ipso! Bildung seit 1876

Das Labyrinth

Eine Praxisarbeit von
Eggert, Adrian Leopold

ipso! Bildung seit 1876

Inhaltsverzeichnis

1 Management Summary

Das Labyrinth-C-Projekt beschäftigt sich mit der Entwicklung eines interaktiven Labyrinth-Spiels in der Programmiersprache C. Ziel der Praxisarbeit war es, ein funktionsfähiges Spiel zu entwickeln, bei dem ein Spieler sich durch ein Labyrinth bewegen, Hindernisse umgehen und ein definiertes Ziel erreichen kann. Besondere Aufmerksamkeit lag auf der modularen Programmierung, sodass einzelne Funktionalitäten wie die Spiellogik, die Labyrinth-Erstellung und die Benutzerschnittstelle in separaten Modulen implementiert wurden. Dies gewährleistet eine klare Struktur, erleichtert die Wartung und ermöglicht eine spätere Erweiterung des Projekts, beispielsweise durch neue Spielmoduse oder grafische Oberflächen.

Die Umsetzung erfolgte Schritt für Schritt: Zunächst wurde die Spiellogik gemacht, einschliesslich der Regeln für Bewegung und Kollisionen. Anschliessend wurde das Labyrinth dynamisch generiert, wobei unterschiedliche Schwierigkeitsgrade berücksichtigt werden. Die Benutzerinteraktion erfolgt über eine textbasierte Schnittstelle, die eine intuitive Steuerung ermöglicht. Parallel zur Implementierung wurde grosser Wert auf die Fehlerbehandlung gelegt, sodass ungültige Eingaben oder ungewöhnliche Spielsituationen korrekt verarbeitet werden.

Die Dokumentation des Projekts erfolgte nach einem standardisierten Aufbau, der die Analyse, die Planung, die Implementierung sowie Tests und Ergebnisse umfasst. Durch die modulare Struktur und die klare Trennung der Verantwortlichkeiten konnte ein stabiler, fehlerfreier Ablauf des Spiels gewährleistet werden. Das Projekt demonstriert die Vorteile strukturierter Softwareentwicklung in C, liefert eine nachvollziehbare Basis für zukünftige Erweiterungen und stellt sicher, dass der gesamte Entwicklungsprozess transparent dokumentiert ist.

Insgesamt wurde ein robustes und gut strukturiertes Softwareprojekt geschaffen, das nicht nur ein funktionsfähiges Spiel bereitstellt, sondern auch als Grundlage für weitere Projekte oder Lernzwecke im Bereich der modularen C-Programmierung dienen könnte. Die dokumentierten Arbeitsschritte ermöglichen eine einfache Nachvollziehbarkeit und bieten wertvolle Einblicke in die Umsetzung von Softwareprojekten nach modernen Entwicklungsprinzipien.

2 Aufgabenstellung

Die Aufgabe des Labyrinth-C-Projekts war es, ein kleines Labyrinth-Spiel in der Programmiersprache C zu entwickeln. Das Spiel sollte so aufgebaut sein, dass ein Spieler sich im Labyrinth bewegen, Hindernisse umgehen und ein Ziel erreichen kann. Wichtig war dabei, dass das Programm nicht als ein grosses Stück Code geschrieben wird, sondern in mehrere kleinen Module aufgeteilt ist. Jedes Modul sollte eine bestimmte Aufgabe haben, zum Beispiel das Labyrinth darstellen, die Bewegungen des Spielers berechnen oder die Eingaben des Benutzers verarbeiten. So wird der Code übersichtlicher und einfacher zu verstehen und zu verändern.

Ein weiteres Ziel war, dass das Spiel auch bei falschen Eingaben stabil bleibt. Wenn ein Spieler zum Beispiel eine ungültige Taste drückt, sollte das Programm richtig reagieren und nicht abstürzen. Gleichzeitig sollte der Code so geschrieben sein, dass später neue Funktionen wie zusätzliche Spielmodi oder eine grafische Anzeige leicht hinzugefügt werden können.

Die Aufgabenstellung verlangte außerdem, dass die Arbeit gut dokumentiert wird. Dazu gehörte, dass erklärt wird, wie die einzelnen Module aufgebaut sind, wie die Spiellogik funktioniert, welche Datenstrukturen verwendet werden und wie das Programm insgesamt umgesetzt wurde. Die Dokumentation sollte nachvollziehbar zeigen, wie das Spiel Schritt für Schritt entwickelt wurde.

