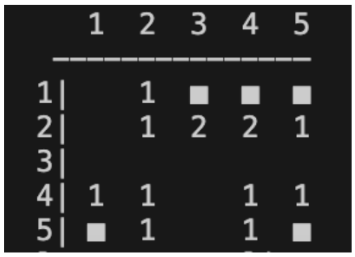
Als erstes soll man die Größe des Spielfelds eingeben, dabei wird die Eingabe auf ungültige Eingaben überprüft und der Benutzer passend über eine falsche Eingabe informiert. Eine falsche Eingabe ist dann der Fall, wenn der Benutzer keine Zahl eingibt, oder keine Zahl zwischen 5 und 15.



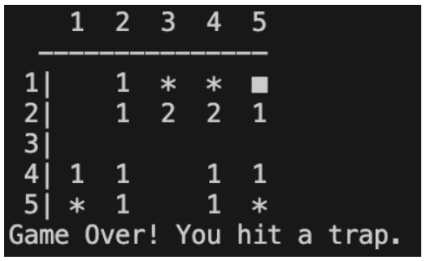
Hier wird das Spielfeld ausgegeben. Die nicht aufgedeckten Felder sind hierbei ein ■ und über den Feldern wird die Zeilen-/Spaltennummer angezeigt, sodass der Benutzer leichter ein Feld auswählen kann.

```
Please enter line of the row: a
Please enter a valid number.
Please enter line of the row: 0
0 is out of the Possible Range (1-5)
Please enter line of the row: 1
Please enter line of the column: 1
```

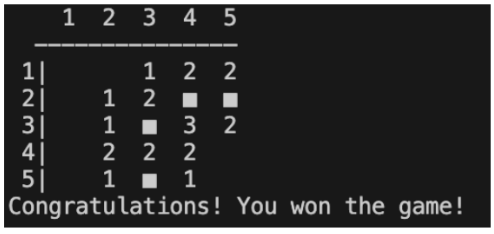
Hier soll der Benutzer die Zeilen-/Spaltennummer eingeben, von dem Feld, welches er auswählen möchte. Es wird außerdem wieder auf falsche Eingaben hingewiesen.



Hier wird das aktualisierte Spielfeld angezeigt. Die Zahlen stehen für die Anzahl der Fallen, welche das Feld um sich herum hat. Dabei kommt dazu, dass es leere Felder gibt, da sie keine Fallen um sich herum haben und somit eigentlich eine 0 sind.



