Projektarbeit Ben Müller

Inhaltsverzeichnis

Aufbau und Funktionen des Spiels

Welche Funktionen sind in meinem Code?

```python
def print_traps_layout() -> None:
    """
    Method to print the layout of the game.
    """
    print()
    print("\t\tSPACESTATION\n")
    st = "    "
    for i in range(N):
        st = st + "      " + str(i + 1)
    print(st)
    for r in range(N):
        st = "     "
        if r == 0:
            for col in range(N):
                st = st + "______"
            print(st)
        st = "     "
        for col in range(N):
            st = st + "|      "
        print(st + "|")
        st = "   " + str(r + 1) + "  "
        for col in range(N):
            st = st + "|   " + str(TRAP_VALUES[r][col]) + "   "
        print(st + "|")
        st = "     "
        for col in range(N):
            st = st + "|______"
        print(st + '|')
    print()
```

Die print_traps_layout Funktion baut das leere Spielfeld auf. Hier wird Über N, die variable für Spalten und Zeilen, iteriert und formatiert.

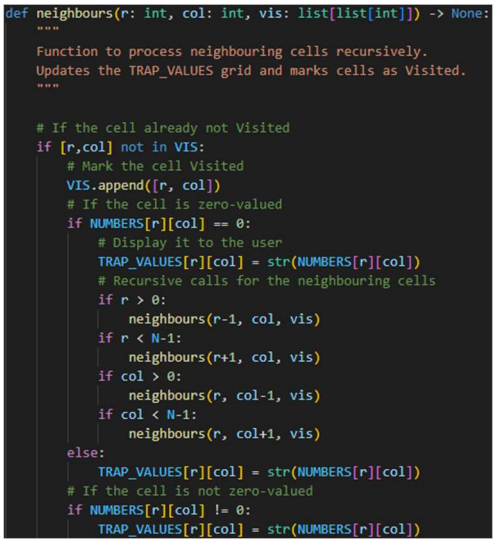
```python
def set_traps() -> None:
    """
    Function for setting up traps in the grid.
    """
    # Track of number of traps already set up
    count = 0
    while count < TRAPS_NO:
        # Random number from all possible grid positions
        val = random.randint(0, N*N-1)
        # Generating row and column from the number
        r = val // N
        col = val % N
        # Place the trap, if it doesn't already have one
        if NUMBERS[r][col] != -1:
            count = count + 1
            NUMBERS[r][col] = -1
```

Die set_traps Funktion generiert

So lange eine zufällige zahl zwischen 0 und in meinem Fall 24 bis die vorgegebene Anzahl an Fallen erreicht ist. Für diese zufälligen Zahlen wird danach die jeweilige Spalte und Zahl ausgerechnet und fügt diese dann der Liste Numbers hinzu.

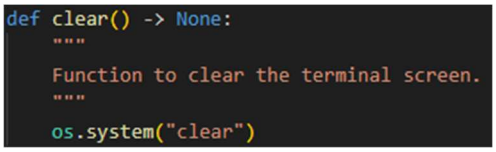
```python
def set_values() -> None:
    """
    Function for setting up the other grid values.
    """
    # Loop for counting each cell value
    for r in range(N):
        for col in range(N):
            # Skip, if it contains a trap
            if NUMBERS[r][col] == -1:
                continue
            # Check up
            if r > 0 and NUMBERS[r-1][col] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check down
            if r < N-1  and NUMBERS[r+1][col] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check left
            if col > 0 and NUMBERS[r][col-1] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check right
            if col < N-1 and NUMBERS[r][col+1] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check top-left
            if r > 0 and col > 0 and NUMBERS[r-1][col-1] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check top-right
            if r > 0 and col < N-1 and NUMBERS[r-1][col+1] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check below-left
            if r < N-1 and col > 0 and NUMBERS[r+1][col-1] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
            # Check below-right
            if r < N-1 and col < N-1 and NUMBERS[r+1][col+1] == -1:
                NUMBERS[r][col] = NUMBERS[r][col] + 1
```

Die set_values Funktion weist jedem Feld des Spiels die Nummer, der umliegenden Fallen, zu. Die Funktion geht nacheinander jede einzelne Möglichkeit ab um die Nummer zu erstellen. Diese Nummer wird auch aufgedeckt wenn man das Feld absucht.



