ipso! Bildung seit 1876

Praxisarbeit Programmiertechnik

Labyrinth Spiel in C von Cristian Cubas

ipso! Bildung seit 1876

Inhaltsverzeichnis

1 Managment Summary

Im Rahmen dieser Praxisarbeit wurde ein Konsolenspiel in der Programmiersprache C entwickelt. Ziel war es, die im Unterricht erlernten Grundlagen der Programmierung praktisch anzuwenden und ein vollständiges Projekt umzusetzen.

Das Projekt wurde in mehreren Schritten realisiert: Zunächst erfolgte die Planung des Spielkonzepts und die Strukturierung des Programms. Anschließend wurde das Spiel modular programmiert, wodurch Übersichtlichkeit und Wartbarkeit sichergestellt wurden. Die Grundversion ermöglichte es, einen Spieler durch ein Labyrinth zu steuern und einen Schatz zu erreichen. In weiteren Schritten wurden zusätzliche Funktionen wie Schrittzähler, Highscore, Menüführung, Farbgestaltung und eine Siegesmeldung ergänzt.

Besonderer Wert wurde auf gut dokumentierten und stabilen Programmcode gelegt. Das Spiel wurde ausführlich getestet, um falsche Eingaben und Abstürze zu vermeiden und die Siegbedingung zuverlässig auszulösen.

Das Ergebnis ist ein funktionierendes und stabiles Konsolenspiel, das die gestellten Anforderungen erfüllt. Gleichzeitig bot die Arbeit wertvolle Lernerfahrungen im Bereich Projektplanung, Modularisierung und Programmierlogik, die für zukünftige Softwareprojekte von Nutzen sind.

2 Anforderung und Aufgabenstellung

Die Aufgabenstellung dieser Praxisarbeit bestand darin, ein Konsolenspiel in der Programmiersprache C zu entwickeln. Im Zentrum stand ein Labyrinth-Spiel, bei dem der Spieler durch Tastatureingaben das Labyrinth durchqueren und einen Schatz finden muss. Das Projekt ist Teil des Fachs Programmiertechnik A & B und sollte die im Unterricht behandelten Inhalte praktisch anwenden.
Ziel war es nicht nur, ein funktionierendes Spiel zu programmieren, sondern auch grundlegende Konzepte der Sprache C sinnvoll einzusetzen, darunter Datenstrukturen, Algorithmen, Programmsteuerung, Modularisierung, Pointer und dynamische Speicherverwaltung.

Funktionale Anforderungen:
– Anzeige des Labyrinths in der Konsole
– Steuerung des Spielers über die Tasten W, A, S und D
– Korrekte Kollisionsprüfung (Hindernisse dürfen nicht durchquert werden)
– Siegbedingung: Erreichen des Schatzes beendet das Spiel
– Menüführung beim Programmstart

Nicht-funktionale Anforderungen:
– Saubere Modularisierung (mehrere Dateien, klare Struktur)
– Fehlertoleranz bei Eingaben (keine Abstürze bei falscher Eingabe)
– Lesbarkeit und Dokumentation des Codes (Kommentare, klare Benennungen)
– Testbarkeit und Stabilität des Programms

Damit wurde ein Projektziel verfolgt, das sowohl die praktischen Programmierfähigkeiten fördert als auch die im Unterricht vermittelten Prinzipien der Softwaretechnik widerspiegelt.

Zur Veranschaulichung ist in Abbildung 1 ein Beispiel für die Darstellung des Labyrinths in der Konsole dargestellt.