Zusammengefasst ging es bei der Aufgabe darum, ein funktionierendes Labyrinth-Spiel zu entwickeln, das sauber in Module aufgeteilt ist, stabil läuft und gut dokumentiert ist. Das Projekt sollte sowohl zeigen wie man in C programmiert, als auch wie man ein Programm übersichtlich und erweiterbar aufbaut.

3 Anforderungen

Das Labyrinth-Spiel muss es dem Spieler ermöglichen, sich durch das Labyrinth zu bewegen, Hindernisse zu umgehen und das Ziel zu erreichen. Der Code soll modular aufgebaut sein, sodass jedes Modul eine klare Aufgabe hat, zum Beispiel Labyrinth-Erstellung, Spielerbewegungen oder Benutzereingaben. Das Programm muss fehlerrobust sein und falsche Eingaben des Benutzers korrekt behandeln, damit es stabil läuft. Außerdem soll die Arbeit gut dokumentiert werden, sodass die Funktionsweise der Module, die Spiellogik und die verwendeten Datenstrukturen nachvollziehbar sind. Schließlich muss das Projekt so strukturiert sein, dass es später leicht erweitert werden kann, etwa um neue Spielmodi oder eine grafische Oberfläche.

4 Design

Das Labyrinth-Spiel wird in einer textbasierten Darstellung im Terminal ausgegeben. Das Spielfeld besteht aus einem rechteckigen Raster, in dem jede Zelle entweder ein Weg, eine Wand oder das Ziel sein kann. Freie Wege werden durch Leerzeichen oder Punkte, und das Ziel wird durch ein eindeutiges Symbol, zum Beispiel `T`, markiert. Die Position des Spielers wird durch ein eigenes Symbol, zum Beispiel `P`, angezeigt, sodass jederzeit sichtbar ist, wo sich der Spieler innerhalb des Labyrinths befindet.

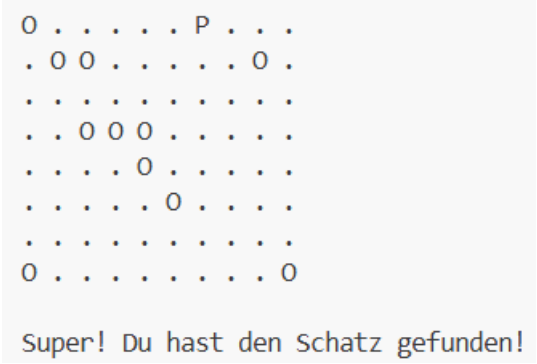

Abbildung 1 Endscreen

Beim Spielstart wird das Labyrinth vollständig angezeigt, und der Spieler kann seine Bewegungen über Tastatureingaben steuern. (W,A,S,D) Jede Bewegung aktualisiert die Darstellung des Spielfelds, sodass der Spieler seine Position im Labyrinth immer in Echtzeit verfolgen kann. Hindernisse wie Wände verhindern, dass der Spieler bestimmte Felder betreten kann, und das Programm prüft jede Eingabe daraufhin, ob sie gültig ist.

Zusätzlich kann das Spiel verschiedene Schwierigkeitsstufen abbilden, indem das Labyrinth komplexer wird, mehr Sackgassen enthält oder das Ziel weiter entfernt liegt. Die textbasierte Darstellung ermöglicht eine klare Übersicht über das Labyrinth und die aktuelle Spielsituation, ohne dass eine grafische Oberfläche erforderlich ist.

Insgesamt sorgt das Design dafür, dass der Spieler jederzeit Orientierung hat, das Labyrinth klar erkennbar ist und das Spielgeschehen verständlich und übersichtlich dargestellt wird.