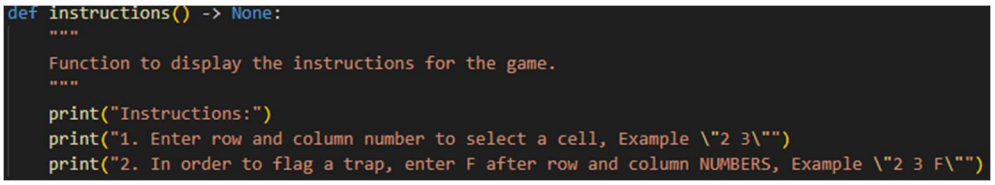
```python
def neighbours(r: int, col: int, vis: list[list[int]]) -> None:
    """
    Function to process neighbouring cells recursively.
    Updates the TRAP_VALUES grid and marks cells as Visited.
    """

    # If the cell already not Visited
    if [r,col] not in VIS:
        # Mark the cell Visited
        VIS.append([r, col])
        # If the cell is zero-valued
        if NUMBERS[r][col] == 0:
            # Display it to the user
            TRAP_VALUES[r][col] = str(NUMBERS[r][col])
            # Recursive calls for the neighbouring cells
            if r > 0:
                neighbours(r-1, col, vis)
            if r < N-1:
                neighbours(r+1, col, vis)
            if col > 0:
                neighbours(r, col-1, vis)
            if col < N-1:
                neighbours(r, col+1, vis)
        else:
            TRAP_VALUES[r][col] = str(NUMBERS[r][col])
        # If the cell is not zero-valued
        if NUMBERS[r][col] != 0:
            TRAP_VALUES[r][col] = str(NUMBERS[r][col])
```

Die neighbours Funktion ist eine Rekursive Funktion, welche, wenn ein Feld mit dem Wert Null aufgedeckt wird, die umliegenden Felder prüft, ob dort auch eine Null ist und deckt dies auch auf. Dies funktioniert, indem die Felder als „Visited“ markiert werden, sobald sie besucht wurde und wird danach ignoriert.



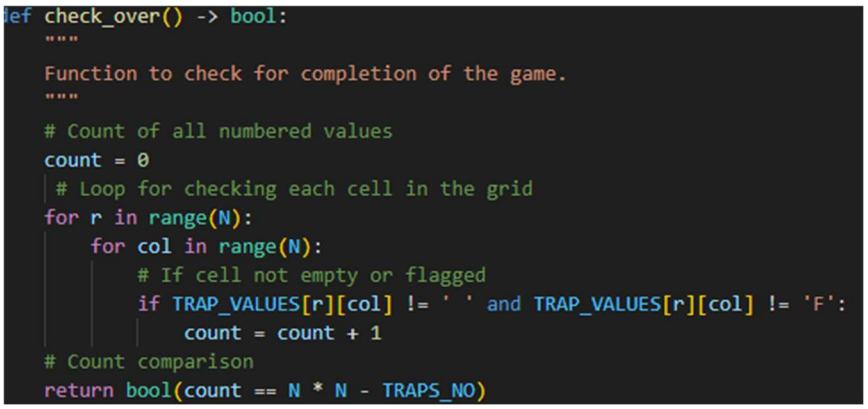
```python
def clear() -> None:
    """
    Function to clear the terminal screen.
    """
    os.system("clear")
```

Die Funktion clear reinigt das Terminal.



```python
def instructions() -> None:
    """
    Function to display the instructions for the game.
    """
    print("Instructions:")
    print("1. Enter row and column number to select a cell, Example \"2 3\"")
    print("2. In order to flag a trap, enter F after row and column NUMBERS, Example \"2 3 F\"")
```

Die Funktion instructions gibt zum Start des Spielst die Regeln und Eingabe vorgaben im Terminal aus.



```python
def check_over() -> bool:
    """
    Function to check for completion of the game.
    """
    # Count of all numbered values
    count = 0
    # Loop for checking each cell in the grid
    for r in range(N):
        for col in range(N):
            # If cell not empty or flagged
            if TRAP_VALUES[r][col] != ' ' and TRAP_VALUES[r][col] != 'F':
                count = count + 1
    # Count comparison
    return bool(count == N * N - TRAPS_NO)
```

Die Funktion check_over überprüft ob alle Felder welche nicht Fallen oder Flaggen gleich der aufgedeckten Felder sind.

```python
def show_traps() -> None:
    """
    Function to display all the trap locations.
    """

    for r in range(N):
        for col in range(N):
            if NUMBERS[r][col] == -1:
                TRAP_VALUES[r][col] = 'M'
```

