

Projektdokumentation: Abandoned Space Station

Inhaltsverzeichnis

- Projektdokumentation: Abandoned Space Station
 - Inhaltsverzeichnis
 - Einleitung
 - Spielfunktionen
 - Verwendete Bibliotheken
 - Architekturbeschreibung
 - Komponentenübersicht
 - Klassenstruktur
 - Hauptklasse: `AbandonedSpaceStation`
 - Datenfluss
 - Benutzeroberfläche
 - Programmablauf
 - Spielinitialisierung
 - Hauptspielschleife
 - Spielabschluss
 - Qualitätssicherung
 - Unit Tests
 - Test Coverage
 - Statische Codeanalyse (Pylint)
 - Typprüfung (MyPy)
 - Fazit und Ausblick

Einleitung

"Abandoned Space Station" ist ein konsolenbasiertes Logikspiel, das von dem klassischen Minensucher-Spielprinzip inspiriert ist. Die Spieler übernehmen die Rolle eines Entdeckers, der eine verlassene Raumstation untersuchen muss. Die Station enthält zahlreiche Gefahrenherde, die es zu vermeiden gilt. Das Ziel des Spiels ist es, alle sicheren Bereiche zu scannen und so die Station zu sichern, ohne dabei Gefahrenzonen auszulösen.

Diese Dokumentation beschreibt detailliert die Funktionalität, Architektur und technischen Aspekte des Spiels sowie die Ergebnisse der Qualitätssicherungsmaßnahmen.

Spielfunktionen

Das Spiel "Abandoned Space Station" bietet folgende Hauptfunktionen:

- 1. Zufallsgenerierte Spielfelder:** Bei jedem Spielstart wird ein neues Spielfeld mit zufällig platzierten Gefahren erstellt.
- 2. Anpassbare Spielfeldgröße:** Spieler können die Größe des Spielfelds (Breite und Höhe) sowie die Anzahl der Gefahren anpassen.
- 3. Gefahrenerkennung:** Beim Scannen sicherer Bereiche wird die Anzahl benachbarter Gefahrenherde angezeigt.
- 4. Strategisches Gameplay:** Spieler müssen logisch denken und Rückschlüsse ziehen, um alle sicheren Bereiche zu identifizieren.
- 5. Spielstatistiken:** Nach Beendigung des Spiels werden Statistiken angezeigt, darunter:
 - Anzahl der gescannten Bereiche
 - Prozentsatz des erkundeten Gebiets
 - Gesamtanzahl der durchgeführten Aktionen
- 6. Fehlerbehandlung:** Das Spiel umfasst eine robuste Fehlerbehandlung für ungültige Benutzereingaben und Ausnahmesituationen.
- 7. Plattformübergreifende Unterstützung:** Das Spiel funktioniert sowohl auf Windows- als auch auf Unix/Linux/Mac-Systemen.

Verwendete Bibliotheken

Die folgende Tabelle zeigt alle verwendeten externen Bibliotheken und deren Versionen:

Bibliothek	Version	Verwendungszweck
pylint	3.2.3	Statische Codeanalyse
coverage	7.5.3	Testabdeckungsanalyse
mypy	1.10.0	Statische Typprüfung

Das Spiel verwendet außerdem die folgenden Standard-Python-Bibliotheken:

- `random`: Für die zufällige Platzierung von Gefahren
- `sys`: Für Systemoperationen und Zugriff auf Befehlszeilenargumente
- `os`: Für betriebssystemspezifische Funktionalitäten
- `unittest`: Für das Testen von Komponenten
- `typing`: Für Typ-Annotationen

Architekturbeschreibung

Komponentenübersicht

Das Projekt folgt einer modularen Struktur mit klarer Trennung von Zuständigkeiten:

1. **Hauptspiellogik** (`game.py`): Enthält die Hauptklasse `AbandonedSpaceStation`, die den Spielzustand verwaltet und die Kernlogik des Spiels implementiert.
2. **Hilfsfunktionen** (`helpers.py`): Stellt allgemeine Hilfsfunktionen bereit, die von verschiedenen Teilen des Spiels verwendet werden, wie z.B. `clear_terminal()` und `process_coordinates()`.
3. **Hauptprogramm** (`main.py`): Dient als Einstiegsplatz für das Spiel, verwaltet Benutzerinteraktionen für Spieleinstellungen und leitet den Spielablauf ein.
4. **Tests**: Separate Testmodule für jede Hauptkomponente des Spiels:
 - `test_game.py`: Tests für die Spiellogik
 - `test_helpers.py`: Tests für Hilfsfunktionen
 - `test_main.py`: Tests für die Main-Funktionen

Klassenstruktur

Hauptklasse: `AbandonedSpaceStation`

Die zentrale Klasse des Spiels mit folgenden Hauptmethoden:

- `__init__()`: Initialisiert ein neues Spielobjekt mit Spielfeldgröße und Gefahrenanzahl
- `_place_hazards()`: Platziert Gefahren zufällig auf dem Spielfeld
- `_count_adjacent_hazards()`: Zählt angrenzende Gefahren für einen bestimmten Bereich
- `scan_area()`: Führt einen Scan an bestimmten Koordinaten durch
- `check_victory_condition()`: Überprüft, ob das Spiel gewonnen wurde
- `display_grid()`: Zeigt das aktuelle Spielfeld an
- `play()`: Hauptspielschleife für den Spielablauf
- `_show_statistics()`: Zeigt Spielstatistiken nach Spielende an

Datenfluss

Der Datenfluss im Spiel folgt diesem Muster:

1. **Initialisierung**: Das Hauptprogramm (`main.py`) erstellt eine Instanz der `AbandonedSpaceStation`-Klasse mit den gewünschten Spielparametern.
2. **Spielablauf**:
 - Die `play()`-Methode steuert den Hauptspielablauf
 - Benutzer geben Koordinaten ein, die durch `process_coordinates()` validiert werden

- Die `scan_area()`-Methode aktualisiert den Spielzustand basierend auf den eingegebenen Koordinaten
- Nach jeder Aktion wird der Spielzustand überprüft und das Spielfeld angezeigt

3. Spielende:

- Bei Spielende (Sieg oder Niederlage) werden die Ergebnisse und Statistiken angezeigt
- Rückkehr zum Hauptprogramm

Benutzeroberfläche

Das Spiel verfügt über eine textbasierte Konsoloberfläche mit folgenden Elementen:

- 1. Startbildschirm:** Zeigt den Spieldaten und bietet die Möglichkeit, die Spielparameter anzupassen.
- 2. Spielanleitung:** Erklärt die grundlegenden Spielregeln und die Bedeutung der Symbole.
- 3. Spielfeld-Anzeige:** Ein zweidimensionales Raster, das den aktuellen Zustand des Spielfelds darstellt:
 - `?`: Unerforschter Bereich
 - `0-8`: Anzahl der angrenzenden Gefahren
 - `H`: Gefahrenbereich (wird bei Spielende oder im Debug-Modus angezeigt)
- 4. Eingabeaufforderung:** Fordert den Benutzer auf, Koordinaten im Format "x y" einzugeben oder "q" zu drücken, um das Spiel zu beenden.
- 5. Statusmeldungen:** Informieren den Benutzer über das Ergebnis der aktuellen Aktion oder über Fehler bei der Eingabe.
- 6. Endergebnisanzeige:** Zeigt bei Spielende an, ob der Spieler gewonnen oder verloren hat, und präsentiert detaillierte Spielstatistiken.

Beispiel für die Spielfeld-Anzeige:

 Kopieren

0	1	2	3	4

0		?	?	?
1		?	1	1
2		?	1	0
3		?	?	1
4		?	?	?

Enter coordinates (x y) or 'q' to quit:

Programmablauf

Spielinitialisierung

1. Das Programm startet über `main.py`.
2. Der Benutzer wird gefragt, ob er die Spielfeldgröße anpassen möchte:
 - Bei Bestätigung werden benutzerdefinierte Einstellungen abgefragt.
 - Andernfalls werden die Standardeinstellungen verwendet.
3. Ein neues `AbandonedSpaceStation`-Objekt wird mit den gewählten Parametern erstellt.
4. Gefahren werden zufällig auf dem Spielfeld platziert.
5. Die Begrüßung und Spielanleitung werden angezeigt.

Flussdiagramm der Spielinitialisierung:

 Kopieren



Hauptspielschleife

1. Die Methode `play()` steuert den Hauptspielablauf.
2. Das aktuelle Spielfeld wird angezeigt.
3. Der Benutzer wird aufgefordert, Koordinaten einzugeben oder das Spiel zu beenden.
4. Die Eingabe wird validiert:
 - Ungültige Eingaben werden abgelehnt und eine erneute Eingabe wird angefordert.
 - Bei gültiger Eingabe wird der entsprechende Bereich gescannt.
5. Nach jedem Scan wird der Spielstatus aktualisiert.
6. Der Vorgang wiederholt sich, bis das Spiel gewonnen, verloren oder vom Benutzer beendet wird.