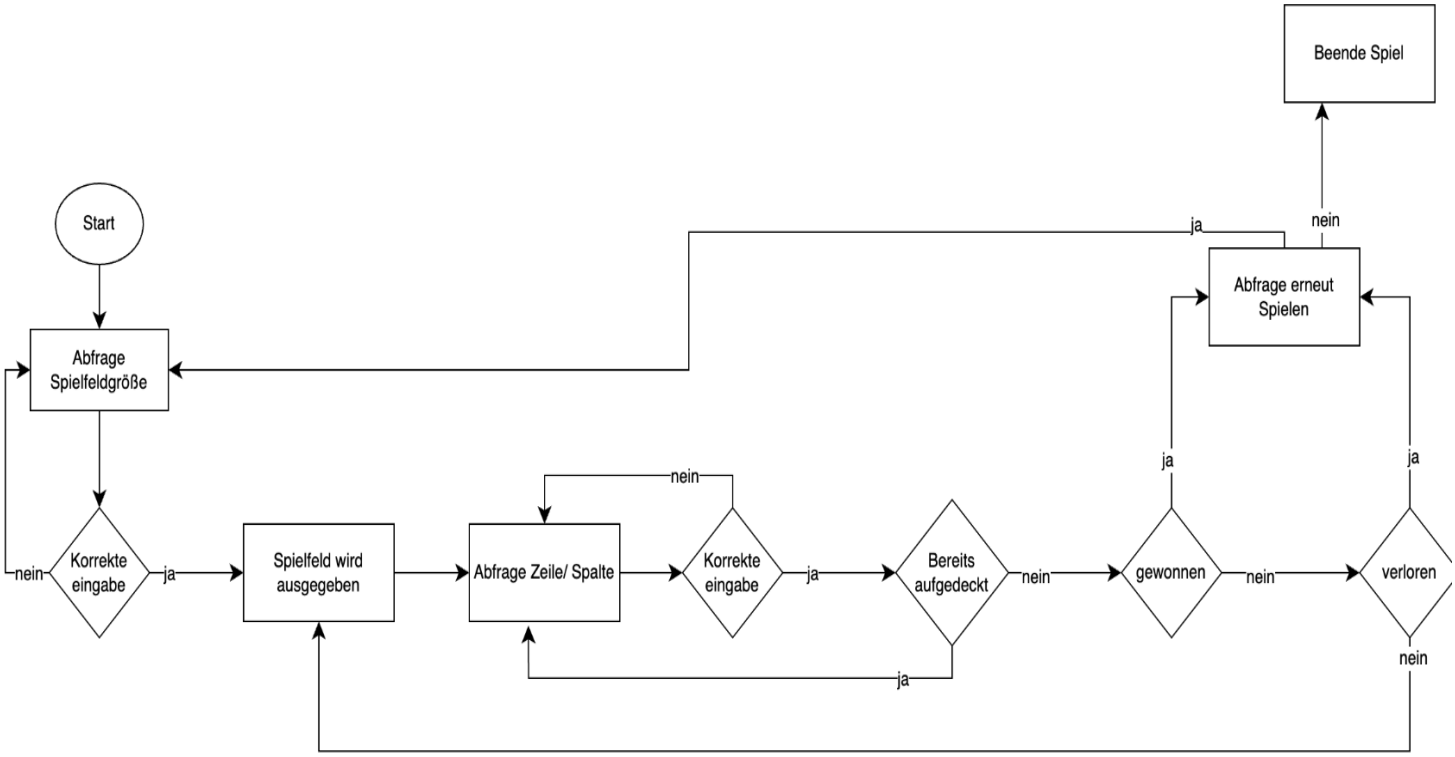
Hier wird das Spielfeld angezeigt, wenn man eine Fallen getroffen hat. Dabei werden alle übrigen Fallenfelder angezeigt und der Benutzer darüber informiert, dass er verloren hat.



Hier wird das Spielfeld angezeigt, wenn man alle freien Felder aufgedeckt hat und somit das Spield gewonnen hat.