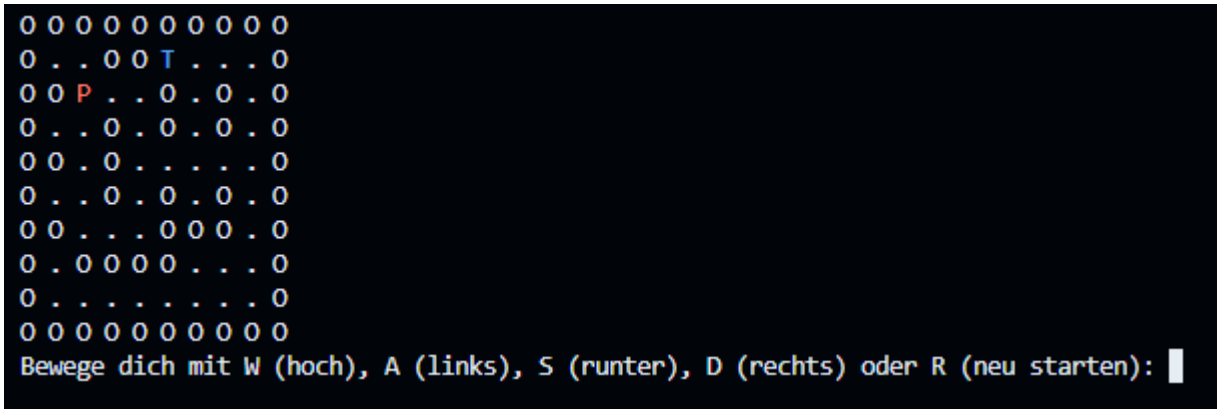

Abbildung 1: Beispielansicht des Labyrinths in der Konsole

3 Design

Bevor die Programmierung begann, wurde ein klares Konzept für das Spiel erstellt. Ziel war es, den Ablauf von Anfang bis Ende zu strukturieren und die Komplexität durch ein einfaches Design handhabbar zu machen.

Das Design gliederte sich in drei Hauptphasen:

1. Initialisierung
– Erzeugen des Spielfelds (Labyrinth als zweidimensionales Array)
– Setzen der Startposition des Spielers
– Platzieren des Schatzes (zufällig zur Steigerung der Spielbarkeit)

2. Spielschleife
– Steuerung über die Tasten W, A, S und D
– Prüfung der Eingabe auf Gültigkeit (keine Bewegung durch Hindernisse oder Wände)
– Aktualisierung und Ausgabe des Spielfelds

3. Spielende
– Nach jedem Zug wird überprüft, ob der Schatz erreicht wurde
– Bei Erfolg: Spielende mit Siegesmeldung und Statistiken (z. B. Schrittanzahl, Highscore)

Zur Symbolwahl wurden einfache, klar erkennbare Zeichen genutzt:
– P = Spieler
– T = Schatz
– O = Hindernis

Später wurde zusätzlich eine Farbgestaltung ergänzt, um die Lesbarkeit zu verbessern.

Durch diese Struktur war jederzeit ein roter Faden vorhanden, wodurch Fehler früh erkannt und behoben werden konnten.

Zur Verdeutlichung des Spielablaufs wurden die einzelnen Schritte in einem Flowgorithm-Diagramm modelliert. Abbildung 2 zeigt die Initialisierung und den Start der Schleife, während Abbildung 3 den weiteren Ablauf mit Eingaben, Bewegungen, Schrittzählung und Siegbedingung darstellt.