Flussdiagramm der Hauptspielschleife:

```
Spielschleife
|
v
Ist das Spiel      --Ja--> Ende
beendet?
|
Nein
|
v
Spielfeld anzeigen
|
v
Koordinaten vom
Benutzer einlesen
|
v
Eingabe = "q"?    --Ja--> Spiel beenden
|
Nein
|
v
Sind die Koordinaten  --Nein--> Fehlermeldung
gültig?
|
Ja
|
v
Bereich scannen
|
v
Ist der Bereich      --Ja--> Spieler verliert
eine Gefahr?
|
Nein
|
v
Anzeigen, wie viele
Gefahren angrenzend sind
|
v
Wurden alle sicheren --Ja--> Spieler gewinnt
Bereiche gescannt?
|
Nein
```

```
|  
+--<-----+  
|  
v  
Zurück zum Anfang  
der Schleife
```

Spielabschluss

1. Bei Niederlage wird eine entsprechende Meldung angezeigt.
2. Bei Sieg wird eine Glückwunschnachricht angezeigt.
3. In beiden Fällen werden die Spielstatistiken präsentiert:
 - Spielfeldgröße
 - Anzahl der Gefahren
 - Anzahl der gescannten Bereiche
 - Prozentsatz der erkundeten sicheren Bereiche
 - Gesamtanzahl der Aktionen

Qualitätssicherung

Unit Tests

Die Anwendung verfügt über umfangreiche Unit-Tests, die alle kritischen Funktionen abdecken. Die Tests sind in drei Hauptdateien organisiert:

1. **test_game.py**: Tests für die Hauptspiellogik
 - Test der Initialisierung
 - Test der Gefahrenplatzierung
 - Test der Zählung benachbarter Gefahren
 - Test des Bereichsscannens
 - Test der Spielgewinn- und Verlustbedingungen
2. **test_helpers.py**: Tests für die Hilfsfunktionen
 - Test der Terminal-Bereinigungsfunktion
 - Test der Koordinatenverarbeitung
3. **test_main.py**: Tests für die Hauptprogrammfunktionen
 - Test der benutzerdefinierten Einstellungen
 - Test der Hauptprogrammlogik
 - Test der Fehlerbehandlung

Testausführungsergebnisse:

Kopieren

```
Ran 24 tests in 0.017s
```

OK

Test Coverage

Der Testabdeckungsbericht zeigt eine hohe Abdeckung des Quellcodes:

Kopieren

Name	Stmts	Miss	Cover
<hr/>			
source/game.py	136	24	82%
source/helpers.py	21	0	100%
source/main.py	65	4	94%
<hr/>			
TOTAL	222	28	87%

Die Gesamtdeckung von 87% deutet auf eine hervorragende Testqualität hin. Die wenigen fehlenden Abdeckungen betreffen hauptsächlich Randfälle und Fehlerbehandlungs Routinen, die schwer automatisiert zu testen sind.

Statische Codeanalyse (Pylint)

Pylint wurde verwendet, um die Codequalität zu bewerten:

Kopieren

```
Your code has been rated at 10.00/10
```

Folgende Fehler wurden in .pylintrc unterdrückt:

wrong-import-position, import-error, line-too-long, too-many-instance-attributes

Diese hängen hauptsächlich mit dem für MyPy benötigten absoluten Pfad zusammen.

Ansonsten erfordert die Programmstruktur die Unterdrückung der weiteren Fehler.

Die Analyse zeigt eine hervorragende Codequalität mit nur geringfügigen Anmerkungen.

Typprüfung (MyPy)

Die MyPy-Typprüfung wurde durchgeführt, um potenzielle Typfehler zu identifizieren:

Success: no issues found in 10 source files

Das fehlerfreie Ergebnis zeigt, dass alle Typ-Annotationen korrekt sind und der Code den strengen Typprüfungen entspricht, die in der mypy.ini-Datei festgelegt wurden.

Fazit und Ausblick

Das Projekt "Abandoned Space Station" ist ein gut strukturiertes, umfassend getestetes Konsolenspiel mit robuster Fehlerbehandlung und benutzerfreundlicher Oberfläche. Die hohe Codequalität, extensive Testabdeckung und die strikte Typprüfung gewährleisten eine zuverlässige und wartbare Anwendung.

Mögliche zukünftige Erweiterungen könnten sein:

- 1. Speichern und Laden von Spielständen:** Ermöglichen, das Spiel zu unterbrechen und später fortzusetzen.
- 2. Schwierigkeitsgrade:** Implementierung verschiedener Schwierigkeitsgrade mit unterschiedlichen Spielregeln.
- 3. Grafikverbesserungen:** Integration von farbiger Konsolenausgabe für verbesserte Lesbarkeit.
- 4. Mehrsprachige Unterstützung:** Hinzufügen von Übersetzungen für internationale Benutzer.
- 5. Highscore-System:** Speichern und Anzeigen der besten Ergebnisse.
- 6. Verschiedene Spielmodi:** Implementierung alternativer Spielmodi, z.B. mit Zeitbegrenzung oder speziellen Herausforderungen.

Die solide Architektur und modulare Struktur des Spiels bilden eine ausgezeichnete Grundlage für diese und weitere potenzielle Verbesserungen.