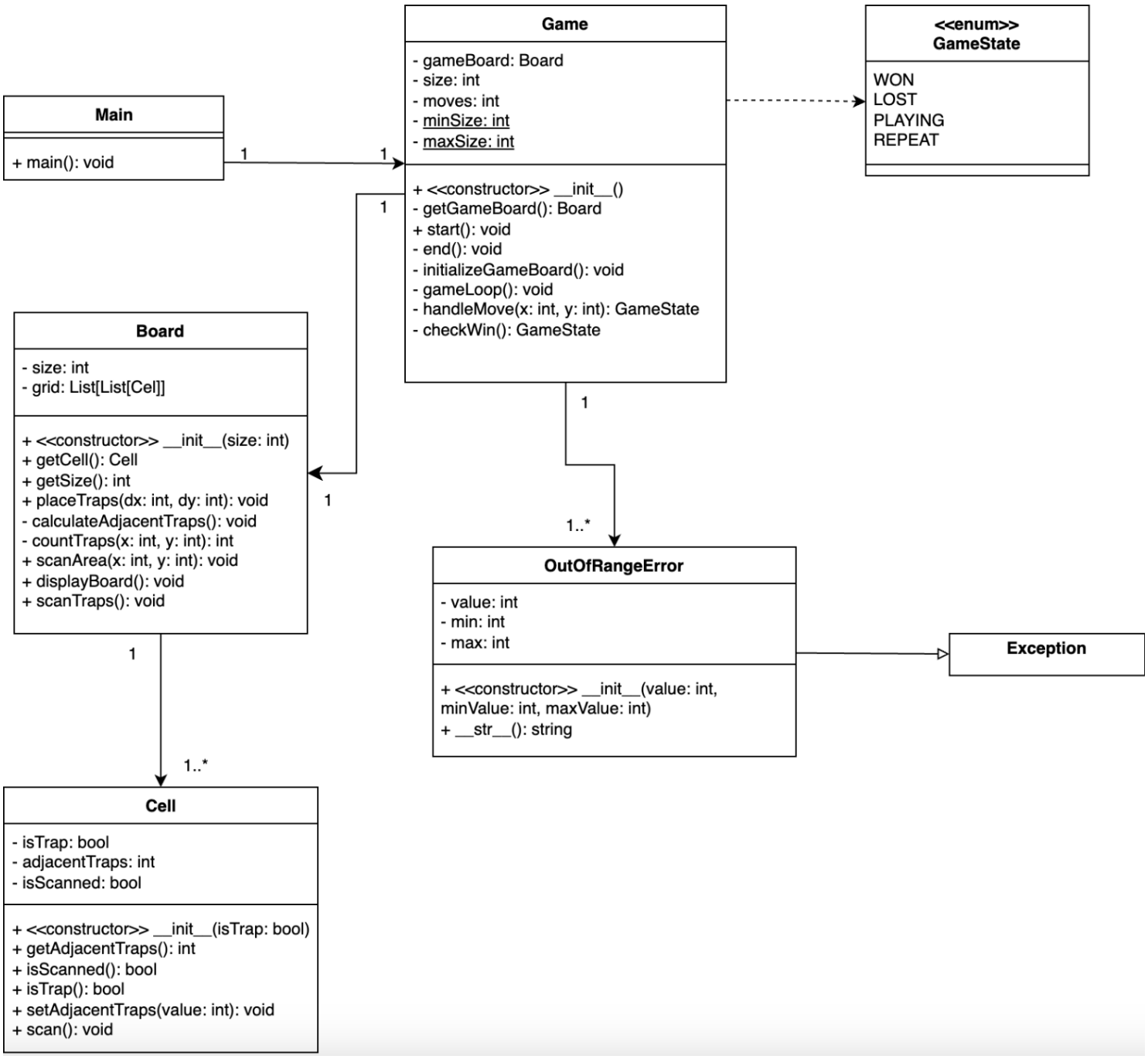
Hier wird der finale Schritt gezeigt, bei der gefragt wird ob der Benutzer erneut spielen möchte oder nicht. Fall er möchte und 'y' eingibt, geht das Spiel von vorne los. Falls er 'n' eingibt, wird das Spiel beendet.

2.2 Programmablauf



3. Architektur und Klassendesign

3.1 Klassendiagramm



3.2 Erklärung der Klassen

main.py

Hier ist der Einstiegspunkt für das Spiel. Es wird ein `Game`-Objekt erstellt und das Spiel gestartet.

Methodenbeschreibung

main() -> None

Beschreibung:
Startet das Spiel, indem ein `Game`-Objekt erstellt und dessen `start()`-Methode aufgerufen wird.

Effekt:

- Erstellt eine Instanz der `Game`-Klasse.
- Ruft die `start()`-Methode auf, um das Spiel zu starten.

Klasse `Game`

Die Klasse `Game` verwaltet die Spiellogik des Spiels. Sie steuert den Ablauf, verarbeitet Spielzüge und überwacht den Spielstatus.

Attribute

- `minSize` (*int*): Minimale Spielfeldgröße (5).
- `maxSize` (*int*): Maximale Spielfeldgröße (15).

Methodenbeschreibung

`__init__(self) -> None`

Beschreibung:
Initialisiert eine neue Instanz des Spiels.

Effekt:

- Setzt `minSize` und `maxSize` für die Spielfeldgröße.
- Initialisiert `_gameBoard` auf `None`.
- Setzt `_size` auf `0` und `_moves` auf `0`.

Attribute:

- `_gameBoard` (*Board* | *None*): Referenz auf das Spielfeld (Standard: `None`).
- `_size` (*int*): Größe des Spielfelds.
- `_moves` (*int*): Anzahl der bereits ausgeführten Züge.