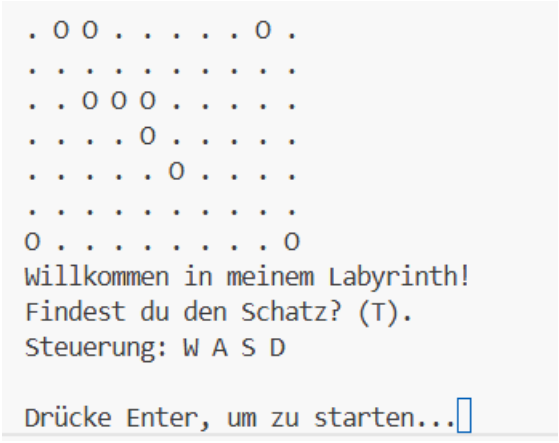

Abbildung 2 Labyrinth in der Konsole

5 Implementierung

Die Implementierung des Labyrinth-Spiels erfolgte in mehreren Schritten und orientierte sich an einer klaren, modularen Struktur. Ziel war es, das Spiel stabil, übersichtlich und erweiterbar zu gestalten. Die Software wurde in verschiedene C-Module unterteilt, wobei jedes Modul eine klar definierte Aufgabe hat. So wurde ein Modul für die Erstellung und Verwaltung des Labyrinths erstellt, ein weiteres Modul für die Spiellogik und die Bewegungen des Spielers und ein zusätzliches Modul für die Benutzereingaben und die Steuerung des Spiels. Für die Darstellung des Labyrinths wurde ein zweidimensionales Array verwendet. Jede Zelle im Array hat einen definierten Zustand: Wand, freier Weg, Spielerposition oder Ziel. Die freien Felder werden durch Leerzeichen oder Punkte, der Spieler durch ein Symbol wie P und das Ziel durch T. Bei jeder Bewegung des Spielers überprüft das Programm die angrenzenden Felder im Array, um Kollisionen mit Wänden zu verhindern. Dabei wird die aktuelle Position des Spielers im Array aktualisiert, und die vorherige Position wird als freies Feld zurückgesetzt.

```
8   // Symbole im Labyrinth
9   #define PLAYER 'P'     // Spieler
10  #define TREASURE 'T'   // Schatz
11  #define EMPTY '.'      // Leeres Feld
12  #define OBSTACLE 'O'   // Hindernis
```

Abbildung 3 Aufbau vom Labyrinth im Visual Studio Code in der .h Datei

Die Spiellogik wurde so implementiert, dass der Spieler vier Bewegungsrichtungen hat: oben, unten, links und rechts. Jede Bewegung wird zunächst auf ihre Gültigkeit geprüft, indem überprüft wird, ob das Ziel-Feld innerhalb des Labyrinths liegt und kein Hindernis enthält. Erst wenn die Bewegung gültig ist, wird die Position des Spielers im Array aktualisiert. Wird das Ziel erreicht, erkennt das Programm automatisch das Ende des Spiels und gibt eine entsprechende Erfolgsmeldung aus.

```
    // Prüfen der Eingabe und neue Position berechnen
    if (input == 'w' || input == 'W') newRow--;       // nach oben
    else if (input == 's' || input == 'S') newRow++;  // nach unten
    else if (input == 'a' || input == 'A') newCol--;  // nach links
    else if (input == 'd' || input == 'D') newCol++;  // nach rechts
    else return; // ungültige Eingabe, keine Bewegung
```

Abbildung 4 Input code aus der .c Datei

Zur Erhöhung der Flexibilität wurden zusätzliche Funktionen implementiert. So kann das Labyrinth dynamisch generiert werden, wobei verschiedene Schwierigkeitsstufen berücksichtigt werden. Das Spiel ist offen für Erweiterungen, wie neue Spielmodi, alternative Layouts oder eine spätere grafische Darstellung.

```
// Funktion: Platziert eine zufällige Position im Labyrinth
void placeRandom(int *row, int *col) {
    do {
        *row = rand() % ROWS;
        *col = rand() % COLS;
    } while (labyrinth[*row][*col] == OBSTACLE || labyrinth[*row][*col] == PLAYER || labyrinth[*row][*col] == TREASURE);
}
```

Abbildung 5 Random Generator Labyrinth

Darüber hinaus wurde Wert auf eine klare Trennung der Aufgaben gelegt. Die Modulstruktur ermöglicht es, einzelne Teile unabhängig zu testen und zu verbessern. Zum Beispiel kann das

Labyrinth-Modul separat getestet werden, ohne dass Eingaben oder Spiellogik betroffen sind. Dies erhöht die Wartbarkeit und erleichtert spätere Erweiterungen erheblich.