Die show_traps Funktion deckt alle Felder, welche eine Falle sind, auf. Das funktioniert indem alle Felder mit dem versteckten Wert -1, welcher eine Falle repräsentiert, durch ein „M“ welches aufgedeckt wird.

```python
def process_input(inp: list[str]) -> tuple[int, int] | None:
    """
    Process user input and validate it.
    Returns a tuple of row and column indices if valid, or None if invalid.
    """
    try:
        val = list(map(int, inp))
        if val[0] > N or val[0] < 1 or val[1] > N or val[1] < 1:
            clear()
            print("Wrong input!")
            instructions()
            return None
        return val[0] - 1, val[1] - 1
    except ValueError:
        clear()
        print("Wrong input!")
        instructions()
        return None
```

Die process_input Funktion kümmert sich um die Benutzereingabe und bestätigt, dass sie richtig ist.

```python
def handle_flag(val: tuple[int, int]) -> None:
    """
    Handle flagging logic for the game.
    """
    r, col = val
    if [r, col] in FLAGS:
        clear()
        print("Flag already set")
        return

    if TRAP_VALUES[r][col] != ' ':
        clear()
        print("Value already known")
        return

    if len(FLAGS) < TRAPS_NO:
        clear()
        print("Flag set")
        FLAGS.append([r, col])
        TRAP_VALUES[r][col] = 'F'
    else:
        clear()
        print("Flags finished")
```

Die handle_flag Funktion beinhaltet die Logik hinter den gesetzten Flaggen. Sie überprüft ob auf dem Feld schon eine Flagge ist oder ob der Wert des Feldes bereits bekannt ist. Ist beides nicht der Fall wird eine Flagge mit „F“ gesetzt. Ein „F“ verhindert jedoch nicht die Auswahl dieses Feldes.

```python
def handle_cell_selection(val: tuple[int, int]) -> bool:
    """
    Handle cell selection logic for the game.
    Returns True if the game is over, False otherwise.
    """
    r, col = val

    if [r, col] in FLAGS:
        FLAGS.remove([r, col])

    if NUMBERS[r][col] == -1:
        TRAP_VALUES[r][col] = 'M'
        show_traps()
        print_traps_layout()
        print("Landed on a trap. GAME OVER!!!!!")
        return True

    if NUMBERS[r][col] == 0:
        vis: list[list[int]] = []
        TRAP_VALUES[r][col] = '0'
        neighbours(r, col, vis)
    else:
        TRAP_VALUES[r][col] = str(NUMBERS[r][col])

    if check_over():
        show_traps()
        print_traps_layout()
        print("Congratulations!!! YOU WIN")
        return True

    return False
```

Die handle_cell_selection Funktion prüft welchen Wert das ausgewählte Feld hat und gibt je nach dem aus ob man Verloren hat oder nicht. Dies wird gemacht indem verglichen wird ob das ausgewählte Feld den versteckten Wert „-1“ hat.

```python
def game_loop() -> None:
    """
    Main function to run the game loop.
    """
    set_traps()
    set_values()
    instructions()

    over = False

    while not over:
        print_traps_layout()

        inp = input("Enter row number followed by space and column number = ").split()

        if len(inp) == 2:
            result = process_input(inp)
            if result is None:
                continue
            over = handle_cell_selection(result)

        elif len(inp) == 3:
            if inp[2].lower() != 'f':
                clear()
                print("Wrong Input!")
                instructions()
                continue
            result = process_input(inp[:2])
            if result is None:
                continue
            handle_flag(result)

        else:
            clear()
            print("Wrong input!")
            instructions()
if __name__ == "__main__":
    game_loop()
```