`_getGameBoard(self) -> Board`

Beschreibung:
Gibt das aktuelle Spielfeld zurück.

Effekt:

- Falls `_gameBoard` nicht initialisiert ist, wird eine `AssertionError` ausgelöst.

Rückgabewert:

- `Board` – Gibt das Spielfeld-Objekt zurück.

---

start(self) -> None

Beschreibung:
Startet das Spiel und beginnt die Spielschleife.

Effekt:

- Zeigt eine Willkommensnachricht an.
- Initialisiert das Spielfeld.
- Ruft die Hauptspiel-Schleife (`_gameLoop`) auf.

_end(self) -> None

Beschreibung:
Beendet das aktuelle Spiel.

Effekt:

- Fragt den Spieler, ob er erneut spielen möchte.
- Startet ein neues Spiel oder beendet das Programm je nach Nutzereingabe.

_initializeGameBoard(self) -> None

Beschreibung:
Erstellt das Spielfeld basierend auf der Nutzereingabe.

Effekt:

- Fragt den Nutzer nach der gewünschten Spielfeldgröße.
- Überprüft die Eingabe auf Gültigkeit.
- Erstellt ein `Board`-Objekt mit der gewählten Größe.

_gameLoop(self) -> None

Beschreibung:
Führt die Hauptspiel-Schleife aus.

Effekt:

- Fragt den Nutzer nach einer Zelle zum Aufdecken.
- Überprüft die Eingabe auf Gültigkeit.
- Führt den Zug mit `_handleMove` aus.
- Überprüft den Spielstatus (gewonnen/verloren).
- Falls das Spiel endet, wird `_end()` aufgerufen.

_handleMove(self, x: int, y: int) -> GameState

Beschreibung:
Führt einen Spielzug an der angegebenen Position aus.

Effekt:

- Überprüft, ob die gewählte Zelle bereits aufgedeckt wurde.
- Falls eine Falle getroffen wurde, verliert der Spieler (`GameState.LOST`).
- Falls keine Falle vorhanden ist, wird das Spielfeld aktualisiert.
- Prüft, ob der Spieler gewonnen hat.

Parameter:

- `x` (*int*): X-Koordinate der gewählten Zelle.
- `y` (*int*): Y-Koordinate der gewählten Zelle.

Rückgabewert:

- `GameState` – Aktueller Spielstatus (`WON`, `LOST`, `PLAYING`, `REPEAT`).

_checkWin(self) -> GameState

Beschreibung:
Prüft ob der Spieler gewonnen hat.

Effekt:

- Überprüft, ob die gewählte Zelle noch nicht aufgedeckt wurde und auch keine Falle ist.
- Falls eine Zelle noch nicht aufgedeckt wurde, geht das Spiel weiter (`GameState.PLAYING`).
- Falls alle Zellen schon aufgedeckt sind und keine davon eine Falle ist, gewinnt der Spieler (`GameState.WON`).

Rückgabewert:

- `GameState` – Aktueller Spielstatus (`WON`, `LOST`, `PLAYING`, `REPEAT`).

Klasse `Board`

Die Klasse `Board` repräsentiert das Spielfeld des Spiels. Sie verwaltet die Spielfeldzellen, platziert Fallen und ermöglicht das Aufdecken von Zellen.

Methodenbeschreibung

`__init__(self, size: int) -> None`

Beschreibung:
Erstellt ein quadratisches Spielfeld mit der angegebenen Größe.

Attribute:

- `_size` (*int*): Die Größe des Spielfelds.
- `_grid` (*list[list[Cell]]*): 2D-Liste, die die Spielfeldzellen speichert.

Parameter:

- `size` (*int*): Größe des Spielfelds.

`getCell(self, x: int, y: int) -> Cell`

Beschreibung:
Gibt die Zelle an den angegebenen Koordinaten zurück.

Rückgabewert:

- `Cell` – Die Zelle an den gegebenen Koordinaten.

`getSize(self) -> int`

Beschreibung:
Gibt die Größe des Spielfelds zurück.

Rückgabewert:

- `int` – Die Größe des 2D Spielfelds.