Abbildung 2: Ablaufdiagramm - Initialisierung und Start der Schleife

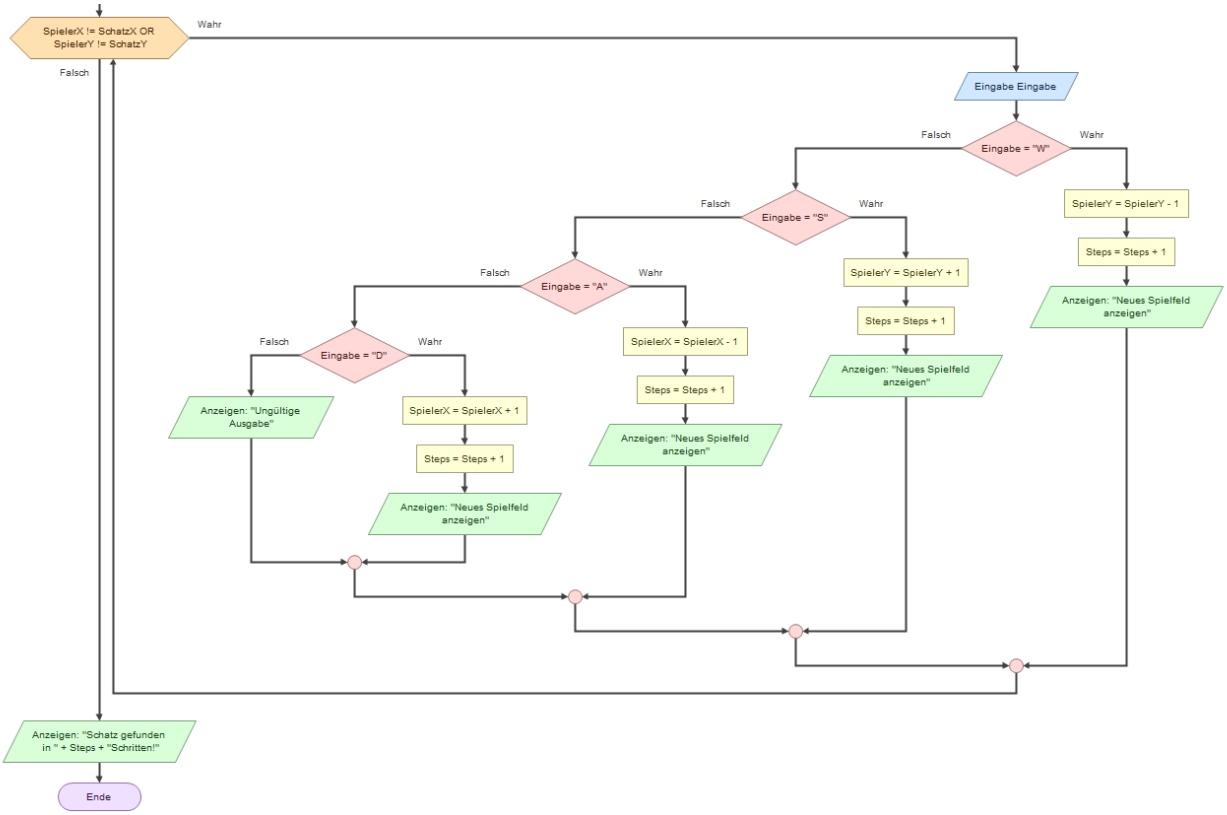

Abbildung 3: Ablaufdiagramm - Eingaben, Bewegungen, Schrittzähler, Spielfeld-Anzeige und Siegbedingung

Das Diagramm macht erkennbar, dass jede Eingabe geprüft, die Spielerposition angepasst und das Spielfeld neu ausgegeben wird, bis schließlich die Siegbedingung erreicht ist.

4 Implementierung

Die Implementierung erfolgte in der Programmiersprache C, wobei ich GitHub Codespaces als Entwicklungsumgebung nutzte. Von Anfang an war mir klar, dass das Projekt modular aufgebaut sein muss, damit Übersichtlichkeit und Wartbarkeit gewährleistet sind. Aus diesem Grund habe ich das Programm in mehrere Dateien aufgeteilt, die jeweils eine klar definierte Aufgabe übernehmen:

Modul	**Aufgaben**
main.c	Einstiegspunkt, enthält die Hauptschleife und steuert den Ablauf bis zum Spielende
game.c / game.h	Zentrale Spiellogik, Regeln und Spielfluss
player.c / player.h	Verwaltung der Spielerbewegung und Position im Labyrinth
labyrinth.c / labyrinth.h	Darstellung und Struktur des Spielfelds, inklusive Hindernissen
utils.c / utils.h	Hilfsfunktionen (z. B. Konsolenausgaben, kleine Berechnungen)
config.h	Zentrale Konfigurationswerte wie Spielfeldgröße und Symbole (Spieler, Schatz, Hindernis)

Da vor Beginn des Projekts keine Vorkenntnisse in C vorhanden waren, war der Einstieg zunächst anspruchsvoll. Viele theoretische Konzepte aus dem Unterricht – etwa Arrays, Pointer oder Modularisierung – wurden erst durch die praktische Anwendung verständlich. Besonders hilfreich war ein schrittweises Vorgehen, ergänzt durch zusätzliche Recherche im Internet.

Die Arbeit wurde bewusst auf mehrere Tage verteilt. So blieb ausreichend Zeit, Fehler zu analysieren und Lösungen auszuprobieren. Fehler wurden nicht als Rückschläge, sondern als Teil des Prozesses betrachtet. Durch systematisches Debugging und kleine Schritte konnte das Programm kontinuierlich verbessert werden.

Eine der größten Herausforderungen war der saubere Aufbau der Dateien. Neue Funktionen wie der Schrittzähler oder die zufällige Platzierung des Schatzes mussten so integriert werden, dass das Programm stabil bleibt. Der Moment, als es gelang, Spieler und Schatz zufällig zu positionieren, war ein entscheidendes Erfolgserlebnis und zeigte, dass die Konzepte verstanden und angewendet werden konnten.

Durch dieses systematische Vorgehen entstand eine funktionierende Grundversion, die anschließend mit zusätzlichen Features erweitert wurde. Das Ergebnis war ein stabiles, modular aufgebautes und erweiterbares Programm, das die Anforderungen vollständig erfüllte.

