ipso! Bildung seit 1876

Labyrinth (C)

Name: Endri Nuredini
Datum: 03.10.2025

Inhaltsverzeichnis

1 Einleitung

Diese Praxisarbeit wurde im Rahmen des Moduls Programmiertechnik A & B erstellt.
Ziel war es, die gelernten Grundlagen der Programmiersprache C praktisch anzuwenden und ein kleines Konsolenspiel zu entwickeln.
Das Spiel sollte ein Labyrinth simulieren, in dem ein Spieler nach einem Schatz sucht.
Die Umsetzung beinhaltete die Planung mit einfachen Diagrammen, die Entwicklung einer passenden Datenstruktur, die Implementierung des Spiels in mehreren Modulen sowie Tests zur Überprüfung der Funktionalität.

Die Arbeit wurde als Einzelprojekt durchgeführt und orientiert sich an einem zeitlichen Aufwand von etwa 25–30 Stunden.
Besonderer Wert lag auf der Anwendung von Algorithmen, Datenstrukturen, dynamischer Speicherverwaltung und robuster Benutzereingaben.
Mit dieser Praxisarbeit konnte ich mein Wissen in C vertiefen und gleichzeitig Erfahrungen mit der Dokumentation und Versionsverwaltung über GitHub sammeln.

2 Management Summary

Dieses Dokument beschreibt die Konzeption und Umsetzung eines einfachen, textbasierten Labyrinth-Spiels in C. Der Spieler ('P') navigiert durch ein zufällig generiertes Spielfeld mit Hindernissen ('O'), um einen Schatz ('T') zu erreichen. Die Arbeit demonstriert grundlegende Algorithmen, Datenstrukturen, Programmsteuerung, Modularisierung, dynamische Speicherverwaltung und Benutzereingaben.

3 Anforderungen & Aufgabenstellung

Ziel war die Entwicklung eines Konsolenspiels mit:
- Anzeige des Labyrinths
- Bewegung mit W/A/S/D (Enter)
- Siegbedingung (Schatz gefunden)
- Beendigung nach Sieg

Randbedingungen: Einzelarbeit, 25–30 Stunden, Einsatz von C, GitHub, Office-Produkten sowie eine PDF/Word-Dokumentation.

4 Design

Das Spiel nutzt eine strukturierte Datenrepräsentation:
- Struktur Maze: Größe, dynamisches Raster, Spieler- und Schatzkoordinaten
- Module: game (Logik/Rendering), maze (Erzeugung), input (Eingabe), utils (Hilfsfunktionen)

Labyrinth wird als dynamisch allokiertes char** (2D-Array) geführt.
Hindernisse = 'O', leere Felder = ' '. Spieler und Schatz werden zufällig gesetzt.

Ablauf (Textdiagramm): Start -> Maze init -> Zeichnen -> Eingabe -> Bewegung prüfen -> Siegbedingung -> Ende.
Ungültige Eingaben werden erkannt und abgewiesen.

5 Implementierung

- game.c/h: Initialisierung, Zeichnen, Regeln, Clear-Screen, Zufallsbereich
- maze.c/h: Platzierung von Hindernissen, Spieler, Schatz
- input.c/h: Eingabe parsen, robustes Lesen
- utils.c/h: Hilfsfunktionen (Sleep)

Speicherverwaltung via malloc/free, Portabilität durch Clear-Screen (Windows/Linux). Build via Makefile.

6 Test

Manuelle Tests:
- Start mit Standardparametern: funktioniert
- Bewegung in Hindernisse/Wände: wird korrekt verhindert
- Schatz erreicht: Spielende mit Siegmeldung

Beispieltestfälle:
- Eingabe 'x' -> Ungültig
- ./labyrinth 8 10 0 -> kein Hindernis
- ./labyrinth 12 24 35 -> dichtes Feld, aber lauffähig

7 Lessons Learned

- Robuste Eingabeprüfung spart Zeit
- Eine klare Datenstruktur (Maze) reduziert Kopplung
- Kleine Tests = effektiver
- Konsolen-UX hängt stark von Details (Buffering, Clear-Screen) ab

8 Wöchentliche Projektdokumentation

Woche 1: Aufgabenstellung gelesen, Datenstruktur entworfen, Prototyp für Labyrinth erstellt, Repo initialisiert.

Woche 2: Module implementiert, Eingabeprüfung hinzugefügt, manuelle Tests durchgeführt, Dokumentation geschrieben, finale Version am 03.10.2025 committet.

9 Anhang

- Quellcode im GitHub-Repository
- Beispielaufrufe: ./labyrinth, ./labyrinth 15 30 20
- Screenshots der Konsole können ergänzt werden.