---

placeTraps(self, dx: int, dy: int) -> None

Beschreibung:
Platziert eine bestimmte Anzahl zufällig verteilter Fallen auf dem Spielfeld, wobei die Startposition (dx, dy) ausgelassen wird.

Effekt:

- Vermeidet, dass an der Startposition eine Falle platziert wird.
- Ruft anschließend die Berechnung benachbarter Fallen für alle Zellen auf.

Parameter:

- dx (*int*): X-Koordinate der Startzelle.
- dy (*int*): Y-Koordinate der Startzelle.

_calculateAdjacentTraps(self) -> None

Beschreibung:
Berechnet für jede Zelle die Anzahl der benachbarten Fallen und speichert diese Information.

Effekt:

- Setzt für jede nicht mit einer Falle belegte Zelle die Anzahl benachbarter Fallen.

_countTraps(self, x: int, y: int) -> int

Beschreibung:
Zählt die Anzahl der Fallen in den benachbarten Zellen der Position (x, y).

Rückgabewert:

- int – Anzahl der angrenzenden Fallen.

Parameter:

- x (*int*): X-Koordinate der Zelle.
- y (*int*): Y-Koordinate der Zelle.

---

scanArea(self, x: int, y: int) -> None

Beschreibung:
Deckt die Zelle an der Position (x, y) auf. Falls dort keine benachbarten Fallen sind, werden angrenzende Zellen ebenfalls aufgedeckt.

Effekt:

- Deckt in einem 3x3 Feld um die Zelle herum die Felder auf.
- Falls ein Feld dabei keine Fallen um sich hat, wird scanArea(x, y) auch für dieses Feld aufgerufen.

Parameter:

- x (*int*): X-Koordinate der zu scannende Zelle.
- y (*int*): Y-Koordinate der zu scannende Zelle.

---

displayBoard(self) -> None

Beschreibung:
Gibt das aktuelle Spielfeld in der Konsole aus.

Effekt:

- Zeigt das Spielfeld mit Zeilen- und Spaltennummern zur besseren Orientierung an.

---

scanTraps(self) -> None

Beschreibung:
Deckt alle Fallen auf dem Spielfeld auf und zeigt das vollständige Spielfeld an.

Effekt:

- Markiert alle Fallen als aufgedeckt.
- Gibt das aktualisierte Spielfeld aus.

Klasse `Cell`

Die Klasse `Cell` repräsentiert eine einzelne Zelle im Spielfeld (`gameBoard`). Sie speichert Informationen darüber, ob eine Zelle eine Falle ist, ob sie schon aufgedeckt wurde und wie viele Fallen sich in ihrer direkten Nachbarschaft befinden.

Methodenbeschreibung

`__init__(self, isTrap: bool = False)`

Beschreibung:
Initialisiert eine Zelle.

Parameter:

- `isTrap` (*bool*): Gibt an, ob die Zelle eine Falle enthält (Standard: `False`).

Attribute:

- `_isTrap` (*bool*): Speichert, ob die Zelle eine Falle ist.
- `_adjacentTraps` (*int*): Speichert die Anzahl der Fallen in den benachbarten Zellen (Standard: `0`).
- `_isScanned` (*bool*): Gibt an, ob die Zelle bereits aufgedeckt wurde (Standard: `False`).

`getAdjacentTraps(self) -> int`

Beschreibung:
Gibt die Anzahl der benachbarten Fallen der Zelle zurück.

Rückgabewert:

- `int` – Anzahl angrenzender Fallen.

---

isScanned(self) -> bool

Beschreibung:
Gibt zurück, ob die Zelle bereits aufgedeckt wurde.

Rückgabewert:

- `bool` – `True`, wenn die Zelle aufgedeckt wurde, sonst `False`.

---

isTrap(self) -> bool

Beschreibung:
Gibt zurück, ob sich in der Zelle eine Falle befindet.

Rückgabewert:

- `bool` – `True`, wenn die Zelle eine Falle ist, sonst `False`.

---

setAdjacentTraps(self, value: int) -> None

Beschreibung:
Setzt die Anzahl benachbarter Fallen für diese Zelle.