Abbildung 4 zeigt den Menücode aus main.c. Er stellt den Einstiegspunkt des Spiels dar, bei dem der Spieler zwischen Spielstart und Beenden wählen kann. Ungültige Eingaben werden ebenfalls abgefangen

```c

int main() {
    srand(time(NULL));

    char choice;
    do {
        clearScreen();
        printf("+------------------------+\n");
        printf("|  LABYRINTH-GAME-CUBI   |\n");
        printf("|  1 = Spiel starten     |\n");
        printf("|  2 = Beenden           |\n");
        printf("+------------------------+\n");
        printf("Deine Wahl: ");
        scanf(" %c", &choice);

        if (choice == '1') {
            startGame();
        } else if (choice == '2') {
            printf("Spiel wird beendet. Auf Wiedersehen!\n");
        } else {
            printf("Ungültige Eingabe!\n");
        }

    } while (choice != '2');

    return 0;
}
```

Abbildung 4: Menüauszug aus main.c - Einstieg in das Spiel

4.1 Projekttagebuch

4.1.1 Tag 1 – Planung

Am ersten Tag stand die Planung im Vordergrund. Ich habe die Aufgabenstellung gründlich gelesen und mir Notizen gemacht, welche Funktionen das Spiel auf jeden Fall enthalten muss: ein Labyrinth, die Spielerbewegung, eine Siegbedingung und eine klare Darstellung in der Konsole. Bereits hier habe ich entschieden, dass ich das Projekt modular aufbauen möchte, da ich wusste, dass der Code sonst schnell unübersichtlich wird.

Ich habe erste Skizzen des Labyrinths gezeichnet und ausprobiert, welche Symbole geeignet sind. Der Spieler sollte mit dem Buchstaben **P** dargestellt werden, der Schatz mit **T** und die Hindernisse mit **O**. Auch über die Steuerung habe ich nachgedacht. Mir war klar, dass ich die klassischen WASD-Tasten verwenden möchte, da sie in vielen Spielen Standard sind und sofort verständlich wirken.

Am Ende des ersten Tages hatte ich ein klares Bild davon, wie das Projekt aufgebaut sein soll. Besonders wichtig war mir, dass die Struktur stimmt, da ich mir eingestehen musste, dass ich als Anfänger in C ohne eine saubere Grundlage schnell den Überblick verlieren würde.

4.1.2 Tag 2- Grundversion

Am zweiten Tag begann ich mit der eigentlichen Umsetzung. Zuerst habe ich das Labyrinth als zweidimensionales Array aufgebaut und eine einfache Ausgabe in der Konsole programmiert. Das war der Moment, in dem das Projekt erstmals „lebendig" wurde, da ich ein sichtbares Spielfeld auf dem Bildschirm hatte.

Anschließend habe ich die Spielfigur implementiert. Es war für mich eine Herausforderung, die Eingaben korrekt einzulesen und dafür zu sorgen, dass sich der Spieler nur innerhalb des Labyrinths bewegen kann. Fehler gehörten hier zum Alltag: mal verschwand der Spieler aus dem Spielfeld, mal konnte er einfach durch Hindernisse hindurchlaufen. Doch Schritt für Schritt gelang es mir, diese Probleme zu lösen.

Am Ende des zweiten Tages funktionierte die Grundversion: Der Spieler konnte sich durch das Labyrinth bewegen, Hindernisse blockierten die Bewegung und sobald der Schatz erreicht wurde, endete das Spiel mit einer Siegesmeldung. Es war ein grosser Erfolgsmoment, da ich zum ersten Mal wirklich ein spielbares Ergebnis vor mir hatte.

4.1.3 Tag 3 – Erweiterung und Finalisierung

Der dritte Tag stand ganz im Zeichen der Erweiterungen. Die Grundversion funktionierte zwar, wirkte aber noch sehr roh. Deshalb wollte ich das Spiel benutzerfreundlicher und spannender machen.

Zunächst habe ich einen **Schrittzähler** eingebaut, der jede Bewegung des Spielers mitzählt. Danach folgte ein **Highscore-System**, das speichert, wie viele Schritte der Spieler bis zum Schatz benötigt hat. Das gab dem Spiel einen gewissen Wettbewerbsgedanken, da man nun versuchen konnte, mit weniger Zügen zum Ziel zu kommen.

Ein weiterer wichtiger Punkt war das **Spielermenü**. Beim Start sollte der Spieler die Möglichkeit haben, Regeln anzuzeigen oder das Spiel direkt zu beenden. Dadurch wirkte das Spiel nicht mehr wie ein „einfacher Code", sondern wie ein kleines Programm mit Menüführung.

Um die Darstellung attraktiver zu machen, habe ich **Farben** in der Konsole eingeführt: der Spieler wurde farbig markiert, ebenso der Schatz und die Hindernisse. Das erleichterte die Orientierung und machte das Spiel übersichtlicher. Zum Abschluss habe ich das **Siegesfenster** überarbeitet, sodass am Ende eine klare und motivierende Gewinnmeldung erschien, ergänzt durch Statistiken wie die benötigten Schritte.

Am Ende dieses Tages hatte das Spiel einen professionelleren Charakter und war deutlich interessanter zu spielen.