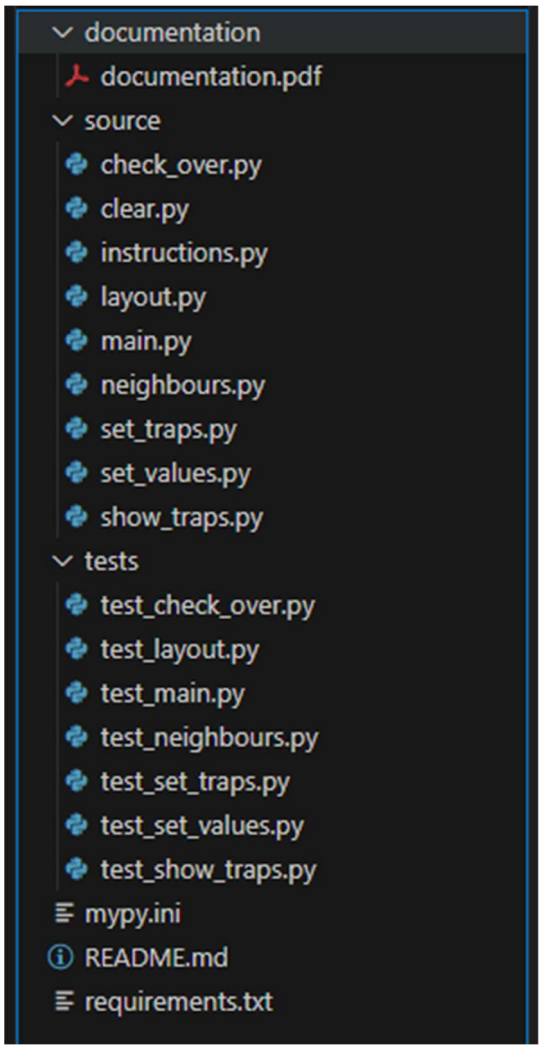
Die game_loop Funktion vereint alle bisher genannten Funktionen und lässt den Benutzer solange spielen, bis er das Spiel gewonnen oder verloren hat.

Versionen aller Bibliotheken

In der requirements.txt findet man alle Bibliotheken, welche für dieses Spiel benötigt werden.

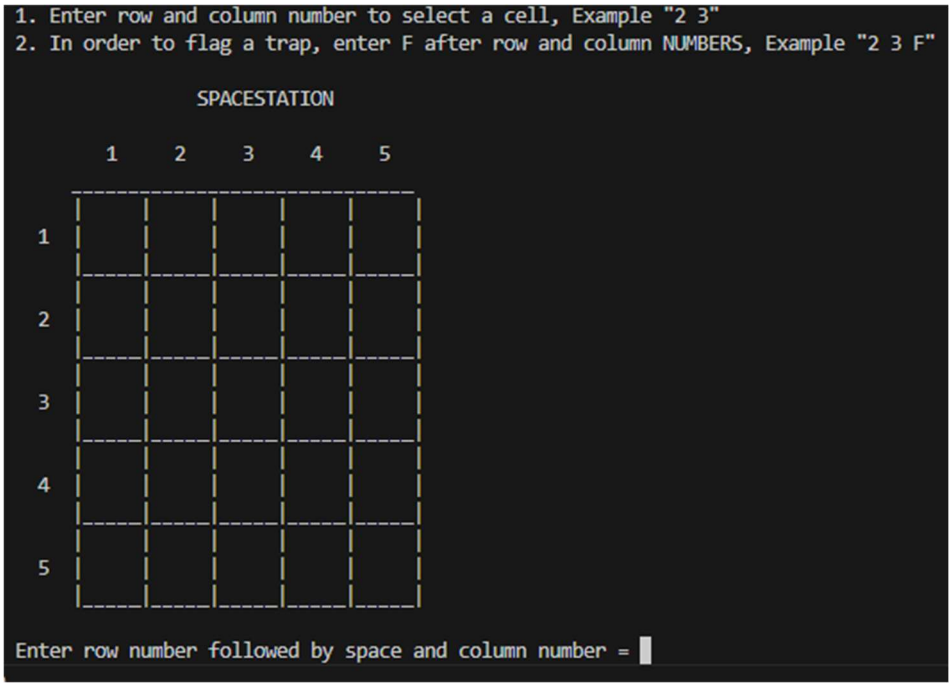
```
astroid==3.3.9
coverage==7.8.0
dill==0.3.9
isort==6.0.1
mccabe==0.7.0
mypy==1.15.0
mypy-extensions==1.0.0
platformdirs==4.3.7
pylint==3.3.6
tomlkit==0.13.2
```

Architekturbeschreibung



Ich habe mich an die vorgegebene Struktur gehalten und das ist meine endgültige Struktur