Parameter:

- `value` (*int*): Die Anzahl angrenzender Fallen.

---

scan(self) -> None

Beschreibung:
Markiert die Zelle als aufgedeckt.

Effekt:

- Setzt `_isScanned` auf `True`, sodass die Zelle nicht erneut aufgedeckt wird.

Klasse OutOfRangeError

Die Klasse OutOfRangeError ist für den Fehler, dass eine Benutzereingabe außerhalb eines vorgegebenen Wertebereichs ist.

Methodenbeschreibung

__init__(self, value: int, minValue: int, maxValue: int) -> None

Beschreibung:
Erstellt Fehlerobjekt für zu hohe oder zu niedrige Eingaben.

Parameter:

- value (*int*): Der ungültige Wert.
- minValue (*int*): Die minimale erlaubte Grenze.
- maxValue (*int*): Die maximale erlaubte Grenze.

Attribute:

- _value (*int*): Speichert den ungültigen Wert.
- _min (*int*): Speichert die untere Grenze des gültigen Bereichs.
- _max (*int*): Speichert die obere Grenze des gültigen Bereichs.

__str__(self) -> str

Beschreibung:
Gibt eine formatierte Fehlermeldung zurück, die angibt, dass der Wert außerhalb des gültigen Bereichs liegt.

Rückgabewert:

1. *str* – Fehlermeldung im Format "<Wert> is out of the Possible Range (<min>-<max>)"

4. Ordnerstruktur

```
PYTHON_ABGABE
├── src
│   ├── __init__.py
│   ├── cell.py
│   ├── errors.py
│   ├── game.py
│   ├── board.py
│   └── main.py
├── tests
│   ├── __init__.py
│   ├── testBoard.py
│   ├── testCell.py
│   ├── testErrors.py
│   ├── testGame.py
│   └── testMain.py
├── .pylintrc
├── mypy.ini
├── README.md
└── requirements.txt
```

5. Versionen der Bibliotheken

```
astroid==3.3.8
coverage==7.6.12
dill==0.3.9
isort==6.0.1
mccabe==0.7.0
mypy==1.15.0
mypy-extensions==1.0.0
platformdirs==4.3.6
pylint==3.3.4
tomlkit==0.13.2
typing_extensions==4.12.2
```

6. Testergebnisse

mypy

```
(venv) I750588@LFPC7YMWX2 Python_Abgabe % mypy .
Success: no issues found in 12 source files
```

pylint

- **src**

```
(venv) I750588@LFPC7YMWX2 Python_Abgabe % pylint src/

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

  - Bei `game.py` `pylint: disable=too-few-public-methods` bei der `GameState(Enum)`, da eine Enumeration keine Klassen besitzt.

- **tests**

```
(venv) I750588@LFPC7YMWX2 Python_Abgabe % pylint tests/

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

  - Bei `testGame.py` `pylint: disable=protected-access` für alle privaten Attribute/ Methoden, welche von `game.py` verwendet werden, da diese eigentlich geschützt sind und nicht zugänglich für andere sein sollte.

unittest

```
(venv) I750588@LFPC7YMWX2 Python_Abgabe % coverage run -m unittest discover -s tests
.........-----------------------------------------
Goodbye!
----------------------------------------
.-----------------Welcome-----------------
...2 is out of the Possible Range (5-15)
16 is out of the Possible Range (5-15)
..Game Over! You hit a trap.
..
----------------------------------------------------------------------
Ran 19 tests in 0.013s

OK
```

coverage

```
(venv) I750588@LFPC7YMWX2 Python_Abgabe % coverage report
Name                    Stmts   Miss  Cover
-------------------------------------------
src/__init__.py             0      0   100%
src/board.py               71     15    79%
src/cell.py                15      0   100%
src/errors.py               7      0   100%
src/game.py                93     17    82%
src/main.py                 6      1    83%
tests/testBoard.py         39      1    97%
tests/testCell.py          18      1    94%
tests/testErrors.py         9      1    89%
tests/testGame.py          66      1    98%
tests/testMain.py          11      1    91%
-------------------------------------------
TOTAL                     335     38    89%
```