Insgesamt sorgt die detaillierte Implementierung dafür, dass das Labyrinth-Spiel stabil, funktionsfähig und leicht verständlich ist. Die Kombination aus Modularität, robuster Spiellogik, fehlerrobuster Benutzereingabe und flexibler Labyrinth-Darstellung gewährleistet ein hochwertiges, gut strukturiertes Softwareprojekt, das sowohl für den aktuellen Einsatz als auch für zukünftige Erweiterungen vorbereitet ist.

```
// Labyrinth-Array
char labyrinth[ROWS][COLS] = {
    {'O',' ',' ',' ',' ',' ',' ',' ',' ',' '},
    {' ','O','O',' ',' ',' ',' ',' ','O',' '},
    {' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
    {' ',' ','O','O','O',' ',' ',' ',' ',' '},
    {' ',' ',' ',' ','O',' ',' ',' ',' ',' '},
    {' ',' ',' ',' ',' ','O',' ',' ',' ',' '},
    {' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
    {'O',' ',' ',' ',' ',' ',' ',' ',' ','O'}
};
```

Abbildung 6 Labyrinth Array

Damit das Labyrinth-Spiel stabil und zuverlässig läuft, wurden alle Funktionen genau getestet und mögliche Fehler abgefangen. Jede Funktion wurde einzeln geprüft, um sicherzustellen, dass sie richtig arbeitet. Getestet wurde zum Beispiel, ob der Spieler sich korrekt bewegen kann, Hindernisse wie Wände richtig erkannt werden und das Spiel endet, wenn das Ziel erreicht wird.
Besonders wichtig war die **Fehlerbehandlung**. Das Programm erkennt falsche Eingaben, wie ungültige Tasten oder Bewegungen in Wände, und reagiert darauf, ohne abzustürzen. Der Spieler kann eine neue Eingabe machen. So bleibt das Spiel stabil, auch wenn Fehler passieren.
Die **Qualitätssicherung** wurde durch die modulare Struktur erleichtert. Jedes Modul – Labyrinth-Erstellung, Spiellogik oder Eingaben – konnte einzeln getestet werden. So lassen sich Fehler leichter finden und beheben. Alle Tests und Änderungen wurden dokumentiert, damit nachvollziehbar ist, was geprüft wurde und welche Ergebnisse erzielt wurden.
Durch diese Tests, die Fehlerbehandlung und die sorgfältige Qualitätssicherung läuft das Labyrinth-Spiel stabil und zuverlässig. Gleichzeitig ist es so aufgebaut, dass neue Funktionen oder Erweiterungen später einfach hinzugefügt werden können.

6 Fazit

Das Labyrinth-C-Projekt wurde erfolgreich abgeschlossen und erfüllt alle gestellten Anforderungen. Das entwickelte Spiel erlaubt es dem Spieler, sich durch das Labyrinth zu bewegen, Hindernisse zu umgehen und das Ziel zu erreichen. Durch die modulare Struktur ist der Code übersichtlich, gut wartbar und leicht erweiterbar.

Die Spiellogik funktioniert zuverlässig: Bewegungen werden korrekt überprüft, Kollisionen mit Wänden werden verhindert und das Spiel erkennt das Erreichen des Ziels automatisch. Ungültige Eingaben werden abgefangen, wodurch das Programm stabil bleibt. Verschiedene Tests und die sorgfältige Fehlerbehandlung haben sichergestellt, dass das Spiel in allen Szenarien zuverlässig arbeitet.

Darüber hinaus ist das Projekt gut dokumentiert. Die einzelnen Module, die Spiellogik, die Datenstrukturen und die Entwicklungsprozesse wurden klar beschrieben. Dies ermöglicht es, den Entwicklungsweg nachzuvollziehen und erleichtert spätere Erweiterungen, wie neue Labyrinth-Layouts, zusätzliche Spielmodi oder sogar eine grafische Oberfläche.

Insgesamt liefert das Projekt ein stabiles, funktionsfähiges und gut strukturiertes Labyrinth-Spiel. Es demonstriert die Vorteile modularer Programmierung, robuster Fehlerbehandlung und sorgfältiger Qualitätssicherung. Gleichzeitig bietet es eine solide Grundlage für zukünftige Anpassungen und Weiterentwicklungen, wodurch das Projekt sowohl als Lernprojekt als auch als Basis für weiterführende Entwicklungen genutzt werden kann.

```
Super! Du hast den Schatz gefunden!
```

Abbildung 7 Schlusssatz

6.1 Persönliches Fazit:

Während dem ganzen Prozedere habe ich sehr viel gelernt, ich kam nochmal mit allen Funktionen, die ich in den 2. Programmtiertechnik Modulen gelernt habe und konnte die meisten umsetzen.