Nutzerinterface

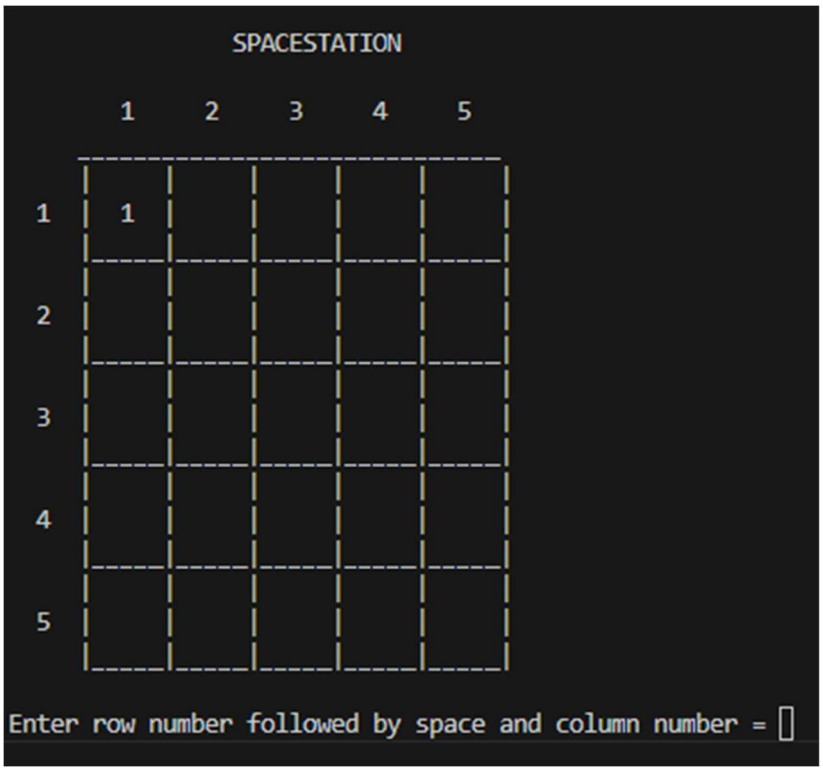


Wenn die main.py ausgeführt wird, sieht der Benutzer das.

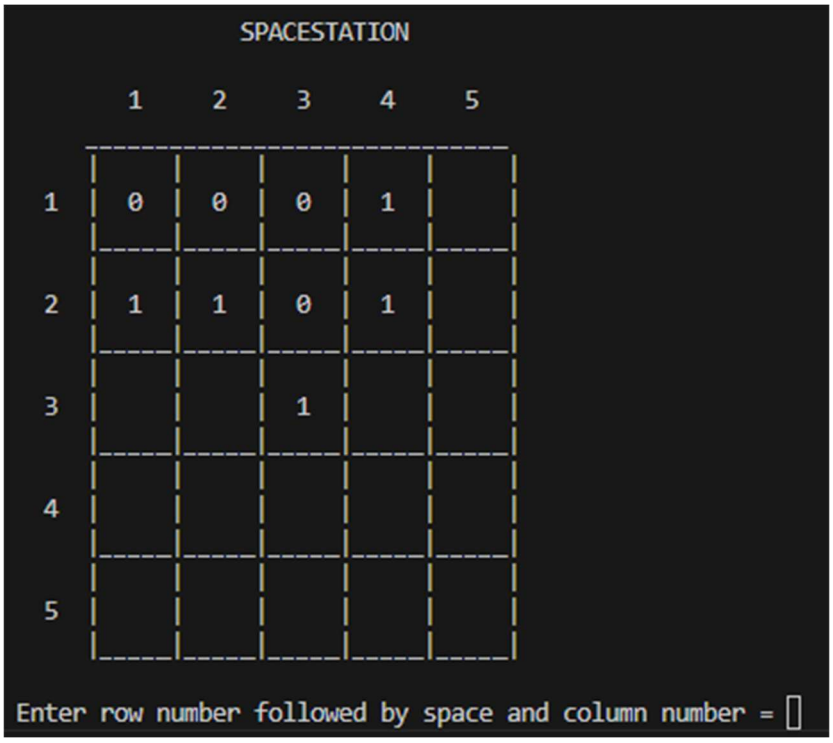
Ganz oben die Beschreibung wie man das Spiel spielt. Darunter dann das formatierte Spielfeld mit nummerierten Zeilen und Spalten. Und ganz unten die Eingabeaufforderung wobei Zeilen- und Spaltennummer und gegebenenfalls ein „F“, um eine Flagge dort zu setzen, erwartet.

Programmablauf

Wenn der Benutzer nun eine gültige Eingabe tätigt, in diesem Fall „1 1“ wird das jeweilige Feld aufgedeckt.