4.1.4 Tag 4 – Test und Feinschliff

Am vierten Tag habe ich das Spiel ausführlich getestet und kleinere Fehler behoben. Dabei habe ich verschiedene Szenarien ausprobiert: Spielerstart in der Nähe des Schatzes, Spielerstart weit entfernt, Schatz am Rand oder in einer Sackgasse. Außerdem habe ich absichtlich falsche Eingaben gemacht, um zu prüfen, ob das Spiel stabil bleibt.

Die Tests zeigten mir, dass die Fehlertoleranz funktioniert: Falsche Eingaben wurden abgefangen, das Spiel blieb laufstabil. Auch die Siegbedingung löste zuverlässig aus. Ich stellte fest, dass manche Ausgaben in der Konsole noch unübersichtlich wirkten, und habe deshalb Kleinigkeiten an der Formatierung und den Farben verbessert.

Zum Schluss habe ich alle Module nochmals durchgesehen und Kommentare ergänzt, um den Code lesbarer zu machen. Das war wichtig, weil ich weiss, dass nicht nur die Funktionalität, sondern auch die Verständlichkeit bewertet wird.

Am Ende des vierten Tages konnte ich zufrieden feststellen: Das Spiel war vollständig, getestet und dokumentiert. Für mich war es eine wertvolle Erfahrung, ein komplettes Projekt von der Idee bis zur Endversion selbst umzusetzen.

5 Test

Nachdem die Grundversion des Spiels funktionierte und zusätzliche Erweiterungen implementiert waren, wurde eine systematische Testphase durchgeführt. Ziel war es, die Stabilität, Funktionalität und Fehlertoleranz des Programms zu überprüfen. Dabei wurden sowohl reguläre Spielabläufe als auch Sonderfälle getestet.

Durchgeführte Tests:
– Normale Spielabläufe mit regulären Eingaben
– Startpositionen nahe und fern zum Schatz
– Platzierungen des Schatzes am Rand oder in Sackgassen
– Falsche Eingaben (z. B. Zahlen oder unzulässige Buchstaben)
– Längere Spielrunden mit vielen Schritten (Prüfung des Zählers und Highscores)

Testergebnisse:
Alle Tests zeigten, dass das Spiel stabil läuft. Der Spieler bewegt sich nur innerhalb der Spielfeldgrenzen, Hindernisse blockieren korrekt, und das Erreichen des Schatzes löst zuverlässig die Siegbedingung aus. Falsche Eingaben werden abgefangen, sodass das Programm nicht abstürzt. Auch Schrittzähler und Highscore-System funktionierten erwartungsgemäß. Kleinere Anpassungen an Farben und Ausgaben erhöhten zusätzlich die Lesbarkeit.

Testübersicht in Tabellenform:

Testfall	Eingabe	Erwartetes Ergebnis	Tatsächliches Ergebnis	OK/NOK
Normale Bewegung	W, A, S, D	Spieler bewegt sich in die gewünschte Richtung, bleibt im Spielfeld	Bewegung korrekt, keine Fehler	OK
Hindernisprüfung	Bewegung gegen „O“	Spieler darf Feld nicht betreten	Bewegung blockiert	OK
Schatzprüfung	Spieler auf Schatzfeld	Siegmeldung erscheint	Siegmeldung korrekt	OK
Ungültige Eingabe	Zahl oder Sonderzeichen	Meldung „Ungültige Eingabe“	Meldung erscheint, Spiel läuft weiter	OK
Schrittzähler	Mehrere Bewegungen	Schrittzahl erhöht sich bei jeder gültigen Bewegung	Schrittzahl wird korrekt hochgezählt	OK
Highscore-Test	Mehrere Runden spielen	Highscore speichert die geringste Schrittzahl	Highscore korrekt gespeichert	OK

Insgesamt bestätigte die Testphase, dass das Spiel die Anforderungen erfüllt und eine robuste, benutzerfreundliche Anwendung darstellt.

6 Lesson Learnd

Während der Arbeit an dieser Praxisarbeit konnten zahlreiche Erfahrungen gesammelt werden. Besonders deutlich wurde, wie wichtig eine saubere Projektstruktur ist. Durch die Aufteilung in Module ließ sich der Code leichter verstehen, erweitern und pflegen. Der anfängliche Impuls, alles in einer Datei umzusetzen, hätte schnell zu Unübersichtlichkeit geführt. Erst durch die Modularisierung wurde klar, wie entscheidend Struktur für die Softwareentwicklung ist.

Auch die im Unterricht behandelten Grundlagen – insbesondere Arrays, Pointer und Funktionen – wurden durch die praktische Umsetzung greifbar. Viele Konzepte, die zuvor abstrakt wirkten, erhielten durch das Projekt einen konkreten Anwendungsbezug. Beispiele hierfür waren die Verwaltung der Spielfeldmatrix oder die Implementierung der Spielerbewegung.

Ein weiterer wichtiger Lernpunkt war die Fehlertoleranz. Anfangs führten schon kleine Fehler oder falsche Eingaben zu Programmabstürzen. Mit zunehmender Erfahrung wurde jedoch ein systematisches Vorgehen beim Debugging entwickelt, wodurch das Spiel stabiler und robuster wurde.

Zudem erwies es sich als vorteilhaft, die Arbeit in kleine Etappen aufzuteilen. Durch regelmäßige Pausen blieb Zeit zum Reflektieren, wodurch Lösungsansätze schneller gefunden wurden. Besonders motivierend war der Moment, als es gelang, Spieler und Schatz zufällig zu positionieren. Dieses Erfolgserlebnis zeigte, dass die zentralen Konzepte verstanden und erfolgreich angewendet werden konnten.

Insgesamt hat die Arbeit verdeutlicht, dass auch kleine Projekte ein ideales Übungsfeld bieten, um Programmierlogik und Softwaretechnik praxisnah zu erlernen. Für zukünftige Projekte wurde insbesondere die Bedeutung einer guten Planung und Struktur erkannt, um effizienter und zielgerichteter arbeiten zu können.

7 Anhang

Abbildung A1: Startmenü des Spiels (Konsole)

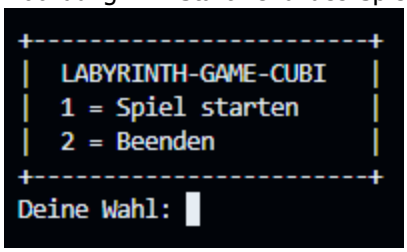

Abbildung A2: Spielfeldansicht mit Spieler (P), Schatz (T) und Hindernissen (O)

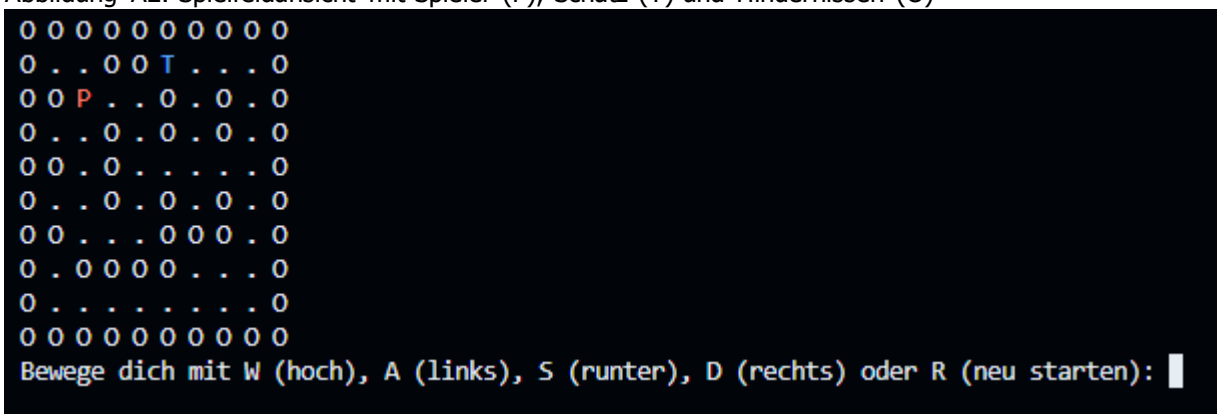

Abbildung A3: Gesamtübersicht des Spielablaufs als Flowgorithm-Diagramm

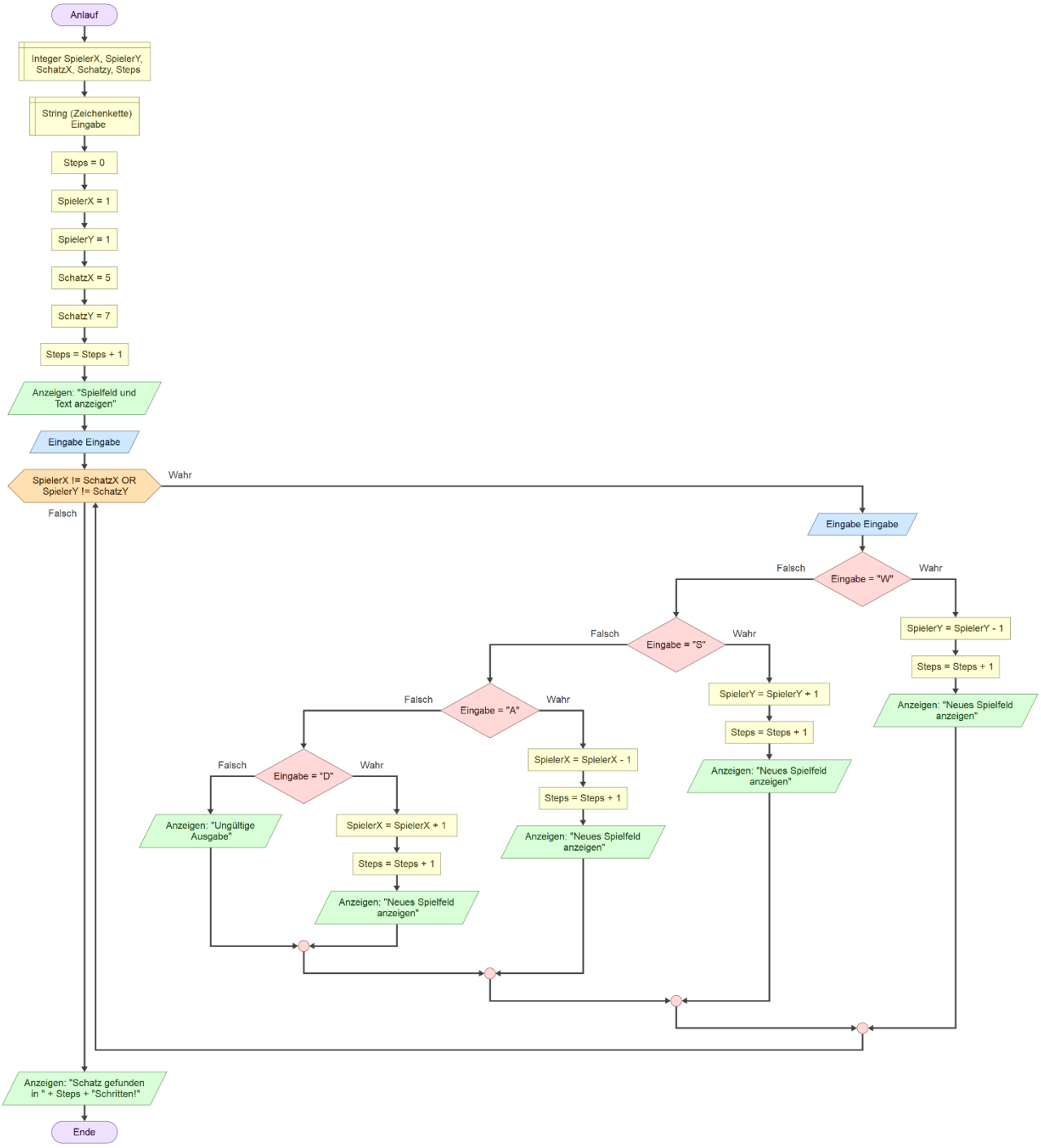

Abbildung A4: Siegmeldung mit Schrittanzahl

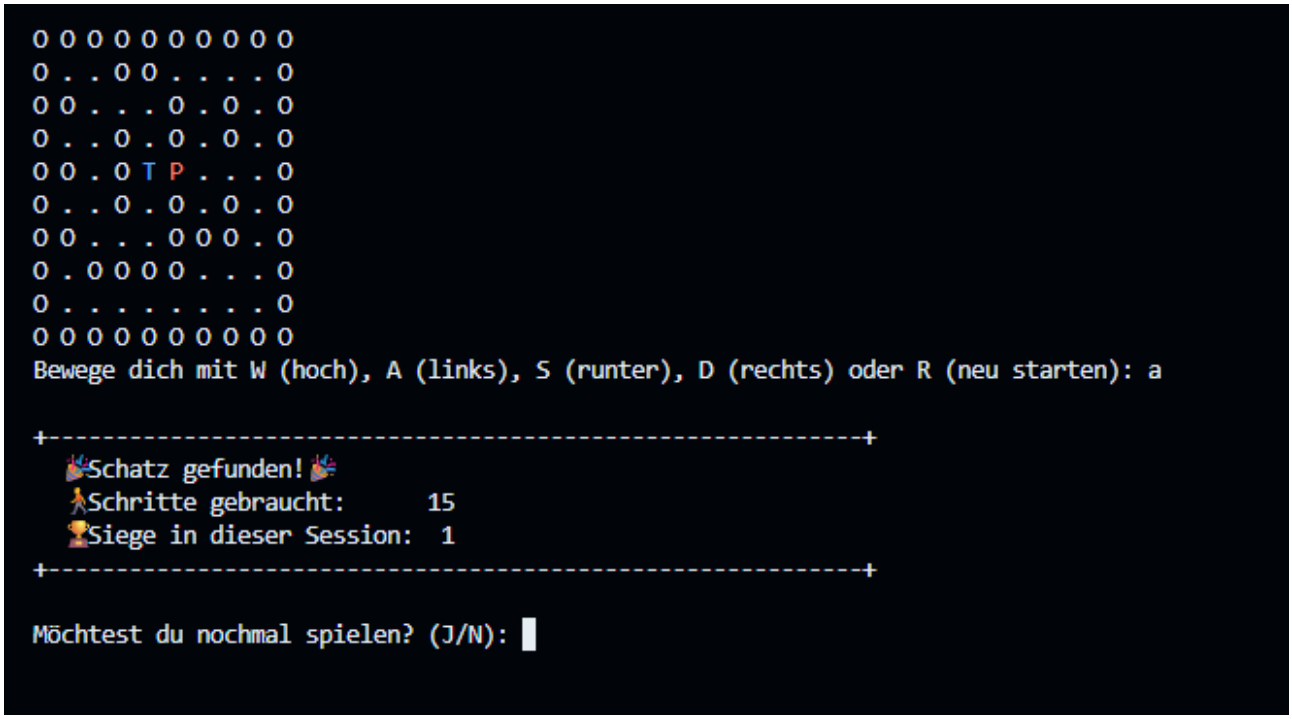

Abbildung A5: Kollision mit der Want

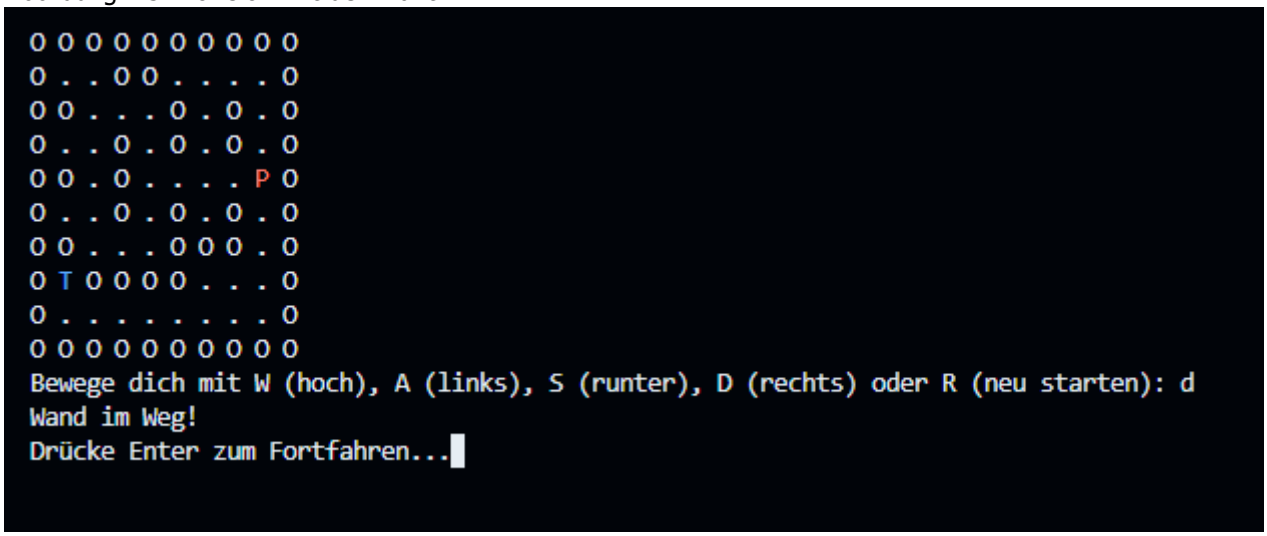

Abbildung A6: Codeausschnitt aus main.c - Spielschleife

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "game.h"
#include "utils.h"

int main() {
    srand(time(NULL));

    char choice;
    do {
        clearScreen();
        printf("+------------------------+\n");
        printf("|  LABYRINTH-GAME-CUBI   |\n");
        printf("|  1 = Spiel starten     |\n");
        printf("|  2 = Beenden           |\n");
        printf("+------------------------+\n");
        printf("Deine Wahl: ");
        scanf(" %c", &choice);

        if (choice == '1') {
            startGame();
        } else if (choice == '2') {
            printf("Spiel wird beendet. Auf Wiedersehen!\n");
        } else {
            printf("Ungültige Eingabe!\n");
        }

    } while (choice != '2');

    return 0;
}
```