6.1.1 Zeitmanagement:

Ich habe mir zu oft zwischen den verschiedenen Arbeitstagen zu viel Zeit gelassen, und musste oft wieder sehr viel machen, um reinzukommen. Es wäre besser gewesen hätte man das an einem Stück bzw. in einer Woche gemacht.

6.1.2 Allgemein:

Die Aufgabe war nicht leicht, aber ich finde, dass ich das gemäss Aufgabenstellung recht gut gelöst habe. Ich bin zufrieden und freue mich auf zukünftig mehr Challenges in dem Bereich!

7 Zusätzliche Dateien

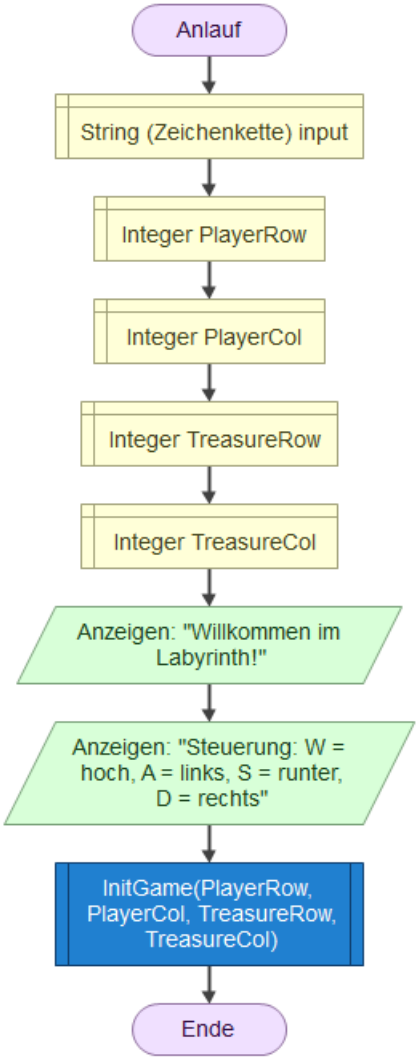

Abbildung 7 Hauptprogramm

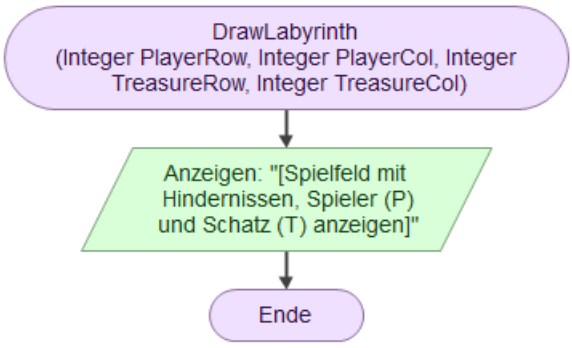

Abbildung 8 Labyrinth darstellen

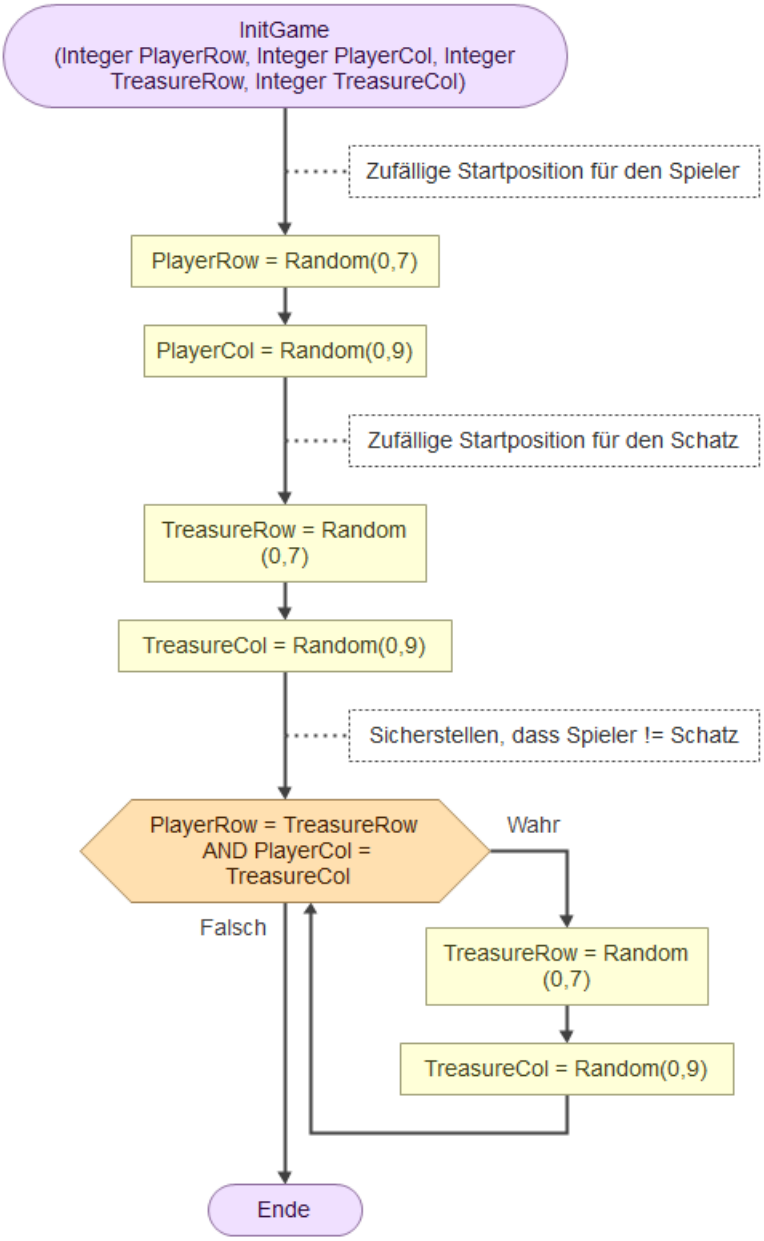

Abbildung 9 Randomgenerator

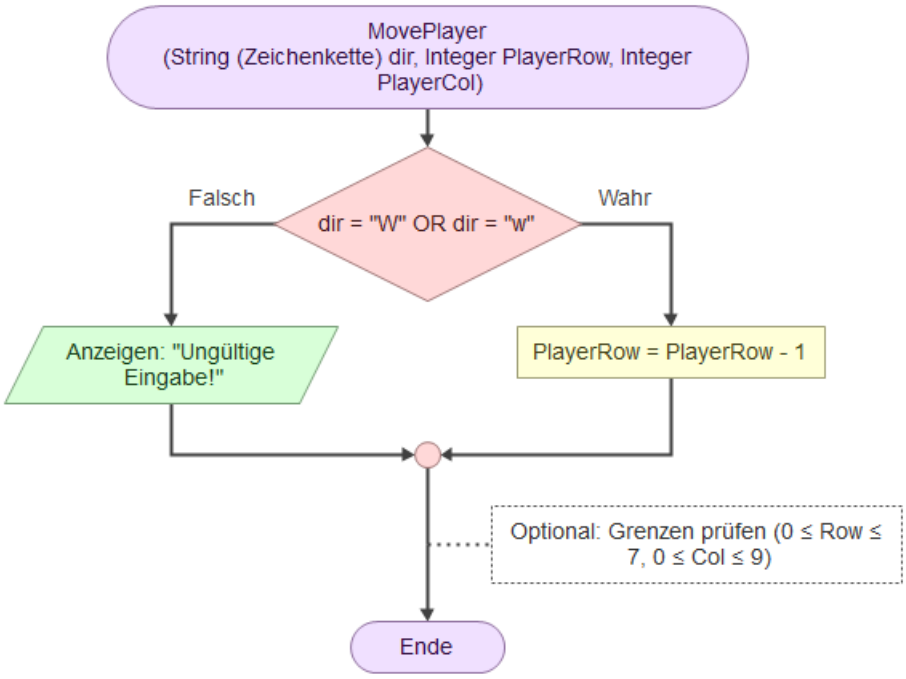

Abbildung 10 Spielerbewegung