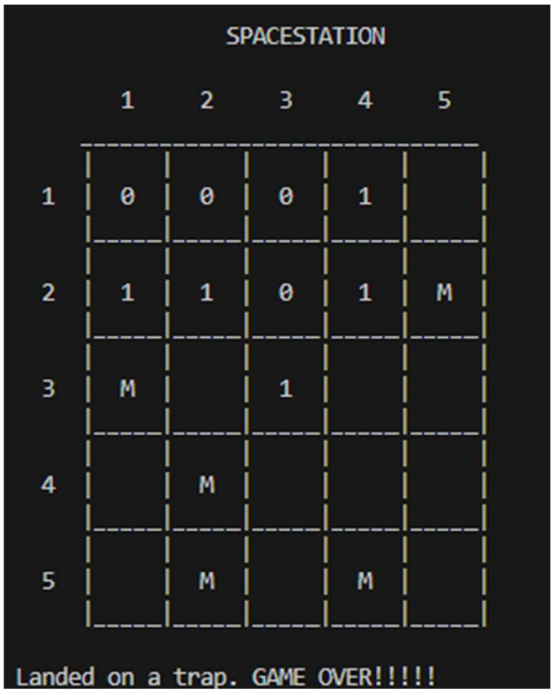


Der Spieler hat nun wieder die Möglichkeit eine Eingabe zu tätigen. In einem anderen Spiel war die Eingabe wieder „1 1“ aber das gewählte Feld hat den Wert Null, weshalb mehrere Felder aufgedeckt wurden.

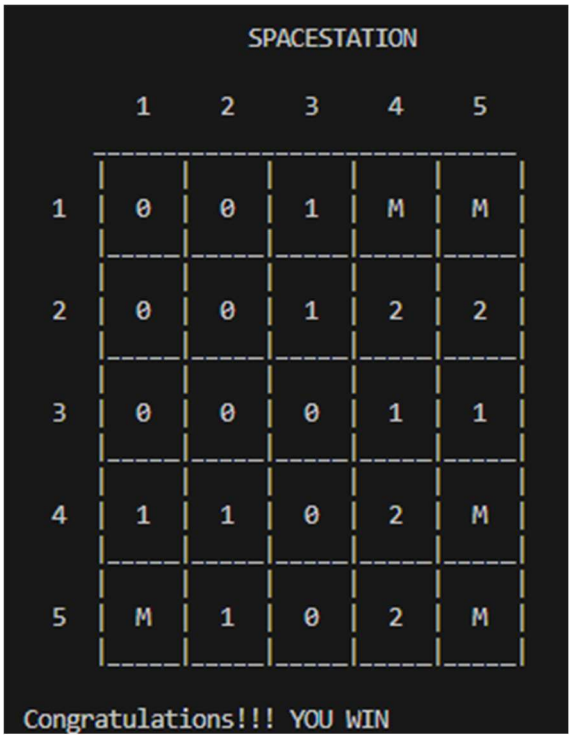


Bei diesem Spielfeld ist der Spieler sicher, dass unter „3 1“ eine Falle liegt, weshalb er mit der Eingabe „3 1 F“ eine Flagge auf dieses Feld setzt.

Wenn der Spieler jedoch trotz seiner gesetzten Flagge dieses Feld nun auswählt, und somit auf Eine Falle trifft, werden alle Fallen aufgedeckt und das Spiel ist verloren.



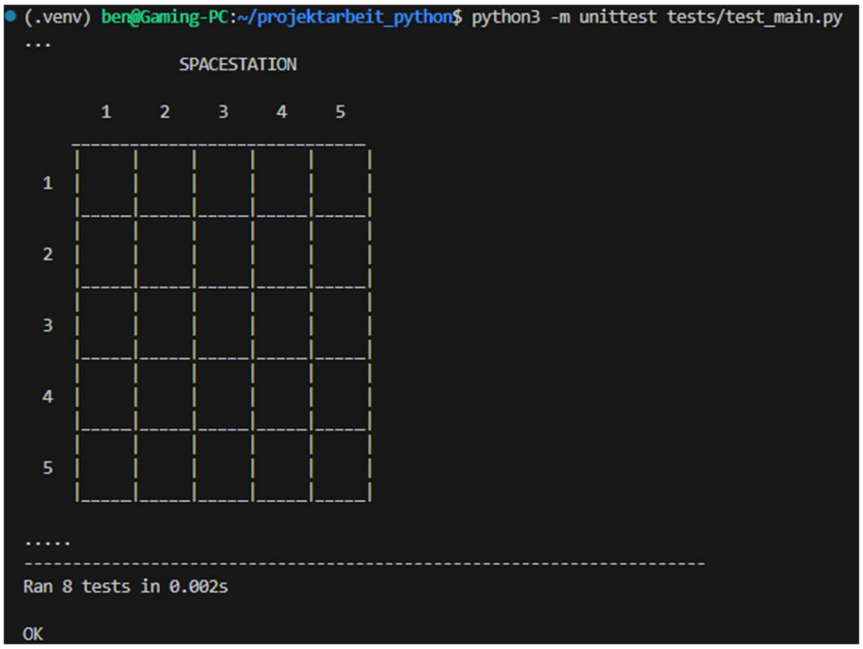
Das Spiel ist gewonnen, wenn alle Felder ohne Falle aufgedeckt wurden. Es werden auch in diesem Fall am Ende des Spiels alle Fallen aufgedeckt.

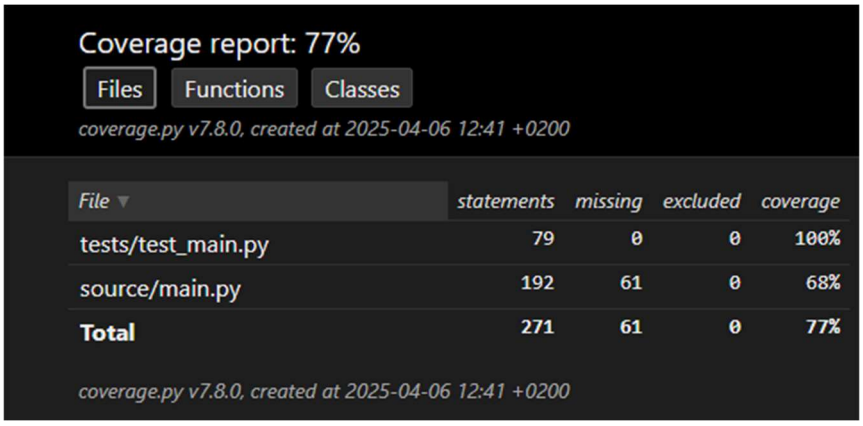


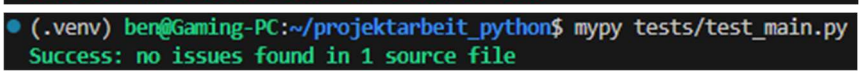
Test-Ergebnisse

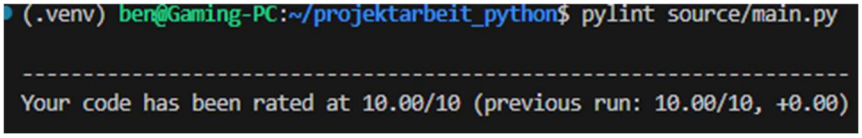
Nun noch alle Test-Ergebnisse:

main.py/test_main.py:

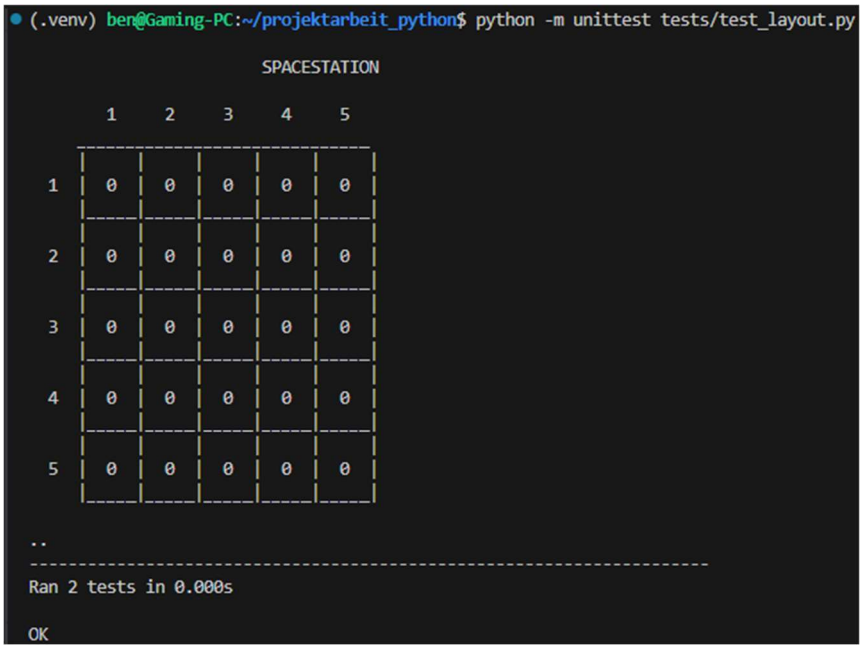


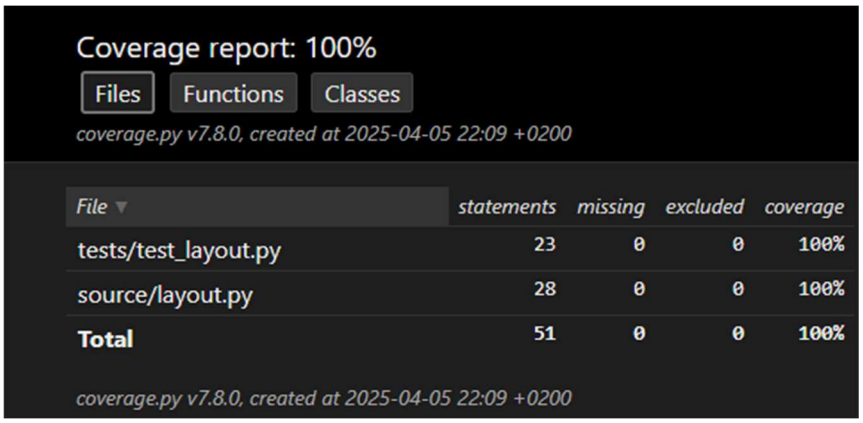


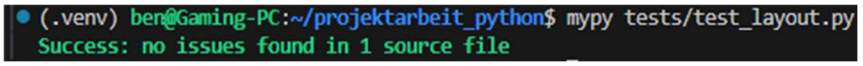


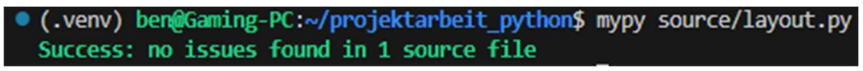


layout.py/test_layout.py:









set_traps.py/test_set_traps.py:

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ python3 -m unittest tests/test_set_traps.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Coverage report: 100%

Files | Functions | Classes

coverage.py v7.8.0, created at 2025-04-05 22:42 +0200

File	statements	missing	excluded	coverage
tests/test_set_traps.py	21	0	0	100%
source/set_traps.py	13	0	0	100%
Total	34	0	0	100%

coverage.py v7.8.0, created at 2025-04-05 22:42 +0200

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy source/set_traps.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy tests/test_set_traps.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ pylint source/set_traps.py

----------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.46/10, +1.54)
```

set_values.py/test_set_values.py:

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ python3 -m unittest tests/test_set_values.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Coverage report: 100%

Files | Functions | Classes

coverage.py v7.8.0, created at 2025-04-05 22:59 +0200

File	statements	missing	excluded	coverage
tests/test_set_values.py	25	0	0	100%
source/set_values.py	24	0	0	100%
Total	49	0	0	100%

coverage.py v7.8.0, created at 2025-04-05 22:59 +0200

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy source/set_values.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy tests/test_set_values.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ pylint source/set_values.py

----------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.17/10, +0.83)
```

neighbours.py/test_neighbours.py:

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ python3 -m unittest tests/test_neighbours.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

Coverage report: 100%

Files | Functions | Classes

coverage.py v7.8.0, created at 2025-04-05 23:47 +0200

File	statements	missing	excluded	coverage
tests/test_neighbours.py	36	0	0	100%
source/neighbours.py	25	0	0	100%
Total	**61**	**0**	**0**	**100%**

coverage.py v7.8.0, created at 2025-04-05 23:47 +0200

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy source/neighbours.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy tests/test_neighbours.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ pylint source/neighbours.py

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.26/10, +0.74)
```

check_over.py/test_check_over.py:

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ python3 -m unittest tests/test_check_over.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Coverage report: 98%

Files | Functions | Classes

coverage.py v7.8.0, created at 2025-04-06 11:42 +0200

File	statements	missing	excluded	coverage
tests/test_check_over.py	32	0	0	100%
source/check_over.py	12	1	0	92%
Total	**44**	**1**	**0**	**98%**

coverage.py v7.8.0, created at 2025-04-06 11:42 +0200

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy source/check_over.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ mypy tests/test_check_over.py
Success: no issues found in 1 source file
```

```
(.venv) ben@Gaming-PC:~/projektarbeit_python$ pylint source/check_over.py

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.33/10, +1.67)
```

show_traps.py/test_show_traps.py: